Action Refinement in End-Based Choice Settings

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von
Diplom-Mathematiker Harald Fecher
aus Offenbach

Mannheim, 2003

Dekan: Professor Dr. Herbert Popp, Universität Mannheim
Referent: Professor Dr. Mila Majster-Cederbaum, Universität Mannheim
Korreferent: Professor Dr. Franz Stetter, Universität Mannheim

Tag der mündlichen Prüfung: 7. Juli 2003

Contents

List of Figures **5**

List of Tables **7**

1 Introduction **9**

1.1 Reactive Systems for Concurrency 10

1.2 Hierarchical System Design 12

1.3 End-Based and Start-Based View of the Choice Operator 13

1.4 Contributions of this Thesis 14

1.5 Outline of this Thesis 15

2 Preliminaries **17**

2.1 Notations 17

2.1.1 Set 17

2.1.2 Relation and Function 18

2.2 Transition System 20

2.3 Complete Partial Orders 22

2.4 Approximation Closedness 23

2.4.1 Proofs 25

3 Standard Action Refinement **29**

3.1 Different Approaches 29

3.2 Syntax 30

3.3 Denotational True Concurrency Semantics for $\mathsf{PA}_{\mathsf{sr}}$ 31

3.3.1 Bundle Event Structures 31

3.3.2 Closed Bundle Event Structures ($\mathbf{CBES}$) 34

3.3.3 Operators on $\mathbf{CBES}$ 36

3.3.4 Denotational Meaning for $\mathsf{PA}_{\mathsf{sr}}$ 38

3.4 Operational Semantics for $\mathsf{PA}_{\mathsf{sr}}$ . . . 40
3.4.1 Modified Operational Semantics . . . 44
3.5 Discussion . . . 45
3.6 Proofs . . . 46
3.6.1 Proof of Theorem 3.10 . . . 46
3.6.2 Proof of Proposition 3.18 . . . 47
3.6.3 Proof of Theorem 3.25 . . . 48

4 Modeling the End-Based View in $\mathbf{CBES}$ **57**
4.1 An End-Based Refinement Operator on $\mathbf{CBES}$ . . . 57
4.2 Equivalences . . . 60
4.2.1 Standard Equivalence Notions . . . 60
4.2.2 ICT-Equivalence on $\mathbf{CBES}$ . . . 62
4.2.3 UI-Bisimilarity on $\mathbf{CBES}$ . . . 64
4.2.4 FUI-Bisimilarity on $\mathbf{CBES}$ . . . 65
4.2.5 Comparison of Equivalences . . . 66
4.2.6 Coarsest Congruence . . . 67
4.3 Discussion . . . 68
4.4 Proofs . . . 69
4.4.1 Proof of the Congruence Results . . . 69
4.4.2 Proof of Proposition 4.20 . . . 75
4.4.3 Proof of Proposition 4.22 . . . 77

5 Terminating by Action Execution **79**
5.1 Motivation . . . 79
5.2 Syntax . . . 80
5.3 Operational Semantics for $\mathsf{PA}_{\mathsf{st}}$ . . . 81
5.4 Denotational Semantics for $\mathsf{PA}_{\mathsf{st}}$ . . . 83
5.4.1 Termination Bundle Event Structure ($\mathbf{TBES}$) . . . 84
5.4.2 Extended Termination Bundle Event Structure ($\mathbf{ETBES}$) . . . 85
5.4.3 Operators on $\mathbf{ETBES}$ . . . 90
5.4.4 Denotational Meaning for $\mathsf{PA}_{\mathsf{st}}$ . . . 93
5.4.5 Extended Termination Precursor Event Structures ($\mathbf{ETPES}$) . . . 94
5.4.6 Correspondence between $\mathbf{ETBES}$ and $\mathbf{ETPES}$. . . . 97
5.5 Discussion . . . 98
5.6 Proofs . . . 98

5.6.1 Proof of Theorem 5.28. 98
5.6.2 Proof of Theorem 5.36. 103
5.6.3 Proof of Proposition 5.39. 104
5.6.4 Proof of Theorem 5.40. 106

6 End-Based View in $\mathbf{ETBES}$ 109
6.1 An End-Based Refinement Operator on $\mathbf{ETBES}$ 109
6.2 Equivalences for $\mathbf{ETBES}$ 110
6.2.1 ICT-Equivalence on $\mathbf{ETBES}$ 111
6.2.2 FUI-Equivalence on $\mathbf{ETBES}$ 112
6.2.3 Comparison of Equivalences 114
6.3 Discussion 114
6.4 Proofs 115
6.4.1 Proof of Lemma 6.2 115
6.4.2 Proofs of the Coarsest Congruence Results 115

7 Start-Based together with End-Based Choice 121
7.1 Motivation 121
7.2 Syntax 122
7.3 Denotational Semantics for $\mathsf{PA}_{\mathsf{se}}$ 122
7.3.1 Start-End Bundle Event Structures ($\mathbf{SEBES}$) 122
7.3.2 Operators on $\mathbf{SEBES}$ 125
7.3.3 Denotational Meaning for $\mathsf{PA}_{\mathsf{se}}$ 127
7.4 Operational Semantics for $\mathsf{PA}_{\mathsf{se}}$ 127
7.5 Consistency of the Semantics for $\mathsf{PA}_{\mathsf{se}}$ 135
7.6 Equivalence 137
7.7 Axiomatization 138
7.7.1 Soundness 142
7.7.2 Completeness 145
7.8 Proofs 146
7.8.1 Proof of the Consistency Results 146
7.8.2 Proof of the Congruence Results 159
7.8.3 Proof of Theorem 7.34 160
7.8.4 Proof of Theorem 7.39 161

8 Conclusion 165

Bibliography **167**

Index **179**

Zusammenfassung **185**

List of Figures

1.1 Nuclear Power Plant Example 11

2.1 A Transition System Example 21
2.2 Trace-Equivalent but not Bisimilar Transition Systems 22

3.1 A Bundle Event Structure 32
3.2 An Extended Bundle Event Structure 33
3.3 Transition System Derived from $\mathbf{CBES}$ 36
3.4 Illustration of the Ref^s Operator 38
3.5 Examples of the Denotational Semantics of $\mathsf{EXP}_{\mathsf{sr}}$ 39
3.6 Example of a Process Derivation with respect to $\longrightarrow^s_{\mathrm{decl}}$ 43

4.1 Start-Based versus End-Based Refinement 58
4.2 End-Based Refinement in $\mathbf{CBES}$ (1) 60
4.3 Trace Equivalent but not Bisimilar cbes 61
4.4 End-Based Refinement in $\mathbf{CBES}$ (2) 62
4.5 Some Closed Bundle Event Structures 62
4.6 Non ICT-Equivalent cbes 63
4.7 ICT-Equivalent cbes 63
4.8 ICT-Equivalent and UI-Equivalent cbes 65
4.9 FUI-Equivalence Differs from UI-Equivalence 66
4.10 Relations Between the Equivalences 67
4.11 Counterexample of Coarsest Congruence 68

5.1 Some Extended Termination Bundle Event Structures 87
5.2 Transition System Derived from $\mathbf{ETBES}$ 89
5.3 Some Extended Termination Precursor Event Structures 95
5.4 Transition System Derived from $\mathbf{ETPES}$ 96

6.1 End-Based Refinement in **ETBES** 111
6.2 Non ICT-Equivalent eTbes 112
6.3 ICT-Equivalent eTbes 112
6.4 FUI-Equivalent eTbes (1) 113
6.5 FUI-Equivalent eTbes (2) 113
6.6 Relations Between the Equivalences 114

7.1 Some Start-End Bundle Event Structures 124
7.2 Illustration of the Stack Technique 128

8.1 Relations Between the Equivalences 165
8.2 Hierarchy of Event Structures 166

List of Tables

3.1 Transition Rules for $\longrightarrow^{s}_{\mathrm{decl}}$ 42
3.2 Modified Transition Rules for $\longrightarrow^{s}_{\mathrm{decl}}$ 45
3.3 Event-Based Transition Rules with respect to $\longrightarrow^{s}_{\mathrm{decl}}$ 50

5.1 Transition Rules for $\longrightarrow^{t}_{\mathrm{decl}}$ 82
5.2 Event Based Transition Rules with respect to $\longrightarrow^{t}_{\mathrm{decl}}$ 100

7.1 Transition Rules for $\longrightarrow^{c}_{\mathrm{decl}}$ (1) 132
7.2 Transition Rules for $\longrightarrow^{c}_{\mathrm{decl}}$ (2) 133
7.3 Transition Rules for $\longrightarrow^{c}_{\mathrm{decl}}$ (3) 134
7.4 Transition Rules for $\longrightarrow^{z}_{\mathrm{decl}}$ (1) 140
7.5 Transition Rules for $\longrightarrow^{z}_{\mathrm{decl}}$ (2) 141
7.6 Axioms for the Non-Refinement Operators 143
7.7 Axioms for the Refinement Operators 144
7.8 Event Based Transition Rules with respect to $\longrightarrow^{c}_{\mathrm{decl}}$ (1) 147
7.9 Event Based Transition Rules with respect to $\longrightarrow^{c}_{\mathrm{decl}}$ (2) 148
7.10 Event Based Transition Rules with respect to $\longrightarrow^{c}_{\mathrm{decl}}$ (3) 149

Chapter 1

Introduction

Formal methods in computer science, like methods for specification and verification[1], have become more and more important [51, 55], as the complexity of programs and processes that have to be controlled by machines has enormously increased over the years. No system designer can consider the totality of a program/process of a system in detail. Therefore, communication between different people about programs/processes has to take place. This communication usually subject to misunderstandings. Hence, it is important to possess formalisms that allow to talk more precisely about programs/processes, for example about the properties they should satisfy. Such languages are called specification formalisms. They allow to describe processes or properties of programs. Examples of specification languages are

- *descriptive/property-based* formalisms. Typical examples are logical frameworks, e.g. [100, 123, 124, 167]. They have the advantage of being intuitive, concise and abstract, i.e. they only consider the relevant details.

- *imperative/operational-based* formalisms. Examples are transition systems [121, 179], process algebras [18, 27, 84, 108, 138] or finite automata [110, 154]. They have the advantage of being close to actual implementations. Especially process algebras can be considered as action-based programming languages. Consequently, design descriptions are less likely to omit required attributes of the intended design, as it is possible in the property-based approaches to specification.

For safety critical tasks, for example air traffic control or the supervision of nuclear power plants, it is essential to guarantee the correctness of the involved program, i.e. verification (e.g. [83, 105]) becomes necessary. Furthermore, it is important to eliminate system design errors as early as possible, since they produce huge costs [161]. Such an error reduction can be achieved by verification based on the *dual language approach* (see for example [101, 151]), i.e. by using different description languages (a property- together with an operational-based one) in the system design phase and verifying their consistency, e.g. by using *model checking* [24, 52].

[1] Verification means the correctness of a program relative to the considered mathematical model. It can never guarantee the correctness of a program running on a concrete computer, as this also depends on further circumstances.

For concurrent processes, i.e. systems where each process may proceed more or less independently, it is important to use formal methods, since their interaction is not easily handled, especially when communication or synchronization takes place. The formal methods that are important for this thesis are discussed in Section 1.1 and in Section 1.2. Section 1.1 introduces the kind of systems that are considered, namely reactive true concurrent systems. Section 1.2 motivates the necessity of hierarchical system design in specification formalism. The new formalism on which the thesis is based is illustrated in Section 1.3. The contribution of the thesis is given in Section 1.4, and the outline of the thesis is given in Section 1.5.

1.1 Reactive Systems for Concurrency

Reactive systems are systems whose behavior depends on the environment, which means that the environment is able to influence the future behavior of such a system. In other words, the environment interacts with the system. Consider for example a computer system that controls a nuclear power plant. It runs continuously until the environment demands a shut down, for example by human demand or because the temperature of the reactor reaches a critical state.

Reactive systems are usually modeled by means of actions in order to describe the different activities of the system and of the environment, i.e. the system communicates with the environment via actions. Actions are usually considered to be atomic [34, 112] (they have no intermediate states) and instantaneous, i.e. durationless. Consequently, they can only be observed at a specific point in time. Furthermore, actions are divided into observable actions and internal actions. The execution of observable actions depends on the environment, whereas the environment has no influence on the execution of internal actions.

In the above nuclear power plant example the actions may be: $a_1 \hat{=}$ 'the reactor runs for another minute', $a_2 \hat{=}$ 'a human being demands the shut down of the reactor', $a_3 \hat{=}$ 'the temperature of the reactor has reached a critical state' and $a_4 \hat{=}$ 'the reactor is shutting down'. The reactive system can be described by the causal dependencies of these actions. This is done for example by:

- describing all possible (finite) execution sequences of the system. This description is called the *trace semantics* of a system, e.g. [48, 107].
- describing the actions each state allows (those which may be executed at a state) together with the information to which state the execution of the actions will lead. Typical models are *labeled transition systems*, which are introduced in Section 2.2. The labeled transition system obtained from the above nuclear power plant example is depicted in Figure 1.1.

In this thesis, we will consider concurrent reactive systems. This means that actions may be executed in parallel. Different models of concurrency are for example presented in [179]. These models can be divided into two different approaches:

- *Interleaving*: Here the parallelism between actions means that the execution of these actions can happen in any order. This is useful, if e.g. a sequential program has to be specified where the programmer is allowed to have a much greater degree of flexibility in order to carry out the implementation. So the programmer may decide the order of the execution of the actions. Transition systems are typical interleaving models.

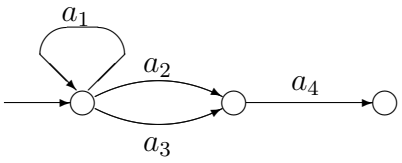


Figure 1.1: Nuclear Power Plant Example

- *True concurrency*: Here, actions may be executed simultaneously. This is necessary, e.g. if physical processes, where actions may happen simultaneously, have to be specified. When an implementation is realized on a system with more than one processor, true concurrency within the model is essential for specification. Typical models for true concurrent computations are for example *petri nets* [155], *event structures* [177], *pomsets* [153] and *causal trees* [61].

Process algebras are typically used as specification languages for concurrent reactive systems. The best known process algebras are

- the *Calculus of Communicating Systems* (CCS) [138],
- the *Communicating Sequential Processes* (CSP) [106, 108],
- the *Theory of Communicating Sequential Processes* (TCSP) [47],
- the *Algebra of Communicating Processes* (ACP) [25, 26] and
- the *Language of Temporal Ordering Specification* (LOTOS) [32].

It is necessary to give meanings, i.e. semantics, to languages. In general, three different kinds of semantics are used:

Operational semantics: It expresses the meaning of terms by execution steps, for example the observable behavior of a machine when it runs the program. The operational semantics of process algebras is typically given by transition rules in the style of Plotkin [9, 150], which yield transition systems.

Denotational semantics: Process expressions are interpreted in a mathematical model where the meaning of an expression is defined in terms of the meanings of its components, i.e. it is derived compositionally. Denotational true concurrency semantics of process algebras are given, for example in terms of event structures [36, 125, 145, 178].

Axiomatic semantics: Here, properties of process terms can be derived. This is typically done by axiom systems. In particular, axiomatic semantics are used for verification.

Standard process algebras can only describe the functional behavior of a system, i.e. the order of the execution of the actions. This is only suitable for a limited number of applications, since applications might be influenced by time and probability aspects. The examination of these further aspects is usually called performance analysis. For the purpose of performance analysis, process algebras are extended by

- *time*: Here, time may pass between actions. Some approaches allow actions with a fixed duration [10, 57, 99], i.e. actions are not necessarily instantaneous. There are discrete time versions [17, 103, 143] and dense time versions [17, 77, 128, 164, 175]. Classification properties of timed process algebras are given in [142].

- *priority*: Formalisms are added to the basic process algebras such that some executions of actions have priority over others, as for examples in [50, 53, 129]. Such approaches can be classified by static versus dynamic priorities and by global pre-emption versus local pre-emption [54].

- *probability*: Here, a probabilistic choice operator is used to model probabilistic behavior. This choice operator contains numbers, which determines the probability that the first (respectively the second) process is chosen. Examples are [88, 115, 129, 165].

- *stochastic*: The actions are considered to have a duration and this duration may vary. The duration of an action is given by a distribution. Examples of stochastic process algebras can be found in [11, 45, 104].

- *combinations*: For example in [29, 43], the above features are modeled in single process algebras.

1.2 Hierarchical System Design

Whether a system satisfies a formula is automatically decidable by means of model checking techniques if the underlying transition model has a finite number of states. Nevertheless, most systems have too many sates, hence model checking techniques cannot be used directly (the calculation takes too much time). This problem is usually called the *state explosion problem* [24, 52, 148].

Hierarchical system design [109] , a design that is developed on different levels of abstraction, can be used in order to handle the state space explosion problem. For example, a developer divides the intended design (usually complex) into various 'sub-designs'. He will develop the sub-designs by enriching them step by step with details, i.e. changing the level of abstraction. If the properties are preserved in some sense for every step between the levels of abstraction, it may be sufficient to verify the most abstract level, which has in general less states, to show the property of the concrete level. Design formalisms have to support hierarchical system design styles, as argued in [13, 55, 135, 169].

In reactive system models, hierarchical system design is usually done by action refinement operators[2] [7, 91, 98]. Intuitively, action refinement means the refinement of actions ($a, b, ...$), in a process P by more complex processes ($P_a, P_b, ...$). In other words, the occurrence of action a in the behavior of P is replaced by the behavior of P_a. How action refinement can be exploited for verification can be seen e.g. in [82, 111, 130, 132, 134, 162].

[2] In some approaches action refinement is not considered to be a usual operator as in [158], where it is considered to be an implementation relation.

1.3 End-Based and Start-Based View of the Choice Operator

In models of concurrent systems, actions are usually considered to be instantaneous, i.e. durationless, as mentioned before. However, if real time aspects of systems have to be modeled and/or action refinement operators are employed, we have to take into account that actions consume time[3].

A standard operator for modeling reactive systems is the choice between two processes ($P_1 + P_2$), i.e. if P_1 executes the next action, P_2 is disabled and vice versa. If durational actions are considered, it is not clear when an action has to be considered to be executed. In particular, it is not clear at which point in time an action triggers a choice – at the beginning, at the end or anywhere in the middle of its duration?

The consequence of this decision is illustrated by the following example. Consider a process that consists of a choice between actions a and b. The duration of a is 3 and the duration of b is 1. In addition, action a may start at time 0 and action b may start at time 1. If the choice is triggered at the beginning, then a triggers the choice before b starts. On the other hand, if the choice is triggered by the end of an action, the choice is triggered by b, i.e. a does not finish.

In the standard approach, a choice is triggered by the start of an involved event (action) [10, 99, 133, 141, 174]. But it is reasonable to consider approaches where choices are determined by the ending of actions:

- In stochastic approaches, it is common to consider a *race policy* approach [12, 29, 104], i.e. the fastest action triggers the choice. Consequently, a choice has to be triggered at the end of the action's duration, since it is usually not known a priori which action is the fastest.

- The end-based point of view is of interest for hierarchical system development, where complex activities are specified by single actions in the first system design steps. This is illustrated by the following example.

  Example 1.1 *Consider the example of a plane that runs into problems and has to land as fast as possible. Two airports (in the same city) come into consideration for the emergency landing. The pilot sends an SOS-signal to both airports. Both airports start their preparations for the emergency landing. The pilot will choose the airport that is the first to respond to be ready. On an abstract level the pilot can be modeled by*

  $$P_{ab} = send; ((ok_1; L_1) + (ok_2; L_2))$$

  where $send$ denotes the sending of the SOS-Signal, ok_i is the response of the i-th airport A_i (that runs in parallel to P_{ab} synchronizing over ok_i), and L_i denotes the landing on the i-th airport. Furthermore, ; denotes the sequential composition of two processes. The choice in P_{ab} is either triggered by ok_1 or by ok_2, as usual.

  In practice, the airports will send more detailed information, e.g. that the maneuvering area is free, fire service is ready, and so on. In other words, actions ok_1 and ok_2 are time-consuming. Then the choice in P_{ab} has to be considered as end-based, since the choice should be made when the first airport has completed its preparations.

[3] Action refinement operators can, for example, split an action into a start- and an end-action, hence the action's duration can be modeled in some sense.

This is easily understood when we consider the next system design phase, where the ok_i actions are specified in more detail, i.e. they are refined by a process M_i. The choice of the pilot is triggered when either M_1 or M_2 terminates and not when the first action is executed by M_1 or M_2. In particular, the actions of M_2 that are executed before the termination of M_1 remain visible, i.e. they are not made undone after the termination of M_1, and vice versa. This makes clear that the end-based choice can be viewed as some kind of parallelism, where M_1 and M_2 run in parallel until one of them terminates.

The possibility of late decisions is also motivated and examined in Z [166].

1.4 Contributions of this Thesis

The goal of this thesis is to make a first step to establish the end-based choice operator in reactive true concurrent systems. More precisely, we want to establish an end-based choice operator in untimed reactive systems that contain action refinement operators. This is done by investigating a process algebra that contains these operators and by giving a semantic foundation (operational, denotational, axiomatic) to this process algebra. Furthermore, the consequences of considering an end-based rather than a start-based choice are examined, in particular with respect to equivalence notions.

The contributions of this thesis are explained in more detail in the following. We have to give operational semantics to a process algebra that contains end-based choice and action refinement operators. This leads to some problems which we first consider in the start-based setting: Action refinement operators in event structures where the choice is considered as start-based are well established. A corresponding definition for the operational semantics of process algebras is not obvious when non-atomic action refinement is considered. The typical approach to the substitution of the refining process (either statically or dynamically) [5, 144], sometimes called *syntactical action refinement*, does not always correspond to the refinement of event structures [93]. In this thesis, a new possibility to define an operational semantics for action refinement that corresponds to the denotational semantics is given in a start-based choice setting. Here, it is not necessary to introduce new syntactic terms in order to give the operational semantics.

Furthermore, by considering the end-based view in bundle event structures, we have recognized that bundle event structures [125, 126] fail to be a complete partial order[4]. Therefore, we present a new technique in order to define complete partial orders for event structures that are based on the bundle technique. This is necessary in this thesis, since we use event structures that are based on the bundle technique as denotational semantics for our end-based process algebra. These new techniques will be applied first in the start-based setting.

The first step to the end-based approach is the definition of an action refinement operator on (extended) bundle event structures where the conflict relation is considered as end-based. Furthermore, new equivalences are defined in order to obtain equivalences that are congruences for the end-based action refinement operator, since the standard equivalences are no congruences for this operator. The new equivalences fail to be the coarsest with respect to trace and bisimulation equivalence. This results from the fact that processes terminate by the execution of a special termination action and not by the execution of the 'final' executed action.

[4]The theory of complete partial orders allows to define denotations of recursive processes.

Before we continue to present a process algebra with an end-based choice operator, we take a closer look at the termination philosophy that a process should terminate by the execution of its 'final' action and not (as usual) by an additional termination action. We will argue that this termination philosophy is especially of interest if process algebras with a disrupt operator (as it is implicitly the case for process algebras that contain action refinement and end-based choice operators) are considered. Therefore, two new kinds of event structures, which allow more general disabling, are given to obtain denotational semantics of process algebras that contain disruption and model termination by the execution of the 'final' action. One event structure models the 'non-disabling' of events rather than the disabling of events. This is done by making use of a witness relation. The other event structure is a generalization of Winskel's event structures. We show that there is consistency [19, 68] between the operational and the denotational semantics of a process algebra which contains disruption and which is based on the new termination philosophy. The expressive power of our new kind of event structure is also examined: We verify that these structures have the same expressive power and they are more expressive than the standard event structures with respect to event traces. Furthermore, we adapt the equivalences that are defined in the context of the end-based view to this new type of event structures. We show that one of the adapted equivalences is the coarsest congruence for the end-based refinement operator with respect to trace (respectively bisimulation) equivalence.

We argue that it is useful to have also a start-based choice operator whenever an end-based choice operator is considered, since an end-based choice can model a kind of start-based choice in the case of synchronized parallel execution. Therefore, we will finally consider a process algebra that includes a start-based and an end-based choice together with an action refinement operator. A class of event structures with two relations for disabling is introduced in order to give a denotational semantics. An operational semantics, which is consistent with the denotational semantics, is given. We define the coarsest congruence with respect to bisimulation equivalence and we present an axiom system for this equivalence. Moreover, we show that the axiom system is sound and complete for guarded and finite state processes.

Parts of this thesis are published in [78, 80].

1.5 Outline of this Thesis

Chapter 2 contains some preliminaries: First the notions used in this thesis are introduced. Then transition systems and parts of the partial orders theory are presented. The final section of Chapter 2 contains the results of the new approximation closedness property, which is later used in order to define classes of event structures that yield complete partial orders.

In Chapter 3, the standard action refinement operator, i.e. the one that is based on the start-based choice, is presented and examined. That chapter includes the new operational semantics and the modification of bundle event structures that yield a complete partial order.

The action refinement operator that considers an end-based choice is given in Chapter 4. It is defined on the modified extended bundle event structures mentioned before. Congruences for this refinement operator will be introduced. We show that none of the new equivalences is the coarsest one with respect to trace/bisimulation equivalence. It is also argued that extended bundle event structures are not appropriate to model the end-based view. In addition, some standard equivalences are summarized in this chapter.

In Chapter 5, we investigate a process algebra that includes a disrupt operator. Here, termination is determined by the execution of the 'final' action. In this chapter, the classes of event structures that allow more general disabling are introduced and their expressive power are examined. Chapter 5 also contains a denotational semantics in the class of event structures based on the witness approach. A consistency result between the operational and the denotational semantics is shown.

The end-based view, introduced in Chapter 4, is adapted in Chapter 6 to one of the classes of event structures presented in Chapter 5. Here, the adapted equivalences of Chapter 4 are the coarsest congruences with respect to trace (respectively bisimulation) equivalence. We also argue that some kinds of start-based choices can be modeled with an end-based choice operator together with a parallel operator where some actions have to be synchronized.

A process algebra that contains a start-based and an end-based choice operator at the same time is introduced in Chapter 7. A denotational semantics of this process algebra, which also contains a refinement operator, is presented there. Moreover, a consistent operational semantics is given. Chapter 7 also contains the definition and the axiom system of the coarsest congruence with respect to bisimulation equivalence.

Finally, a conclusion is given in Chapter 8.

Chapter 2

Preliminaries

2.1 Notations

In this section, we present some basic notations that are used in this thesis. Subsection 2.1.1 considers notions related to sets, whereas Subsection 2.1.2 considers notions related to relations and functions. In this section, M, M_1, M_2 and M_3 denote arbitrary sets.

2.1.1 Set

- $\mathbb{N}^+$ denotes the *positive natural numbers*, i.e. $\mathbb{N}$ without 0.
- $M_1 \backslash M_2 = \{m \in M_1 \mid m \notin M_2\}$
- $|M|$ denotes the *cardinality* of set M.
- M is *countable* if $|M| \leq |\mathbb{N}|$.
- $\mathcal{P}(M) = \{A \mid A \subseteq M\}$
- $\mathcal{P}_{fin}(M) = \{A \subseteq M \mid |A| < |\mathbb{N}|\}$
- $\mathcal{P}_{count}(M) = \{A \subseteq M \mid |A| \leq |\mathbb{N}|\}$
- $M^n = \underbrace{M \times \cdots \times M}_{n-\text{times}} = \{(m_1, \cdots, m_n) \mid m_i \in M\}$ where $n \in \mathbb{N}^+$.
- $M^\star$ denotes the *set of all strings* – including the *empty string* ϵ – over set M. Here, a *string* of M is a finite sequence of elements of M. We sometimes write strings $m_1 ... m_n$ where $m_i = m$ for all i as m^n.

  Furthermore, if $m \in M$, $\sigma \in M^\star$ with $\sigma = m_1 \cdots m_n$ and $i \in \mathbb{N}$ with $i \leq n$ then

  - the i-th element of σ is denoted by $\sigma[i]$.
  - $m \cdot \sigma = m m_1 \cdots m_n$

- $\sigma\backslash i$ denotes the deletion of the i-th element of σ, i.e. $\sigma\backslash i = m_1 \cdots m_{i-1} m_{i+1} \cdots m_n$.
- $\sigma \pm (i, m)$ denotes the replacement of the i-th element of σ by m, i.e. $\sigma \pm (i, m) = m_1 \cdots m_{i-1} m m_{i+1} \cdots m_n$.
- the length of σ is denoted by $|\sigma|$, i.e. $|\sigma| = n$.

The following definition introduces a universe of *events*. Events are used in the following to denote different occurrences of actions. We need this universe in order to guarantee that event structures, which are defined in the following chapters, are sets rather than classes. This enables us to apply the theory of complete partial orders, which is presented in Section 2.3, directly.

Definition 2.1 (Universe of Events) *Let $\bullet$, $\star_1$, $\star_2$ and $\star$ be arbitrary, but fixed, pairwise different symbols. Then the* universe of events*, denoted $\mathcal{U}$, is an arbitrary, but fixed, countable set such that $\bullet \in \mathcal{U}$, $\star \notin \mathcal{U}$ and $\forall e, e' \in \mathcal{U} : (e, e'), (\star_1, e), (\star_2, e), (\star, e), (e, \star) \in \mathcal{U}$*[1].

We use $s_1 ... s_n \bullet$ where $s_i \in \mathcal{U} \cup \{\star_1, \star_2, \star\}$ as an abbreviation for $(s_1, (s_2, ...(s_n, \bullet)...))$.

2.1.2 Relation and Function

- For any binary relation $\natural \subseteq M_1 \times M_2$ we write
  - $m_1 \natural m_2$ if and only if $(m_1, m_2) \in \natural$ and
  - $\_\natural m_2$ for the set $\{m \in M_1 \mid m \natural m_2\}$.
- $\mathrm{Id}^M \subseteq M \times M$ denotes the *identity relation*, i.e. $\mathrm{Id}^M = \{(m, m) \mid m \in M\}$. The index M is omitted if it is clear from the context.
- If $\natural_1 \subseteq M_1 \times M_2$ and $\natural_2 \subseteq M_2 \times M_3$ are two binary relations, then $\natural_1 \circ \natural_2$ denotes the binary relation given by $\{(m_1, m_3) \in M_1 \times M_3 \mid \exists m_2 \in M_2 : m_1 \natural_1 m_2 \wedge m_2 \natural_2 m_3\}$
- $\langle M, \sqsubseteq \rangle$ denotes the set M ordered by the partial order $\sqsubseteq$.
- $M_1 \to M_2$ (or $M_2^{M_1}$) denotes the set of all functions from M_1 to M_2. We denote that f is a function from M_1 to M_2 by $f : M_1 \to M_2$. The function from M_1 to M_2 that maps every element of M_1 to $m \in M_2$ is denoted by $\mathrm{cons}_m^{M_1 \to M_2}$, where index $M_1 \to M_2$ is omitted if it is clear from the context. Furthermore, if $f : M_1 \to M_2$ and $f'' : M_2 \to M_3$ then
  - $f'' \circ f$ is the function from M_1 to M_3 given by $(f'' \circ f)(m) = f''(f(m))$.
  - $f(M)$, where $M \subseteq M_1$, denotes the *image of M under f*, i.e. $f(M) = \{f(m) \mid m \in M\}$.
  - $f \restriction M$ where $M \subseteq M_1$ is the function from M to M_2 with $(f \restriction M)(m) = f(m)$ for any $m \in M$.

[1] It is clear that such an $\mathcal{U}$ exists.

  - $f[m_1 \to m_2]$ where $m_1 \in M_1$, $m_2 \in M_2$ is the function from M_1 to M_2 with $(f[m_1 \to m_2])(m) = \begin{cases} f(m) & \text{if } m \neq m_1 \\ m_2 & \text{otherwise} \end{cases}$.
  - if f is bijective, then f^{-1} denotes the *inverse function* of f, i.e. $f^{-1} : M_2 \to M_1$ with $f^{-1}(m_2) = m_1 \Leftrightarrow f(m_1) = m_2$.

- $M_1 \rightharpoonup M_2$ denotes the set of all *partial functions* from M_1 to M_2. We denote that f is a partial function from M_1 to M_2 by $f : M_1 \rightharpoonup M_2$. The partial function from M_1 to M_2 that is everywhere undefined is denoted by $\bot^{M_1 \rightharpoonup M_2}$. The partial function from M_1 to M_2 that maps every element from M_1 to $m \in M_2$ is denoted by $\mathrm{cons}_m^{M_1 \rightharpoonup M_2}$. The index $M_1 \rightharpoonup M_2$ is omitted in both cases if it is clear from the context.

  Furthermore, if $f, f' : M_1 \rightharpoonup M_2$, $f'' : M_2 \rightharpoonup M_3$ and $f''' : M_3 \rightharpoonup M_2$ then

  - the *domain* of f, denoted by $\mathrm{dom}(f)$, is the set $\{m \in M_1 \mid f(m) \text{ is defined}\}$.
  - We define $f \cup f'$ by considering f and f' as relations.
  - $f'' \circ f$ is the partial function from M_1 to M_3 given by $(f'' \circ f)(m) = f''(f(m))$ if $m \in \mathrm{dom}(f) \wedge f(m) \in \mathrm{dom}(f'')$ and undefined otherwise.
  - $f(M)$, where $M \subseteq M_1$, denotes the *image of M under f*, i.e. $f(M) = \{f(m) \mid m \in M\}$.
  - We write $f(m_1) \simeq f'''(m_3)$ to denote that $f(m_1)$ is defined $\Leftrightarrow$ $f'''(m_3)$ is defined $\wedge$ $f(m_1)$ is defined $\Rightarrow$ $f(m_1) = f'''(m_3)$.
    $f \simeq f'$ holds if and only if $\forall m_1 \in M_1 : f(m_1) \simeq f'(m_1)$.
  - $f \restriction M$ where $M \subseteq M_1$ is the partial function from M_1 to M_2 with $(f \restriction M)(m) \simeq f(m)$ whenever $m \in M$ and undefined otherwise.
  - $f[m_1 \to m_2]$ where $m_1 \in M_1$, $m_2 \in M_2$ is the partial function from M_1 to M_2 with $(f[m_1 \to m_2])(m) \simeq \begin{cases} f(m) & \text{if } m \neq m_1 \\ m_2 & \text{otherwise} \end{cases}$.
  - f is *injective* (*surjective*, *bijective*) from $M_1' \subseteq M_1$ to $M_2' \subseteq M_2$ if and only if $\mathrm{dom}(f) = M_1'$, $f(M_1') \subseteq M_2'$ and $f \restriction M_1'$ is an injective function from M_1' to M_2' (respectively surjective, bijective). We call f *injective* if f is injective between $\mathrm{dom}(f)$ and M_2.
  - if f is bijective, then f^{-1} denotes the *inverse partial function of f*, i.e. $f^{-1} : M_2 \rightharpoonup M_1$ with $f^{-1}(m_2) \simeq m_1 \iff f(m_1) \simeq m_2$.

- $M \to^{fin} \mathbb{N}$ denotes the set of all functions from M to the natural numbers that differs only finitely often from 0, i.e. $M \to^{fin} \mathbb{N} = \{f : M \to \mathbb{N} \mid |\{m \in M \mid f(m) \neq 0\}| < \infty\}$.

- $M_1 \rightharpoondown M_2$ denotes the set of all functions from M_1 to the set of all strings over M_2 that differs only finitely often from the empty string, i.e. $M_1 \rightharpoondown M_2 = \{f : M_1 \to M_2^\star \mid |\{m \in M_1 \mid f(m) \neq \epsilon\}| < \infty\}$.

  The function that maps every element of M_1 to the empty string of $M_2^\star$ is denoted by $\bot^{M_1 \rightharpoondown M_2}$. The index $M_1 \rightharpoondown M_2$ is omitted if it is clear from the context.

- π_i denotes the *projection* to the i-th component of a Cartesian product

- $\cong$ is used to denote isomorphism of structures, i.e. a structure preserving bijective function.

- A binary relation $\natural \subseteq M \times M$ is *preserved by* an operator $F : M \to M$ if and only if $m_1 \natural m_2 \Rightarrow F(m_1) \natural F(m_2)$. Furthermore, $\natural$ is *preserved by* an operator $F_n : M^n \to M$ if and only if $F_n(m_1, ..., m_n) \natural F_n(m'_1, ..., m'_n)$ whenever $m_i \natural m'_i$ for $i = 1, .., n$. F is called *monotonic* for the special case when $\natural$ is a partial order.

Definition 2.2 *Let $\equiv \subseteq M \times M$ and let $\mathcal{F}$ be a set of operators where for every $f \in \mathcal{F}$ there exists $i \in \mathbb{N}$ such that $f : M^i \to M$. Then*

- *$\equiv$ is a* congruence *for $\mathcal{F}$ if and only if $\equiv$ is an equivalence relation and $\equiv$ is preserved by all operators of $\mathcal{F}$*

- *$\equiv_c \subseteq M \times M$ is the* coarsest congruence *for $\mathcal{F}$ with respect to $\equiv$ if and only if*

  - *$\equiv_c \subseteq \equiv$*
  - *$\equiv_c$ is a congruence for $\mathcal{F}$*
  - *whenever $\equiv' \subseteq M \times M$ is a congruence for $\mathcal{F}$ such that $\equiv' \subseteq \equiv$ then $\equiv' \subseteq \equiv_c$*

2.2 Transition System

Labeled transition systems, originally introduced by [121] under the name 'named transition systems', represent a model to describe the behavioral character of a process. This is done by abstracting complex activities into a single action. The duration of actions is often neglected, i.e. the actions are considered to be instantaneous, in order to obtain a simpler, time independent model. Transition systems are used to describe to which state the execution of an action may lead.

Definition 2.3 (Transition System) *A* (labeled) transition system *is a quadruple $(S, L, \longrightarrow, \bar{s})$ with*

- *S, a non-empty set of* states

- *L, a set of* labels

- *$\longrightarrow \subseteq S \times L \times S$, a* transition relation

- *$\bar{s} \in S$, the* initial state.

We will write $p \xrightarrow{\gamma} q$ rather than $(p, \gamma, q) \in \longrightarrow$. The class of all transition systems is denoted by $\mathbf{TS}$.

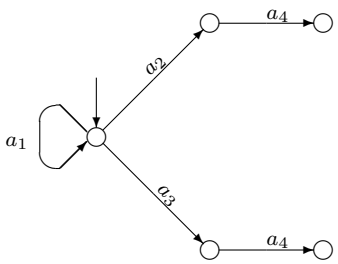


Figure 2.1: A Transition System Example

The intuitive meaning of $p \xrightarrow{\gamma} q$ is that the execution of γ in state p may lead to state q. It is not necessarily uniquely determined to which state the execution of γ in state p leads, since it is possible that $p \xrightarrow{\gamma} q'$ is another transition of the transition system. Examples of transition systems are shown in Figure 1.1 and in Figure 2.1, where the initial state is marked by an arrow without a source.

Transition systems, as well as other models, are often considered as too concrete descriptions. Therefore, equivalences are defined on transition systems in order to identify those which we consider to display the same behavior in some sense.

Two basic equivalences have been defined for transition systems: *trace equivalence* [108], which considers the possible sequences of observable behavior, and *bisimulation equivalence* [136], which also takes the branching structure into account.

Definition 2.4 (Trace Equivalence) *The set of* traces *of a transition system* $(S, L, \longrightarrow, \bar{s})$ *is defined by*

$$T(S, L, \longrightarrow, \bar{s}) = \{(\gamma_i)_{i<n} \mid n \in \mathbb{N} \wedge \exists s_0, \cdots, s_n \in S \wedge s_0 = \bar{s} \wedge \forall i < n : s_i \xrightarrow{\gamma_i} s_{i+1}\}.$$

We sometimes write $T(\bar{s})$ *if* $S, L, \longrightarrow$ *are clear from the context.*

Two transition systems $(S, L, \longrightarrow, \bar{s})$ *and* $(S', L, \longrightarrow', \bar{s}')$ *over the same set of labels* L *are* trace equivalent*, which is denoted by* $(S, L, \longrightarrow, \bar{s}) \sim_t (S', L, \longrightarrow', \bar{s}')$ *or* $\bar{s} \sim_t \bar{s}'$ *for short, if* $T(\bar{s}) = T(\bar{s}')$.

Definition 2.5 (Bisimilarity) *Two transition systems* $(S, L, \longrightarrow, \bar{s})$ *and* $(S', L, \longrightarrow', \bar{s}')$ *over the same set of labels are* bisimilar *(or bisimulation equivalent), denoted by* $(S, L, \longrightarrow, \bar{s}) \sim_b (S', L, \longrightarrow', \bar{s}')$ *or* $\bar{s} \sim_b \bar{s}'$ *for short, if there is a* bisimulation*, i.e. a relation* $\mathcal{R} \subseteq S \times S'$ *such that* $(\bar{s}, \bar{s}') \in \mathcal{R}$ *and for each* $(s_1, s_1') \in \mathcal{R}$ *we have:*

- *if* $s_1 \xrightarrow{\gamma} s_2$*, then there is* s_2' *such that* $(s_2, s_2') \in \mathcal{R}$ *and* $s_1' \xrightarrow{\gamma}{}' s_2'$
- *if* $s_1' \xrightarrow{\gamma}{}' s_2'$*, then there is* s_2 *such that* $(s_2, s_2') \in \mathcal{R}$ *and* $s_1 \xrightarrow{\gamma} s_2$.

Remark 2.6 *Bisimilar transition systems are also trace equivalent.*

Figure 2.2: Trace-Equivalent but not Bisimilar Transition Systems

The transition systems from Figure 1.1 and Figure 2.1 are bisimilar and therefore also trace equivalent. The transition systems presented in Figure 2.2 are trace equivalent but not bisimilar.

Various other equivalences have been defined for transition systems. An overview over equivalences that lie between trace and bisimulation equivalences with respect to their discriminating power is given in [86, 170]

2.3 Complete Partial Orders

To obtain denotational semantics for systems that include recursion, it is usually necessary to employ model with fixpoint theory. *Complete metric spaces* with *contracting* functions, e.g. [74], denote a class where fixpoints always exist. They are used for example in [21, 65, 117] to define a denotational semantics of recursive systems. Another class where fixpoints always exist is given by the *(ω-)complete partial orders* (cpo) with *continuous* functions. An overview of this theory is given in [3]. Complete partial orders are used for example in [4, 21, 76, 147, 149, 178] to obtain a denotational semantics of recursive systems. The definition and the results that are used in the following chapters are summarized in this section. For further details consult [3]. Here, the general cpo theory is restricted to the ω-cpo theory, because the latter is sufficient for our purpose.

Definition 2.7 (ω-complete partial order) *A set D with the partial order $\sqsubseteq$ is a (pointed) ω-*complete partial order *(cpo) if*

- *D has a* least element *($\bot$) with respect to $\sqsubseteq$, i.e. $\forall d \in D : \bot \sqsubseteq d$*
- *for every ω-*chain *$(d_i)_{i\in\mathbb{N}}$, i.e. $\forall j \in \mathbb{N} : d_j \sqsubseteq d_{j+1}$, there exists a* least upper bound *($\bigsqcup_{i\in\mathbb{N}} d_i$) in D, i.e.*
  - *$\forall j \in \mathbb{N} : d_j \sqsubseteq \bigsqcup_{i\in\mathbb{N}} d_i$*
  - *$\forall d \in D : (\forall j \in \mathbb{N} : d_j \sqsubseteq d) \Rightarrow \bigsqcup_{i\in\mathbb{N}} d_i \sqsubseteq d$*

Definition 2.8 (ω-continuous functions) *A function f between cpo D and cpo E, i.e. $f : D \to E$, is ω-*continuous *(or* continuous *for short) if it preserves the least upper bounds of ω-chains, i.e. for all ω-chains $(d_i)_{i\in\mathbb{N}}$ of D we have $f(\bigsqcup_{i\in\mathbb{N}} d_i) = \bigsqcup_{i\in\mathbb{N}} f(d_i)$.*

In particular, every continuous function is monotonic.

Theorem 2.9 *Let M be a set and let $\langle D, \sqsubseteq_D \rangle$ and $\langle E, \sqsubseteq_E \rangle$ be cpos. Then*

- *the function space $M \to D$ with the* pointwise order, *i.e. $g \sqsubseteq g' \Leftrightarrow \forall m \in M : g(m) \sqsubseteq_D g'(m)$, is a cpo. Moreover, $\bigsqcup_{i\in\mathbb{N}} g_i = g$, where $g(m) = \bigsqcup_{i\in\mathbb{N}} g_i(m)$.*
- *and the Cartesian product $D \times E$ with the* componentwise order, *i.e. $(d, e) \sqsubseteq (d', e') \Leftrightarrow (d \sqsubseteq_D d' \wedge e \sqsubseteq_E e')$, is a cpo. Moreover, $\bigsqcup_{i\in\mathbb{N}}(d_i, e_i) = (\bigsqcup_{i\in\mathbb{N}} d_i, \bigsqcup_{i\in\mathbb{N}} e_i)$.*

Lemma 2.10 *Let D, D', E be cpos and $f : (D \times D') \to E$. Then f is continuous if and only if it is* componentwise continuous, *i.e. for all $d' \in D' : f^{(1)}_{d'} : D \to E$, where $f^{(1)}_{d'}(d) = f(d, d')$, is continuous and for all $d \in D : f^{(2)}_{d} : D' \to E$, similarly defined, is continuous.*

Theorem 2.11 *Let D be a cpo and let $f : D \to D$ be a continuous function, then f has a* least fixpoint $\mathrm{fix}(f)$, *which is given by $\bigsqcup_{i\in\mathbb{N}} f^i(\bot)$.*

2.4 Approximation Closedness

In this section, we define when a set $M \subseteq \mathcal{P}(E)$ is approximation closed. These sets are used to guarantee that an ω-chain of *event structures* introduced in later chapters will have a least upper bound.

Definition 2.12 *Let E be a countable set. A* finite, monotone approximation *of E is a sequence $(E_i)_{i\in\mathbb{N}}$ such that $\bigcup_{i\in\mathbb{N}} E_i = E \wedge \forall k : E_k \subseteq E_{k+1} \wedge |E_k| < \infty$.*

It is obvious that every countable set has a finite, monotone approximation.

Definition 2.13 *Let E be a countable set and $M \subseteq \mathcal{P}(E)$. We say that M is* approximation closed with respect to E *if*

- *$X \in M$ whenever*
  $X \subseteq E$ and there is a finite, monotone approximation $(E_i)_{i\in\mathbb{N}}$ of E such that $\forall k \in \mathbb{N} : \exists X_k \in M : X_k \cap E_k = X \cap E_k$.

Example 2.14 *If E is finite, then every $M \subseteq \mathcal{P}(E)$ is approximation closed with respect to E. Another example is $\mathcal{P}(\mathbb{N})$, which is approximation closed with respect to $\mathbb{N}$.*

On the other hand, $\mathcal{P}_{fin}(\mathbb{N})$ is not approximation closed with respect to $\mathbb{N}$, since $\forall n \in \mathbb{N} : \{m \in \mathbb{N} \mid m \leq n\} \in \mathcal{P}_{fin}(\mathbb{N})$ but $\mathbb{N} \notin \mathcal{P}_{fin}(\mathbb{N})$. Also $\mathcal{P}(\mathbb{N})\backslash\{\emptyset\}$ is not approximation closed with respect to $\mathbb{N}$.

Proposition 2.15 *Suppose M_1, M_2 are approximation closed with respect to E. Then $M_1 \cap M_2$ and $M_1 \cup M_2$ are approximation closed with respect to E.*

Proof: The proof is given in Subsection 2.4.1. □

For the following proofs, it is necessary to have further set constructions that yield approximation closed sets. In order to verify the approximation closedness, we construct a set X out of a sequence $(X_n)_{n\in\mathbb{N}}$ of sets (where $X_n \in M$). We show that this constructed X is in M whenever M is approximation closed. The construction of X is generalized in the sense that X is constructed out of two given sequences. This generalization is needed in some of the approximation closedness proofs.

Definition 2.16 *Let E_j be a set and $\kappa_j : \mathbb{N} \to E_j$ such that κ_j is bijective for $j = 1, 2$. Furthermore, let $\vec{X}^{(j)} = (X_n^{(j)})_{n\in\mathbb{N}}$ be a sequence of elements of $\mathcal{P}(E_j)$ for $j = 1, 2$.*

Define $\mathcal{X}(\vec{X}^{(1)}, \kappa_1, \vec{X}^{(2)}, \kappa_2) = (\bigcup_{k\in\mathbb{N}} A_k^{(1)}, \bigcup_{k\in\mathbb{N}} A_k^{(2)})$ where $A_n^{(j)} \subseteq E_j$ and $N_n^{(j)} \subseteq \mathbb{N}$ is given by $A_0^{(1)} = A_0^{(2)} = \emptyset$, $N_0^{(0)} = N_0^{(1)} = N_0^{(2)} = \mathbb{N}$ and for $j = 1, 2$

$$
\begin{array}{rcl}
N_{n+1}^{(0)} & = & N_n^{(2)} \\
N_{n+1}^{(j)} & = & \begin{cases} \{q \in N_{n+1}^{(j-1)} \mid \kappa_j(n) \in X_q^{(j)}\} & \text{if } \kappa_j(n) \in \bigcap_k \bigcup_{i\geq k, i\in N_{n+1}^{(j-1)}} X_i^{(j)} \\ \{q \in N_{n+1}^{(j-1)} \mid \kappa_j(n) \notin X_q^{(j)}\} & \text{otherwise} \end{cases} \\
A_{n+1}^{(j)} & = & \begin{cases} A_n^{(j)} \cup \{\kappa_j(n)\} & \text{if } \kappa_j(n) \in \bigcap_k \bigcup_{i\geq k, i\in N_{n+1}^{(j-1)}} X_i^{(j)} \\ A_n^{(j)} & \text{otherwise} \end{cases}
\end{array}
$$

Note that $|N_n^{(j)}|$ is always infinite, since $\kappa_j(n) \in \bigcap_k \bigcup_{i\geq k, i\in N_{n+1}^{(j-1)}} X_i^{(j)}$ holds exactly when $\kappa_j(n) \in X_i^{(j)}$ for infinitely many $i \in N_{n+1}^{(j-1)}$.

Proposition 2.17 *Suppose M_j is approximation closed with respect to E_j, $\vec{X}^{(j)} = (X_n^{(j)})_{n\in\mathbb{N}}$ is a sequence of elements of M_j and $\kappa_j : \mathbb{N} \to E$ is a bijective function for $j = 1, 2$. Then $\pi_j(\mathcal{X}(\vec{X}^{(1)}, \kappa_1, \vec{X}^{(2)}, \kappa_2)) \in M_j$ for $j = 1, 2$.*

Proof: The proof is given in Subsection 2.4.1. □

As a consequence of Proposition 2.17 we obtain the following corollaries. Their proofs are given in Subsection 2.4.1.

Corollary 2.18 *Suppose M is approximation closed with respect to E and $E' \subseteq E$. Then*

$$\{X \cap E' \mid X \in M\}$$

is approximation closed with respect to E'.

Corollary 2.19 *Suppose M_1, M_2 are approximation closed with respect to E. Then*

$$\{X_1 \cup X_2 \mid X_1 \in M_1 \wedge X_2 \in M_2\}$$

is approximation closed with respect to E.

Corollary 2.20 *Suppose M_i is approximation closed with respect to E_i for $i = 1, 2$. Then*

$$\{\{(e_1, e_2) \in E_1 \times E_2 \mid e_1 \in X_1 \wedge e_2 \in X_2\} \mid X_1 \in M_1 \wedge X_2 \in M_2\},$$
$$\{\{(e_1, e_2) \in E_1 \times E_2 \mid e_i \in X_i\} \mid i \in \{1, 2\} \wedge X_i \in M_i\} \text{ and}$$
$$\{\{(e_1, e_2) \in E_1 \times E_2 \mid e_1 \in X_1 \vee e_2 \in X_2\} \mid X_1 \in M_1 \wedge X_2 \in M_2\}$$

are approximation closed with respect to $E_1 \times E_2$.

Corollary 2.21 *Suppose M is approximation closed with respect to E and for all $e \in E$ let E_e be a set and M_e be a collection of subsets such that M_e is approximation closed with respect to E_e. Then*

$$\{\{(e, \hat{e}) \mid e \in X \wedge \hat{e} \in X_e\} \mid X \in M \wedge \forall e \in E : X_e \in M_e\}$$

is approximation closed with respect to $\{(e, \hat{e}) \mid e \in E \wedge \hat{e} \in E_e\}$.

2.4.1 Proofs

Proof of Proposition 2.15:

$M_1 \cap M_2$: Suppose $X \subseteq E$ and $(E_n)_{n\in\mathbb{N}}$ be a finite monotone approximation of E such that $\forall n \in \mathbb{N} : \exists X_n : X_n \in M_1 \wedge X_n \in M_2 \wedge X \cap E_n = X_n \cap E_n$. By the approximation closedness of M_i we obtain that $X \in M_i$, which completes this case.

$M_1 \cup M_2$: Suppose $X \subseteq E$ and $(E_n)_{n\in\mathbb{N}}$ is a finite monotone approximation of E such that $\forall n \in \mathbb{N} : \exists X_n : (X_n \in M_1 \vee X_n \in M_2) \wedge X \cap E_n = X_n \cap E_n$. Let $N_i = \{n \in \mathbb{N} \mid X_n \in M_i\}$. Then N_1 or N_2 has to be infinite. Without loss of generality, let N_1 be infinite (the other case follows analogously). Then $(E_n)_{n\in N_1}$ is a finite monotone approximation of E such that $\forall n \in N_1 : X_n \in M_1 \wedge X \cap E_n = X_n \cap E_n$. Thus $X \in M_1$ by the approximation closedness of M_1. □

Proof of Proposition 2.17: Let $A_n^{(j)}$ and $N_n^{(j)}$ be defined as in Definition 2.16.
Define $E_n^{(j)} = A_n^{(j)} \cup \left[\left(\bigcup_{r\leq n}\{\kappa_j(r)\}\right) \setminus \left(\bigcup_{i\geq n, i\in N_{n+1}^{(j-1)}} X_i^{(j)}\right)\right]$. Then $E_n^{(j)} \subseteq E_{n+1}^{(j)}$ and

$$\forall n : \kappa_j(n) \in \bigcup_p E_p^{(j)} \tag{2.1}$$

which is verified as follows. Suppose $\kappa_j(n) \in \bigcap_k \bigcup_{i\geq k, i\in N_{n+1}^{(j-1)}} X_i^{(j)}$, then $\kappa_j(n) \in A_{n+1}^{(j)}$. Hence, $\kappa_j(n) \in \bigcup_p E_p^{(j)}$. Now suppose $\kappa_j(n) \notin \bigcap_k \bigcup_{i\geq k, i\in N_{n+1}^{(j-1)}} X_i^{(j)}$. Then there is a k such that $\kappa_j(n) \notin \bigcup_{i\geq k, i\in N_{n+1}^{(j-1)}} X_i^{(j)}$. Define $m = \max\{k, n\}$. We get $\kappa_j(n) \notin \bigcup_{i\geq m, i\in N_{m+1}^{(j-1)}} X_i^{(j)}$ since $N_{m+1}^{(j-1)} \subseteq N_{n+1}^{(j-1)}$. Hence, $\kappa_j(n) \in E_m^{(j)}$ which establishes (2.1).

$$\forall n \in \mathbb{N} : \forall q \in N_{n+1}^{(j-1)} : A_n^{(j)} \subseteq X_q^{(j)} \tag{2.2}$$

This can be proven by induction, where the claim is easily seen to hold in the base case. Now suppose $q \in N_{n+2}^{(j-1)}$ then $q \in N_{n+1}^{(j-1)}$ and so by induction $A_n^{(j)} \subseteq X_q^{(j)}$. If $A_{n+1}^{(j)} = A_n^{(j)}$, the

claim follows. Therefore, suppose $A^{(j)}_{n+1} = A^{(j)}_n \cup \{\kappa_j(n)\}$. Then $\kappa_j(n) \in \bigcap_k \bigcup_{i\geq k, i\in N^{(j-1)}_{n+1}} X^{(j)}_i$. From $N^{(j-1)}_{n+2} \subseteq N^{(j)}_{n+1}$ and the definition of $N^{(j)}_{n+1}$ we get $\kappa_j(n) \in X^{(j)}_q$, which verifies (2.2).

Now we show $\bigcup_p A^{(j)}_p \in M$:
$E^{(j)}_n \cap \bigcup_p A^{(j)}_p = \left(A^{(j)}_n \cap \bigcup_p A^{(j)}_p\right) \cup \left(\left[\left(\bigcup_{r\leq n}\{\kappa_j(r)\}\right) \setminus \left(\bigcup_{i\geq n, i\in N^{(j-1)}_{n+1}} X^{(j)}_i\right)\right] \cap \bigcup_p A^{(j)}_p\right) = A^{(j)}_n \cup \left(A^{(j)}_{n+1} \setminus \left(\bigcup_{i\geq n, i\in N^{(j-1)}_{n+1}} X^{(j)}_i\right)\right)$ by definition. Furthermore, $\bigcap_k \bigcup_{i\geq k, i\in N^{(j-1)}_{n+1}} X^{(j)}_i \subseteq \bigcup_{i\geq n, i\in N^{(j-1)}_{n+1}} X^{(j)}_i$, which yields $A^{(j)}_{n+1} \subseteq \bigcup_{i\geq n, i\in N^{(j-1)}_{n+1}} X^{(j)}_i$ with (2.2) and the fact that $|N^{(j-1)}_{n+1}|$ is infinite. Hence, $E^{(j)}_n \cap \bigcup_p A^{(j)}_p = A^{(j)}_n$. On the other hand, let $q \in N^{(j-1)}_{n+1}$ with $q \geq n$. Such a q exists, since $|N^{(j-1)}_{n+1}|$ is infinite. Then by (2.2) and from the fact that $X^{(j)}_q \subseteq \bigcup_{i\geq n, i\in N^{(j-1)}_{n+1}} X^{(j)}_i$ we have $X^{(j)}_q \cap E^{(j)}_n = \left(X^{(j)}_q \cap A^{(j)}_n\right) \cup \left(X^{(j)}_q \cap \left[\left(\bigcup_{r\leq n}\{\kappa_j(r)\}\right) \setminus \left(\bigcup_{i\geq n, i\in N^{(j-1)}_{n+1}} X^{(j)}_i\right)\right]\right) = A^{(j)}_n$. Hence, $X^{(j)}_q \cap E^{(j)}_n = E^{(j)}_n \cap \bigcup_p A^{(j)}_p$. Therefore, by the approximation closedness of M we get $\bigcup_p A^{(j)}_p \in M$, since $(E^{(j)}_n)_{n\in\mathbb{N}}$ is a finite, monotone approximation of E, which follows from (2.1). □

Proof of Corollary 2.18: Define $M' = \{X \cap E' \mid X \in M\}$. Suppose $X' \subseteq E'$ and $(E'_n)_{n\in\mathbb{N}}$ is a finite, monotone approximation of E' such that $\forall n : \exists X'_n \in M' : X'_n \cap E'_n = X' \cap E'_n$.
By the definition of M', for all $n \in \mathbb{N}$ there exists $X_n \in M$ such that $X_n \cap E'_n = X' \cap E'_n$.
If E is finite, the proof is trivial. Therefore, let $\kappa : \mathbb{N} \to E$ be bijective. Then by Proposition 2.17 we have $\pi_1(\mathcal{X}(\vec{X}, \kappa, \vec{X}, \kappa)) \in M$, where $\vec{X} = (X_n)_{n\in\mathbb{N}}$. Let $A^{(1)}_n$ and $N^{(1)}_n$ be defined as in Definition 2.16.
It remains to prove that $E' \cap \pi_1(\mathcal{X}(\vec{X}, \kappa, \vec{X}, \kappa)) = X'$ by which then $X' \in M'$.

$\subseteq$: Suppose $e \in E' \cap \bigcup_p A^{(1)}_p$. Then there exists $g \in \mathbb{N}$ such that $\kappa(g) = e$. Moreover, there exists n such that $\kappa(g) \in E'_n$, since $\kappa(g) \in E'$. From $\kappa(g) \in \bigcup_p A^{(1)}_p$ we get $\kappa(g) \in \bigcap_k \bigcup_{i\geq k, i\in N^{(0)}_{g+1}} X_i$, which is a subset of $\bigcup_{i\geq n, i\in N^{(0)}_{g+1}} X_i$. Then there is an $i \in \mathbb{N}$ such that $i \geq n$ and $\kappa(g) \in X_i$. Thus $\kappa(g) \in X_i \cap E'_n \overset{(i\geq n)}{=} X_i \cap E'_i \cap E'_n = X' \cap E'_i \cap E'_n$. Hence, $\kappa(g) \in X'$.

$\supseteq$: Suppose $e \in X'$. Then there is $g \in \mathbb{N}$ such that $\kappa(g) = e$. Moreover, there is an n such that $\kappa(g) \in E'_n$, since $X' \subseteq E'$. Then for all $i \geq n$ we have $\kappa(g) \in X' \cap E'_n = X' \cap E'_i \cap E'_n = X_i \cap E'_i \cap E'_n$. Thus $\forall k : \kappa(g) \in \bigcup_{i\geq k, i\in N^{(0)}_{g+1}} X_i$, since $|N^{(0)}_{g+1}|$ is infinite. Hence, $\kappa(g) \in \bigcap_k \bigcup_{i\geq k, i\in N^{(0)}_{g+1}} X_i$. And so by definition $\kappa(g) \in A^{(1)}_{g+1} \subseteq \bigcup_p A^{(1)}_p$. □

Proof of Corollary 2.19: Suppose $X \subseteq E$ and $(E_n)_{n\in\mathbb{N}}$ is a finite monotone approximation of E such that $\forall n \in \mathbb{N} : \exists X^{(i)}_n \in M_i : X \cap E_n = (X^{(1)}_n \cup X^{(2)}_n) \cap E_n$.
If E is finite, the proof is trivial. Therefore, let $\kappa : \mathbb{N} \to E$ be bijective. Then by Proposition 2.17 we have $\pi_j(\mathcal{X}(\vec{X}^{(1)}, \kappa, \vec{X}^{(2)}, \kappa)) \in M_j$, where $\vec{X}^{(i)} = (X^{(i)}_n)_{n\in\mathbb{N}}$. Let $A^{(i)}_n$ and $N^{(i)}_n$ be

defined as in Definition 2.16. It remains to prove that

$$\bigcup_{j\in\{1,2\}} \pi_j(\mathcal{X}(\vec{X}^{(1)},\kappa,\vec{X}^{(2)},\kappa) = X.$$

$\subseteq$: Suppose $e \in (\bigcup_p A_p^{(1)}) \cup (\bigcup_p A_p^{(2)})$. Then $e \in \bigcup_p A_p^{(j)}$ for some $j \in \{1,2\}$. Let $g \in \mathbb{N}$ such that $\kappa(g) = e$ and let $n \in \mathbb{N}$ such that $\kappa(g) \in E_n$. From $\kappa(g) \in \bigcup_p A_p^{(j)}$ we get $\kappa(g) \in \bigcap_k \bigcup_{q\geq k, q\in N_{g+1}^{(j-1)}} X_q^{(j)}$, which is a subset of $\bigcup_{q\geq n, q\in N_{g+1}^{(j-1)}} X_q^{(j)}$. Then there exists $q \in \mathbb{N}$ such that $q \geq n$ and $\kappa(g) \in X_q^{(j)}$. Thus $\kappa(g) \in (X_q^{(1)} \cup X_q^{(1)}) \cap E_n \overset{(q\geq n)}{=} (X_q^{(1)} \cup X_q^{(1)}) \cap E_q \cap E_n = X \cap E_q \cap E_n$. Hence, $e = \kappa(g) \in X$.

$\supseteq$: Suppose $e \in X$. Let $g \in \mathbb{N}$ such that $\kappa(g) = e$ and let $n \in \mathbb{N}$ such that $\kappa(g) \in E_n$. Then for all $q \geq n$ we have $\kappa(g) \in X \cap E_n = X \cap E_q \cap E_n = (X_q^{(1)} \cup X_q^{(2)}) \cap E_q \cap E_n$. Hence, $\forall i \in N_{n+1}^{(1)} : i \geq n \Rightarrow \kappa(g) \in X_i^{(1)} \cup X_i^{(2)}$.

Suppose $\kappa(g) \in X_i^{(2)}$ for infinitely many $i \in N_{g+1}^{(1)}$. Then $\forall k : \kappa(g) \in \bigcup_{i\geq k, i\in N_{g+1}^{(1)}} X_i^{(2)}$, hence $\kappa(g) \in \bigcup_p A_p^{(2)}$.

Now suppose $\kappa(g) \in X_i^{(2)}$ for finitely many $i \in N_{g+1}^{(1)}$. Then $\kappa(g) \in X_i^{(1)}$ for infinitely many $i \in N_{g+1}^{(1)}$. Then $\forall k : \kappa(g) \in \bigcup_{i\geq k, i\in N_{g+1}^{(0)}} X_i^{(1)}$, since $N_{g+1}^{(1)} \subseteq N_{g+1}^{(0)}$. Hence $\kappa(g) \in \bigcup_p A_p^{(1)}$.

□

Proof of Corollary 2.20: Consider the first set: Suppose $X \subseteq E_1 \times E_2$ and $(\tilde{E}_n)_{n\in\mathbb{N}}$ is a finite monotone approximation of $E_1 \times E_2$ such that $\forall n \in \mathbb{N} : \exists X_n^{(1)} \in M_1, X_n^{(2)} \in M_2 : X \cap \tilde{E}_n = \{(e_1,e_2) \in \tilde{E}_n \mid e_1 \in X_n^{(1)} \wedge e_2 \in X_n^{(2)}\}$. If E_j is finite, the proof is much simpler. Therefore, let $\kappa_j : \mathbb{N} \to E_j$ be bijective. Then by Proposition 2.17 we have $\pi_j(\mathcal{X}(\vec{X}_1,,\kappa_1,\vec{X}_2,\kappa_2)) \in M_j$, where $\vec{X}_j = (X_n^{(j)})_{n\in\mathbb{N}}$. Let $A_n^{(i)}$ and $N_n^{(i)}$ be defined as in Definition 2.16. It remains to prove that

$$X = \{(e_1,e_2) \mid e_1 \in \pi_1(\mathcal{X}(\vec{X}_1,,\kappa_1,\vec{X}_2,\kappa_2)) \wedge e_2 \in \pi_2(\mathcal{X}(\vec{X}_1,,\kappa_1,\vec{X}_2,\kappa_2))$$

$\subseteq$: Suppose $(\kappa_1(g_1),\kappa_2(g_2)) \in X$. Let $n \in \mathbb{N}$ such that $(\kappa_1(g_1),\kappa_2(g_2)) \in \tilde{E}_n$. Then for all $q \geq n$ we have $(\kappa_1(g_1),\kappa_2(g_2)) \in X \cap \tilde{E}_n = X \cap \tilde{E}_q \cap \tilde{E}_n = \{(e_1,e_2) \in \tilde{E}_q \mid e_1 \in X_q^{(1)} \wedge e_2 \in X_q^{(2)}\} \cap \tilde{E}_n$. Hence, $\forall i \in N_{g_j+1}^{(j-1)} : i \geq n \Rightarrow \kappa_j(g_j) \in X_i^{(j)}$. Then $\forall k : \kappa_j(g_j) \in \bigcup_{i\geq k, i\in N_{g_j+1}^{(j-1)}} X_i^{(j)}$, hence $\kappa_j(g_j) \in \bigcup_p A_p^{(j)}$.

$\supseteq$: Suppose $\kappa_1(g_1) \in \bigcup_p A_p^{(1)}$ and $\kappa_2(g_2) \in \bigcup_p A_p^{(2)}$. Thus $\kappa_j(g_j) \in A_{g_j+1}^{(j)}$. From (2.2) we have

$$\forall n \in \mathbb{N} : \forall q \in N_{g_j+2}^{(j-1)} : \kappa_j(g_j) \in X_q^{(j)}. \tag{2.3}$$

Let $n \in \mathbb{N}$ such that $(\kappa_1(g_1),\kappa_2(g_2)) \in \tilde{E}_n$. Then by (2.3) there is $q_j \geq n$ with $q_j \in N_{g_j+2}^{(j-1)}$ and $\kappa_j(g_j) \in X_{q_j}^{(j)}$. Define q to be q_1 if $g_1 > g_2$ and otherwise to be q_2. Then

$$\kappa_1(g_1) \in X_q^{(1)} \wedge \kappa_2(g_2) \in X_q^{(2)} \tag{2.4}$$

which can be seen as follows. If $g_1 > g_2$ then $N^{(1)}_{g_2+2} \subseteq N^{(0)}_{g_1+2}$ and if $g_1 \leq g_2$ then $N^{(0)}_{g_1+2} \subseteq N^{(1)}_{g_2+2}$. Hence, (2.4) is an immediate consequence of (2.3).

Therefore, we have $(\kappa_1(g_1), \kappa_2(g_2)) \in \{(e_1, e_2) \mid e_1 \in X^{(1)}_q \wedge e_2 \in X^{(2)}_q\} \cap \tilde{E}_n \overset{(q \geq n)}{=} \{(e_1, e_2) \in \tilde{E}_q \mid e_1 \in X^{(1)}_q \wedge e_2 \in X^{(2)}_q\} \cap \tilde{E}_n = X \cap \tilde{E}_n$.

For the proof of the approximation closedness of the second and the third set of Corollary 2.20 we define $M'_i = \{\{(e_1, e_2) \in E_1 \times E_2 \mid e_i \in X_i\} \mid X_i \in M_i\}$. Then from the approximation closedness of the first set of Corollary 2.20 and from the fact that the set $\{E_i\}$ is approximation closed with respect to E_i we obtain that M'_i is approximation closed with respect to $E_1 \times E_2$. Then the approximation closedness of the second set follows from Proposition 2.15, since $\{\{(e_1, e_2) \in E_1 \times E_2 \mid e_i \in X_i\} \mid i \in \{1, 2\} \wedge X_i \in M_i\} = M'_1 \cup M'_2$. Furthermore, the approximation closedness of the third set follows from Corollary 2.19, since $\{\{(e_1, e_2) \in E_1 \times E_2 \mid e_1 \in X_1 \vee e_2 \in X_2\} \mid X_1 \in M_1 \wedge X_2 \in M_2\} = \{X'_1 \cup X'_2 \mid X'_1 \in M'_1 \wedge X'_2 \in M'_2\}$. □

Proof of Corollary 2.21: Let $\tilde{E} = \{(e, \hat{e}) \mid e \in E \wedge \hat{e} \in E_e\}$ and define $\pi_e(\tilde{X}) = \{\hat{e} \mid (e, \hat{e}) \in \tilde{X}\}$ for $e \in E$ and $\tilde{X} \subseteq \tilde{E}$. Suppose $X \subseteq \tilde{E}$ and $(\tilde{E}_n)_{n \in \mathbb{N}}$ be a finite monotone approximation of $\tilde{E}$ such that $\forall n \in \mathbb{N} : \exists X_n \in M, X^{(e)}_n \in M_e : X \cap \tilde{E}_n = \{(e, \hat{e}) \in \tilde{E}_n \mid e \in X_n \wedge \hat{e} \in X^{(e)}_n\}$.

Without loss of generality, let E be infinite, since otherwise the proof is much simpler. Therefore, let $\kappa : \mathbb{N} \to E$. Then by Proposition 2.17 we have $\pi_1(\mathcal{X}(\vec{X}, \kappa, \vec{X}, \kappa)) \in M$, where $\vec{X} = (X_n)_{n \in \mathbb{N}}$. Let $N^{(i)}_n$ be defined as in Definition 2.16. First we prove

$$X = \{(e, \hat{e}) \in \tilde{E} \mid e \in \pi_1(\mathcal{X}(\vec{X}, \kappa, \vec{X}, \kappa)) \wedge \hat{e} \in \pi_e(X)\} \tag{2.5}$$

From $\hat{e}' \in \pi_{e'}(X)$ we obtain $(e', \hat{e}') \in X$, which establishes $\supseteq$ of (2.5). Now suppose $(e', \hat{e}') \in X$. Let $q \in \mathbb{N}$ such that $e' = \kappa(q)$ and let $n \in \mathbb{N}$ such that $(e', \hat{e}') \in \tilde{E}_n$. Then for all $i \geq n$ we have $(\kappa(q), \hat{e}') \in X \cap \tilde{E}_n = X \cap \tilde{E}_i \cap \tilde{E}_n = \{(e, \hat{e}) \in \tilde{E}_i \mid e \in X_i \wedge \hat{e} \in X^{(e)}_i\} \cap \tilde{E}_n$. Hence, $\forall k : \kappa(q) \in \bigcup_{i \geq k, i \in N^{(0)}_{q+1}} X_i$. Thus $\kappa(q) \in \bigcap_k \bigcup_{i \geq k, i \in N^{(0)}_{q+1}} X_i$. And so by definition $\kappa(q) \in \pi_1(\mathcal{X}(\vec{X}, \kappa, \vec{X}, \kappa))$, which establishes (2.5).

It remains to prove that

$$e' \in \pi_1(\mathcal{X}(\vec{X}, \kappa, \vec{X}, \kappa)) \Rightarrow \pi_{e'}(X) \in M_{e'} \tag{2.6}$$

We have $\pi_{e'}(X) \cap \pi_{e'}(\tilde{E}_n) = \pi_{e'}(X \cap \tilde{E}_n) = \pi_{e'}(\{(e, \hat{e}) \in \tilde{E}_n \mid e \in X_n \wedge \hat{e} \in X^{(e)}_n\}) = X^{(e')}_n \cap \pi_{e'}(\tilde{E}_n)$ whenever $e' \in X_n$. From $e' \in \pi_1(\mathcal{X}(\vec{X}, \kappa, \vec{X}, \kappa))$ we obtain that $e' \in X_n$ for infinitely many n. And so (2.6) follows from the approximation closedness property, since $(\pi_{e'}(\tilde{E}_n))_{n \in N}$, where $N = \{n \in \mathbb{N} \mid e' \in X_n\}$, is a finite monotone approximation of $E_{e'}$. □

Chapter 3

Standard Action Refinement

In this chapter, we first sketch the different action refinement approaches in scientific literature. Then we illustrate the common approach on a concrete setting, i.e. we give a denotational and operational semantics to a process algebra that contains an action refinement operator. Before we present the action refinement operator on extended bundle event structures [125] (used as the denotational model), we will show that the event structures based on the bundle technique fail to yield a complete partial order (with the standard ordering). Therefore, we introduce a new subclass of extended bundle event structures that yields a complete partial order with respect to the standard ordering. This subclass is defined by using the approximation closedness techniques introduced in Section 2.4.

Furthermore, a new technique is used in this section to define an operational semantics that corresponds to the denotational semantics. This technique has the advantage of handling the disrupt expression in a feasible way and of avoiding any further syntactical expression.

3.1 Different Approaches

As mentioned in the introduction, in software design it is useful to have a *top down system design* [180], i.e. to change the level of abstraction until the implementation is obtained from the specification. Expressing simple actions by more concrete processes, called *action refinement*, reflects this methodology in the context of process algebraic settings.

Different approaches for action refinement can be distinguished:

- *atomic action refinement* [34, 64, 69, 94, 97], where the process P to which the action is refined has to be considered atomic, i.e. there are no observable states in between the execution steps of P (all-or-nothing). Motivations for this approach are given in [34, 69, 98].

- *non-atomic action refinement* [6, 31, 49, 60, 70, 90, 111, 114, 122, 144, 157, 173], where, as opposed to the above approach, the process to which the action is refined may interleave with the original system (or with other refinements).
  This approach is on the whole more popular than the atomic approach. For example, if two actions a, b are completely independent, it seems unreasonable to impose a restriction stating that b stays idle while the refinement of a is executed.

- *relaxed forms* [113, 159, 176]. Here, the causal ordering after the refinement of actions is relaxed if the involved actions are considered to be independent. For example, suppose action b has to be preceded by action a and a is refined by the sequential composition of actions a_1 and a_2. Then a_1 may interleave with action b if a_1 and b are defined to be independent.

- *vertical action refinement* [94, 156, 158]. Here, refinement is regarded to be an implementation relation instead of being an operator, as it is done in the above cases.

For a more detailed overview over the different methods of action refinement consult [91, 98].

In this thesis, we consider non-atomic action refinement. The theory of action refinement is well established for denotational, true concurrency semantics of process algebras, i.e. refinement operators are employed in *event structures*, *petri nets* and in other models of concurrency [42, 62, 63, 72, 82, 91, 172]. When presenting an operational semantics of process algebras with a refinement operator in terms of transition systems, the action refinement operator is often handled on the process term level by *syntactic substitution* [5, 6, 144]. In general, this type of operational semantics is incompatible with the standard denotational semantics [93]. Approaches to obtain operational semantics that correspond to the denotational ones are given in [70, 98, 157].

The common non-atomic action refinement approach is illustrated in the rest of this chapter. It also contains a new technique to define an operational semantics that is consistent with the denotational semantics.

3.2 Syntax

We choose a process algebra that is close to *basic LOTOS* [32] except that the symbols are rather taken from [117] and that the process algebra contains an expression for action refinement.

Let $\surd$ and τ be two different elements, which indicate the *termination* and the *internal action*. Furthermore, let Obs be a set such that $\surd, \tau \notin \mathsf{Obs}$ and $|\mathsf{Obs}| > |\mathbb{N}|$. We call Obs the set of *observable actions*. The *set of all actions* $\mathcal{A}ct_{\surd}$ is defined by $\mathcal{A}ct_{\surd} = \{\surd, \tau\} \cup \mathsf{Obs}$. Assume a fixed countable set of *process variables* Var which is disjoint from $\mathcal{A}ct_{\surd}$.

The process algebra expressions $\mathsf{EXP}_{\mathsf{sr}}$ ($_{\mathsf{s}} \triangleq$ start-based, $_{\mathsf{r}} \triangleq$ refinement) are defined by the following BNF-grammar.

$$B \ ::= \ \mathbf{0} \mid \mathbf{1} \mid a.B \mid \tau.B \mid B + B \mid B;B \mid B \,[\!>\, B \mid B \|_A B \mid B[(a \to B)^{a \in A}] \mid B\backslash A \mid x$$

where $x \in \mathsf{Var}$, $a \in \mathsf{Obs}$ and $A \subseteq \mathsf{Obs}$ with $|A| \leq |\mathbb{N}|$[1]. A *process with respect to* $\mathsf{EXP}_{\mathsf{sr}}$ is a pair $\langle \mathrm{decl}, B \rangle$ consisting of a *declaration* $\mathrm{decl} : \mathsf{Var} \to \mathsf{EXP}_{\mathsf{sr}}$ and an expression $B \in \mathsf{EXP}_{\mathsf{sr}}$. Let $\mathsf{PA}_{\mathsf{sr}}$ denote the set of all processes with respect to $\mathsf{EXP}_{\mathsf{sr}}$.

An expression $B \in \mathsf{EXP}_{\mathsf{sr}}$ is also called a process if decl is clear from the context. We sometimes omit $\mathbf{1}$ in an expression, i.e. we write a instead of $a.\mathbf{1}$.

[1] The action relabeling operator [138] can be modeled by the action refinement operator in our process algebra, since $|A|$ may be infinite.

The intuitive meanings of the expressions are as follows. $\mathbf{0}$ is the inactive process, i.e. it can not execute any action. $\mathbf{1}$ is the process that can terminate immediately. The action prefix process $a.B$ ($\tau.B$) is the process that executes a (respectively τ) and revolves to process B. Process $B_1 + B_2$ is a choice between the behaviors described by B_1 and B_2. The choice is determined by the first action that is executed. $B_1; B_2$ is the sequential composition, i.e. B_1 proceeds until it terminates, after which B_2 takes over. $B_1 [> B_2$ is the disruption of B_1 by B_2, i.e. any action from B_2 may disable B_1 as long as B_1 has not terminated. On the other hand, the termination of B_1 disables B_2. $B_1 \|_A B_2$ describes the parallel composition of B_1 and B_2 where both processes have to synchronize on actions of A and on $\surd$ (i.e. the parallel composition terminates if and only if both sides terminate). The intuitive meaning of the refinement expression $B[(a \to B_a)^{a \in A}]$ is that it behaves like process B except that every execution of action a in A is substituted by the behavior of B_a. The hiding process $B \backslash A$ behaves like B except that all actions of A are renamed with τ. The behavior of x is given by the declaration.

3.3 Denotational True Concurrency Semantics for $\mathsf{PA_{sr}}$

Event structures are typically used as denotational true concurrency models for process algebras. (Extended) bundle event structures are investigated in [126, 125] as a denotational model for LOTOS, on which our process algebra is based. Unfortunately, the set of all (extended) bundle event structures does not yield an ω-complete partial order (cpo) with the standard order. We remedy this problem by defining a subclass of the class of extended bundle event structures that yields a cpo.

This section is organized as follows: Subsection 3.3.1 contains the definition of (extended) bundle event structures. We show that they do not yield a cpo with the standard order. The subset that yields a cpo is introduced in Subsection 3.3.2. The operators on these event structures are defined in Subsection 3.3.3. These operators are used in Subsection 3.3.4 in order to present the denotational meaning of a process.

3.3.1 Bundle Event Structures

(Extended) bundle event structures are introduced in [125, 126]. Later on, they are extended to timed versions [39, 40, 120], to stochastic versions [45, 118] and to a probabilistic version [119]. See also [46].

Definition 3.1 (Bundle Event Structure) *A* bundle event structure, bes *for short,* $(E, \sharp, \mapsto, l)$ *is an element of* $\mathcal{P}(\mathcal{U}) \times \mathcal{P}(\mathcal{U} \times \mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U}) \times \mathcal{U}) \times (\mathcal{U} \rightharpoonup \mathcal{A}ct_{\surd})$ *such that*

- $\sharp \subseteq (E \times E)$ *and* $\sharp$ *is irreflexive and symmetric*
- $\mapsto \subseteq \mathcal{P}(E) \times E$
- $\mathrm{dom}(l) = E$
- $\forall X \subseteq E, e \in E : X \mapsto e \Rightarrow (\forall e', e'' \in X : e' \neq e'' \Rightarrow e' \sharp e'')$

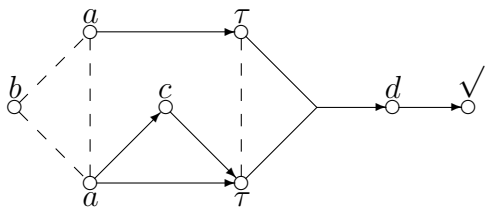


Figure 3.1: A Bundle Event Structure

E is called the set of events, $\sharp$ the (irreflexive) *symmetric conflict relation*, $\mapsto$ the *bundle relation* and l the *action-labeling function* of the bes $(E, \sharp, \mapsto, l)$. X is called a bundle of e if and only if $X \mapsto e$.

The intuitive meaning of $X \mapsto e$ is that before e is enabled, an event of X has to be executed. $e' \sharp e$ means that the execution of e disables e' forever, and vice versa. The action that may be observed when an event is executed is given by the action-labeling function. The last constraint in the definition of bundle event structures is called *bundle stability constraint*. It guarantees the absence of *causal ambiguity*, i.e. exactly one event of a bundle of e is executed before e is enabled, and so no confusion which event causes e arises. Bundle event structures and flow nets [35, 38] have exactly the same expressiveness [37].

Bundle event structures can be used as a semantic model for expressions of $\mathsf{EXP}_{\mathsf{sr}}$ that do not contain disrupt operators. For example, in [125] the expression $\big((a + b.\mathbf{0}) \|_{\{a\}} (a.c + a)\big)\,;d$ is modeled by the bes depicted in Figure 3.1. There, events are illustrated by circles labeled with their corresponding action names; dashed lines indicate conflicts between events and for each bundle $X \mapsto e$ we draw one arrow from all events in X to e.

A bundle event structures has a symmetric conflict relation. Therefore, it is not clear how disruption, for example $a.b \,[> c$, can be modeled as a bes, since disruption is not a symmetrical property. Therefore, [125] introduces extended bundle event structures, where the conflict relation does not need to be symmetric.

Definition 3.2 (Extended Bundle Event Structure) *An* extended bundle event structure *, ebes for short, $\mathcal{E} = (E, \rightsquigarrow, \mapsto, l)$ is an element of $\mathcal{P}(\mathcal{U}) \times \mathcal{P}(\mathcal{U} \times \mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U}) \times \mathcal{U}) \times (\mathcal{U} \rightharpoonup \mathcal{A}ct_{\surd})$ such that*

- $\rightsquigarrow \subseteq (E \times E)$ *and* $\rightsquigarrow$ *is irreflexive*
- $\mapsto \subseteq \mathcal{P}(E) \times E$
- $\mathrm{dom}(l) = E$
- $\forall X \subseteq E, e \in E : X \mapsto e \Rightarrow (\forall e', e'' \in X : e' \neq e'' \Rightarrow e' \rightsquigarrow e'')$

Let $\mathbf{EBES}$ *denote the set of all extended bundle event structures.*

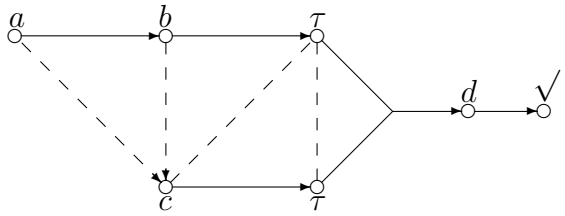


Figure 3.2: An Extended Bundle Event Structure

$\rightsquigarrow$ is called the (irreflexive) *asymmetric conflict relation*. Hereafter, we consider $\mathcal{E}$ to be $(E, \rightsquigarrow, \mapsto, l)$, $\mathcal{E}_i$ to be $(E_i, \rightsquigarrow_i, \mapsto_i, l_i)$ and in general $\mathcal{E}$ to be $(E_{\mathcal{E}}, \rightsquigarrow_{\mathcal{E}}, \mapsto_{\mathcal{E}}, l_{\mathcal{E}})$.

The intuitive meaning of $e' \rightsquigarrow e$ is that the execution of e disables e' forever, but not vice versa. Furthermore, an event can not be in conflict with itself, which is expressed by the irreflexivity of $\rightsquigarrow$. In [125] the expression $((a.b)\,[>c)\,;d$ is modeled by the ebes shown in Figure 3.2. There the conflicts are depicted as dashed arrows or depicted as dashed lines if they are symmetrical.

Remark 3.3 *The bundle stability constraint of extended bundles event structures are dropped to obtain more general class of event structures. The event structures obtained are called* dual event structures *[116]. They allow causal ambiguity, which is examined in [127].*

The standard order for (extended) bundle event structures is introduced as follows.

Definition 3.4 (Restriction of a ebes) *Suppose $\mathcal{E}$ is an extended bundle event structure and $E' \subseteq E$. Then the* restriction of $\mathcal{E}$ to E'*, denoted by $\mathcal{E} \upharpoonright E'$, is $(E', \rightsquigarrow \cap (E' \times E'), \mapsto', l \upharpoonright E')$ where $\mapsto' = \{(X \cap E', e) \mid e \in E' \wedge X \mapsto e\}$.*

Remark 3.5 *A restriction of a restricted ebes is equal to the restriction of that ebes, i.e. if $\mathcal{E} = (E, \rightsquigarrow, \mapsto, l)$ is an ebes and $E_1 \subseteq E_2 \subseteq E$, then $(\mathcal{E} \upharpoonright E_2) \upharpoonright E_1 = \mathcal{E} \upharpoonright E_1$.*

Definition 3.6 (Order on EBES) *Let $\mathcal{E}_i \in \mathbf{EBES}$. Then $\mathcal{E}_1 \trianglelefteq \mathcal{E}_2$ if and only if $E_1 \subseteq E_2$ and $\mathcal{E}_1 = \mathcal{E}_2 \upharpoonright E_1$*

Remark 3.7 *Two different ebes which are comparable with respect to $\trianglelefteq$ must have different sets of events, i.e. $\mathcal{E}_1 \trianglelefteq \mathcal{E}_2 \wedge E_2 \subseteq E_1$ implies $\mathcal{E}_1 = \mathcal{E}_2$.*

Langerak [125] constructs the following minimal upper bound of an ω-chain.

Definition 3.8 *Let $(\mathcal{E}_i)_{i \in \mathbb{N}}$, where $\mathcal{E}_i = (E_i, \rightsquigarrow_i, \mapsto_i, l_i)$, be an ω-chain with respect to $\trianglelefteq$. Then define $\bigsqcup_i \mathcal{E}_i$ to be the ebes $(\bigcup_i E_i, \bigcup_i \rightsquigarrow_i, \mapsto, \bigcup_i l_i)$ where $\mapsto = \{(X, e) \mid \forall k : e \in E_k \Rightarrow (X \cap E_k) \mapsto_k e\}$.*

Non-Completeness of $\langle \mathbf{EBES}, \trianglelefteq \rangle$.

$\langle \mathbf{EBES}, \trianglelefteq \rangle$ does not yield a cpo, as there are ω-chains with more than one minimal upper bound. This situation can arise, for example, if $(X \cap E_k)_{k \in \mathbb{N}}$ is strictly increasing and each $(X \cap E_k, e)$ is a bundle of $\mathcal{E}_i$ whenever $i \geq k$. We illustrate this by an example, where we consider $\mathbb{N}$ to be a subset of $\mathcal{U}$ by identifying $\star^n\bullet$ with $n \in \mathbb{N}$.

Let $\hat{\mathcal{E}}_n$ be the ebes where the set of events consists of the elements of $M_n = \{1, ..., n\}$ and an additional element e. Furthermore, the bundles of e are all subsets of M_n. Formally,

$$\hat{\mathcal{E}}_n = (M_n \cup \{e\}, M_n \times M_n \backslash \mathrm{Id}, \{(X, e) \mid X \in \mathcal{P}(M_n)\}, \mathrm{cons}_a)$$

for $n \in \mathbb{N}$, where Id denotes the *identity relation*.

It is obvious that $(\hat{\mathcal{E}}_n)_{n \in \mathbb{N}}$ is a chain in $\langle \mathbf{EBES}, \trianglelefteq \rangle$, hence $\bigsqcup_n \hat{\mathcal{E}}_n$ is a minimal upper bound of $(\hat{\mathcal{E}}_n)_{n \in \mathbb{N}}$ with respect to $\trianglelefteq$. From Definition 3.8 we get

$$\bigsqcup_n \hat{\mathcal{E}}_n = (\mathbb{N} \cup \{e\}, \mathbb{N} \times \mathbb{N} \setminus \mathrm{Id}, \{(X, e) \mid X \in \mathcal{P}(\mathbb{N})\}, \mathrm{cons}_a).$$

In words, any subset of the natural numbers combined with e is a bundle in $\bigsqcup_n \hat{\mathcal{E}}_n$. If we restrict the bundles to the finite subsets of the natural numbers, i.e.

$$\hat{\mathcal{E}}^{fin} = (\mathbb{N} \cup \{e\}, \mathbb{N} \times \mathbb{N} \setminus \mathrm{Id}, \{(X, e) \mid X \in \mathcal{P}_{fin}(\mathbb{N})\}, \mathrm{cons}_a),$$

we also get a minimal upper bound of $(\hat{\mathcal{E}}_n)_{n \in \mathbb{N}}$ with respect to $\trianglelefteq$. Furthermore, $\hat{\mathcal{E}}^{fin}$ and $\bigsqcup_n \hat{\mathcal{E}}_n$ are incomparable with respect to $\trianglelefteq$ by Remark 3.7. Hence, the ω-chain $(\hat{\mathcal{E}}_n)_{n \in \mathbb{N}}$ does not have a least upper bound and therefore $\langle \mathbf{EBES}, \trianglelefteq \rangle$ fails to be a cpo.

The above counterexample is also a counterexample for the non-completeness of the class of bundle event structures under the given partial order, since every event structure in the example is a bundle event structure.

3.3.2 Closed Bundle Event Structures ($\mathbf{CBES}$)

We want to restrict the ebes we have just considered in such a way that only one of the minimal upper bounds from the counterexample of Subsection 3.3.1 is allowed. Therefore, we only take those ebes into account which are closed under some special kind of finite approximation and thus rule out $\hat{\mathcal{E}}^{fin}$. We define this closedness property by using the results of *approximation closedness* introduced in Section 2.4.

Definition 3.9 (Closed Bundle Event Structure) *A* closed bundle event structure *(*cbes*)* $\mathcal{E} = (E, \rightsquigarrow, \mapsto, l)$ *is an element of* $\mathcal{P}(\mathcal{U}) \times \mathcal{P}(\mathcal{U} \times \mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U}) \times \mathcal{U}) \times (\mathcal{U} \rightharpoonup \mathcal{A}ct_{\surd})$ *such that*

- $\rightsquigarrow \subseteq (E \times E)$ *and* $\forall e \in E : \neg(e \rightsquigarrow e)$
- $\mapsto \subseteq \mathcal{P}(E) \times E$
- $\mathrm{dom}(l) = E$

- $\forall X \subseteq E, e \in E : X \mapsto e \Rightarrow (\forall e', e'' \in X : e' \neq e'' \Rightarrow e' \rightsquigarrow e'')$
- $\forall e \in E : \_ \mapsto e$ *is approximation closed with respect to* E

Let $\mathbf{CBES}$ *denote the set of all closed bundle event structures.*

Since every cbes is also an ebes (only a further constraint is added), we have $\mathbf{CBES} \subset \mathbf{EBES}$. Furthermore, every ebes that has a finite set of events satisfies the closedness condition, and therefore is an element of $\mathbf{CBES}$. Hence, the ebes shown in Figure 3.2 is also a cbes. Furthermore, $\bigsqcup_n \hat{\mathcal{E}}_n$ from the counterexample of Subsection 3.3.1 is a cbes, whereas $\hat{\mathcal{E}}^{fin}$ is not, which follows from Example 2.14.

Theorem 3.10 (Complete Partial Order) *The ordered set* $\langle \mathbf{CBES}, \trianglelefteq \rangle$ *is an* ω*-complete partial order, where* $\bigsqcup_n \mathcal{E}_n$ *from Definition 3.8 is the least upper bound.*

Proof: The proof is given in Subsection 3.6.1. □

Remark 3.11 *The set of bundle event structures can be restricted in the same way as* $\mathbf{EBES}$ *to obtain a cpo.*

Remark 3.12 *There is another possibility of defining an order on* $\mathbf{EBES}$ *that yields a complete partial order: An ebes is smaller than another one if it has less events but more bundles (*$\{(X \cap E_1, e) \mid X \mapsto_2 e \wedge e \in E_1\} \subseteq \mapsto_1$*), i.e. events in the greater ebes can be enabled earlier as in the smaller one. There, the ebes do not have to satisfy the approximation closedness constraints to yield a complete partial order, i.e.* $\mathbf{EBES}$ *with this order yields a cpo. But not all standard operators (compare Subsection 3.3.3) are continuous with respect to this order.*

Transition system from a cbes.

Here, we describe how to obtain a transition system from a cbes, which is later used to establish a consistency result for the denotational and the operational semantics. First, we specify the *initial events*, i.e. those events which do not have a causal constraint, then we specify the events that correspond to termination.

Definition 3.13 *The set of* initial events *of cbes* $\mathcal{E}$ *is defined by*

$$\mathrm{init}(\mathcal{E}) = \{e \in E \mid \neg(\exists X : X \mapsto e)\}.$$

The set of successful termination events *of cbes* $\mathcal{E}$ *is defined by*

$$\mathrm{exit}(\mathcal{E}) = \{e \in E \mid l(e) = \surd\}.$$

In order to obtain a transition system from a cbes, the *remainder* [20, 126, 131] of a cbes with respect to an initial event is defined. The remainder denotes the event structure after the execution of this initial event. Remainders are used to obtain a transition relation for $\mathbf{CBES}$.

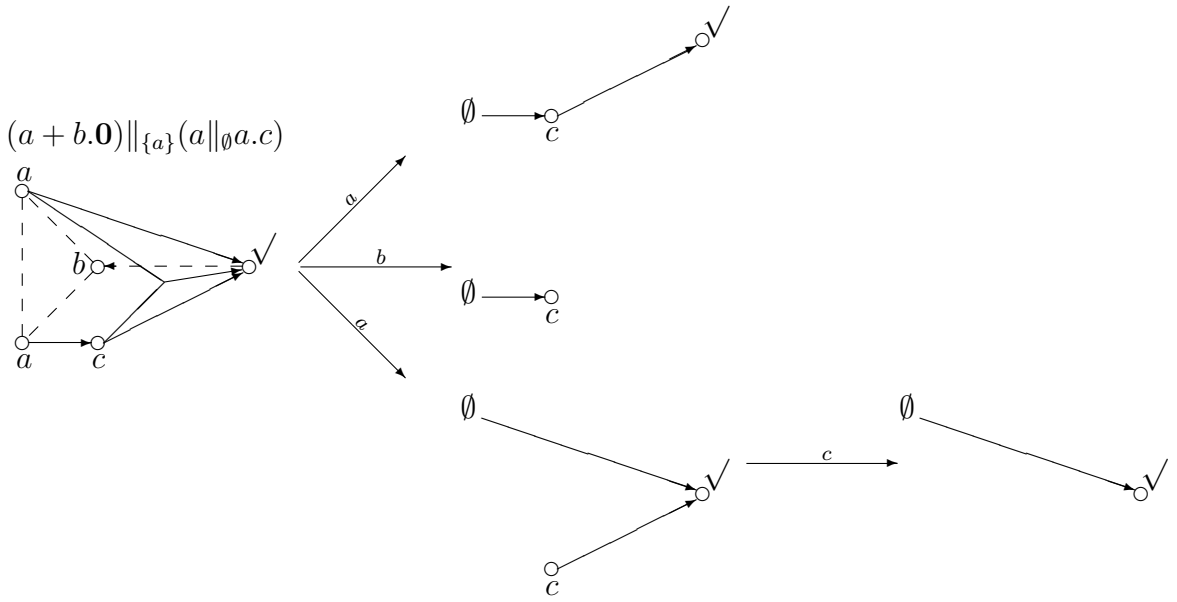


Figure 3.3: Transition System Derived from $\mathbf{CBES}$

Definition 3.14 (Remainder of a cbes) *Let $\mathcal{E} \in \mathbf{CBES}$ and $e \in \mathrm{init}(\mathcal{E})$. Then the* remainder *$\mathcal{E}_{[e]}$ is given by $(E', \leadsto', \mapsto', l')$ where*

$$\begin{array}{rcl} E' & = & \{e' \in E \mid e' \neq e \wedge \neg(e' \leadsto e)\} \\ \leadsto' & = & \leadsto \cap (E' \times E') \\ \mapsto' & = & \{(X \cap E', e') \mid e' \in E' \wedge X \mapsto e' \wedge e \notin X\} \\ l' & = & l \restriction E' \end{array}$$

Lemma 3.15 *Let $\mathcal{E} \in \mathbf{CBES}$ and $e \in \mathrm{init}(\mathcal{E})$. Then $\mathcal{E}_{[e]} \in \mathbf{CBES}$.*

Proof: Define $\tilde{\mathcal{E}} = (E, \leadsto, \{(X, \tilde{e}) \in \mapsto \mid e \notin X\}, l)$. We will show that $\tilde{\mathcal{E}} \in \mathbf{CBES}$. Let $X \subseteq E$ and $(E_n)_{n \in \mathbb{N}}$ be a finite, monotone approximation of E such that $\forall n : \exists X_n : X_n \tilde{\mapsto} \tilde{e} \wedge X \cap E_n = X_n \cap E_n$. Then there is $m \in \mathbb{N}$ such that $e \in E_m$. Thus, $e \notin X$. Furthermore, we have $X \mapsto \tilde{e}$ by the approximation closedness condition of $\mathcal{E}$. Hence, $X \tilde{\mapsto} \tilde{e}$.
It is easy to check that $\mathcal{E}_{[e]} = \tilde{\mathcal{E}} \restriction E_{\mathcal{E}_{[e]}}$. And so the result follows by Lemma 3.27. □

Definition 3.16 *The transition relation $\hookrightarrow \subseteq \mathbf{CBES} \times \mathcal{A}ct_\surd \times \mathbf{CBES}$ is defined by $\hookrightarrow = \{(\mathcal{E}, l(e), \mathcal{E}_{[e]}) \mid \mathcal{E} \in \mathbf{CBES} \wedge e \in \mathrm{init}(\mathcal{E})\}$.*

An example of a transition system obtained from $\hookrightarrow$ is given in Figure 3.3.

3.3.3 Operators on $\mathbf{CBES}$

Here, we present the operators on $\mathbf{CBES}$ that are later used to define the denotational semantics. They are taken from [125] except for some slight modifications. For example, we model the disjoint union directly and we introduce more conflicts. This is done in order to obtain

a closer connection between the operational and the denotational semantics. The refinement operator, which does not appear in [125], is an adapted version of [79, 133].

Definition 3.17 (Operators on $\mathcal{E}$) *Let $A \subseteq \mathsf{Obs}$. Then define*

$\widehat{.} : (\mathsf{Obs} \cup \{\tau\}) \times \mathbf{CBES} \to \mathbf{CBES}$ *with* $a\widehat{.}\mathcal{E} = (\{\bullet\} \cup (\{\star_1\} \times E), \tilde{\leadsto}, \tilde{\mapsto}, \tilde{l})$ *where*

$$\begin{array}{rcl} \tilde{\leadsto} & = & \{((\star_1, e), (\star_1, e')) \mid (e, e') \in \leadsto\} \\ \tilde{\mapsto} & = & \{(\{\star_1\} \times X, (\star_1, e)) \mid (X, e) \in \mapsto\} \cup \{\{(\bullet\}, (\star_1, e)) \mid e \in \mathrm{init}(\mathcal{E})\} \\ \tilde{l}(\tilde{e}) & = & \begin{cases} l(e) & \textit{if } \tilde{e} = (\star_1, e) \\ a & \textit{if } \tilde{e} = \bullet \end{cases} \end{array}$$

$\widehat{+} : \mathbf{CBES} \times \mathbf{CBES} \to \mathbf{CBES}$ *with* $\mathcal{E}_1 \widehat{+} \mathcal{E}_2 = (\tilde{E}, \tilde{\leadsto}, \tilde{\mapsto}, \tilde{l})$ *where*

$$\begin{array}{rcl} \tilde{E} & = & (\{\star_1\} \times E_1) \cup (\{\star_2\} \times E_2) \\ \tilde{\leadsto} & = & \{((\star_i, e_i), (\star_j, e_j)) \mid i \neq j \wedge e_j \in \mathrm{init}(E_j)\} \cup \{((\star_i, e), (\star_i, e')) \mid e \leadsto_i e'\} \\ \tilde{\mapsto} & = & \{(\{\star_i\} \times X, (\star_i, e)) \mid X \mapsto_i e\} \\ \tilde{l}((\star_i, e)) = l_i(e) & & \end{array}$$

$\widehat{;} : \mathbf{CBES} \times \mathbf{CBES} \to \mathbf{CBES}$ *with* $\mathcal{E}_1 \widehat{;} \mathcal{E}_2 = (\tilde{E}, \tilde{\leadsto}, \tilde{\mapsto}, \tilde{l})$ *where*

$$\begin{array}{rcl} \tilde{E} & = & (\{\star_1\} \times E_1) \cup (\{\star_2\} \times E_2) \\ \tilde{\leadsto} & = & \{((\star_i, e), (\star_i, e')) \mid e \leadsto_i e' \vee (i = 1 \wedge e \neq e' \wedge e' \in \mathrm{exit}(\mathcal{E}_1))\} \\ \tilde{\mapsto} & = & \{(\{\star_i\} \times X, (\star_i, e)) \mid X \mapsto_i e\} \cup \\ & & \{(\{\star_1\} \times \mathrm{exit}(\mathcal{E}_1), (\star_2, e)) \mid e \in \mathrm{init}(\mathcal{E}_2)\} \\ \tilde{l}((\star_i, e)) = & & \begin{cases} l_1(e) & \textit{if } i = 1 \wedge e \notin \mathrm{exit}(\mathcal{E}_1) \\ \tau & \textit{if } i = 1 \wedge e \in \mathrm{exit}(\mathcal{E}_1) \\ l_2(e) & \textit{if } i = 2 \end{cases} \end{array}$$

$\widehat{[>} : \mathbf{CBES} \times \mathbf{CBES} \to \mathbf{CBES}$ *with* $\mathcal{E}_1 \widehat{[>} \mathcal{E}_2 = (\tilde{E}, \tilde{\leadsto}, \tilde{\mapsto}, \tilde{l})$ *where*

$$\begin{array}{rcl} \tilde{E} & = & (\{\star_1\} \times E_1) \cup (\{\star_2\} \times E_2) \\ \tilde{\leadsto} & = & \{((\star_i, e), (\star_i, e')) \mid e \leadsto_i e'\} \cup (\{\star_1\} \times E_1)) \times (\{\star_2\} \times \mathrm{init}(\mathcal{E}_2)) \cup \\ & & (\{\star_2\} \times E_2)) \times (\{\star_1\} \times \mathrm{exit}(\mathcal{E}_1)) \\ \tilde{\mapsto} & = & \{(\{\star_i\} \times X, (\star_i, e)) \mid X \mapsto_i e\} \\ \tilde{l}((\star_i, e)) = l_i(e) & & \end{array}$$

$\widehat{\|}_A : \mathbf{CBES} \times \mathbf{CBES} \to \mathbf{CBES}$ *with* $\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2 = (\tilde{E}, \tilde{\leadsto}, \tilde{\mapsto}, \tilde{l})$ *where*

$$\begin{array}{rcl} \tilde{E} & = & (E_1^f \times \{\star\}) \cup (\{\star\} \times E_2^f) \cup E^s \\ E_i^f & = & \{e \in E_i \mid l_i(e) \notin A \cup \{\surd\}\} \\ E^s & = & \{(e_1, e_2) \in E_1 \times E_2 \mid l_1(e_1) = l_2(e_2) \in A \cup \{\surd\}\} \\ \tilde{\leadsto} & = & \{((e_1, e_2), (e_1', e_2')) \mid e_1 \leadsto_1 e_1' \vee e_2 \leadsto_2 e_2' \vee \\ & & (e_1 = e_1' \neq \star \wedge e_2 \neq e_2') \vee (e_2 = e_2' \neq \star \wedge e_1 \neq e_1')\} \\ \tilde{\mapsto} & = & \{(\{(e_1', e_2') \in \tilde{E} \mid e_i' \in X_i\}, (e_1, e_2)) \mid X_i \mapsto_i e_i\} \\ \tilde{l}((e_1, e_2)) = & & \begin{cases} l_1(e_1) & \textit{if } e_2 = \star \\ l_2(e_2) & \textit{otherwise} \end{cases} \end{array}$$

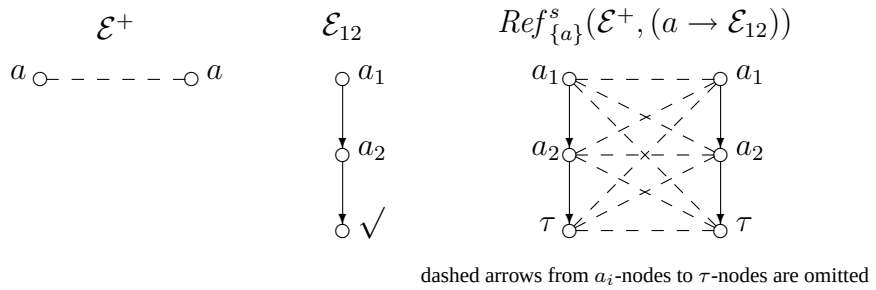


Figure 3.4: Illustration of the Ref^s Operator

$Ref^s_A : \mathbf{CBES} \times (A \to \mathbf{CBES}) \to \mathbf{CBES}$ *with* $Ref^s_A(\mathcal{E}, \theta) = (\tilde{E}, \tilde{\rightsquigarrow}, \tilde{\mapsto}, \tilde{l})$ *where*

$$
\begin{array}{rcl}
\tilde{E} & = & \{(e, \hat{e}) \mid e \in E \wedge l(e) \in A \wedge \hat{e} \in E_{\theta(l(e))}\} \cup \\
 & & \{(e, e) \in E \times E \mid l(e) \notin A\} \\
\tilde{\rightsquigarrow} & = & \{((e_1, \hat{e}_1), (e_2, \hat{e}_2)) \mid e_1 \rightsquigarrow e_2 \vee (e_1 = e_2 \wedge l(e_1) \in A \wedge \hat{e}_1 \neq \hat{e}_2 \wedge \\
 & & (\hat{e}_1 \rightsquigarrow_{\theta(l(e_1))} \hat{e}_2 \vee \hat{e}_2 \in \mathrm{exit}(\theta(l(e_1)))))\} \\
\tilde{\mapsto} & = & \{(\{e\} \times X', (e, \hat{e})) \mid l(e) \in A \wedge X' \mapsto_{\theta(l(e))} \hat{e}\} \cup \\
 & & \{(\tilde{X}, (e, \hat{e})) \mid \exists X : X \mapsto e \wedge (l(e) \in A \Rightarrow \hat{e} \in \mathrm{init}(\theta(l(e)))) \wedge \\
 & & \tilde{X} = \{(e', \hat{e}') \in \tilde{E} \mid e' \in X \wedge (l(e') \in A \Rightarrow \hat{e}' \in \mathrm{exit}(\theta(l(e'))))\}\}
\end{array}
$$

$$
\tilde{l}((e, \hat{e})) = \begin{cases} l(e) & \textit{if } l(e) \notin A \\ l_{\theta(l(e))}(\hat{e}) & \textit{if } l(e) \in A \wedge l_{\theta(l(e))}(\hat{e}) \neq \surd \\ \tau & \textit{if } l(e) \in A \wedge l_{\theta(l(e))}(\hat{e}) = \surd \end{cases}
$$

$\widehat{\backslash} A : \mathbf{CBES} \to \mathbf{CBES}$ *with* $\mathcal{E}\widehat{\backslash} A = (E, \rightsquigarrow, \mapsto, \tilde{l})$ *where*

$$
\tilde{l}(e) = \begin{cases} l(e) & \textit{if } l(e) \notin A \\ \tau & \textit{if } l(e) \in A \end{cases}
$$

A small example that illustrates how the refinement operator Ref^s behaves is given in Figure 3.4. There, $(a \to \mathcal{E}_{12})$ denotes the function from $\{a\}$ to $\mathbf{CBES}$ that maps a to $\mathcal{E}_{12}$.

Proposition 3.18 *All operators of Definition 3.17 are well defined, i.e. they really yield elements of* $\mathbf{CBES}$. *Moreover all operators from Definition 3.17 are continuous with respect to* $\trianglelefteq$.

Proof: The proof is given in Subsection 3.6.2. □

3.3.4 Denotational Meaning for $\mathsf{PA}_{\mathsf{sr}}$

First, we define the denotational semantics of expressions ($\mathsf{EXP}_{\mathsf{sr}}$) with respect to variable assignments, i.e. functions from Var to $\mathbf{CBES}$. Then variable assignments are derived from declarations, which are used to define the denotational semantics of processes ($\mathsf{PA}_{\mathsf{sr}}$).

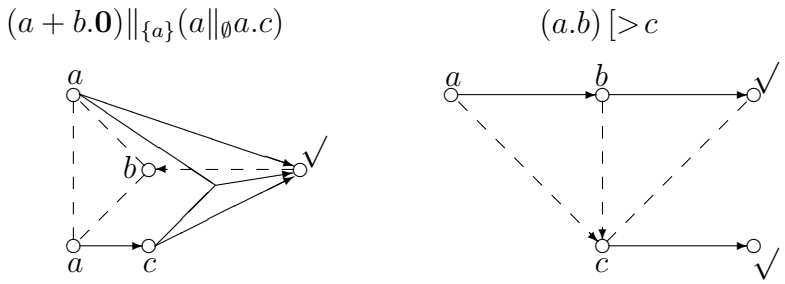


Figure 3.5: Examples of the Denotational Semantics of $\mathsf{EXP_{sr}}$

Definition 3.19 *Let* $[\![\_]\!]_\_ : \mathsf{EXP_{sr}} \times (\mathsf{Var} \to \mathbf{CBES}) \to \mathbf{CBES}$ *be defined as follows (where* $\rho : \mathsf{Var} \to \mathbf{CBES}$*)*

$$
\begin{array}{ll}
[\![\mathbf{0}]\!]_\rho = (\emptyset,\emptyset,\emptyset,\emptyset) & [\![\mathbf{1}]\!]_\rho = (\{\bullet\},\emptyset,\emptyset,\{(\bullet,\surd)\}) \\
[\![a.B]\!]_\rho = a\widehat{\ .}[\![B]\!]_\rho & [\![\tau.B]\!]_\rho = \tau\widehat{\ .}[\![B]\!]_\rho \\
[\![B_1 + B_2]\!]_\rho = [\![B_1]\!]_\rho \widehat{+} [\![B_2]\!]_\rho & [\![B_1; B_2]\!]_\rho = [\![B_1]\!]_\rho \widehat{\ ;}\ [\![B_2]\!]_\rho \\
[\![B_1\,[\!> B_2]\!]_\rho = [\![B_1]\!]_\rho \widehat{[\!>} [\![B_2]\!]_\rho & [\![B_1 \|_A B_2]\!]_\rho = [\![B_1]\!]_\rho \widehat{\|}_A [\![B_2]\!]_\rho \\
[\![B[(a \to B_a)^{a\in A}]]\!]_\rho = Ref^s_A([\![B]\!]_\rho, (a \to [\![B_a]\!]_\rho)^{a\in A}) & \\
[\![B\backslash A]\!]_\rho = [\![B]\!]_\rho \widehat{\backslash} A & [\![x]\!]_\rho = \rho(x)
\end{array}
$$

Examples of how $[\![\_]\!]_\_$ behaves are given in Figure 3.5.

To apply the cpo theory we need the following lemma.

Lemma 3.20 $[\![B]\!]_\_$ *is continuous for every* $B \in \mathsf{EXP_{sr}}$.

Proof: By structural induction. We only present the case $B_1 + B_2$.

$$
\begin{array}{rcl}
[\![B_1 + B_2]\!]_{\bigsqcup_i \rho_i} & = & \widehat{+}([\![B_1]\!]_{\bigsqcup_i \rho_i}, [\![B_2]\!]_{\bigsqcup_i \rho_i}) \\
 & \overset{\text{by induction}}{=} & \widehat{+}(\bigsqcup_i [\![B_1]\!]_{\rho_i}, \bigsqcup_i [\![B_2]\!]_{\rho_i}) \\
 & \overset{\text{Theorem 2.9}}{=} & \widehat{+}(\bigsqcup_i ([\![B_1]\!]_{\rho_i}, [\![B_2]\!]_{\rho_i})) \\
 & \overset{\text{Prop. 3.18}}{=} & \bigsqcup_i \widehat{+}([\![B_1]\!]_{\rho_i}, [\![B_2]\!]_{\rho_i}) \\
 & = & \bigsqcup_i [\![B_1 + B_2]\!]_{\rho_i}
\end{array}
$$

The other cases follow analogously. □

Now we are ready to give the meaning of a declaration.

Suppose $\mathrm{decl} : \mathsf{Var} \to \mathsf{EXP_{sr}}$. Then define $\mathcal{F}_{\mathrm{decl}} : (\mathsf{Var} \to \mathbf{CBES}) \to (\mathsf{Var} \to \mathbf{CBES})$ with $\mathcal{F}_{\mathrm{decl}}(\rho)_{(x)} = [\![\mathrm{decl}(x)]\!]_\rho$. From Lemma 3.20 it follows that $\mathcal{F}_{\mathrm{decl}}$ is continuous. Therefore, from Theorem 2.11 we get $\{\!|\_|\!\} : (\mathsf{Var} \to \mathsf{EXP_{sr}}) \to (\mathsf{Var} \to \mathbf{CBES})$ with $\{\!|\mathrm{decl}|\!\} = \mathrm{fix}(\mathcal{F}_{\mathrm{decl}}) = \bigsqcup_n \mathcal{F}^n_{\mathrm{decl}}(\bot)$ is well defined.

We define the denotation of a process as follows.

Definition 3.21 (Denotational Semantics)
Define $[\![\_]\!] : \mathsf{PA_{sr}} \to \mathbf{CBES}$ *by* $[\![\langle \mathrm{decl}, B\rangle]\!] = [\![B]\!]_{\{[\mathrm{decl}]\}}$.

3.4 Operational Semantics for $\mathsf{PA_{sr}}$

Operational semantics that coincide with the denotational semantics have been given for process algebras with action refinement in [70, 98, 157]. Operational semantics for the ST-approache[2], which can be straightforwardly adjusted to operational semantics for action refinement, have been examined in [44, 49, 98, 102]. The papers based on the ST-approach do not explicitly examine consistency between denotational and operational semantics. The underlying languages used in the cited papers do not contain disruption. Moreover, most of the operational semantics defined in these papers are inappropriate for languages that contain disruption like our language does. This is argued as follows.

In [44, 49, 70], a refinement expression $B[a \to Q]$ is modeled by invoking Q in parallel with the remaining process whenever a is activated in B. This leads to problems if a disrupt operator is involved. For example, action a can be disrupted during its execution. But in the above approach Q is executed, even though a is disrupted.

On the other hand, in [98, 157] the execution of an action is only allowed if all active actions are still executable afterwards. This is not a reasonable approach if a language with a disrupt operator is considered, since this operator can remove active actions in a reasonable way.

The operational semantics mentioned above have to extend the syntax of the underlying languages in order to define the operational semantics for refinement: Further syntactical operators have to be added to the languages in [44, 49, 70]. And [98, 157] extends the syntactical expressions by moving to an event-based language, which leads to a very discriminating theory.

We will present a new approach to an operational semantics that corresponds to the denotational semantics. This approach can handle the disrupt operator and, moreover, no extension of the syntactical expressions is necessary.

The operational semantics is defined by using transition rules that simulate the execution of a refined action within the refinement construction. For example, the process $(a.B)[a \to a_1.a_2]$ evolves into $(a.B)[a \to a_2]$ by executing a_1 provided a does not occur in B. If a occurs in B as in $a.a$ and if we proceed in the same way, we lose the information to which term the a_s occurring in B have to be refined, i.e. $(a.a)[a \to a_1.a_2]$ must not evolve to $(a.a)[a \to a_2]$ by executing a_1. To circumvent this problem, we rename the executed action a by a fresh action name a' and extend the refinement by an additional refinement, i.e. $(a.a)[a \to a_1.a_2]$ evolves into $(a'.a)[(a \to a_1.a_2), (a' \to a_2)]$ by executing a_1, where $a' \neq a$. This is still not sufficient, since we have to trigger the choice when an action is renamed in the choice operator, i.e. $(b + (a.B))[a \to a_1.a_2]$ evolves into $(a'.B)[(a \to a_1.a_2), (a' \to a_2)]$ by executing a_1. Otherwise, $(b + (a.B))[a \to a_1.a_2]$ would be able to execute b after a_1. This is counterintuitive, and therefore the choice has to be triggered. We model the renaming of the action by extending the transition relation using additional transition labels. The choices are also triggered by those additional transition labels.

[2] In the ST-approach, the execution of an action is split into the two different events of the start and the ending (termination) of an action, where the ending is uniquely related to its start.

Before we introduce the transition rules for $\mathsf{EXP_{sr}}$, we have to determine all actions which occur in a given expression. We need this in order to determine a fresh action name for the renaming of an action. Furthermore, we need $|\mathsf{Obs}| > |\mathbb{N}|$ to guarantee the existence of a fresh action name.

Definition 3.22 *The function $\tilde{\mathcal{L}} : \mathsf{EXP_{sr}} \to \mathcal{P}_{count}(\mathsf{Obs})$ is defined as follows.*

$$\begin{array}{ll}
\tilde{\mathcal{L}}(\mathbf{0}) = \tilde{\mathcal{L}}(\mathbf{1}) = \tilde{\mathcal{L}}(x) = \emptyset & \tilde{\mathcal{L}}(a.B) = \{a\} \cup \tilde{\mathcal{L}}(B) \\
\tilde{\mathcal{L}}(\tau.B) = \tilde{\mathcal{L}}(B) & \tilde{\mathcal{L}}(B_1 \|_A B_2) = \tilde{\mathcal{L}}(B_1) \cup \tilde{\mathcal{L}}(B_2) \cup A \\
\tilde{\mathcal{L}}(B_1 + B_2) = \tilde{\mathcal{L}}(B_1; B_2) = & \tilde{\mathcal{L}}(B_1 \,[\!> B_2) = \tilde{\mathcal{L}}(B_1) \cup \tilde{\mathcal{L}}(B_2) \\
\tilde{\mathcal{L}}(B\backslash A) = \tilde{\mathcal{L}}(B) \cup A & \tilde{\mathcal{L}}(B[(a \to B_a)^{a \in A}]) = \tilde{\mathcal{L}}(B) \cup A \cup \bigcup_{a \in A} \tilde{\mathcal{L}}(B_a)
\end{array}$$

Then, define $\mathcal{L} : \mathsf{PA_{sr}} \to \mathcal{P}_{count}(\mathsf{Obs})$ by $\mathcal{L}(\langle \mathrm{decl}, B\rangle) = \tilde{\mathcal{L}}(B) \cup \bigcup_{x \in \mathsf{Var}} \tilde{\mathcal{L}}(\mathrm{decl}(x))$.

The set A has to be added in the cases of the parallel operator and the hiding operator, since otherwise confusion may arise. For example, if action a is renamed b in $a\|_{\{b\}}\mathbf{1}$, we obtain a change in the behavior, since the process is deadlocked after the renaming.

The transition rules of $\longrightarrow^s_{\mathrm{decl}} \subseteq \mathsf{EXP_{sr}} \times (\mathcal{A}ct_{\surd} \cup (\mathsf{Obs} \times \mathsf{Obs})) \times \mathsf{EXP_{sr}}$ are presented in Table 3.1, where γ is an element of $\mathcal{A}ct_{\surd} \cup (\mathsf{Obs} \times \mathsf{Obs})$. We write $\longrightarrow^s$ if decl is clear from the context. Here $\xrightarrow{(a,b)}$ means that one executable action a is renamed b and that all choices which would be triggered by the execution of this a are taken. $\xrightarrow{(a,b)}$ can be interpreted that action a has been started and that this action a will finish by executing action b.

In rule A_2, action a is relabeled by b. Rule C makes no difference between labels from $\mathcal{A}ct_{\surd}$ and $(\mathsf{Obs} \times \mathsf{Obs})$. Hence, in both cases the choice is triggered.

The disrupt operator $[\!>$ also disrupts the process if an (a, b) transition takes place, since in this chapter the start of an action triggers the disruption.

The parallel operator works asynchronously exactly for those actions which are not in $A \cup \{\surd\}$. For synchronization purposes, (a, b) is regarded to be in A if and only if $a \in A$. The rule for synchronization is divided into P_2 and P_3, since in the case where (a, b) is executed, we have to take care that further derivations will synchronize on b.

Rule Ref_1 considers the case when an action is executed which is not refined. In this case, only the term which is refined is modified. Rule Ref_2 considers the case when an action $\hat{a}$ which is refined by a non-terminating process is executed. Here $\hat{a}$ is renamed with a fresh action name, which is ensured by $b \notin A \cup \mathcal{L}(\langle \mathrm{decl}, B\rangle)$. Such a b always exists, since $|\mathsf{Obs}| > |\mathbb{N}|$. Furthermore, we keep all present refinements and add a new refinement for the executed action $\hat{a}$, which is now labeled with b. The case when the refined process terminates is considered in rule Ref_3. In this case, $\hat{a}$ terminates. Therefore, it has to be removed, which is done by taking the transition labeled with $\hat{a}$. The refinement remains unaffected.

The remaining rules are the standard ones [32].

Rules P_1, P_3, Ref_1, Res_1 sometimes derive processes with undesired behavior by executing (a, b). For example, $a \,\|_{\{b\}}\mathbf{1} \xrightarrow{(a,b)} b\|_{\{b\}}\mathbf{1}$, which deadlocks. The undesired behavior can only appear if action b occurs in the considered process. Such undesired behavior does not cause any problems, since we are only interested in transitions labeled with elements of $\mathcal{A}ct_{\surd}$. The only situation where we need (a, b) transitions is in rule Ref_2, but there we take care that b is fresh.

$$T : \frac{}{\mathbf{1} \xrightarrow{\surd} \mathbf{0}} \qquad A_1 : \frac{}{a.B \xrightarrow{a} B} \qquad A_2 : \frac{a, b \in \mathsf{Obs}}{a.B \xrightarrow{(a,b)} b.B} \qquad C : \frac{B_1 \xrightarrow{\gamma} B'}{\begin{array}{c} B_1 + B_2 \xrightarrow{\gamma} B' \\ B_2 + B_1 \xrightarrow{\gamma} B' \end{array}}$$

$$S_1 : \frac{B_1 \xrightarrow{\gamma} B_1' \quad \gamma \neq \surd}{B_1; B_2 \xrightarrow{\gamma} B_1'; B_2} \qquad S_2 : \frac{B_1 \xrightarrow{\surd} B_1'}{B_1; B_2 \xrightarrow{\tau} B_2}$$

$$I_1 : \frac{B_1 \xrightarrow{\gamma} B_1' \quad \gamma \neq \surd}{B_1 [> B_2 \xrightarrow{\gamma} B_1' [> B_2} \qquad I_2 : \frac{B_1 \xrightarrow{\surd} B_1'}{B_1 [> B_2 \xrightarrow{\surd} B_1'} \qquad I_3 : \frac{B_2 \xrightarrow{\gamma} B_2'}{B_1 [> B_2 \xrightarrow{\gamma} B_2'}$$

$$P_1 : \frac{B_1 \xrightarrow{\gamma} B_1' \quad \gamma \notin \{\surd\} \cup A \cup (A \times \mathsf{Obs})}{\begin{array}{c} B_1 \|_A B_2 \xrightarrow{\gamma} B_1' \|_A B_2 \\ B_2 \|_A B_1 \xrightarrow{\gamma} B_2 \|_A B_1' \end{array}}$$

$$P_2 : \frac{B_1 \xrightarrow{\gamma} B_1' \quad B_2 \xrightarrow{\gamma} B_2' \quad \gamma \in \{\surd\} \cup A}{B_1 \|_A B_2 \xrightarrow{\gamma} B_1' \|_A B_2'} \qquad P_3 : \frac{B_1 \xrightarrow{(a,b)} B_1' \quad B_2 \xrightarrow{(a,b)} B_2' \quad a \in A}{B_1 \|_A B_2 \xrightarrow{(a,b)} B_1' \|_{A \cup \{b\}} B_2'}$$

$$Ref_1 : \frac{B \xrightarrow{\gamma} B' \quad \gamma \notin A \cup (A \times \mathsf{Obs})}{B[(a \to B_a)^{a \in A}] \xrightarrow{\gamma} B'[(a \to B_a)^{a \in A}]}$$

$$Ref_2 : \frac{B \xrightarrow{(\hat{a},b)} B' \quad \hat{a} \in A \quad b \notin A \cup \mathcal{L}(\langle \mathrm{decl}, B \rangle) \quad B_{\hat{a}} \xrightarrow{\gamma} B'' \quad \gamma \neq \surd}{B[(a \to B_a)^{a \in A}] \xrightarrow{\gamma} B'[(a \to B_a)^{a \in A}, (b \to B'')]}$$

$$Ref_3 : \frac{B \xrightarrow{\hat{a}} B' \quad \hat{a} \in A \quad B_{\hat{a}} \xrightarrow{\surd} B''}{B[(a \to B_a)^{a \in A}] \xrightarrow{\tau} B'[(a \to B_a)^{a \in A}]}$$

$$Res_1 : \frac{B \xrightarrow{\gamma} B' \quad \gamma \notin A \cup (A \times \mathsf{Obs})}{B \backslash A \xrightarrow{\gamma} B' \backslash A} \qquad Res_2 : \frac{B \xrightarrow{a} B' \quad a \in A}{B \backslash A \xrightarrow{\tau} B' \backslash A}$$

$$Rec : \frac{\mathrm{decl}(x) \xrightarrow{\gamma} B'}{x \xrightarrow{\gamma} B'}$$

Table 3.1: Transition Rules for $\longrightarrow^{s}_{\mathrm{decl}}$

Example 3.23 *We illustrate the transition rules by presenting a derivation path of the process* $\big(a\|_{\{a\}}(a + b.\mathbf{0})\big)\,[a \to c][c \to d]$ *in Figure 3.6. In the first step, action* a *is renamed* c*. This seems to be strange, since* c *will be refined to* d*. But it does not cause a problem, because the operational semantics refines* c *to* c' *before it gets under the influence of* $[c \to d]$*. The choice is triggered by the renaming transition as it is illustrated by the first derivation step.*

$$\begin{array}{c}
\big(a\|_{\{a\}}(a+b.\mathbf{0})\big)\,[a\to c][c\to d]\\
\downarrow d\\
\big(c\|_{\{a,c\}}c\big)\,[a\to c, c\to c'][c\to d, c'\to \mathbf{1}]\\
\downarrow \tau\\
\big(c''\|_{\{a,c,c''\}}c''\big)\,[a\to c, c\to c', c''\to \mathbf{1}][c\to d, c'\to \mathbf{1}]\\
\downarrow \tau\\
\big(\mathbf{1}\|_{\{a,c,c''\}}\mathbf{1}\big)\,[a\to c, c\to c', c''\to \mathbf{1}][c\to d, c'\to \mathbf{1}]\\
\downarrow \surd\\
\big(\mathbf{0}\|_{\{a,c,c''\}}\mathbf{0}\big)\,[a\to c, c\to c', c''\to \mathbf{1}][c\to d, c'\to \mathbf{1}]
\end{array}$$

Figure 3.6: Example of a Process Derivation with respect to $\longrightarrow^s_{\mathrm{decl}}$

Definition 3.24 (Operational Semantics) *The operational semantics* $\mathcal{O}^s : \mathsf{PA_{sr}} \to \mathbf{TS}$ *is given by* $\mathcal{O}^s(\langle \mathrm{decl}, B\rangle) = (\mathsf{EXP_{sr}}, \mathcal{A}ct_{\surd}, \longrightarrow_{\mathcal{O}^s}, B)$ *where* $\longrightarrow_{\mathcal{O}^s} = \longrightarrow^s_{\mathrm{decl}} \cap (\mathsf{EXP_{sr}} \times \mathcal{A}ct_{\surd} \times \mathsf{EXP_{sr}})$.

We still have to argue that the presented operational semantics is reasonable. The fact that the transitions labeled with elements of $\mathsf{Obs} \times \mathsf{Obs}$ always generate an infinite transition system is problematic. More precisely, if process B can execute $a \in \mathsf{Obs}$ then it can execute (a, b) for every $b \in \mathsf{Obs}$, which yields a transition system with uncountably infinite branches. Moreover, it is also possible to derive undesirable infinite derivations, for example $a_1.\mathbf{1} \xrightarrow{(a_1,a_2)} \cdots \xrightarrow{(a_i,a_{i+1})} \cdots$. Nevertheless, this does not cause any problem since only the transitions labeled with elements of $\mathcal{A}ct_{\surd}$ are used to define the operational semantics (Definition 3.24).

But we still have an infinitely branching transition system by rule Ref_2, for example $a[a \to a_1.a_2] \xrightarrow{a_1} b[a \to a_1.a_2, b \to a_2]$ for all $b \in \mathsf{Obs}\backslash\{a\}$. This is also no problem, since all the expressions are α-*equivalent* [22], i.e. they can be translated into each other by action renaming of the bound actions. In other words, in rule Ref_2, it is not important which $b \notin A \cup \mathcal{L}(\langle \mathrm{decl}, B\rangle)$ is chosen, since all choices yield α-equivalent expressions.

It is also possible to apply techniques, similar to [44], in order to obtain unique action renaming, which makes α-conversion obsolete. This technique is used in Chapter 7 to obtain an operational semantics (Section 7.4).

Our operational semantics is a meaningful semantics with respect to the denotational semantics, since the transition system derived from the denotational semantics is bisimilar to the operational semantics.

Theorem 3.25 (Consistency) *Let* $\langle \mathrm{decl}, B\rangle \in \mathsf{PA_{sr}}$. *Then the transition systems* $\mathcal{O}^s(\langle \mathrm{decl}, B\rangle)$ *and* $(\mathbf{CBES}, \mathcal{A}ct_{\surd}, \hookrightarrow, [\![\langle \mathrm{decl}, B\rangle]\!])$ *are bisimilar.*

Proof: The proof is given in Subsection 3.6.3. □

3.4.1 Modified Operational Semantics

The operational semantics we have just presented yields unnecessarily long terms, since we have to copy each refinement, even though it is used only once. Consider, for example, the process $(a.B')[a \to a_1.a_2.\cdots a_n.\mathbf{1}]$. Then we have $(a.B')[a \to a_1.a_2.\cdots a_n.\mathbf{1}] \xrightarrow{a_1} \cdots \xrightarrow{a_i} (a^{(i)}.B')[(a \to a_1.a_2.\cdots a_n.\mathbf{1}), (a^{(1)} \to a_2.\cdots a_n.\mathbf{1}), \cdots, (a^{(i)} \to a_{i+1}.\cdots a_n.\mathbf{1})]$ for $i \leq n$ and suitable actions $a^{(j)}$. Here, the refinement of $a^{(j)} \to a_{j+1}.\cdots a_n.\mathbf{1}$ is an unnecessary information for all $j < i$, since $a^{(j)}$ does not occur in $(a^{(i)}.B')$.

Our operational semantics does not only yield unnecessary long terms, it also yields infinite transition systems in cases where finite ones would be sufficient. For example, let $\mathrm{decl}(X) = a.X$, then $X[a \to b] \left(\xrightarrow{b}\xrightarrow{\tau}\right)^i X[a \to b, a^{(1)} \to \mathbf{1}, \cdots a^{(i)} \to \mathbf{1}]$. Therefore, we get an infinite transition system. But it is sufficient that $X[a \to b]$ evolves to $X[a \to b]$ by executing $(b\tau)^i$, since $X[a \to b]$ contains all necessary information.

To circumvent the unnecessary copy of the refinement, we divide Obs into two parts. One part (Obs_a) is used for active actions, i.e. actions which have been renamed. The other part (Obs_P) is used for actions in the original expression, i.e. in the expression on which the transition rules were applied. We then know that an executable action of Obs_a only appears 'once' in the process. More precisely, suppose B only contains actions of Obs_P and $B \xrightarrow{\gamma_1} \cdots \xrightarrow{\gamma_i} B' \xrightarrow{(b,c)} B''$, where $b \in \mathsf{Obs}_a$ and $c \notin \mathcal{L}(\langle \mathrm{decl}, B' \rangle)$. Then $B'[b \to \tilde{B}] \sim B''[c \to \tilde{B}]$. This gives us the advantage of keeping actions of Obs_a unrenamed.

Furthermore, when an action of Obs_a finishes, we remove it from those positions in the expressions in which they were inserted by rule P_3 and Ref_2. Therefore we obtain the advantage of generating more finite state transition systems. This can also be achieved when no α-conversion is considered, for example by choosing always the 'smallest' action of Obs_a that does not occur for action renaming.

The ideas mentioned above are formalized as follows. Let $\mathsf{Obs}_P, \mathsf{Obs}_a \subseteq \mathsf{Obs}$ with $|\mathsf{Obs}_P| = |\mathsf{Obs}_a| = |\mathbb{N}|$ and $\mathsf{Obs}_P \cap \mathsf{Obs}_a = \emptyset$. Then the modified transition rules are those of Table 3.1 where A_2, P_2, Ref_2 and Ref_3 are replaced by the transition rules presented in Table 3.2.

In rule A_2^m, it is now only possible to start (renaming) actions of Obs_P, since actions of Obs_a are considered to be active, and therefore they can not be started again. Furthermore, the action a in rule A_2^m can only be replaced by an action of Obs_a, since a becomes active.

Rule P_2 is modified by removing action γ from the synchronization set if it is an active action. This is possible, since γ terminates in this rule, and therefore it does not appear in $B_1' \|_{A'} B_2'$ when it is from Obs_a.

The rule Ref_2 is split into two rules. Rule $Ref_{2.1}^m$ considers the case when an action of Obs_P is refined. In this case the rule stays the same. And rule $Ref_{2.2}^m$ considers the case when an active action is refined. Here we do not rename the action nor do we change the expression which gets refined at all, i.e. we do not change B. We only check that $\hat{a}$ is an initial action, which is done by $B \xrightarrow{\hat{a}} B'$. This is necessary for the soundness, since we have a disrupt operator, and so not every started action which has not terminated has to be active. The change in the refinement in rule $Ref_{2.2}^m$ is directly done at $\hat{a}$, i.e. $\hat{a}$ is refined by B''.

In rule Ref_3^m we remove, similar to rule P_2^m, action γ from the expression if it is an active action.

$$A_2^m : \frac{a \in \mathsf{Obs}_P \quad b \in \mathsf{Obs}_a}{a.B \xrightarrow{(a,b)} b.B}$$

$$P_2^m : \frac{B_1 \xrightarrow{\gamma} B_1' \quad B_2 \xrightarrow{\gamma} B_2' \quad \gamma \in \{\surd\} \cup A \quad A' = A \backslash (\{\gamma\} \cap \mathsf{Obs}_a)}{B_1 \|_A B_2 \xrightarrow{\gamma} B_1' \|_{A'} B_2'}$$

$$Ref_{2.1}^m : \frac{B \xrightarrow{(\hat{a},b)} B' \quad \hat{a} \in A \cap \mathsf{Obs}_P \quad b \notin A \cup \mathcal{L}(\langle \mathrm{decl}, B \rangle) \quad B_{\hat{a}} \xrightarrow{\gamma} B'' \quad \gamma \neq \surd}{B[(a \to B_a)^{a \in A}] \xrightarrow{\gamma} B'[(a \to B_a)^{a \in A}, (b \to B'')]}$$

$$Ref_{2.2}^m : \frac{B \xrightarrow{\hat{a}} B' \quad \hat{a} \in A \cap \mathsf{Obs}_a \quad B_{\hat{a}} \xrightarrow{\gamma} B'' \quad \gamma \neq \surd}{B[(a \to B_a)^{a \in A}] \xrightarrow{\gamma} B[(a \to B_a)^{a \in A \backslash \{\hat{a}\}}, (\hat{a} \to B'')]}$$

$$Ref_3^m : \frac{B \xrightarrow{\hat{a}} B' \quad \hat{a} \in A \quad B_{\hat{a}} \xrightarrow{\surd} B'' \quad A' = A \backslash (\{\hat{a}\} \cap \mathsf{Obs}_a)}{B[(a \to B_a)^{a \in A}] \xrightarrow{\tau} B'[(a \to B_a)^{a \in A'}]}$$

Table 3.2: Modified Transition Rules for $\longrightarrow^s_{\mathrm{decl}}$

The modified transition rules and the original transition rules yield bisimilar transitions systems:

Theorem 3.26 *Suppose* $\langle \mathrm{decl}, B \rangle \in \mathsf{PA}_{\mathsf{sr}}$ *and* $\mathcal{L}(\langle \mathrm{decl}, B \rangle) \subseteq \mathsf{Obs}_P$.
Then $\mathcal{O}_m^s(\langle \mathrm{decl}, B \rangle)$ *and* $(\mathbf{CBES}, \mathcal{A}ct_{\surd}, \hookrightarrow, [\![\langle \mathrm{decl}, B \rangle]\!])$ *are bisimilar, where* $\mathcal{O}_m^s$ *is derived as* $\mathcal{O}^s$ *except that the modified transition rules are used.*

Proof: Similar to the proof of Theorem 3.25. □

3.5 Discussion

In this chapter, we have reproduced the standard action refinement approach (the choice is considered to be start-based) in order to give an introduction to the subject of action refinement. We had to define a subclass of (extended) bundle event structures, called closed bundle event structures, to yield a reasonable complete partial order on these kinds of event structures. Furthermore, we have presented a new technique to define an operational semantics on process algebras containing action refinement operators such that the operational and the denotational semantics are consistent. With this technique it is not necessary to extend the syntactical expressions of the process algebra in order to define the operational semantics.

In the following chapter, we take a first look on the end-based view. More precisely, we define an action refinement operator on closed bundle event structures that considers the conflict relation to be end-based triggered. We show that the start-based and the end-based views lead to different theories, for example the standard equivalences are not preserved by the end-based refinement operator. Therefore, we introduce new equivalences that are congruences for the end-based

refinement operator. Finally, we argue that closed bundle event structures are not appropriate to be used for the end-based view.

3.6 Proofs

3.6.1 Proof of Theorem 3.10

First we show that $\mathbf{CBES}$ is closed under restriction.

Lemma 3.27 *Let $\mathcal{E} \in \mathbf{CBES}$ and $E' \subseteq E$. Then $\mathcal{E} \upharpoonright E' \in \mathbf{CBES}$.*

Proof: Is an immediate consequence of Corollary 2.18. □

The following lemma is later used to verify the uniqueness of the minimal upper bounds in $\mathbf{CBES}$.

Lemma 3.28 *Two cbes are equal if and only if they coincide on every finite restriction, i.e. $\forall \mathcal{E}, \mathcal{E}' \in \mathbf{CBES} : (E = E' \wedge \forall \tilde{E} \in \mathcal{P}_{fin}(E) : \mathcal{E} \upharpoonright \tilde{E} = \mathcal{E}' \upharpoonright \tilde{E}) \Leftrightarrow \mathcal{E} = \mathcal{E}'$.*

Proof: It is easy to check that the conflict relations and the action-labeling functions coincide. Let $(E_n)_{n\in\mathbb{N}}$ be a finite, monotone approximation of E. Suppose $X \mapsto e$ (the other inclusion follows by symmetrical arguments). Then $e \in E_n \Rightarrow (X \cap E_n \mapsto_{\mathcal{E}\upharpoonright E_n} e)$. From the assumption we get $\mapsto_{\mathcal{E}\upharpoonright E_n} = \mapsto_{\mathcal{E}'\upharpoonright E_n}$. Hence, $X \mapsto' e$ by the approximation closedness condition of $\mathcal{E}'$. □

Proof of Theorem 3.10: We have to verify the following facts (where $\mathcal{E} = \bigsqcup_n \mathcal{E}_n$):

Reflexivity: Obvious.

Transitivity: Follows from Remark 3.5.

Antisymmetry: This is an immediate consequence of Remark 3.7.

Least element: It is easily seen that $(\emptyset, \emptyset, \emptyset, \emptyset)$ is a least element.

$\mathcal{E}$ is a cbes: The only non-trivial fact is the approximation closedness condition.

Suppose $e \in E$, $X \subseteq E$ and $(E'_k)_{k\in\mathbb{N}}$ is a finite, monotone approximation of E where $X \cap E'_k \in \{X' \cap E'_k \mid X' \mapsto e\}$ holds.

From the definition of $\bigsqcup_n \mathcal{E}_n$ we get that for all k there is a $X_k \subseteq \bigcup_{n\in\mathbb{N}} E_n$ such that $X_k \cap E'_k = X \cap E'_k$ and $\forall n : e \in E_n \Rightarrow (X_k \cap E_n) \mapsto_n e$. Now suppose $e \in E_n$, then we have $(X_k \cap E_n \cap E'_k) \mapsto_{\mathcal{E}_n\upharpoonright(E_n\cap E'_k)} e$ whenever $e \in E'_k$. Furthermore, $X_k \cap E_n \cap E'_k = X \cap E_n \cap E'_k$. Therefore, $X \cap E_n \mapsto_n e$, since $\mathcal{E}_n \in \mathbf{CBES}$ and $(E_n \cap E'_k)_{k\in\mathbb{N}}$ is a finite, monotone approximation of E_n. And so by definition $X \mapsto e$.

$\mathcal{E}$ is an upper bound: Trivial, except for the bundle relation.

Suppose $X \mapsto e$ and $e \in E_k$. Then $(X \cap E_k) \mapsto_k e$ by definition of $\bigsqcup_n \mathcal{E}_n$.

Suppose $X_k \mapsto_k e$. Let $\Phi : \mathbb{N} \to \mathcal{P}(E)$ such that $\Phi(n) = X_k \cap E_n$ if $n \leq k$ and if $n > k$ then $\Phi(n) \in \{X' \mid X' \mapsto_n e \wedge X' \cap E_{n-1} = \Phi(n-1)\}$. The existence of such a kind of function follows from the axiom of choice, since $\mathcal{E}_{n-1} \trianglelefteq \mathcal{E}_n$ implies that the above set is nonempty.

Furthermore, we show by induction that $E_n \cap \Phi(i) = \Phi(n)$ whenever $i \geq n$. If $i = n$, then the statement follows, since $\Phi(j) \subseteq E_j$. Suppose $i > n$ then $E_n \cap \Phi(i) = E_n \cap E_{i-1} \cap \Phi(i) = E_n \cap \Phi(i-1)$ and the rest follows by induction.

Therefore, $E_n \cap \bigcup_i \Phi(i) = \Phi(n)$, since $(\Phi(i))_{i \in \mathbb{N}}$ is monotone. And from the definition of Φ we have $\Phi(n) \mapsto_n e$ whenever $e \in E_n$. Hence, $\bigcup_i \Phi(i) \mapsto e$ and $E_k \cap \bigcup_i \Phi(i) = X_k$.

$\mathcal{E}$ is the least upper bound: Suppose $\forall n : \mathcal{E}_n \trianglelefteq \mathcal{E}'$. Then $E \subseteq E'$. Hence, $\mathcal{E}' \restriction E \in \mathbf{CBES}$ by Lemma 3.27.

Suppose $\tilde{E} \in \mathcal{P}_{fin}(E)$. Then there is an n such that $\tilde{E} \subseteq E_n$. Therefore, $(\mathcal{E}' \restriction E) \restriction \tilde{E} = \mathcal{E}' \restriction \tilde{E} = (\mathcal{E}' \restriction E_n) \restriction \tilde{E} = \mathcal{E}_n \restriction \tilde{E} = (\mathcal{E} \restriction E_n) \restriction \tilde{E} = \mathcal{E} \restriction \tilde{E}$. Hence, $\mathcal{E}' \restriction E = \mathcal{E}$ by Lemma 3.28, i.e. $\mathcal{E} \trianglelefteq \mathcal{E}'$. □

3.6.2 Proof of Proposition 3.18

First, we show the well definedness and then the continuity.

Lemma 3.29 *All operators of Definition 3.17 are well defined, i.e. they really yield elements of* $\mathbf{CBES}$.

Proof: We only present the cases of the parallel and the refinement operator. The well-definedness of the other operators can be easily checked.

$\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2$: Let $(\tilde{e}_1, \tilde{e}_2) \in E_{\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2}$ Define $M_i = \{X \mid X \mapsto_i \tilde{e}_i\}$. Then M_i is approximation closed with respect to $E_i \cup \{\star\}$. Thus by Corollary 2.20 $M' = \{\{(e_1, e_2) \in (E_1 \cup \{\star\}) \times (E_2 \cup \{\star\}) \mid e_i \in X_i\} \mid i \in \{1,2\} \wedge X_i \in M_i\}$ is approximation closed with respect to $(E_1 \cup \{\star\}) \times (E_2 \cup \{\star\})$. And so the approximation closedness of $\mapsto_{\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2}$ follows from Corollary 2.18, since $\{X \mid X \mapsto_{\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2} (\tilde{e}_1, \tilde{e}_2)\} = \{X' \cap E_{\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2} \mid X' \in M'\}$.

$Ref^s_A(\mathcal{E}, \theta)$: Let $(e', \hat{e}') \in E_{Ref^s_A(\mathcal{E},\theta)}$. Define $M = \{X \mid X \mapsto e'\}$ and

$$M_1 = \begin{cases} \{\{e'\} \times \hat{X} \mid \hat{X} \mapsto_{\theta(l(e'))} \hat{e}'\} & \text{if } l(e') \in A \\ \emptyset & \text{otherwise} \end{cases}.$$

Then M_1 is approximation closed with respect to $E_{Ref^s_A(\mathcal{E},\theta)}$. Furthermore, let

$$E_e = \begin{cases} E_{\theta(l(e))} & \text{if } l(e) \in A \\ \{e\} & \text{otherwise} \end{cases} \text{ and } M_e = \begin{cases} \{\text{exit}(\theta(l(e)))\} & \text{if } l(e) \in A \\ \{\{e\}\} & \text{otherwise} \end{cases}.$$

Obviously, M_e is approximation closed with respect to E_e.

From Corollary 2.21 we obtain that

$$M_2 = \begin{cases} \emptyset & \text{if } l(e') \in A \wedge \hat{e}' \notin \text{init}(\theta(l(e'))) \\ \{\{(e, \hat{e}) \mid e \in X \wedge \hat{e} \in X_e\} \mid X \in M \wedge X_e \in M_e\} & \text{otherwise} \end{cases}$$

is approximation closed with respect to $E_{Ref^s_A(\mathcal{E},\theta)}$. And so the approximation closedness of $\mapsto_{Ref^s_A(\mathcal{E},\theta)}$ follows from Proposition 2.15, since $\{X' \mid X' \mapsto_{Ref^s_A(\mathcal{E},\theta)} (e', \hat{e}')\} = M_1 \cup M_2$.

The other conditions are easy to check. □

To simplify the verification of the continuity, we use Winskel's *continuity on events* [178].

Definition 3.30 *Let D be a cpo. An operator $F : D \to \mathbf{CBES}$ is* continuous on events *if and only if F is monotonic and for every ω-chain $(d_i)_{i\in\mathbb{N}}$ in D we have $E_{F(\bigsqcup_i d_i)} \subseteq E_{\bigsqcup_i F(d_i)}$.*

Lemma 3.31 *Let D be a cpo and let $F : D \to \mathbf{CBES}$. Then F is continuous if and only if F is continuous on events.*

Proof: Let $\sqsubseteq$ be the partial order of D. Continuity on events is obviously implied by continuity. Now let $(d_i)_{i\in\mathbb{N}}$ be an ω-chain in D. Then

$$
\begin{array}{lll}
\forall i : d_i \sqsubseteq \bigsqcup_i d_i & \overset{F \text{ is monotonic}}{\Rightarrow} & \forall i : F(d_i) \trianglelefteq F(\bigsqcup_i d_i) \\
& \Rightarrow & \bigsqcup_i F(d_i) \trianglelefteq F(\bigsqcup_i d_i) \\
& \overset{F \text{ continuous on events}}{\Rightarrow} & \bigsqcup_i F(d_i) \trianglelefteq F(\bigsqcup_i d_i) \wedge E_{F(\bigsqcup_i d_i)} \subseteq E_{\bigsqcup_i F(d_i)} \\
& \overset{\text{Remark 3.7}}{\Rightarrow} & \bigsqcup_i F(d_i) = F(\bigsqcup_i d_i),
\end{array}
$$

which completes the proof. □

Lemma 3.32 *All operators from Definition 3.17 are continuous with respect to $\trianglelefteq$.*

Proof: It is straightforward to check that every operator from Definition 3.17 considered componentwise is continuous on events. Hence, the statement follows by Lemma 3.31 and Lemma 2.10. □

3.6.3 Proof of Theorem 3.25

We introduce an event-based transition relation. Then we show that its corresponding transition system is bisimilar to $\mathcal{O}^s(\langle \mathrm{decl}, B\rangle)$ and that it is bisimilar to $(\mathbf{CBES}, \mathcal{A}ct_{\surd}, \hookrightarrow, [\![\langle \mathrm{decl}, B\rangle]\!])$. And so Theorem 3.25 follows by the transitivity of bisimilarity [138, page 90]. The event-based transition relation is particularly introduced to handle unguarded recursion.

Event-Based Transition Systems.

We want to define event-based transition rules such that the corresponding action occurrence is denoted by the event which corresponds to this action occurrence in the denotational semantics. Therefore, the information of the original positions (events) has to be kept in the derived expressions. This is done by defining the expression set $\mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}}$, which contains exactly those elements generated by

$$C \quad ::= \quad B \mid C; B \mid C \,[\!> B \mid C \|_A C \mid C[(a \to C)^{a\in A}] \mid C\backslash A \mid \lceil C \rceil_i$$

where $a \in \mathsf{Obs}$, $B \in \mathsf{EXP}_{\mathsf{sr}}$, $i \in \{1,2\}$ and $A \subseteq \mathsf{Obs}$ with $|A| \leq |\mathbb{N}|$. The symbols in the definition of $\mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}}$, e.g. $[>$, are overloaded, since they are also used in the definition of $\mathsf{EXP}_{\mathsf{sr}}$. Hence, the unique derivation of an expression of $\mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}}$ is contradicted. Nevertheless, it does not harm our theory (both have the same transition rule) and therefore, we use the same symbols, especially in order to reduce the numbers of the transition rules.

We do not need to extend the declaration, i.e. we define $\mathsf{PA}^{\mathsf{e}}_{\mathsf{sr}} = (\mathsf{Var} \to \mathsf{EXP}_{\mathsf{sr}}) \times \mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}}$.

The function $\mathcal{L}$ is adapted to $\mathcal{L}' : \mathsf{PA}^{\mathsf{e}}_{\mathsf{sr}} \to \mathcal{P}_{count}(\mathsf{Obs})$ as follows.

$$
\begin{array}{l}
\mathcal{L}'(\langle \mathrm{decl}, B\rangle) = \mathcal{L}(\langle \mathrm{decl}, B\rangle)\\
\mathcal{L}'(\langle \mathrm{decl}, C; B\rangle) = \mathcal{L}'(\langle \mathrm{decl}, C\,[> B\rangle) = \mathcal{L}'(\langle \mathrm{decl}, C\rangle) \cup \mathcal{L}'(\langle \mathrm{decl}, B\rangle)\\
\mathcal{L}'(\langle \mathrm{decl}, C_1 \|_A C_2\rangle) = \mathcal{L}'(\langle \mathrm{decl}, C_1\rangle) \cup \mathcal{L}'(\langle \mathrm{decl}, C_2\rangle) \cup A\\
\mathcal{L}'(\langle \mathrm{decl}, C[(a \to C_a)^{a\in A}]\rangle) = \mathcal{L}'(\langle \mathrm{decl}, C\rangle) \cup A \cup \bigcup_{a\in A} \mathcal{L}'(\langle \mathrm{decl}, C_a\rangle)\\
\mathcal{L}'(\langle \mathrm{decl}, C\backslash A\rangle) = \mathcal{L}'(\langle \mathrm{decl}, C\rangle) \cup A\\
\mathcal{L}'(\langle \mathrm{decl}, \lceil C\rceil_i\rangle) = \mathcal{L}'(\langle \mathrm{decl}, C\rangle)
\end{array}
$$

The event transition rules $\longrightarrow'_{\mathrm{decl}} \subseteq \mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}} \times ((\mathcal{A}ct_{\surd} \cup \mathsf{Obs} \times \mathsf{Obs}) \times \mathcal{U}) \times \mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}}$ are presented in Table 3.3. The elements of $\mathcal{U}$ in the transitions labels encode the position of the execution such that they correspond exactly to the events labeled by the denotational semantics. The original event positions are kept by using the $\lceil\_\rceil_i$ expressions. Another possibility to encode this information is presented in [38].

The First Bisimilarity Result.

We define a relation between $\mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}}$ and $\mathsf{EXP}_{\mathsf{sr}}$ which yields a bisimulation. An expression C of $\mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}}$ and an expression B of $\mathsf{EXP}_{\mathsf{sr}}$ are related if C results in B by removing all $\lceil\_\rceil$ expressions. This is formalized by the following function, where we also count the $\lceil\_\rceil$ symbols in C.

Definition 3.33 $\Xi : \mathbb{N} \times \mathsf{EXP}_{\mathsf{sr}} \to \mathcal{P}(\mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}})$ *is defined as follows*

$$
\begin{array}{l}
\Xi(0, B) = \{B\}\\
\Xi(n+1, B) = \{\lceil \tilde{C}\rceil_i \mid i \in \{1,2\} \wedge \tilde{C} \in \Xi(n, B)\}\\
\qquad\qquad \textit{if } B \in \{\mathbf{0}, \mathbf{1}, a.B_1, \tau.B_1, B_1 + B_2, x\}\\
\Xi(n+1, B_1; B_2) = \{\lceil \tilde{C}\rceil_i \mid i \in \{1,2\} \wedge \tilde{C} \in \Xi(n, B_1; B_2)\} \cup\\
\quad \{C_1; B_2 \mid C_1 \in \Xi(n+1, B_1)\}\\
\Xi(n+1, B_1\,[> B_2) = \{\lceil \tilde{C}\rceil_i \mid i \in \{1,2\} \wedge \tilde{C} \in \Xi(n, B_1\,[> B_2)\} \cup\\
\quad \{C_1\,[> B_2 \mid C_1 \in \Xi(n+1, B_1)\}\\
\Xi(n+1, B_1 \|_A B_2) = \{\lceil \tilde{C}\rceil_i \mid i \in \{1,2\} \wedge \tilde{C} \in \Xi(n, B_1 \|_A B_2)\} \cup\\
\quad \{C_1 \|_A C_2 \mid \exists m \in \mathbb{N} : m \leq n+1 \wedge C_1 \in \Xi(m, B_1) \wedge C_2 \in \Xi(n+1-m, B_2)\}\\
\Xi(n+1, B[(a \to B_a)^{a\in A}]) = \{\lceil \tilde{C}\rceil_i \mid i \in \{1,2\} \wedge \tilde{C} \in \Xi(n, B[(a \to B_a)^{a\in A}])\} \cup\\
\quad \{C[(a \to C_a)^{a\in A}] \mid \exists m \in \mathbb{N}, (m_a)_{a\in A} \in \mathbb{N}^A : m + \textstyle\sum_{a\in A} m_a = n+1 \wedge\\
\qquad C_a \in \Xi(m_a, B_a) \wedge C \in \Xi(m, B)\}\\
\Xi(n+1, B\backslash A) = \{\lceil \tilde{C}\rceil_i \mid i \in \{1,2\} \wedge \tilde{C} \in \Xi(n, B\backslash A)\} \cup\\
\quad \{C\backslash A \mid C \in \Xi(n+1, B)\}
\end{array}
$$

$$T' : \frac{}{\mathbf{1} \xrightarrow[\bullet]{\surd} \mathbf{0}} \qquad A'_1 : \frac{}{a.B \xrightarrow[\bullet]{a} \lceil B \rceil_1} \qquad A'_2 : \frac{a, b \in \mathsf{Obs}}{a.B \xrightarrow[\bullet]{(a,b)} b.B} \qquad C' : \frac{B_1 \xrightarrow[e]{\gamma} C}{\begin{array}{l} B_1 + B_2 \xrightarrow[(\star_1,e)]{\gamma} \lceil C \rceil_1 \\ B_2 + B_1 \xrightarrow[(\star_2,e)]{\gamma} \lceil C \rceil_2 \end{array}}$$

$$S'_1 : \frac{C \xrightarrow[e]{\gamma} C' \quad \gamma \neq \surd}{C; B \xrightarrow[(\star_1,e)]{\gamma} C'; B} \qquad S'_2 : \frac{C \xrightarrow[e]{\surd} C'}{C; B \xrightarrow[(\star_1,e)]{\tau} \lceil B \rceil_2}$$

$$I'_1 : \frac{C \xrightarrow[e]{\gamma} C' \quad \gamma \neq \surd}{C \lceil\!> B \xrightarrow[(\star_1,e)]{\gamma} C' \lceil\!> B} \qquad I'_2 : \frac{C \xrightarrow[e]{\surd} C'}{C \lceil\!> B \xrightarrow[(\star_1,e)]{\surd} \lceil C' \rceil_1} \qquad I'_3 : \frac{B \xrightarrow[e]{\gamma} B'}{C \lceil\!> B \xrightarrow[(\star_2,e)]{\gamma} \lceil B' \rceil_2}$$

$$P'_1 : \frac{C_1 \xrightarrow[e]{\gamma} C'_1 \quad \gamma \notin \{\surd\} \cup A \cup A \times \mathsf{Obs}}{\begin{array}{c} C_1 \|_A C_2 \xrightarrow[(e,\star)]{\gamma} C'_1 \|_A C_2 \\ C_2 \|_A C_1 \xrightarrow[(\star,e)]{\gamma} C_2 \|_A C'_1 \end{array}}$$

$$P'_2 : \frac{C_1 \xrightarrow[e]{\gamma} C'_1 \quad C_2 \xrightarrow[e']{\gamma} C'_2 \quad \gamma \in \{\surd\} \cup A}{C_1 \|_A C_2 \xrightarrow[(e,e')]{\gamma} C'_1 \|_A C'_2} \qquad P'_3 : \frac{C_1 \xrightarrow[e]{(a,b)} C'_1 \quad C_2 \xrightarrow[e']{(a,b)} C'_2 \quad a \in A}{C_1 \|_A C_2 \xrightarrow[(e,e')]{(a,b)} C'_1 \|_{A \cup \{b\}} C'_2}$$

$$Ref'_1 : \frac{C \xrightarrow[e]{\gamma} C' \quad \gamma \notin A \cup A \times \mathsf{Obs}}{C[(a \to C_a)^{a \in A}] \xrightarrow[(e,e)]{\gamma} C'[(a \to C_a)^{a \in A}]}$$

$$Ref'_2 : \frac{C \xrightarrow[e]{(\hat{a},b)} C' \quad \hat{a} \in A \quad b \notin A \cup \mathcal{L}'(\langle \mathrm{decl}, C \rangle) \quad C_{\hat{a}} \xrightarrow[\hat{e}]{\gamma} C'' \quad \gamma \neq \surd}{C[(a \to C_a)^{a \in A}] \xrightarrow[(e,\hat{e})]{\gamma} C'[(a \to C_a)^{a \in A}, (b \to C'')]}$$

$$Ref'_3 : \frac{C \xrightarrow[e]{\hat{a}} C' \quad \hat{a} \in A \quad C_{\hat{a}} \xrightarrow[\hat{e}]{\surd} C''}{C[(a \to C_a)^{a \in A}] \xrightarrow[(e,\hat{e})]{\tau} C'[(a \to C_a)^{a \in A}]}$$

$$Res'_1 : \frac{C \xrightarrow[e]{\gamma} C' \quad \gamma \notin A \cup A \times \mathsf{Obs}}{C \backslash A \xrightarrow[e]{\gamma} C' \backslash A} \qquad Res'_2 : \frac{C \xrightarrow[e]{a} C' \quad a \in A}{C \backslash A \xrightarrow[e]{\tau} C' \backslash A}$$

$$Rec' : \frac{\mathrm{decl}(x) \xrightarrow[e]{\gamma} C}{x \xrightarrow[e]{\gamma} C} \qquad N' : \frac{C \xrightarrow[e]{\gamma} C'}{\lceil C \rceil_i \xrightarrow[(\star_i,e)]{\gamma} \lceil C' \rceil_i}$$

Table 3.3: Event-Based Transition Rules with respect to $\longrightarrow^s_{\mathrm{decl}}$

The well-definedness of Ξ is easily seen. Furthermore, Ξ has no effect on the action names occurred in the processes, i.e.

Lemma 3.34 $C \in \Xi(n, B) \Rightarrow \mathcal{L}(\langle \mathrm{decl}, B \rangle) = \mathcal{L}'(\langle \mathrm{decl}, C \rangle)$

Proof: By structural induction on B combined with n. □

Lemma 3.35 *Let* $B \in \mathsf{EXP}_{\mathsf{sr}}$ *then* $\mathcal{O}^s(\langle \mathrm{decl}, B\rangle)$ *is bisimilar to* $(\mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}}, \mathcal{A}ct_{\surd}, \longrightarrow'_{\mathcal{O}^s}, B)$ *where* $C \stackrel{\alpha}{\longrightarrow}'_{\mathcal{O}^s} C' \Leftrightarrow \exists e \in \mathcal{U} : C \stackrel{\alpha}{\stackrel{e}{\longrightarrow}}'_{\mathrm{decl}} C'$.

Proof: Define $\mathcal{R} = \{(B, C) \in \mathsf{EXP}_{\mathsf{sr}} \times \mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}} \mid \exists n : C \in \Xi(n, B)\}$. In order to verify that $\mathcal{R}$ is a bisimulation, we show

$$(B \stackrel{\gamma}{\longrightarrow}^s B' \wedge C \in \Xi(n, B)) \Rightarrow \exists e, C', m : C \stackrel{\gamma}{\stackrel{e}{\longrightarrow}}' C' \wedge C' \in \Xi(m, B') \tag{3.1}$$

The proof of (3.1) works by induction on the depth of inference of $B \stackrel{\alpha}{\longrightarrow} B'$, combined with the value of n. Then (3.1) can be easily checked with the following procedure:

- applying rule N whenever $C = \lceil \tilde{C} \rceil_i$. In these cases n is reduced by one and $B \stackrel{\alpha}{\longrightarrow}^s B'$ is unaffected. Therefore the hypothesis concludes the result.
- applying the correspondent rules of $B \stackrel{\alpha}{\longrightarrow}^s B'$ whenever C is different from $\lceil \tilde{C} \rceil_i$. In these cases the depth of inference is reduced and n gets not increased. Therefore the hypothesis concludes the result. In the case of rule Ref_2, we also use Lemma 3.34 for the conclusion.

Another fact is

$$(C \stackrel{\gamma}{\stackrel{e}{\longrightarrow}}' C' \wedge C \in \Xi(n, B)) \Rightarrow \exists B', m : B \stackrel{\gamma}{\longrightarrow}^s B' \wedge C' \in \Xi(m, B') \tag{3.2}$$

This equation can be seen by induction on the depth of inference of $C \stackrel{\gamma}{\stackrel{e}{\longrightarrow}}' C'$. In the case of rule Ref'_2, we also use Lemma 3.34.
Now we are ready to verify that $\mathcal{R}$ is a bisimulation:

- It is clear that $(B, B) \in \mathcal{R}$.
- Suppose $(B_1, C_1) \in \mathcal{R}$ and $B_1 \stackrel{\alpha}{\longrightarrow}_{\mathcal{O}^s} B_2$. Then $B_1 \stackrel{\alpha}{\longrightarrow}^s{}_{\mathrm{decl}} B_2$ by definition. Hence, $\exists e, C_2, m : C_1 \stackrel{\alpha}{\stackrel{e}{\longrightarrow}}' C_2 \wedge C_2 \in \Xi(m, B_2)$ by (3.1). Thus $C_1 \stackrel{\alpha}{\longrightarrow}'_{\mathcal{O}^s} C_2$ and $(B_2, C_2) \in \mathcal{R}$, as required.
- Suppose $(B_1, C_1) \in \mathcal{R}$ and $C_1 \stackrel{\alpha}{\longrightarrow}'_{\mathcal{O}^s} C_2$. Then $C_1 \stackrel{\alpha}{\stackrel{e}{\longrightarrow}}' C_2$ for some e. Hence, $\exists B_2, m : B_1 \stackrel{\alpha}{\longrightarrow}^s{}_{\mathrm{decl}} B_2 \wedge C_2 \in \Xi(m, B_2)$ by (3.2). Thus $B_1 \stackrel{\alpha}{\longrightarrow}_{\mathcal{O}^s} B_2$ and $(B_2, C_2) \in \mathcal{R}$, as required. □

Remark 3.36 *The transition systems mentioned in Lemma 3.35 are not isomorphic. Consider, for example,* decl *with* $\mathrm{decl}(x) = a; x$. *Then the expression* x *yields a finite transition system with respect to* $\mathcal{O}^s$*, whereas an infinite one is derived with respect to* $\longrightarrow'_{\mathcal{O}^s}$.

The Second Bisimilarity Result.

First, we show that the denotation of a variable is the same as the denotation of its corresponding expression.

Lemma 3.37 *Let* $\mathrm{decl} : \mathsf{Var} \to \mathsf{EXP}_{\mathsf{sr}}$ *and* $x \in \mathsf{Var}$. *Then* $[\![\langle \mathrm{decl}, x\rangle]\!] = [\![\langle \mathrm{decl}, \mathrm{decl}(x)\rangle]\!]$.

Proof: We have $[\![\langle \mathrm{decl}, x\rangle]\!] = [\![x]\!]_{\{[\![\mathrm{decl}]\!]\}} = \{[\![\mathrm{decl}]\!]\}(x) = \mathrm{fix}(\mathcal{F}_{\mathrm{decl}})(x) = \mathcal{F}_{\mathrm{decl}}(\{[\![\mathrm{decl}]\!]\})(x) = [\![\mathrm{decl}(x)]\!]_{\{[\![\mathrm{decl}]\!]\}} = [\![\langle \mathrm{decl}, \mathrm{decl}(x)\rangle]\!]$. □

We extend the denotational semantics to $\mathsf{PA}^{\mathsf{e}}_{\mathsf{sr}}$.

Definition 3.38 (Denotational semantics of $\mathsf{PA}^{\mathsf{e}}_{\mathsf{sr}}$)

$\widehat{Shift}_i : \mathbf{CBES} \to \mathbf{CBES}$ *with* $\widehat{Shift}_i(\mathcal{E}) = (\{\star_i\} \times E, \tilde{\rightsquigarrow}, \tilde{\mapsto}, \tilde{l})$ *where*

$$\begin{array}{lcl} \tilde{\rightsquigarrow} & = & \{((\star_i, e), (\star_i, e')) \mid (e, e') \in \rightsquigarrow\} \\ \tilde{\mapsto} & = & \{(\{\star_i\} \times X, (\star_i, e)) \mid (X, e) \in \mapsto\} \\ \tilde{l}(\star_i, e) = l(e) & & \end{array}$$

Furthermore, define $[\![\_]\!]' : \mathsf{PA}^{\mathsf{e}}_{\mathsf{sr}} \to \mathbf{CBES}$ *by*

$$\begin{array}{l} [\![\langle \mathrm{decl}, B\rangle]\!]' = [\![\langle \mathrm{decl}, B\rangle]\!] \\ [\![\langle \mathrm{decl}, C; B\rangle]\!]' = [\![\langle \mathrm{decl}, C\rangle]\!]' \,\widehat{;}\, [\![\langle \mathrm{decl}, B\rangle]\!]' \\ [\![\langle \mathrm{decl}, C\,[> B\rangle]\!]' = [\![\langle \mathrm{decl}, C\rangle]\!]' \widehat{[>} [\![\langle \mathrm{decl}, B\rangle]\!]' \\ [\![\langle \mathrm{decl}, C_1 \|_A C_2\rangle]\!]' = [\![\langle \mathrm{decl}, C_1\rangle]\!]' \widehat{\|}_A [\![\langle \mathrm{decl}, C_2\rangle]\!]' \\ [\![\langle \mathrm{decl}, C[(a \to C_a)^{a\in A}]\rangle]\!]' = Ref^s_A([\![\langle \mathrm{decl}, C\rangle]\!]', (a \to [\![\langle \mathrm{decl}, C_a\rangle]\!]')^{a\in A}) \\ [\![\langle \mathrm{decl}, C\backslash A\rangle]\!]' = [\![\langle \mathrm{decl}, C_2\rangle]\!]' \widehat{\backslash} A \\ [\![\langle \mathrm{decl}, \lceil C\rceil_i\rangle]\!]' = \widehat{Shift}_i([\![\langle \mathrm{decl}, C\rangle]\!]') \end{array}$$

It is clear that $[\![\_]\!]'$ is well defined.

Lemma 3.39 *Suppose* $\langle \mathrm{decl}, C\rangle \in \mathsf{PA}^{\mathsf{e}}_{\mathsf{sr}}$, $b \in \mathsf{Obs}$ *and* $b \notin \mathcal{L}'(\langle \mathrm{decl}, C\rangle)$. *Then for all* $e \in E_{[\![\langle \mathrm{decl}, C\rangle]\!]'}$ *we have* $l_{[\![\langle \mathrm{decl}, C\rangle]\!]'}(e) \neq b$.

Proof: First we show for any $\mathrm{decl} : \mathsf{Var} \to \mathsf{EXP}_{\mathsf{sr}}$ that for all $n \in \mathbb{N}$ and $B \in \mathsf{EXP}_{\mathsf{sr}}$ we have

$$b \notin \mathcal{L}(\langle \mathrm{decl}, B\rangle) \Rightarrow \forall e \in \pi_1([\![B]\!]_{\mathcal{F}^n_{\mathrm{decl}}(\bot)}) : b \neq \pi_4([\![B]\!]_{\mathcal{F}^n_{\mathrm{decl}}(\bot)})(e) \tag{3.3}$$

This can be easily checked by induction on n combined with the structure of B where the lexicographic order is used.

The main statement follows by structural induction on C. We only present the case $C = B \in \mathsf{EXP}_{\mathsf{sr}}$. By Lemma 3.20 we get that $[\![\langle \mathrm{decl}, B\rangle]\!] = \bigsqcup_n [\![B]\!]_{\mathcal{F}^n_{\mathrm{decl}}(\bot)}$. Then there is m such that $e \in \pi_1([\![B]\!]_{\mathcal{F}^m_{\mathrm{decl}}(\bot)})$. And so the result follows by (3.3). □

We introduce the following definition in order to obtain an adequate transition relation in $\mathbf{CBES}$ which is labeled with elements of $\mathsf{Obs} \times \mathsf{Obs}$.

Definition 3.40 *Let $\mathcal{E} \in \mathbf{CBES}$, $b \in \mathsf{Obs}$ and $e \in \mathrm{init}(\mathcal{E})$ such that $l(e) \in \mathsf{Obs}$. Then $\mathcal{E}_{[e/b]}$ of $\mathcal{E}$ is given by $(E', \leadsto', \mapsto', l')$ where*

$$\begin{array}{rcl} E' & = & \{e' \in E \mid \neg(e' \leadsto e)\} \\ \leadsto' & = & \leadsto \cap (E' \times E') \\ \mapsto' & = & \{(X \cap E', e') \mid e' \in E' \wedge X \mapsto e'\} \\ l'(e') & = & \begin{cases} b & \text{if } e' = e \\ l(e') & \text{otherwise} \end{cases} \end{array}$$

Lemma 3.41 *Let $\mathcal{E} \in \mathbf{CBES}$, $b \in \mathsf{Obs}$ and $e \in \mathrm{init}(\mathcal{E})$ such that $l(e) \in \mathsf{Obs}$. Then $\mathcal{E}_{[e/b]} \in \mathbf{CBES}$.*

Proof: Define $\tilde{\mathcal{E}} = (E, \leadsto, \mapsto, \tilde{l})$ with $\tilde{l}(e) = b$ and $\forall e' \in E \backslash \{e\} : \tilde{l}(e') = l(e')$. It is obvious that $\tilde{\mathcal{E}} \in \mathbf{CBES}$. Furthermore, $\mathcal{E}_{[e/b]} = \tilde{\mathcal{E}} \restriction \pi_1(\mathcal{E}_{[e/b]})$. And so the result follows by Lemma 3.27. □

Lemma 3.42 *Suppose $\mathcal{E}, \mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_a \in \mathbf{CBES}$. Then*

$$\begin{array}{rcl} (a \widehat{.} \mathcal{E})_{[\bullet]} & = & \widehat{Shift}_1(\mathcal{E}) \\ (\mathcal{E}_1 \widehat{+} \mathcal{E}_2)_{[(\star_i, e)]} & \simeq & \widehat{Shift}_i(\mathcal{E}_{i[e]}) \\ (\mathcal{E}_1 \widehat{;} \mathcal{E}_2)_{[(\star_1, e)]} & \simeq & \begin{cases} \widehat{Shift}_2(\mathcal{E}_2) & \text{if } l_1(e) = \surd \wedge e \in \mathrm{init}(\mathcal{E}_1) \\ \mathcal{E}_{1[e]} \widehat{;} \mathcal{E}_2 & \text{otherwise} \end{cases} \\ (\mathcal{E}_1 \widehat{[>} \mathcal{E}_2)_{[(\star_i, e)]} & \simeq & \begin{cases} \widehat{Shift}_1(\mathcal{E}_{1[e]}) & \text{if } l_1(e) = \surd \wedge i = 1 \\ \mathcal{E}_{1[e]} \widehat{[>} \mathcal{E}_2 & \text{if } l_1(e) \neq \surd \wedge i = 1 \\ \widehat{Shift}_2(\mathcal{E}_{2[e]}) & \text{if } i = 2 \end{cases} \\ (\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2)_{[(e_1, e_2)]} & \simeq & \begin{cases} \mathcal{E}_{1[e_1]} \widehat{\|}_A \mathcal{E}_2 & \text{if } e_2 = \star \wedge l_1(e_1) \notin A \cup \{\surd\} \\ \mathcal{E}_1 \widehat{\|}_A \mathcal{E}_{2[e_2]} & \text{if } e_1 = \star \wedge l_2(e_2) \notin A \cup \{\surd\} \\ \mathcal{E}_{1[e_1]} \widehat{\|}_A \mathcal{E}_{2[e_2]} & \text{if } l_1(e_1) = l_2(e_2) \in A \cup \{\surd\} \end{cases} \\ Ref^s_A(\mathcal{E}, \theta)_{[(e, \hat{e})]} & \simeq & \begin{cases} Ref^s_A(\mathcal{E}_{[e]}, \theta) & \text{if } (l_{\theta(l(e))}(\hat{e}) = \surd \wedge \hat{e} \in \mathrm{init}(\theta(l(e)))) \vee \\ & \quad (l(e) \notin A \wedge e = \hat{e}) \\ Ref^s_{A \cup \{b\}}(\mathcal{E}_{[e/b]}, \theta \cup \{(b, \theta(l(e))_{[\hat{e}]})\}) & \text{if } l_{\theta(l(e))}(\hat{e}) \neq \surd \end{cases} \\ & & \text{whenever } b \notin A \wedge \forall \tilde{e} \in E : l(\tilde{e}) \neq b \\ (\mathcal{E} \widehat{\backslash} A)_{[e]} & \simeq & \mathcal{E}_{[e]} \widehat{\backslash} A \\ \widehat{Shift}_i(\mathcal{E})_{[(\star_i, e)]} & \simeq & \widehat{Shift}_i(\mathcal{E}_{[e]}) \end{array}$$

Proof: Straightforward. □

Lemma 3.43 *Suppose $\mathcal{E}, \mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_a \in \mathbf{CBES}$ and $b \in \mathsf{Obs}$. Then*

$$\begin{array}{rcl} (a \widehat{.} \mathcal{E})_{[\bullet/b]} & = & b \widehat{.} \mathcal{E} \quad \text{if } a \in \mathsf{Obs} \\ (\mathcal{E}_1 \widehat{+} \mathcal{E}_2)_{[(\star_i, e)/b]} & \simeq & \widehat{Shift}_i(\mathcal{E}_{i[e/b]}) \\ (\mathcal{E}_1 \widehat{;} \mathcal{E}_2)_{[(\star_1, e)/b]} & \simeq & \mathcal{E}_{1[e/b]} \widehat{;} \mathcal{E}_2 \end{array}$$

$$
\begin{array}{rcl}
(\mathcal{E}_1 \widehat{[>} \mathcal{E}_2)_{[(\star_i,e)/b]} & \simeq & \left\{ \begin{array}{ll} \mathcal{E}_{1[e/b]} \widehat{[>} \mathcal{E}_2 & \textit{if } i = 1 \\ \widehat{Shift}_2(\mathcal{E}_{2[e/b]}) & \textit{if } i = 2 \end{array} \right. \\
(\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2)_{[(e_1,e_2)/b]} & \simeq & \left\{ \begin{array}{ll} \mathcal{E}_{1[e_1/b]} \widehat{\|}_A \mathcal{E}_2 & \textit{if } e_2 = \star \wedge l_1(e_1) \notin A \cup \{\surd\} \\ \mathcal{E}_1 \widehat{\|}_A \mathcal{E}_{2[e_2/b]} & \textit{if } e_1 = \star \wedge l_2(e_2) \notin A \cup \{\surd\} \\ \mathcal{E}_{1[e_1/b]} \widehat{\|}_{A \cup \{b\}} \mathcal{E}_{2[e_2/b]} & \textit{if } l_1(e_1) = l_2(e_2) \in A \cup \{\surd\} \end{array} \right. \\
 & & \textit{when } b \notin A \wedge \forall i \in \{1,2\} : \forall e \in E_i : l_i(e) \neq b \\
Ref^s_A(\mathcal{E}, \theta)_{[(e,\hat{e})/b]} & \simeq & \left\{ \begin{array}{ll} Ref^s_A(\mathcal{E}_{[e/b]}, \theta) & \textit{if } l(e) \notin A \wedge e = \hat{e} \\ Ref^s_{A \cup \{b'\}}(\mathcal{E}_{[e/b']}, \theta \cup \{(b', \theta(l(e))_{[\hat{e}/b]})\}) & \textit{if } l(e) \in A \end{array} \right. \\
 & & \textit{whenever } b' \notin A \wedge \forall \tilde{e} \in E : l(\tilde{e}) \neq b' \\
(\mathcal{E} \widehat{\setminus} A)_{[e/b]} & \simeq & \mathcal{E}_{[e/b]} \widehat{\setminus} A \quad \textit{when } b \notin A \wedge l(e) \notin A \\
\widehat{Shift}_i(\mathcal{E})_{[(\star_i,e)/b]} & \simeq & \widehat{Shift}_i(\mathcal{E}_{[e/b]})
\end{array}
$$

Proof: Straightforward. □

Lemma 3.44 *Suppose* $\langle \mathrm{decl}, C \rangle \in \mathsf{PA}^{\mathsf{e}}_{\mathsf{sr}}$ *and* $C \xrightarrow{\frac{\gamma}{e}}{}' C'$. *Then*

$$
\begin{array}{rcl}
(\gamma = (a,b) \wedge b \notin \mathcal{L}'(\langle \mathrm{decl}, C \rangle)) & \Rightarrow & e \in \mathrm{init}(\mathcal{E}) \wedge l(e) = a \wedge \mathcal{E}' = \mathcal{E}_{[e/b]} \\
\gamma \in \mathcal{A}ct_{\surd} & \Rightarrow & e \in \mathrm{init}(\mathcal{E}) \wedge l(e) = \gamma \wedge \mathcal{E}' = \mathcal{E}_{[e]}
\end{array}
$$

with $\mathcal{E} = [\![\langle \mathrm{decl}, C \rangle]\!]'$ *and* $\mathcal{E}' = [\![\langle \mathrm{decl}, C' \rangle]\!]'$.

Proof: We use induction on the depth of inference of $C \xrightarrow{\frac{\gamma}{e}}{}' C'$. Then the equation can be verified by case analysis on the derivation rules, where Lemma 3.42 and Lemma 3.43 are used. In the cases of P'_3, Ref'_2 and Res'_1, we also use Lemma 3.39. And in the case of Rec', we make use of Lemma 3.37. □

Lemma 3.45 *Let* $\langle \mathrm{decl}, C \rangle \in \mathsf{PA}^{\mathsf{e}}_{\mathsf{sr}}$, $e \in \mathrm{init}([\![\langle \mathrm{decl}, C \rangle]\!]')$ *and* $\alpha = l_{[\![\langle \mathrm{decl}, C \rangle]\!]'}(e)$. *Then*

$$
\exists C' \in \mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}} : C \xrightarrow{\frac{\alpha}{e}}{}' C' \wedge (\alpha \in \mathsf{Obs} \Rightarrow \forall b \in \mathsf{Obs} : \exists C'' \in \mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}} : C \xrightarrow{\frac{(\alpha,b)}{e}}{}' C'')
$$

Proof: First we show for any $\mathrm{decl} : \mathsf{Var} \to \mathsf{EXP}_{\mathsf{sr}}$:

$$
\begin{array}{l}
\forall n \in \mathbb{N} : \forall B \in \mathsf{EXP}_{\mathsf{sr}} : e \in \mathrm{init}([\![B]\!]_{\mathcal{F}^n_{\mathrm{decl}}(\perp)}) \wedge \alpha = \pi_4([\![B]\!]_{\mathcal{F}^n_{\mathrm{decl}}(\perp)})(e) \\
\Rightarrow \exists C' \in \mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}} : B \xrightarrow{\frac{\alpha}{e}}{}' C' \wedge (\alpha \in \mathsf{Obs} \Rightarrow \forall b \in \mathsf{Obs} : \exists C'' \in \mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}} : B \xrightarrow{\frac{(\alpha,b)}{e}}{}' C'')
\end{array} \tag{3.4}
$$

This is done by induction on n combined with the structure of B, where the lexicographic order is used. Furthermore, a case analysis on the structure of B is made. Here, we only present the case $B = x$: $e \in \mathrm{init}([\![x]\!]''_{\mathcal{F}^n_{\mathrm{decl}}(\perp)})$ implies that $n > 0$. Therefore, $[\![x]\!]_{\mathcal{F}^n_{\mathrm{decl}}(\perp)} = \mathcal{F}^n_{\mathrm{decl}}(\perp)(x) = [\![\mathrm{decl}(x)]\!]_{\mathcal{F}^{n-1}_{\mathrm{decl}}(\perp)}$. The rest follows by induction, since n is reduced. Thus (3.4) is established.
The main statement follows now by structural induction on C. We only present the case $C = B \in \mathsf{EXP}_{\mathsf{sr}}$. By Lemma 3.20 we get that $[\![\langle \mathrm{decl}, B \rangle]\!] = \bigsqcup_n [\![B]\!]_{\mathcal{F}^n_{\mathrm{decl}}(\perp)}$. Then it is easily seen that there is an m such that $e \in \mathrm{init}([\![B]\!]_{\mathcal{F}^m_{\mathrm{decl}}(\perp)})$ and $\alpha = \pi_4([\![B]\!]_{\mathcal{F}^m_{\mathrm{decl}}(\perp)})$. And so the result follows by (3.4). □

Lemma 3.46 *Let* $\langle \mathrm{decl}, C\rangle \in \mathsf{PA}^{\mathsf{e}}_{\mathsf{sr}}$*, then the transition systems* $(\mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}}, \mathcal{A}ct_{\surd}, \stackrel{\alpha}{\longrightarrow}\!\!{}'_{\mathcal{O}^s}, C)$ *and* $(\mathbf{CBES}, \mathcal{A}ct_{\surd}, \hookrightarrow, [\![\langle \mathrm{decl}, C\rangle]\!]')$ *are bisimilar, where* $\longrightarrow'_{\mathcal{O}^s}$ *is defined as in Lemma 3.35.*

Proof: Define $\mathcal{R} = \{(C', [\![\langle \mathrm{decl}, C'\rangle]\!]') \mid C' \in \mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}}\}$. Then $(C, [\![\langle \mathrm{decl}, C\rangle]\!]') \in \mathcal{R}$ by definition.

Suppose $C_1 \in \mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}}$ and $C_1 \stackrel{\alpha}{\longrightarrow}\!\!{}'_{\mathcal{O}^s} C_2$. Then $C_1 \stackrel{\alpha}{\stackrel{e}{\longrightarrow}}{}' C_2$ for some e. Hence, by Lemma 3.44 $[\![\langle \mathrm{decl}, C_1\rangle]\!]' \stackrel{\alpha}{\hookrightarrow} [\![\langle \mathrm{decl}, C_2\rangle]\!]'$, as required.

Suppose $C_1 \in \mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}}$ and $[\![\langle \mathrm{decl}, C_1\rangle]\!]' \stackrel{\alpha}{\hookrightarrow} \mathcal{E}_2$. Then there is $e \in \mathrm{init}([\![\langle \mathrm{decl}, C_1\rangle]\!]')$ such that $\mathcal{E}_2 = [\![\langle \mathrm{decl}, C_1\rangle]\!]'_{[e]}$ and $\alpha = \pi_4([\![\langle \mathrm{decl}, C_1\rangle]\!]')(e)$. From Lemma 3.45 we get the existence of $C_2 \in \mathsf{EXP}^{\mathsf{e}}_{\mathsf{sr}}$ such that $C_1 \stackrel{\alpha}{\stackrel{e}{\longrightarrow}}{}' C_2$. Moreover, $[\![\langle \mathrm{decl}, C_1\rangle]\!]'_{[e]} = [\![\langle \mathrm{decl}, C_2\rangle]\!]'$ by Lemma 3.44, which concludes the proof. □

Chapter 4

Modeling the End-Based View in CBES

A choice in concurrent systems is usually taken by the start of actions. In this chapter, we propose the alternative view that a choice is determined by the ending of actions, called end-based view, as this alternative has relevant applications and interesting implications, as illustrated in Section 1.3.

Another advantage of the end-based approach is that it can simplify the action refinement approach for timed systems. More precisely, an action refinement operator for timed bundle event structures [116, 120] is presented in [81, 133]. The authors extend each Q in a refinement $P[a \to Q]$ by an additional internal event, which corresponds to the start of Q. This is necessary in order to guarantee the start-based choice, i.e. to guarantee that the choice is triggered at the start of Q (at the time when action a starts to be executed in P). In other words, the choice is not triggered when Q executes its first action, which may be executed after a time period has passed. This approach is reasonable if generative systems[1], rather than reactive systems, are considered. The problem is that an undesired internal choice may be introduced by action refinement in reactive systems, for example refining a to a' and b to b' in $a + b$ yields $\tau.a' + \tau.b'$ in [81, 133]. We have the opinion that the introduction of additional internal events can be avoided in an end-based choice setting.

The different points of view (start-based vs end-based) lead to different refinement operators on $\mathbf{CBES}$. We introduce a refinement operator on closed bundle event structures for the end-based view. Furthermore, we show that the standard equivalences are not preserved by this refinement operator. Therefore, we introduce and study new equivalences that are preserved by our refinement operator.

4.1 An End-Based Refinement Operator on CBES

The decision at which time a choice is triggered does not influence the theory of most untimed semantic models. This situation changes when models that contain an action refinement operator are considered. Action refinement can, for example, split an action into a start- and an

[1]Generative systems are systems that are considered independently of the environment, i.e. the behavior is generated. Consequently, the branching structure is considered with respect to the abstractions of the internal actions.

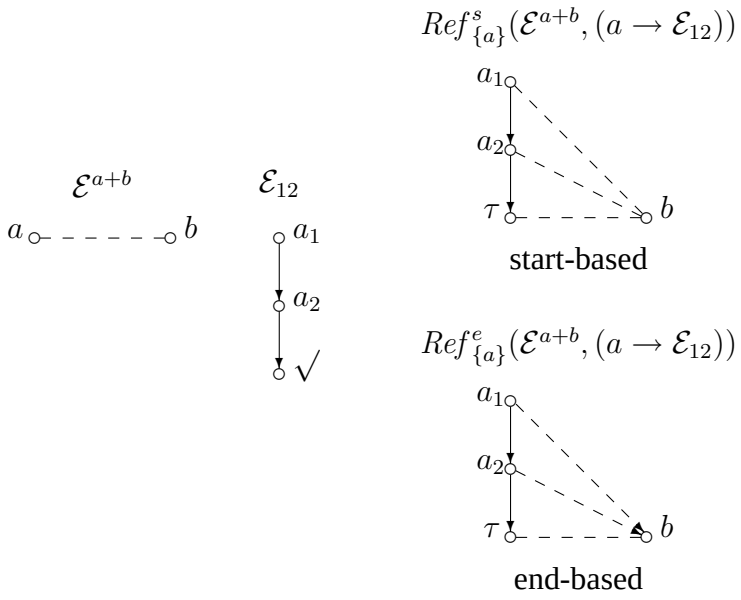


Figure 4.1: Start-Based versus End-Based Refinement

end-action. More precisely, if action a is refined to a_1 followed by a_2, i.e. a_2 executes after a_1, in a process consisting of a choice between actions a and b, then the execution sequence a_1, b is allowed in an end-based setting, which is not the case in a start-based setting. This example is illustrated in Figure 4.1, where we depict extended (or closed) bundle event structures (see Subsection 3.3.1 and Subsection 3.3.2) after the refinement in the start-based and in the end-based approaches. It can be seen in this figure that a_1 is in conflict with b in the start-based approach, whereas it is possible that a_1 precedes b in the end-based approach (a non-symmetric relation is used, depicted by dashed arrows). Thus the sequence a_1, b is only a trace of the event structure corresponding to the end-based view.

In the standard approach to action refinement, a choice is triggered by the start of actions, for example in [6, 63, 91, 98, 133, 172]. Here, we develop an action refinement operator for an untimed event structure with respect to the end-based point of view. Two constraints have to be imposed on the event structure in order to give a reasonable definition of such a refinement operator.

1. Each event in an event structure represents a unique occurrence of an action. This is necessary, since otherwise the occurrence of an action could be started more than once. In *prime event structures* [145] such a unique representation can not be guaranteed, as pointed out, for example, in [91, Section 2.3].

2. An event structure must allow to model disruption, since a disrupt operation can result from the end-based refinement operator, which will be discussed in more detail later in this section. This is for example not the case for *prime event structures, flow event structures* [36, 38] and *stable event structures* [178].

Therefore, we choose to define the end-based refinement operator on *closed bundle event structures* (**CBES**), which are introduced in Subsection 3.3.2.

The definition of the end-based refinement operator differs from the classical definition (Definition 3.17) with respect to the conflict relation: Only the termination events of the refining processes are used in our approach to define the conflict relation whereas every event (or every initial event) is used in the standard approach. More precisely, in the classical definition we have: If e is in conflict with e' and e (e' respectively) is refined to $\mathcal{E}_e$ ($\mathcal{E}_{e'}$ respectively), then every (initial) event of $\mathcal{E}_e$ is placed in conflict with every (initial) event of $\mathcal{E}_{e'}$ and vice versa. In our definition, we place every event of $\mathcal{E}_e$ in conflict with the termination events of $\mathcal{E}_{e'}$, i.e. they may only be executed if they are executed before every termination event of $\mathcal{E}_{e'}$. And, of course, we put every event of $\mathcal{E}_{e'}$ in conflict with the termination events of $\mathcal{E}_e$. By this approach, we guarantee that a choice is triggered by the ending of actions, i.e. by a termination event of the refining process. Formally:

Let τ, Obs, $\mathcal{A}ct_{\surd}$ and Var be defined as in Section 3.2.

Definition 4.1 (Refinement Operator) *Let $A \subseteq$ Obs. Then define $Ref^e_A : \mathbf{CBES} \times (A \to \mathbf{CBES}) \to \mathbf{CBES}$ by $Ref^e_A(\mathcal{E}, \theta) = (\tilde{E}, \tilde{\leadsto}, \tilde{\mapsto}, \tilde{l})$ where*

$$
\begin{array}{rcl}
\tilde{E} & = & \{(e,\hat{e}) \mid e \in E \wedge l(e) \in A \wedge \hat{e} \in E_{\theta(l(e))}\} \cup \\
 & & \{(e,e) \in E \times E \mid l(e) \notin A\} \\
\tilde{\leadsto} & = & \{((e_1,\hat{e}_1),(e_2,\hat{e}_2)) \mid (e_1 \leadsto e_2 \wedge (l(e_2) \in A \Rightarrow \hat{e}_2 \in \mathrm{exit}(\theta(l(e_2))))) \vee \\
 & & (e_1 = e_2 \wedge l(e_1) \in A \wedge \hat{e}_1 \neq \hat{e}_2 \wedge (\hat{e}_1 \leadsto_{\theta(l(e_1))} \hat{e}_2 \vee \hat{e}_2 \in \mathrm{exit}(\theta(l(e_1)))))\} \\
\tilde{\mapsto} & = & \{(\{e\} \times X', (e,\hat{e})) \mid l(e) \in A \wedge X' \mapsto_{\theta(l(e))} \hat{e}\} \cup \\
 & & \{(\tilde{X}, (e,\hat{e})) \mid \exists X : X \mapsto e \wedge (l(e) \in A \Rightarrow \hat{e} \in \mathrm{init}(\theta(l(e)))) \wedge \\
 & & \tilde{X} = \{(e',\hat{e}') \in \tilde{E} \mid e' \in X \wedge (l(e') \in A \Rightarrow \hat{e}' \in \mathrm{exit}(\theta(l(e'))))\}\}
\end{array}
$$

$$
\tilde{l}((e,\hat{e})) = \begin{cases} l(e) & \text{if } l(e) \notin A \\ l_{\theta(l(e))}(\hat{e}) & \text{if } l(e) \in A \wedge l_{\theta(l(e))}(\hat{e}) \neq \surd \\ \tau & \text{if } l(e) \in A \wedge l_{\theta(l(e))}(\hat{e}) = \surd \end{cases}
$$

Lemma 4.2 *The refinement operator Ref^e is well defined, i.e. it yields elements of* **CBES**.

Proof: The approximation closedness of $Ref^e_A(\mathcal{E}, \theta)$ follows from the approximation closedness of $Ref^s_A(\mathcal{E}, \theta)$ (Lemma 3.18), since they do not differ in their bundle relations. The other conditions are easy to check. □

Example 4.3 *Figure 4.2 illustrates how the refinement operator (Ref^e) behaves. For a better understanding, we augment the examples by process term descriptions of the systems (see Section 3.2). Furthermore, $(a \to \mathcal{E}_{12})$ denotes the function from $\{a\}$ to* **CBES** *that maps a to $\mathcal{E}_{12}$. The effect of the classical (start-based) action refinement operator on this example is depicted in Figure 3.4.*

Figure 4.2 illustrates that the events labeled by a_1 are not in conflict with each other in $\mathcal{E}^+_{ref}$. Only the events that correspond to the termination of $\mathcal{E}_{12}$ (they are labeled by τ in $\mathcal{E}^+_{ref}$) disable the other actions. In other words, both actions a in $\mathcal{E}^+_{ref}$ may start and execute their actions independently until one of them terminates.

Remark 4.4 *Our refinement operator allows the modeling of a disrupt mechanism as it is used, for example, in LOTOS [32] and in Chapter 3: Suppose the process described by $\mathcal{E}_2$ can disrupt*

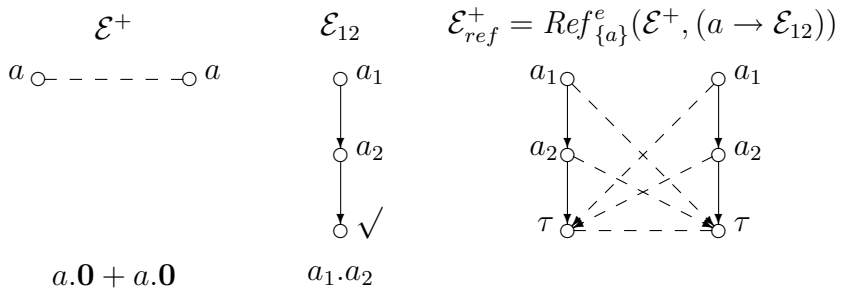


Figure 4.2: End-Based Refinement in CBES (1)

the process described by $\mathcal{E}_1$, i.e. $\mathcal{E}_1 \widehat{[>} \mathcal{E}_2$ where $\widehat{[>}$ is introduced in Definition 3.17. Then let $\mathcal{E}_1 \widehat{+} \mathcal{E}^a$ be the process that is obtained by taking a choice between $\mathcal{E}_1$ and the event structure $\mathcal{E}^a$ from Figure 4.4 where label a does not appear in $\mathcal{E}_1$. Then the event structures $\mathcal{E}_1 \widehat{[>} \mathcal{E}_2$ and $Ref^e_{\{a\}}(\mathcal{E}_1 + \mathcal{E}^a, (a \to \mathcal{E}_2))$ posses the same behavior.

4.2 Equivalences

First, we give a short overview of some standard equivalence notions and argue that they are not reasonable for an end-based action refinement operator, since they are not preserved by the end-based refinement operator. Then we introduce new equivalences that are preserved by the end-based refinement operator. These equivalences are examined with respect to their discriminating power. Finally, the issue concerning the coarsest equivalence with respect to trace (respectively bisimulation) equivalence is discussed.

4.2.1 Standard Equivalence Notions

Trace and *strong bisimulation equivalences* for CBES are defined as follows.

Definition 4.5 (Trace Equivalence) *Two $\mathcal{E}, \mathcal{E}' \in$ CBES are* trace equivalent, *denoted by $\mathcal{E} \sim_t \mathcal{E}'$, if and only if the transition systems $(\mathbf{CBES}, \mathcal{A}ct_{\surd}, \hookrightarrow, \mathcal{E})$ and $(\mathbf{CBES}, \mathcal{A}ct_{\surd}, \hookrightarrow, \mathcal{E}')$ (where $\hookrightarrow$ is defined in Definition 3.16) are trace equivalent (Definition 2.4).*

Definition 4.6 (Strong Bisimulation Equivalence) *Two $\mathcal{E}, \mathcal{E}' \in$ CBES are* strong bisimilar *(or strong bisimulation equivalent), denoted by $\mathcal{E} \sim_b \mathcal{E}'$, if and only if the transition systems $(\mathbf{CBES}, \mathcal{A}ct_{\surd}, \hookrightarrow, \mathcal{E})$ and $(\mathbf{CBES}, \mathcal{A}ct_{\surd}, \hookrightarrow, \mathcal{E}')$ are bisimilar (Definition 2.5).*

The cbes obtained from $(a + b.\mathbf{0}) \|_{\{a\}} (a \|_{\emptyset} a.c)$, depicted in Figure 3.5 (and in Figure 3.3), and the cbes obtained from $b.\mathbf{0} + a.c.\mathbf{0}$ and from $b.\mathbf{0} + a.c.\mathbf{0} + a.\mathbf{0}$, depicted in Figure 4.3 are all trace equivalent, whereas only the first and the third one are strong bisimilar.

Further equivalences that are based on interleaving [86, 89, 170] can be defined in a straightforward manner for cbes if one uses their corresponding transition system.

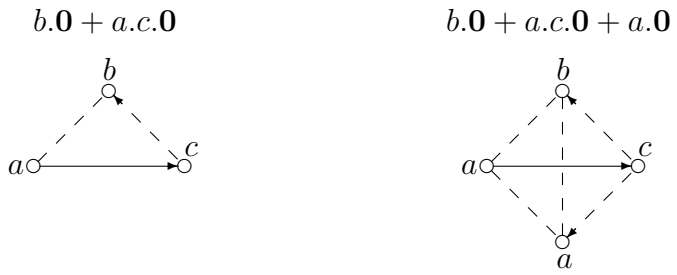


Figure 4.3: Trace Equivalent but not Bisimilar cbes

ST semantics, originally defined in [92], turns out to be the coarsest congruence for action refinement in the start-based setting [6, 85, 96, 173]. In the ST-approach, actions are not considered to be atomic, as in standard interleaving semantics. Instead, the execution of an action is split into the two distinguished events of the start and the ending (termination) of an action, where the ending is uniquely related to its start.

Further true concurrency equivalences are:

Step equivalence: Here, a finite multiset of actions, i.e. a finite set of events, may be executed at once, as opposed to the interleaving approach, where only single actions may be executed at once. In [152] trace and bisimulation versions of this equivalence have been proposed.

Pomset equivalence: Here a finite partially ordered set of events, more precisely a pomset [153], may be executed. Pomsets are equivalence classes with respect to the action labeling and the order. The order of a pomset corresponds to the causality order of the events. In [33] trace and bisimulation versions of this equivalence have been proposed.

History preserving equivalence: Here, the causal order in which events have been executed is additionally taken into account. There are different versions depending on to what degree the past information is taken into account. There exists *weak, normal* and *hereditary history preserving bisimulations* [23, 67, 91].

[91] examines which equivalence notions are preserved by a start-based action refinement operator in a configuration structure setting. There it is shown that pomset trace equivalence, history preserving bisimulation and hereditary history preserving bisimulation are preserved by a start-based action refinement operator. As mentioned before, the ST-equivalence is also preserved by a start-based action refinement. All other equivalences of this subsection are not preserved by a start-based action refinement.

The action refinement operator in an end-based setting is not compatible with the equivalence notions mentioned. This can be seen as follows: In the case of the end-based refinement operator (Ref^e) any equivalence that implies trace equivalence (Definition 4.5) and that identifies a and $a + a$ (like all the equivalences mentioned do) is not preserved. This is the case, because $\mathcal{E}^a_{ref}$ from Figure 4.4 and $\mathcal{E}^+_{ref}$ from Figure 4.2 are not trace equivalent. Resource bisimulation [58, 59], which is defined on transition systems, is the only equivalence known to us that does not identify $a + a$ and a.

In the following subsections, we present new equivalences which are indeed congruences for our refinement operator. For simplicity, we introduce the following definition, which determines

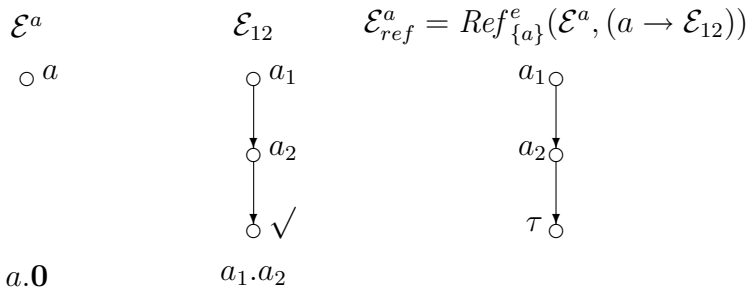


Figure 4.4: End-Based Refinement in CBES (2)

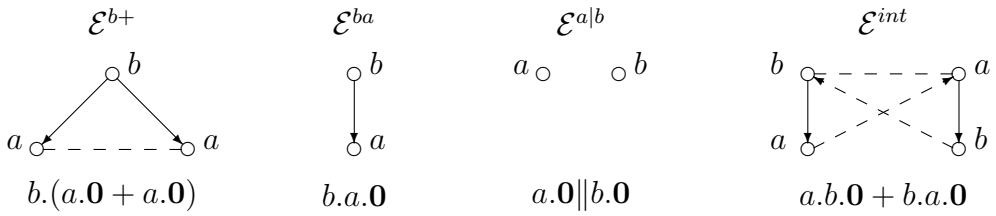


Figure 4.5: Some Closed Bundle Event Structures

the initial events of an event structure with respect to their labels.

Definition 4.7 *Define* $\mathrm{init}_A(\mathcal{E}) = \{e \in \mathrm{init}(\mathcal{E}) \mid l(e) \in A\}$, *where* $A \subseteq \mathcal{A}ct_\surd$. *Furthermore, we write* $\mathrm{init}_a(\mathcal{E})$ *as a short hand for* $\mathrm{init}_{\{a\}}(\mathcal{E})$.

4.2.2 ICT-Equivalence on CBES

The first considered equivalence notion is derived from trace equivalence. An equivalence notion which is a congruence for the end-based refinement operator has to distinguish between $\mathcal{E}^+$ from Figure 4.2 and $\mathcal{E}^a$ from Figure 4.4. One way to achieve this is to guarantee that the number of the initial events which are labeled by the same action have to be equal, i.e. $\mathcal{E}$ and $\mathcal{E}'$ can only be equivalent if $|\mathrm{init}_a(\mathcal{E})| = |\mathrm{init}_a(\mathcal{E})|$ for all $a \in \mathsf{Obs}$. Moreover, we also have to guarantee a relationship between the numbers of the initial events with the same label of the corresponding remainders of the event structures. For example, consider $\mathcal{E}^{b+}$ and $\mathcal{E}^{ba}$ from Figure 4.5. Then $(b, a_1, a_1) \in T(Ref^e_{\{a\}}(\mathcal{E}^{b+}, (a \to \mathcal{E}_{12})))$ but $(b, a_1, a_1) \notin T(Ref^e_{\{a\}}(\mathcal{E}^{ba}, (a \to \mathcal{E}_{12})))$, where $T(\mathcal{E})$ are the traces (Definition 2.4) of the derived transition system of $\mathcal{E}$ (compare with Definition 4.5). Hence, $Ref^e_{\{a\}}(\mathcal{E}^{b+}, (a \to \mathcal{E}_{12}))$ and $Ref^e_{\{a\}}(\mathcal{E}^{ba}, (a \to \mathcal{E}_{12}))$ are not trace equivalent.

Further difficulties become evident by a closer look at $\mathcal{E}^{a|b}$ and $\mathcal{E}^{int}$ from Figure 4.5: $\mathcal{E}^{a|b}$ and $\mathcal{E}^{int}$ satisfy our above criterion, but $(a_1, b, a_1) \in T(Ref^e_{\{a\}}(\mathcal{E}^{int}, (a \to \mathcal{E}_{12})))$ and $(a_1, b, a_1) \notin T(Ref^e_{\{a\}}(\mathcal{E}^{a|b}, (a \to \mathcal{E}_{12})))$, i.e. our tentative relation is not a congruence for the refinement.

Therefore, we introduce the *initial event traces* of a cbes. They consist of an event execution sequence and of a subset of the initial events for every execution step. Two cbes, $\mathcal{E}_1$ and $\mathcal{E}_2$,

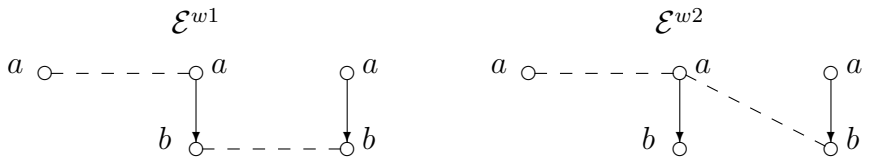


Figure 4.6: Non ICT-Equivalent cbes

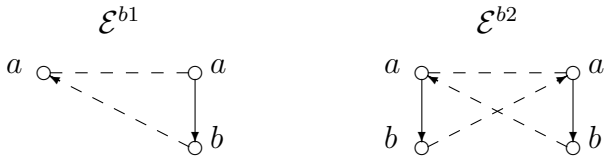


Figure 4.7: ICT-Equivalent cbes

are considered to be equivalent if every initial event trace of $\mathcal{E}_1$ can be mapped by an injective function f to an initial event trace of $\mathcal{E}_2$ and vice versa. Furthermore, this function has to be *labeling preserving*, i.e. $\forall e_1 \in E_1 : l_1(e_1) = l_2(f(e_1))$. The equivalence is precisely given by the following definition, where $\mathcal{E}_{[e]}$ is defined in Definition 3.14.

Definition 4.8 (ICT-equivalence) *Let $\mathcal{E} \in \mathbf{CBES}$. Then the* initial event traces *of $\mathcal{E}$ are defined by $T^{ic}(\mathcal{E}) = \{(e_i, \gamma_i)_{i\leq n} \mid n \in \mathbb{N} \wedge \exists \mathcal{E}_0, \cdots, \mathcal{E}_{n+1} : \mathcal{E}_0 = \mathcal{E} \wedge \forall i \leq n : \mathcal{E}_{i[e_i]} = \mathcal{E}_{i+1} \wedge \gamma_i \in \mathcal{P}_{fin}(\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_i))\}$.*

Two cbes, $\mathcal{E}$ and $\mathcal{E}'$, are initial corresponding trace equivalent *(ICT-equivalent), denoted by $\mathcal{E} \sim_{ICT} \mathcal{E}'$, if*

- *for every $(e_i, \gamma_i)_{i\leq n} \in T^{ic}(\mathcal{E})$ there exists an injective, labeling-preserving function $f : (\bigcup_{i\leq n}(\gamma_i \cup \{e_i\})) \to E'$ such that $(f(e_i), f(\gamma_i))_{i\leq n} \in T^{ic}(\mathcal{E}')$ and*
- *for every $(e'_i, \gamma'_i)_{i\leq n} \in T^{ic}(\mathcal{E}')$ there exists an injective, labeling-preserving function $f' : (\bigcup_{i\leq n}(\gamma'_i \cup \{e'_i\})) \to E$ such that $(f'(e'_i), f'(\gamma'_i))_{i\leq n} \in T^{ic}(\mathcal{E})$*

$\mathcal{E}^{b+}$ and $\mathcal{E}^{ba}$, and also $\mathcal{E}^{a|b}$ and $\mathcal{E}^{int}$ from Figure 4.5 are not ICT-equivalent. In addition, the event structures from Figure 4.6 are not ICT-equivalent either. This holds, since in $\mathcal{E}^{w1}$ it is possible that both events labeled by b become enabled, which is not the case for $\mathcal{E}^{w2}$. Examples of ICT-equivalent event structures are given in Figure 4.7 and in Figure 4.8.

Proposition 4.9 *Two ICT-equivalent cbes are also trace equivalent, i.e. $\sim_{ICT} \subset \sim_t$.*

Proof: It follows from the fact that every trace is also an initial trace, where the second component is always empty. □

Theorem 4.10 *ICT-equivalence is a congruence for the refinement operator Ref^e, i.e. $\mathcal{E} \sim_{ICT} \mathcal{E}' \wedge \forall a \in A : \theta(a) \sim_{ICT} \theta'(a)$ implies that $Ref^e_A(\mathcal{E}, \theta) \sim_{ICT} Ref^e_A(\mathcal{E}', \theta')$. Moreover, it is also a congruence for the operators $\widehat{.}$, $\widehat{+}$, $\widehat{;}$, $\widehat{[>}$, $\widehat{\|}_A$ and $\widehat{\setminus A}$, which are defined in Definition 3.17.*

Proof: The proof is given in Subsection 4.4.1. □

We are also interested in a congruence which implies strong bisimilarity (Definition 4.6). ICT-equivalence does not yield such an equivalence.

Lemma 4.11 *Strong bisimilarity does not follow from ICT-equivalence.*

Proof: The cbes $\mathcal{E}^{b1}$ and $\mathcal{E}^{b2}$ from Figure 4.7 are not bisimilar but ICT-equivalent. □

4.2.3 UI-Bisimilarity on CBES

An equivalence that is derived from bisimulation equivalence and that is a congruence for the end-based refinement operator has to relate the initial events, as it is done by ICT-equivalence. Therefore, we extend the definition of a bisimulation relation by a third component which denotes a bijection between the initial events.

Definition 4.12 (UI-Bisimulation) *A* unique initial bisimulation *(UI-bisimulation) $\mathcal{R}$ is a subset of $\mathbf{CBES} \times \mathbf{CBES} \times (\mathcal{U} \rightharpoonup \mathcal{U})$ such that whenever $(\mathcal{E}_1, \mathcal{E}_2, f) \in \mathcal{R}$, then*

- $\mathrm{dom}(f) = \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_1)$,
- *f is a labeling-preserving bijection between $\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_1)$ and $\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_2)$,*
- *$e_1 \in \mathrm{init}(\mathcal{E}_1)$ implies that there is e_2 and f' such that $l_1(e_1) = l_2(e_2)$ and $f \cup f'$ is an injective function and $(\mathcal{E}_{1[e_1]}, \mathcal{E}_{2[e_2]}, f') \in \mathcal{R}$ and $l_1(e_1) \in \mathsf{Obs} \Rightarrow e_2 = f(e_1)$*
- *$e_2 \in \mathrm{init}(\mathcal{E}_2)$ implies that there is e_1 and f' such that $l_1(e_1) = l_2(e_2)$ and $f \cup f'$ is an injective function and $(\mathcal{E}_{1[e_1]}, \mathcal{E}_{2[e_2]}, f') \in \mathcal{R}$ and $l_2(e_2) \in \mathsf{Obs} \Rightarrow e_2 = f(e_1)$*

We say that $\mathcal{E}_1, \mathcal{E}_2$ are UI-bisimilar *(or* UI-equivalent*), denoted by $\mathcal{E}_1 \sim_{UI} \mathcal{E}_2$, if and only if there is a UI-bisimulation $\mathcal{R}$ and an $f : \mathcal{U} \rightharpoonup \mathcal{U}$ such that $(\mathcal{E}_1, \mathcal{E}_2, f) \in \mathcal{R}$.*

The event structures from Figure 4.7 are not UI-equivalent, whereas the event structures from Figure 4.8 are UI-equivalent.

The condition in Definition 4.12 that $f \cup f'$ has to be a function ensures that the identification of the initial events of $\mathcal{E}_1$ is preserved after the execution, i.e. $f \upharpoonright (\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_1) \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{1[e_1]})) = f' \upharpoonright (\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_1) \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{1[e_1]}))$. The condition that $f \cup f'$ is an *injective* function guarantees that an initial event e_1' of $\mathcal{E}_1$ is kept after the execution if and only if $f(e_1')$ is kept after the corresponding execution, i.e. $e_1' \in \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_1) \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{1[e_1]}) \Leftrightarrow f(e_1') \in \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_2) \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{2[e_2]})$. This means that the identification of the initial events of $\mathcal{E}_2$ is also preserved after the execution, i.e. $f^{-1} \upharpoonright (\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_2) \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{2[e_2]})) = f'^{-1} \upharpoonright (\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_2) \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{2[e_2]}))$.

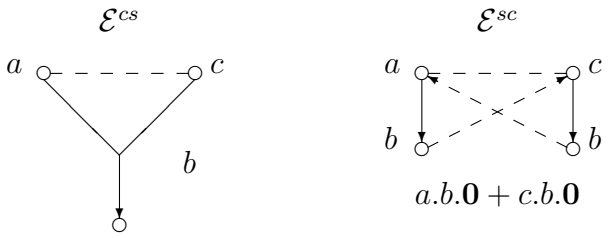


Figure 4.8: ICT-Equivalent and UI-Equivalent cbes

Proposition 4.13 *Two UI-equivalent cbes are also strong bisimilar, i.e.* $\sim_{UI} \subset \sim_b$.

Proof: It follows from the fact that every UI-bisimulation restricted to its first and second component is a bisimulation. □

Theorem 4.14 *UI-equivalence is a congruence for the refinement operator* Ref^e*, i.e.* $\mathcal{E} \sim_{UI} \mathcal{E}' \wedge \forall a \in A : \theta(a) \sim_{UI} \theta'(a)$ *implies that* $Ref^e_A(\mathcal{E}, \theta) \sim_{UI} Ref^e_A(\mathcal{E}', \theta')$*. Moreover, it is also a congruence for the operators* $\widehat{.}$, $\widehat{+}$, $\widehat{;}$, $\widehat{[>}$, $\widehat{\|}_A$ *and* $\widehat{\backslash}A$*, which are defined in Definition 3.17.*

Proof: The proof is given in Subsection 4.4.1. □

4.2.4 FUI-Bisimilarity on CBES

UI-equivalence has to preserve the correspondence of all initial events. This condition is not necessary in order to obtain a congruence that is contained in strong bisimilarity[2]. It is sufficient to guarantee that the correspondence of any finite subset of the initial events is preserved. This is formalized by the following definition.

Definition 4.15 (FUI-Bisimulation) *A* finite unique initial bisimulation *(FUI-bisimulation)* $\mathcal{R}$ *is a subset of* $\mathbf{CBES} \times \mathbf{CBES} \times (\mathcal{U} \rightharpoonup \mathcal{U})$ *such that whenever* $(\mathcal{E}_1, \mathcal{E}_2, f) \in \mathcal{R}$*, then*

- $\mathrm{dom}(f) = \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_1)$
- f *is a labeling-preserving bijection between* $\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_1)$ *and* $\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_2)$
- $e_1 \in \mathrm{init}(\mathcal{E}_1) \wedge I \in \mathcal{P}_{fin}(\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_1))$ *implies that there exists* e_2 *and* f' *such that* $l_1(e_1) = l_2(e_2)$ *and* $(\mathcal{E}_{1[e_1]}, \mathcal{E}_{2[e_2]}, f') \in \mathcal{R}$ *and* $l_1(e_1) \in \mathsf{Obs} \Rightarrow e_2 = f(e_1)$ *and* $f \upharpoonright (I \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{1[e_1]})) = f' \upharpoonright I$ *and* $f^{-1} \upharpoonright (f(I) \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{2[e_2]})) = f'^{-1} \upharpoonright f(I)$
- $e_2 \in \mathrm{init}(\mathcal{E}_2) \wedge I \in \mathcal{P}_{fin}(\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_2))$ *implies that there exists* e_2 *and* f' *such that* $l_1(e_1) = l_2(e_2)$ *and* $(\mathcal{E}_{1[e_1]}, \mathcal{E}_{2[e_2]}, f') \in \mathcal{R}$ *and* $l_2(e_2) \in \mathsf{Obs} \Rightarrow e_2 = f(e_1)$ *and* $f \upharpoonright (f^{-1}(I) \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{1[e_1]})) = f' \upharpoonright f^{-1}(I)$ *and* $f^{-1} \upharpoonright (I \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{2[e_2]})) = f'^{-1} \upharpoonright I$

We say that $\mathcal{E}_1, \mathcal{E}_2$ *are* FUI-bisimilar *(or* FUI-equivalent*), denoted by* $\mathcal{E}_1 \sim_{FUI} \mathcal{E}_2$*, if and only if there is a FUI-bisimulation* $\mathcal{R}$ *and an* $f : \mathcal{U} \rightharpoonup \mathcal{U}$ *such that* $(\mathcal{E}_1, \mathcal{E}_2, f) \in \mathcal{R}$*.*

[2]Nevertheless, UI-equivalence is of interest if infinite events can be executed at a single execution step.

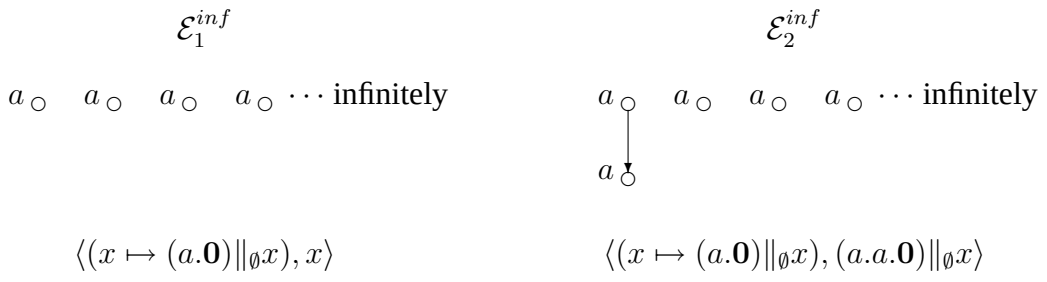


Figure 4.9: FUI-Equivalence Differs from UI-Equivalence

Remark 4.16 *Obviously, any cbes with a finite set of events is UI-equivalent if and only if it is FUI-equivalent. Furthermore, UI-equivalence differs from FUI-equivalence, since the event structures from Figure 4.9 are FUI-equivalent but not UI-equivalent.*

Proposition 4.17 *Two FUI-equivalent cbes are also strong bisimilar, i.e.* $\sim_{FUI} \subset \sim_b$.

Proof: It follows from the fact that every FUI-bisimulation restricted to its first and second component is a bisimulation. □

Theorem 4.18 *FUI-equivalence is a congruence for the refinement operator* Ref^e*, i.e.* $\mathcal{E} \sim_{FUI} \mathcal{E}' \wedge \forall a \in A : \theta(a) \sim_{FUI} \theta'(a)$ *implies that* $Ref^e_A(\mathcal{E}, \theta) \sim_{FUI} Ref^e_A(\mathcal{E}', \theta')$*. Moreover, it is also a congruence for the operators* $\widehat{.}$, $\widehat{+}$, $\widehat{;}$, $\widehat{[>}$, $\widehat{\|_A}$ *and* $\widehat{\setminus A}$*, which are defined in Definition 3.17.*

Proof: The proof is given in Subsection 4.4.1. □

Remark 4.19 *In Theorem 4.10, Theorem 4.14 and Theorem 4.18 the general parallel operator is allowed, i.e. no restriction on the synchronization set is made. But this operator has to be handled carefully, since it is not clear if it matches the intuitive meaning for an end-based setting: It is reasonable that the parallel operator introduces some start-based choices. For example, we expect that the expression* $(a + a) \|_{\{a\}} a.\mathbf{0}$ *can only start one* a*-action whereas its corresponding event structure, which is isomorphic to* $\mathcal{E}^+$ *depicted in Figure 4.2, can start two* a*-actions.*

This problem can be solved by the requirement demanding that actions which can be potentially refined must not appear in the synchronization set. A solution where no restriction on the synchronization set is necessary is obtained if one considers start-based together with end-based disabling in a single setting, which is done in Chapter 7.

4.2.5 Comparison of Equivalences

First, the connection between ICT-equivalence, UI-equivalence and FUI-equivalence is presented.

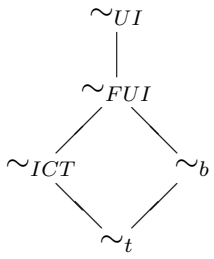


Figure 4.10: Relations Between the Equivalences

Proposition 4.20 *If two cbes are UI-equivalent, then they are also FUI-equivalent and if two cbes are FUI-equivalent, then they are also ICT-equivalent, i.e.*

$$\sim_{UI} \subset \sim_{FUI} \subset \sim_{ICT} .$$

Proof: The proof is given in Subsection 4.4.2. □

From Proposition 4.17 and Lemma 4.11, we obtain that the second inclusion in Proposition 4.20 is strict. The strictness of the first inclusion follows from Remark 4.16. From Lemma 4.11 and from the fact that a and $a + a$ are bisimilar but not ICT-equivalent, we obtain that ICT-equivalence is not comparable with strong bisimilarity. Furthermore, all equivalence notions from Section 4.2.1 can not be contained in $\sim_{ICT}$. Consequently, they can not be contained in $\sim_{UI}$ or in $\sim_{FUI}$, since they identify a and $a + a$. All connection between the equivalences that have been introduced is summarized in Figure 4.10: If two equivalences are connected via a line, then the lower one identifies more elements than the upper one.

4.2.6 Coarsest Congruence

In this subsection, we define the coarsest congruence for the refinement operator Ref^e with respect to strong bisimilarity. It is different from FUI-equivalence, i.e. FUI-equivalence is not the coarsest congruence with respect to strong bisimilarity. Furthermore, ICT-equivalence fails to be the coarsest congruence with respect to trace equivalence.

Definition 4.21 *Define* $\sim_c \subseteq \mathbf{CBES} \times \mathbf{CBES}$ *by* $\mathcal{E}_1 \sim_c \mathcal{E}_2$ *if and only if* $\forall A \subseteq \mathsf{Obs} : \forall \theta : A \to \mathbf{CBES} : Ref^e_A(\mathcal{E}_1, \theta) \sim_b Ref^e_A(\mathcal{E}_2, \theta)$.

Proposition 4.22 *The relation* $\sim_c$ *is the coarsest congruence for the refinement operator* Ref^e *with respect to* $\sim_b$.

Proof: The proof is given in Subsection 4.4.3. □

Unfortunately, FUI-equivalence is a proper subset of $\sim_c$, i.e. FUI-equivalence is not the coarsest congruence for Ref^e. This is illustrated by the following example.

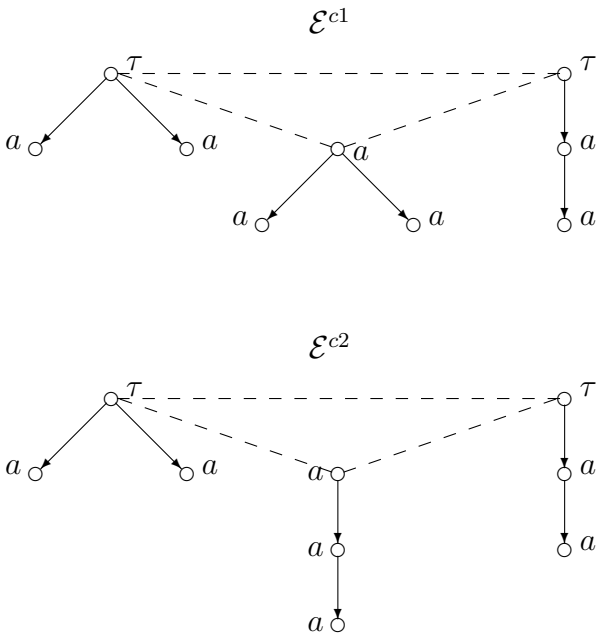


Figure 4.11: Counterexample of Coarsest Congruence

Example 4.23 *Consider $\mathcal{E}^{c1}$ and $\mathcal{E}^{c2}$ from Figure 4.11. They are not ICT-equivalent. Therefore, they are not FUI-equivalent either, since after executing action a, the number of the initial actions does not coincide. But $\mathcal{E}^{c1} \sim_c \mathcal{E}^{c2}$, which can be seen as follows. If a is not refined, then both cbes, which are bisimilar, keep unchanged under the refinement. Now suppose that a is refined. Then we do not have a problem to find a corresponding bisimilar process for every execution step as long as the refinement of a does not terminate. When the refinement terminates, the process executes τ. If '$\mathcal{E}^{c1}$' executes this τ, then '$\mathcal{E}^{c2}$' can execute its τ shown on the left to yield a bisimilar cbes. If '$\mathcal{E}^{c2}$' executes this τ, then '$\mathcal{E}^{c1}$' can execute its τ shown on the right to yield a bisimilar cbes.*

In addition, ICT-equivalence is not the coarsest congruence for Ref^e with respect to $\sim_t$, since $\sim_c$ is a congruence for Ref^e and $\sim_c \subseteq \sim_b \subseteq \sim_t$ but $\sim_c \not\subseteq \sim_{ICT}$ by Example 4.23.

4.3 Discussion

The reason why $\sim_{ICT}$ and $\sim_{FUI}$ fail to be the coarsest congruences for Ref^e stems from the fact that Ref^e renames events labeled by $\surd$ to τ. This renaming is necessary for the well-definedness of this operator, since a definition that removes these events will not result in an element of **CBES** or will not respect the intuitive meaning.

Nevertheless, it seems reasonable to have a refinement operator without such a relabeling. This corresponds to the philosophy that the 'final' executed action terminates the process [14, 28]. This kind of action refinement operator can only be defined in event structures where it is possible that sets of events rather than single events can disable other events. Therefore, we first

establish such a kind of event structures in Chapter 5 before we examine process algebras with an end-based choice operator. In Chapter 5, the new event structures are considered only in the context of disruption. The action refinement operator with the end-based choice view is applied on the new event structures in Chapter 6.

4.4 Proofs

4.4.1 Proof of the Congruence Results

We introduce an event-based refinement, which is used to verify Theorem 4.10. This refinement differs from Ref^e, because it assigns event structures to each event and not only to action names.

Definition 4.24 $\underline{Ref}^e_A : \mathbf{CBES} \times (\mathcal{U} \to \mathbf{CBES}) \to \mathbf{CBES}$ *with* $\underline{Ref}^e_A(\mathcal{E}, \vartheta) = (\tilde{E}, \tilde{\leadsto}, \tilde{\mapsto}, \tilde{l})$ *where*

$$\begin{array}{rcl}
\tilde{E} & = & \{(e,\hat{e}) \mid e \in E \wedge l(e) \in A \wedge \hat{e} \in E_{\vartheta(e)}\} \cup \\
 & & \{(e,e) \in E \times E \mid l(e) \notin A\} \\
\tilde{\leadsto} & = & \{((e_1,\hat{e}_1),(e_2,\hat{e}_2)) \mid (e_1 \leadsto e_2 \wedge (l(e_2) \in A \Rightarrow \hat{e}_2 \in \mathrm{exit}(\vartheta(e_2)))) \vee \\
 & & (e_1 = e_2 \wedge l(e_1) \in A \wedge \hat{e}_1 \neq \hat{e}_2 \wedge (\hat{e}_1 \leadsto_{\vartheta(e_1)} \hat{e}_2 \vee \hat{e}_2 \in \mathrm{exit}(\vartheta(e_1))))\} \\
\tilde{\mapsto} & = & \{(\{e\} \times X', (e,\hat{e})) \mid l(e) \in A \wedge X' \mapsto_{\vartheta(e)} \hat{e}\} \cup \\
 & & \{(\tilde{X}, (e,\hat{e})) \mid \exists X : X \mapsto e \wedge (l(e) \in A \Rightarrow \hat{e} \in \mathrm{init}(\vartheta(e))) \wedge \\
 & & \tilde{X} = \{(e,\hat{e}') \in \tilde{E} \mid e \in X \wedge (l(e) \in A \Rightarrow \hat{e}' \in \mathrm{exit}(\vartheta(e)))\}\}
\end{array}$$

$$\tilde{l}((e,\hat{e})) = \begin{cases} l(e) & \text{if } l(e) \notin A \\ l_{\vartheta(e)}(\hat{e}) & \text{if } l(e) \in A \wedge l_{\vartheta(e)}(\hat{e}) \neq \surd \\ \tau & \text{if } l(e) \in A \wedge l_{\vartheta(e)}(\hat{e}) = \surd \end{cases}$$

The advantage of $\underline{Ref}^e_A$ is that event execution of $\underline{Ref}^e_A(\mathcal{E}, \vartheta)$ can be reduced to the event execution of $\mathcal{E}$ and ϑ, as it is shown in the following lemma.

Lemma 4.25 *Suppose* $\mathcal{E} \in \mathbf{CBES}$ *and* $\vartheta : \mathcal{U} \to \mathbf{CBES}$. *Then*

$$\underline{Ref}^e_A(\mathcal{E}, \vartheta)_{[(e,\hat{e})]} \simeq \begin{cases} \underline{Ref}^e_A(\mathcal{E}_{[e]}, \vartheta) & \text{if } l(e) \notin A \wedge e = \hat{e} \\ \underline{Ref}^e_A(\mathcal{E}_{[e]}, \vartheta[e \to \vartheta(e)_{[\hat{e}]}]) & \text{if } l_{\vartheta(e)}(\hat{e}) = \surd \wedge l(e) \in A \\ \underline{Ref}^e_A(\mathcal{E}, \vartheta[e \to \vartheta(e)_{[\hat{e}]}])\} & \text{if } l_{\vartheta(e)}(\hat{e}) \neq \surd \wedge l(e) \in A \end{cases}$$

Furthermore, $\underline{Ref}^e_A(\mathcal{E}_{[e]}, \vartheta[e \to \mathcal{E}']) \simeq \underline{Ref}^e_A(\mathcal{E}_{[e]}, \vartheta)$ *holds for any* $\mathcal{E}' \in \mathbf{CBES}$.

Proof: Straightforward. □

In order to simplify the proof of Theorem 4.10 we introduce a variant of the initial corresponding traces:

Definition 4.26 *Let* $\mathcal{E}, \mathcal{E}' \in \mathbf{CBES}$. *Then define* $\tilde{T}^{ic}(\mathcal{E}) = \{((e_i,\gamma_i)_{i \leq n-1}, \gamma_n) \mid n \in \mathbb{N} \wedge \exists \mathcal{E}_0, \cdots, \mathcal{E}_n : \mathcal{E}_0 = \mathcal{E} \wedge \forall i \leq n-1 : \mathcal{E}_{i[e_i]} = \mathcal{E}_{i+1} \wedge \forall j \leq n : \gamma_j \in \mathcal{P}_{fin}(\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_j))\}$. *Furthermore, define* $\mathcal{E} \approx_{ICT} \mathcal{E}'$ *if*

- *for every* $((e_i,\gamma_i)_{i\leq n-1},\gamma_n)\in\tilde{T}^{ic}(\mathcal{E})$ *there is an injective, labeling-preserving function* $f:(\gamma_n\cup\bigcup_{i\leq n-1}(\gamma_i\cup\{e_i\}))\rightarrow E'$ *such that* $((f(e_i),f(\gamma_i))_{i\leq n-1},f(\gamma_n))\in\tilde{T}^{ic}(\mathcal{E}')$
- *and symmetrically as in Definition 4.8.*

Lemma 4.27 *The* initial corresponding trace equivalence *coincides with the equivalence defined in Definition 4.26, i.e.* $\sim_{ICT}=\approx_{ICT}$.

Proof: Suppose $(e_i,\gamma_i)_{i\leq n}\in T^{ic}(\mathcal{E})$ and $\mathcal{E}\approx_{ICT}\mathcal{E}'$. Then $((e_i,\gamma_i)_{i\leq n},\emptyset)\in\tilde{T}^{ic}(\mathcal{E})$, and so there is $f:(\bigcup_{i\leq n}(\gamma_i\cup\{e_i\}))\rightarrow E'$ such that $((f(e_i),f(\gamma_i))_{i\leq n},\emptyset)\in\tilde{T}^{ic}(\mathcal{E}')$. This implies $(f(e_i),f(\gamma_i))_{i\leq n}\in T^{ic}(\mathcal{E}')$, as required.

Now suppose $((e_i,\gamma_i)_{i\leq n-1},\gamma_n)\in\tilde{T}^{ic}(\mathcal{E})$ and $\mathcal{E}\sim_{ICT}\mathcal{E}'$. We proceed by making a case analysis:

$\gamma_n=\emptyset$: Similar to the above reasoning.

$\gamma_n\neq\emptyset$: Let $e_n\in\gamma_n$. Then $(e_i,\gamma_i)_{i\leq n}\in T^{ic}(\mathcal{E})$ and so there is a function $f:(\bigcup_{i\leq n}(\gamma_i\cup\{e_i\}))\rightarrow E'$ such that $(f(e_i),f(\gamma_i))_{i\leq n}\in T^{ic}(\mathcal{E}')$. This implies $((f(e_i),f(\gamma_i))_{i\leq n-1},\gamma_n)\in\tilde{T}^{ic}(\mathcal{E}')$ which completes the proof, since $\bigcup_{i\leq n}(\gamma_i\cup\{e_i\})=\gamma_n\cup\bigcup_{i\leq n-1}(\gamma_i\cup\{e_i\})$. □

Proof of Theorem 4.10: Here, we only present the proof for the refinement operator Ref^e, since the other cases are easier.

Let ϑ be a function such that $\forall e\in E:l(e)\in A\Rightarrow\vartheta(e)=\theta(l(e))$, and let ϑ' be a function such that $\forall e'\in E':l'(e')\in A\Rightarrow\vartheta'(e')=\theta'(l'(e'))$. Obviously, $\underline{Ref}^e_A(\mathcal{E},\vartheta)=Ref^e_A(\mathcal{E},\theta)$ and $\underline{Ref}^e_A(\mathcal{E}',\vartheta')=Ref^e_A(\mathcal{E}',\theta')$. For simplicity, we write $\tilde{\mathcal{E}}$ for $Ref^e_A(\mathcal{E},\theta)$ and $\tilde{\mathcal{E}}'$ for $Ref^e_A(\mathcal{E}',\theta')$ respectively.

Suppose $((e_i,\hat{e}_i),\tilde{\gamma}_i)_{i\leq n}\in T^{ic}(\tilde{\mathcal{E}})$. The case in which an element of $T^{ic}(\tilde{\mathcal{E}}')$ is taken follows by symmetrical arguments. As an immediate consequence of Lemma 4.25 we know that there exist $\mathcal{E}_0,...,\mathcal{E}_{n+1}$ and $\vartheta_0,...,\vartheta_{n+1}$ such that for $i\leq n$ we have

$$\begin{array}{l}\mathcal{E}_0=\mathcal{E}\wedge\vartheta_0=\vartheta\wedge e_i\in\mathrm{init}(\mathcal{E}_i)\wedge(l_i(e_i)\in A\Rightarrow\hat{e}_i\in\mathrm{init}(\vartheta_i(e_i)))\wedge\\(l_i(e_i)\notin A\Rightarrow(\mathcal{E}_{i[e_i]}=\mathcal{E}_{i+1}\wedge\vartheta_i=\vartheta_{i+1}))\wedge\\((l_i(e_i)\in A\wedge l_{\vartheta_i(e_i)}(\hat{e}_i)=\surd)\Rightarrow(\mathcal{E}_{i[e_i]}=\mathcal{E}_{i+1}\wedge\vartheta_i[e_i\rightarrow\vartheta_i(e_i)_{[\hat{e}_i]}]=\vartheta_{i+1}))\wedge\\((l_i(e_i)\in A\wedge l_{\vartheta_i(e_i)}(\hat{e}_i)\neq\surd)\Rightarrow(\mathcal{E}_i=\mathcal{E}_{i+1}\wedge\vartheta_i[e_i\rightarrow\vartheta_i(e_i)_{[\hat{e}_i]}]=\vartheta_{i+1})).\end{array}$$

Define $I=\{i\in\{0,\cdots,n\}\mid l(e_i)\notin A\vee l_{\vartheta_i(e_i)}(\hat{e}_i)=\surd\}$. Assume $\{k_0,...,k_{|I|-1}\}=I$ and $k_i<k_{i+1}$. Then we have $\mathcal{E}_{k_0}=\mathcal{E}$ and $\mathcal{E}_{k_j[e_{k_j}]}=\mathcal{E}_{k_{j+1}}$. Furthermore, define $\gamma_{k_j}=\left(\bigcup_{i>k_{j-1}}^{\min\{k_j,n\}}(\pi_1(\tilde{\gamma}_i)\cup\{e_i\})\right)\cap\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{k_j})$ for $j\leq|I|$ where $k_{-1}=-1$ and $k_{|I|}=n+1$. Hence, $((e_{k_i},\gamma_{k_i})_{j<|I|},\gamma_{n+1})\in\tilde{T}^{ic}(\mathcal{E})$. From $\mathcal{E}\sim_{ICT}\mathcal{E}'$ and Lemma 4.27 we know that there exists an injective, labeling preserving $f:(\gamma_{n+1}\cup\bigcup_{j<|I|}(\gamma_{k_j}\cup\{e_{k_j}\}))\rightarrow E'$ such that $((f(e_{k_i}),f(\gamma_{k_i}))_{j<|I|},f(\gamma_{n+1}))\in\tilde{T}^{ic}(\mathcal{E}')$.

Now define $I^e=\{j\in\{0,\cdots,n\}\mid e=e_j\}$ for $e\in E$ with $l(e)\in A$. Let $\{k^e_0,...,k^e_{|I^e|-1}\}=I^e$ with $k^e_i<k^e_{i+1}$. Define $\gamma^e_{k^e_j}=\bigcup_{i>k^e_{j-1}}^{\min\{k^e_j,n\}}\{\hat{e}\mid(e,\hat{e})\in\tilde{\gamma}_i\wedge l_{\vartheta_i(e)}(\hat{e})\in\mathsf{Obs}\}$ for $j\leq|I^e|$, where

$k^e_{-1} = -1$ and $k^e_{|I|} = n+1$. Furthermore, we have $\vartheta_{k^e_0}(e) = \theta(l(e))$ and $\vartheta_{k^e_j}(e)_{[\hat{e}_{k^e_j}]} = \vartheta_{k^e_{j+1}}(e)$. Hence, $((\hat{e}_{k^e_j}, \gamma^e_{k^e_j})_{j<|I^e|}, \gamma^e_{n+1}) \in \tilde{T}^{ic}(\theta(l(e)))$. From $\theta(l(e)) \sim_{ICT} \theta'(l(e))$ and Lemma 4.27 we know that there exists an injective, labeling preserving $f^e : (\gamma^e_{n+1} \cup \bigcup_{j<|I^e|}(\gamma^e_{k_j} \cup \{\hat{e}_{k^e_j}\})) \to E_{\theta'(l(e))}$ such that $((f^e(\hat{e}_{k^e_j}), f^e(\gamma^e_{k^e_j}))_{j<|I^e|}, f^e(\gamma^e_{n+1})) \in \tilde{T}^{ic}(\theta'(l(e)))$.

Define $\tilde{f} : (\bigcup_{i\leq n}(\tilde{\gamma}_i \cup \{(e_i, \hat{e}_i)\})) \to \tilde{E}'$ as follows:

$$\tilde{f}(e,\hat{e}) = \begin{cases} (f(e), f(e)) & \text{if } l(e) \notin A \\ (f(e), f^e(\hat{e})) & \text{otherwise} \end{cases} .$$

Then it is easily seen that $\tilde{f}$ is an injective and labeling-preserving function. Furthermore, define $\mathcal{E}'_0 = \mathcal{E}'$, $\vartheta'_0 = \vartheta'$ and

$$\begin{aligned} \mathcal{E}'_{i+1} &= \begin{cases} \mathcal{E}'_{i[f(e_i)]} & \text{if } l'(f(e_i)) \notin A \vee l_{\vartheta'(f(e_i))}(f^{e_i}(\hat{e}_i)) = \surd \\ \mathcal{E}'_i & \text{otherwise} \end{cases} \\ \vartheta'_{i+1} &= \begin{cases} \vartheta'_i & \text{if } l'(f(e_i)) \notin A \\ \vartheta'_i[f(e_i) \to \vartheta'_i(f(e_i))_{[f^{e_i}(\hat{e}_i)]}] & \text{otherwise} \end{cases} . \end{aligned}$$

$\mathcal{E}'_j$ and ϑ'_j are well defined, because $((f^e(\hat{e}_{k^e_j}), f^e(\gamma^e_{k^e_j}))_{j<|I^e|}, f^e(\gamma^e_{n+1})) \in \tilde{T}^{ic}(\theta'(l(e)))$ and $((f(e_{k_i}), f(\gamma_{k_i}))_{j<|I|}, f(\gamma_{n+1})) \in \tilde{T}^{ic}(\mathcal{E}')$. More precisely, for any j and any $e \in E$ with $l(e) \in A$, we have $\mathcal{E}'_{k_{j+1}} = \mathcal{E}'_{k_j[f(e_{k_j})]}$ and $\vartheta'_{k^e_{j+1}}(f(e)) = \vartheta'_{k^e_j}(f(e))_{[f^e(\hat{e}_{k^e_j})]}$.

Furthermore, define $\tilde{\mathcal{E}}'_i = Ref^e_A(\mathcal{E}'_i, \theta'_i)$ for $i \leq n+1$. Then $\tilde{\mathcal{E}}'_0 = \tilde{\mathcal{E}}'$ and $\tilde{\mathcal{E}}_{i+1} = \tilde{\mathcal{E}}'_{i[\tilde{f}(e_i,\hat{e}_i)]}$, which follows by Lemma 4.25.

Suppose $(e,\hat{e}) \in \tilde{\gamma}_i$. Let $m \in \{0, ..., |I|\}$ such that $k_{m-1} < i \leq k_m$. Hence, $e \in \gamma_{k_m}$, which implies $f(e) \in f(\gamma_{k_m}) \subseteq \text{init}(\mathcal{E}'_{k_m})$. And so from the definition of $\mathcal{E}'_j$ we get $f(e) \in \text{init}(\mathcal{E}'_i)$, since $k_{m-1} < i \leq k_m$. Furthermore, if $l(e) \in A$, then take $\hat{m} \leq |I^e|$ such that $k^e_{\hat{m}-1} < i \leq k^e_{\hat{m}}$. Hence, $\hat{e} \in \gamma^e_{k_{\hat{m}}}$, which implies $f^e(\hat{e}) \in f^e(\gamma^e_{k_{\hat{m}}}) \subseteq \text{init}(\vartheta'_{k^e_{\hat{m}}}(f(e)))$. And so from the definition of ϑ'_j we get $f^e(\hat{e}) \in \text{init}(\vartheta'_i(f(e)))$. Therefore, $\tilde{f}(e,\hat{e}) \in \text{init}(\tilde{\mathcal{E}}'_i)$.

Thus, we have shown $(\tilde{f}(e_i,\hat{e}_i), \tilde{f}(\tilde{\gamma}_i))_{i\leq n} \in T^{ic}(\tilde{\mathcal{E}}')$, which completes this proof. □

Proof of Theorem 4.14: We only present here the proof for the refinement operator Ref^e, since the other cases are easier.

Let $\mathcal{R}$ and $\mathcal{R}_a$ be unique initial bisimulations such that $(\theta(a), \theta'(a), g_a) \in \mathcal{R}_a$ and $(\mathcal{E}, \mathcal{E}', g) \in \mathcal{R}$. Then define

$$\begin{aligned} \mathcal{R}_{Ref} = \{ \; & (\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta}), \underline{Ref}^e_A(\tilde{\mathcal{E}}', \tilde{\vartheta}'), \tilde{f}) \mid \exists f : (\tilde{\mathcal{E}}, \tilde{\mathcal{E}}', f) \in \mathcal{R} \; \wedge \\ & (\forall \tilde{e} \in \tilde{E} : (\tilde{l}(\tilde{e}) \in A \wedge \tilde{e} \notin \text{init}(\tilde{\mathcal{E}})) \Rightarrow \tilde{\vartheta}(\tilde{e}) = \theta(\tilde{l}(\tilde{e}))) \; \wedge \\ & (\forall \tilde{e}' \in \tilde{E}' : (\tilde{l}'(\tilde{e}') \in A \wedge \tilde{e}' \notin \text{init}(\tilde{\mathcal{E}}')) \Rightarrow \tilde{\vartheta}'(\tilde{e}') = \theta'(\tilde{l}'(\tilde{e}'))) \; \wedge \\ & \forall e \in \tilde{E} : \exists f_e : (e \in \text{init}_A(\tilde{\mathcal{E}}) \Rightarrow (\tilde{\vartheta}(e), \tilde{\vartheta}'(f(e)), f_e) \in \mathcal{R}_{\tilde{l}(e)}) \; \wedge \\ & \tilde{f}(e,\hat{e}) \simeq \begin{cases} (f(e), f(e)) & \text{if } e \in \text{init}(\tilde{\mathcal{E}}) \wedge \tilde{l}(e) \in \mathsf{Obs}\backslash A \\ (f(e), f_e(\hat{e})) & \text{if } e \in \text{init}_A(\tilde{\mathcal{E}}) \wedge \hat{e} \in \text{init}_{\mathsf{Obs}}(\tilde{\vartheta}(e)) \end{cases} \} \end{aligned}$$

Suppose $(\underline{Ref}^e_A(\tilde{\mathcal{E}},\tilde{\vartheta}), \underline{Ref}^e_A(\tilde{\mathcal{E}}',\tilde{\vartheta}'), \tilde{f}) \in \mathcal{R}_{Ref}$ and let f,f_e be the corresponding functions.

Obviously, $\tilde{f}$ is always an isomorphism between $\mathrm{init}_{\mathsf{Obs}}(\underline{Ref}^e_A(\tilde{\mathcal{E}},\tilde{\vartheta}))$ and $\mathrm{init}_{\mathsf{Obs}}(\underline{Ref}^e_A(\tilde{\mathcal{E}}',\tilde{\vartheta}'))$ in the above definition. Furthermore, $\tilde{f}$ is labeling-preserving.

In the following, we will show that $\mathcal{R}_{Ref}$ is a UI-bisimulation. Therefore, suppose $(e,\hat{e}) \in \mathrm{init}(\underline{Ref}^e_A(\tilde{\mathcal{E}},\tilde{\vartheta}))$. We proceed by making a case analysis.

$\tilde{l}(e) \notin A \vee l_{\tilde{\vartheta}(e)}(\hat{e}) = \surd$: From Lemma 4.25 we get $\underline{Ref}^e_A(\tilde{\mathcal{E}},\tilde{\vartheta})_{[(e,\hat{e})]} = \underline{Ref}^e_A(\tilde{\mathcal{E}}_{[e]},\tilde{\vartheta})$. Since $(\tilde{\mathcal{E}},\tilde{\mathcal{E}}',f) \in \mathcal{R}$, there is $\dot{e}$ and f' such that $\tilde{l}(e) = \tilde{l}'(\dot{e})$ and $(\tilde{\mathcal{E}}_{[e]},\tilde{\mathcal{E}}'_{[\dot{e}]},f') \in \mathcal{R}$ and $l(e) \in \mathsf{Obs} \Rightarrow f(e) = \dot{e}$ and $f \cup f'$ is an injective function.
Let $f'_{e'} = \begin{cases} g_a & \text{if } \tilde{l}(e) = a \wedge e' \notin \mathrm{init}(\tilde{\mathcal{E}}) \\ f_{e'} & \text{otherwise} \end{cases}$. Define $\tilde{f}'$ as follows

$$\tilde{f}'(e',\hat{e}') \simeq \begin{cases} (f'(e'),f'(e')) & \text{if } e' \in \mathrm{init}(\tilde{\mathcal{E}}_{[e]}) \wedge \tilde{l}(e') \in \mathsf{Obs}\backslash A \\ (f'(e'),f'_{e'}(\hat{e}')) & \text{if } e' \in \mathrm{init}_A(\tilde{\mathcal{E}}_{[e]}) \wedge \hat{e}' \in \mathrm{init}_{\mathsf{Obs}}(\tilde{\vartheta}(e')) \end{cases}$$

Then $\tilde{f}'$ and $\tilde{f}$ coincide on $\mathrm{init}_{\mathsf{Obs}}(\underline{Ref}^e_A(\tilde{\mathcal{E}},\tilde{\vartheta})) \cap \mathrm{init}_{\mathsf{Obs}}(\underline{Ref}^e_A(\tilde{\mathcal{E}},\tilde{\vartheta})_{[(e,\hat{e})]})$, since we have $f \restriction (\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_1) \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{1[e_1]})) = f' \restriction (\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_1) \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{1[e_1]}))$. Thus, $\tilde{f}' \cup \tilde{f}$ is a function. Suppose $(\tilde{f}' \cup \tilde{f})(e',\hat{e}') = (\tilde{f}' \cup \tilde{f})(e'',\hat{e}'')$. Then $(f' \cup f)(e') = (f' \cup f)(e'')$ and therefore $e' = e''$. The equality of $\hat{e}'$ and $\hat{e}''$ can now be easily derived from the injectivity of $\tilde{f}'$ or of $\tilde{f}$. Thus, $\tilde{f}' \cup \tilde{f}$ is an injective function.

Suppose $\tilde{l}(e') \in A$, then $e' \notin \mathrm{init}(\tilde{\mathcal{E}}) \Rightarrow \tilde{\vartheta}(e') = \theta(\tilde{l}(e'))$. Additionally, we conclude from $e' \notin \mathrm{init}(\tilde{\mathcal{E}})$ that $f'(e') \notin \mathrm{init}(\tilde{\mathcal{E}}')$, since $f \cup f'$ is injective. And so we get $e' \notin \mathrm{init}(\tilde{\mathcal{E}}) \Rightarrow \tilde{\vartheta}'(f'(e')) = \theta'(\tilde{l}'(f'(e')))$. Thus, $e' \in \mathrm{init}(\tilde{\mathcal{E}}_{[e]}) \wedge \tilde{l}(e') \in A \Rightarrow (\tilde{\vartheta}(e'),\tilde{\vartheta}'(f'(e')),f'_{e'}) \in \mathcal{R}_{\tilde{l}(e)}$. Hence, $(\underline{Ref}^e_A(\tilde{\mathcal{E}}_{[e]},\tilde{\vartheta}), \underline{Ref}^e_A(\tilde{\mathcal{E}}'_{[\dot{e}]},\tilde{\vartheta}'),\tilde{f}') \in \mathcal{R}_{Ref}$ by definition.

Moreover, $\tilde{l}(e) \notin A$ implies $\tilde{l}'(\dot{e}) \notin A$. From Lemma 4.25 we obtain that $\underline{Ref}^e_A(\tilde{\mathcal{E}}'_{[\dot{e}]},\tilde{\vartheta}') = \underline{Ref}^e_A(\tilde{\mathcal{E}}',\tilde{\vartheta}')_{[(\dot{e},\dot{e})]}$. Furthermore, $l_{\underline{Ref}^e_A(\tilde{\mathcal{E}},\tilde{\vartheta})}(e,\hat{e}) \in \mathsf{Obs} \Rightarrow \tilde{f}(e,\hat{e}) = (f(e),f(e)) = (\dot{e},\dot{e})$.

Now consider the case when $\tilde{l}(e) \in A \wedge l_{\tilde{\vartheta}(e)}(\hat{e}) = \surd \wedge \hat{e} \in \mathrm{init}(\tilde{\vartheta}(e))$. Then we have $(\tilde{\vartheta}(e),\tilde{\vartheta}'(f(e)),f_e) \in \mathcal{R}_{\tilde{l}(e)}$. Hence, there exists $\ddot{e} \in \mathrm{init}(\tilde{\vartheta}'(f(e)))$ such that $l_{\tilde{\vartheta}'(f(e))}(\ddot{e}) = \surd$. Thus $\underline{Ref}^e_A(\tilde{\mathcal{E}}'_{[\dot{e}]},\tilde{\vartheta}') = \underline{Ref}^e_A(\tilde{\mathcal{E}}'_{[f(e)]},\tilde{\vartheta}'[f(e) \to \tilde{\vartheta}'(f(e))_{[\ddot{e}]}) = \underline{Ref}^e_A(\tilde{\mathcal{E}}',\tilde{\vartheta}')_{[(f(e),\ddot{e})]}$ by Lemma 4.25. This completes the case, since $l_{\underline{Ref}^e_A(\tilde{\mathcal{E}},\tilde{\vartheta})}(e,\hat{e}) \notin \mathsf{Obs}$.

$\tilde{l}(e) \in A \wedge l_{\tilde{\vartheta}(e)}(\hat{e}) \neq \surd$: By Lemma 4.25 we get $\underline{Ref}^e_A(\tilde{\mathcal{E}},\tilde{\vartheta})_{[(e,\hat{e})]} = \underline{Ref}^e_A(\tilde{\mathcal{E}},\tilde{\vartheta}[e \to \tilde{\vartheta}(e)_{[\hat{e}]}])$. Furthermore, $(\tilde{\vartheta}(e),\tilde{\vartheta}'(f(e)),f_e) \in \mathcal{R}_{\tilde{l}(e)}$. And so there is $\ddot{e}$ and f'_e such that $l_{\tilde{\vartheta}(e)}(\hat{e}) = l_{\tilde{\vartheta}'(f(e))}(\ddot{e})$ and $(\tilde{\vartheta}(e)_{[\hat{e}]},\tilde{\vartheta}'(f(e))_{[\ddot{e}]},f'_e) \in \mathcal{R}_{\tilde{l}(e)}$ and $l_{\tilde{\vartheta}(e)}(\hat{e}) \in \mathsf{Obs} \Rightarrow \ddot{e} = f'_e(\hat{e})$ and $f_e \cup f'_e$ is an injective function. Define $\tilde{f}'$ by

$$\tilde{f}'(e',\hat{e}') \simeq \begin{cases} (f(e'),f(e')) & \text{if } e' \in \mathrm{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}) \wedge \tilde{l}(e') \notin A \\ (f(e'),f_{e'}(\hat{e}')) & \text{if } e' \in \mathrm{init}_A(\tilde{\mathcal{E}}) \wedge \hat{e}' \in \mathrm{init}_{\mathsf{Obs}}(\tilde{\vartheta}(e')) \wedge e' \neq e \\ (f(e),f'_e(\hat{e}')) & \text{if } \hat{e}' \in \mathrm{init}_{\mathsf{Obs}}(\tilde{\vartheta}(e)_{[\hat{e}]}) \wedge e' = e \end{cases}$$

Then $\tilde{f}'$ and $\tilde{f}$ coincide on $\mathrm{init}_{\mathsf{Obs}}(\underline{Ref}^e_A(\tilde{\mathcal{E}},\tilde{\vartheta})) \cap \mathrm{init}_{\mathsf{Obs}}(\underline{Ref}^e_A(\tilde{\mathcal{E}},\tilde{\vartheta})_{[(e,\hat{e})]})$. Thus, $\tilde{f}' \cup \tilde{f}$ is a function. Suppose $(\tilde{f}' \cup \tilde{f})(e',\hat{e}') = (\tilde{f}' \cup \tilde{f})(e'',\hat{e}'')$. Then $f(e') = f(e'')$ by definition,

hence $e' = e''$. If $\tilde{l}(e') \notin A$, then $\hat{e}' = \hat{e}''$ immediately follows. If $\tilde{l}(e') \in A \wedge e' \neq e$, then $\tilde{f}(e', \hat{e}') = \tilde{f}'(e', \hat{e}')$, and so $\hat{e}' = \hat{e}''$ follows by the injectivity of $\tilde{f}$. Now suppose $\tilde{l}(e') \in A \wedge e' = e$. Then $(f'_e \cup f_e)(\hat{e}') = (f'_e \cup f_e)(\hat{e}'')$, hence $\hat{e}' = \hat{e}''$. Therefore, $\tilde{f}' \cup \tilde{f}$ is an injective function.

Thus, $(\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta}[e \to \tilde{\vartheta}(e)_{[\ddot{e}]}]), \underline{Ref}^e_A(\tilde{\mathcal{E}}', \tilde{\vartheta}'[f(e) \to \tilde{\vartheta}(f(e))_{[\ddot{e}]}]), \tilde{f}') \in \mathcal{R}_{Ref}$ by definition. Moreover, $\tilde{l}(e) \in A \wedge l_{\tilde{\vartheta}(e)}(\hat{e}) \neq \surd \wedge \hat{e} \in \mathrm{init}(\tilde{\vartheta}(e))$ implies that $\tilde{l}'(f(e)) \in A \wedge l_{\tilde{\vartheta}'(f(e))}(\ddot{e}) \neq \surd$. And so by Lemma 4.25 we have $\underline{Ref}^e_A(\tilde{\mathcal{E}}', \tilde{\vartheta}'[f(e) \to \tilde{\vartheta}(f(e))_{[\ddot{e}]}]) = \underline{Ref}^e_A(\tilde{\mathcal{E}}', \tilde{\vartheta}')_{[(f(e),\ddot{e})]}$. This completes the case, since $l_{\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta})}(e, \hat{e}) \in \mathsf{Obs} \Rightarrow \tilde{f}(e, \hat{e}) = (f(e), \ddot{e})$.

The last condition of the UI-bisimulation can be derived by symmetrical arguments. Thus, we proved that $\mathcal{R}_{Ref}$ is a unique initial bisimulation.

Obviously, $\underline{Ref}^e_A(\mathcal{E}, \vartheta) = Ref^e_A(\mathcal{E}, \theta)$ whenever $\forall e \in E : l(e) \in A \Rightarrow \vartheta(e) = \theta(l(e))$. Define

$$\tilde{g}(e', \hat{e}') \simeq \begin{cases} (g(e'), g(e')) & \text{if } e' \in \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}) \wedge l(e') \notin A \\ (g(e'), g_{l(e')}(\hat{e}')) & \text{if } e' \in \mathrm{init}(\mathcal{E}) \wedge l(e') \in A \wedge \hat{e}' \in \mathrm{init}_{\mathsf{Obs}}(\theta(l(e'))) \end{cases}.$$

Then $(Ref^e_A(\mathcal{E}, \theta), Ref^e_A(\mathcal{E}', \theta'), \tilde{g}) \in \mathcal{R}_{Ref}$, which completes this proof. □

Proof of Theorem 4.18: We only present here the proof for the refinement operator Ref^e, the other cases are easier.

Let $\mathcal{R}$ and $\mathcal{R}_a$ be unique initial bisimulations such that $(\theta(a), \theta'(a), g_a) \in \mathcal{R}_a$ and $(\mathcal{E}, \mathcal{E}', g) \in \mathcal{R}$. Then define

$$\begin{aligned} \mathcal{R}_{Ref} \;=\; \{ & (\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta}), \underline{Ref}^e_A(\tilde{\mathcal{E}}', \tilde{\vartheta}'), f) \mid \exists \tilde{f} : \exists \tilde{I} \in \mathcal{P}_{fin}(\mathrm{init}_A(\tilde{\mathcal{E}})) : \exists F : \mathcal{U} \to (\mathcal{U} \rightharpoonup \mathcal{U}) : \\ & (\tilde{\mathcal{E}}, \tilde{\mathcal{E}}', \tilde{f}) \in \mathcal{R} \;\wedge \\ & (\forall \tilde{e} \in \tilde{E} \backslash \tilde{I} : (\tilde{l}(\tilde{e}) \in A \Rightarrow \tilde{\vartheta}(\tilde{e}) = \theta(\tilde{l}(\tilde{e})) \wedge (\tilde{e} \in \mathrm{init}(\tilde{\mathcal{E}}) \Rightarrow F(\tilde{e}) = g_{\tilde{l}(\tilde{e})})) \;\wedge \\ & (\forall \tilde{e}' \in \tilde{E}' \backslash \tilde{f}(\tilde{I}) : (\tilde{l}'(\tilde{e}') \in A \Rightarrow \tilde{\vartheta}'(\tilde{e}') = \theta'(\tilde{l}'(\tilde{e}'))) \;\wedge \\ & (\forall \tilde{e} \in \tilde{I} : (\tilde{\vartheta}(\tilde{e}), \tilde{\vartheta}'(\tilde{f}(\tilde{e})), F(\tilde{e})) \in \mathcal{R}_{\tilde{l}(\tilde{e})}) \;\wedge \\ & (\forall \tilde{e} \in \mathrm{init}(\tilde{\mathcal{E}}) : (\tilde{l}(\tilde{e}) \notin A \Rightarrow F(\tilde{e}) = \mathrm{cons}_{\tilde{f}(\tilde{e})}) \;\wedge \\ & f(e, \hat{e}) \simeq \begin{cases} (\tilde{f}(e), F(e)(\hat{e})) & \text{if } (e, \hat{e}) \in \mathrm{init}_{\mathsf{Obs}}(\underline{Ref}^e(\tilde{\mathcal{E}}, \tilde{\vartheta}) \\ \text{undefined} & \text{otherwise} \end{cases} \} \end{aligned}$$

First, we observe that for all $(\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta}), \underline{Ref}^e_A(\tilde{\mathcal{E}}', \tilde{\vartheta}'), f) \in \mathcal{R}_{Ref}$ we have: f is always an isomorphism between $\mathrm{init}_{\mathsf{Obs}}(\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta}))$ and $\mathrm{init}_{\mathsf{Obs}}(\underline{Ref}^e_A(\tilde{\mathcal{E}}', \tilde{\vartheta}'))$, f is labeling-preserving and

$$f^{-1}(e', \hat{e}') = (\tilde{f}^{-1}(e'), F(\tilde{f}^{-1}(e'))^{-1}(\hat{e}')) \tag{4.1}$$

where $\tilde{f}, F$ are its corresponding functions. This holds, since $f(\tilde{f}^{-1}(e'), F(\tilde{f}^{-1}(e'))^{-1}(\hat{e}')) = (e', \hat{e}')$.

Suppose $(\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta}), \underline{Ref}^e_A(\tilde{\mathcal{E}}', \tilde{\vartheta}'), f) \in \mathcal{R}_{Ref}$, let $\tilde{I}$ be the corresponding set and let $\tilde{f}$,F be the corresponding functions.

In the following, we will show that $\mathcal{R}_{Ref}$ is a FUI-bisimulation. Therefore, suppose $(e, \hat{e}) \in \mathrm{init}(\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta})) \wedge I \in \mathcal{P}_{fin}(\mathrm{init}_{\mathsf{Obs}}(\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta})))$. We proceed by making a case analysis.

$\tilde{l}(e) \notin A$: Then $\hat{e} = e$ and from Lemma 4.25 we get $\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta})_{[(e,\hat{e})]} = \underline{Ref}^e_A(\tilde{\mathcal{E}}_{[e]}, \tilde{\vartheta})$.

Define $\tilde{I}' = \tilde{I} \cup \pi_1(I)$. Since $(\tilde{\mathcal{E}}, \tilde{\mathcal{E}}', \tilde{f}) \in \mathcal{R} \wedge \tilde{I}' \in \mathcal{P}_{fin}(\text{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}))$, there is e' and $\tilde{f}'$ such that $\tilde{l}(e) = \tilde{l}'(e')$ and $(\tilde{\mathcal{E}}_{[e]}, \tilde{\mathcal{E}}'_{[e']}, \tilde{f}') \in \mathcal{R}$ and $\tilde{l}(e) \in \mathsf{Obs} \Rightarrow \tilde{f}(e) = e'$ and $\tilde{f} \upharpoonright (\tilde{I}' \cap \text{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}_{[e]})) = \tilde{f}' \upharpoonright \tilde{I}'$ and $\tilde{f}^{-1} \upharpoonright (f(\tilde{I}') \cap \text{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}'_{[e']})) = \tilde{f}'^{-1} \upharpoonright f(\tilde{I}')$.

We have $\tilde{l}(e) \notin A$ implies $\tilde{l}'(e') \notin A$. And so by Lemma 4.25 we obtain $\underline{Ref}^e_A(\tilde{\mathcal{E}}'_{[e']}, \tilde{\vartheta}') = \underline{Ref}^e_A(\tilde{\mathcal{E}}', \tilde{\vartheta}')_{[(e',e')]}$. Furthermore, the labels of (e', e') and (e, e) coincide and $l_{\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta})}(e, e) \in \mathsf{Obs} \Rightarrow f(e, e) = (\tilde{f}(e), \tilde{f}(e)) = (e', e')$.

Now we define F' and f' by

$$F'(e'') = \begin{cases} \text{cons}_{\tilde{f}'(\tilde{e})} & \text{if } l(e'') \notin A \wedge e'' \in \text{init}(\tilde{\mathcal{E}}) \\ F(e) & \text{otherwise} \end{cases} \tag{4.2}$$

$$f'(e'', \hat{e}'') = \begin{cases} (\tilde{f}'(e''), F'(e'')(\hat{e}'')) & \text{if } (e'', \hat{e}'') \in \text{init}_{\mathsf{Obs}}(\underline{Ref}^e(\tilde{\mathcal{E}}_{[e]}, \tilde{\vartheta}) \\ \text{undefined} & \text{otherwise} \end{cases} . \tag{4.3}$$

Then $(\underline{Ref}^e_A(\tilde{\mathcal{E}}_{[e]}, \tilde{\vartheta}), \underline{Ref}^e_A(\tilde{\mathcal{E}}'_{[e']}, \tilde{\vartheta}'), f') \in \mathcal{R}_{Ref}$ where $\tilde{I}' \cap \text{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}_{[e]})$ is its corresponding set and $\tilde{f}'$,F' are its corresponding functions.

Furthermore, from $\tilde{f} \upharpoonright (\tilde{I}' \cap \text{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}_{[e]})) = \tilde{f}' \upharpoonright \tilde{I}'$ we obtain that f' and f coincide on $I \cap \text{init}_{\mathsf{Obs}}(\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta})_{[(e,\hat{e})]})$.

Suppose $(e''', \hat{e}''') \in \text{init}_{\mathsf{Obs}}(\underline{Ref}^e_A(\tilde{\mathcal{E}}', \tilde{\vartheta}')_{[(e',e')]}) \cap f(I)$. Then from (4.1) and from the fact that $\tilde{f}^{-1} \upharpoonright (f(\tilde{I}') \cap \text{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}'_{[e']})) = \tilde{f}'^{-1} \upharpoonright f(\tilde{I}')$ we obtain the following equation $f'^{-1}(e''', \hat{e}''') = (\tilde{f}'^{-1}(e'''), F'(\tilde{f}'^{-1}(e'''))^{-1}(\hat{e}''')) = (\tilde{f}^{-1}(e'''), F(\tilde{f}^{-1}(e'''))^{-1}(\hat{e}''')) = f^{-1}(e''', \hat{e}''')$, which completes this case.

$\tilde{l}(e) \in A \wedge l_{\tilde{\vartheta}(e)}(\hat{e}) = \surd$: By Lemma 4.25 $\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta})_{[(e,\hat{e})]} = \underline{Ref}^e_A(\tilde{\mathcal{E}}_{[e]}, \tilde{\vartheta}[e \rightarrow \tilde{\vartheta}(e)_{[\hat{e}]}]) = \underline{Ref}^e_A(\tilde{\mathcal{E}}_{[e]}, \tilde{\vartheta})$.

Define $\tilde{I}' = \tilde{I} \cup \pi_1(I)$. Since $(\tilde{\mathcal{E}}, \tilde{\mathcal{E}}', \tilde{f}) \in \mathcal{R} \wedge \tilde{I}' \in \mathcal{P}_{fin}(\text{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}))$, there is e' and $\tilde{f}'$ such that $\tilde{l}(e) = \tilde{l}'(e')$ and $(\tilde{\mathcal{E}}_{[e]}, \tilde{\mathcal{E}}'_{[e']}, \tilde{f}') \in \mathcal{R}$ and $\tilde{l}(e) \in \mathsf{Obs} \Rightarrow \tilde{f}(e) = e'$ and $\tilde{f} \upharpoonright (\tilde{I}' \cap \text{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}_{[e]})) = \tilde{f}' \upharpoonright \tilde{I}'$ and $\tilde{f}^{-1} \upharpoonright (f(\tilde{I}') \cap \text{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}'_{[e']})) = \tilde{f}'^{-1} \upharpoonright f(\tilde{I}')$.

Furthermore, $(\tilde{\vartheta}(e), \tilde{\vartheta}'(e'), F(e)) \in \mathcal{R}_{\tilde{l}(e)}$ and $\hat{e} \in \text{init}(\tilde{\vartheta}(e))$. Hence, there exists $\hat{e}' \in \text{init}(\vartheta)$ such that $l_{\tilde{\vartheta}(e)}(\hat{e}) = l_{\tilde{\vartheta}'(e')}(\hat{e}')$ and $l_{\tilde{\vartheta}(e)}(\hat{e}) \in \mathsf{Obs} \Rightarrow F(e)(\hat{e}) = \hat{e}'$.

By Lemma 4.25 we get $\underline{Ref}^e_A(\tilde{\mathcal{E}}'_{[e']}, \tilde{\vartheta}'[e' \rightarrow \tilde{\vartheta}'(e')_{[\hat{e}']}]) = \underline{Ref}^e_A(\tilde{\mathcal{E}}', \tilde{\vartheta}')_{[(e',\hat{e}')]}$. Furthermore, the labels of $(e', \hat{e}')$ and $(e, \hat{e})$ coincide and $l_{\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta})}(e, \hat{e}) \in \mathsf{Obs}$ implies that $f(e, \hat{e}) = (\tilde{f}(e), F(e)(\hat{e})) = (e', \hat{e}')$.

Now define F' and f' as in (4.2) and (4.3). Then $(\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta})_{[(e,\hat{e})]}, \underline{Ref}^e_A(\tilde{\mathcal{E}}', \tilde{\vartheta}')_{[(e',\hat{e}')]}, f') = (\underline{Ref}^e_A(\tilde{\mathcal{E}}_{[e]}, \tilde{\vartheta}), \underline{Ref}^e_A(\tilde{\mathcal{E}}'_{[e']}, \tilde{\vartheta}'), f') \in \mathcal{R}_{Ref}$ where $\tilde{I}' \cap \text{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}_{[e]})$ is its corresponding set and $\tilde{f}'$,F' are its corresponding functions (please note that $e \notin E_{\tilde{\mathcal{E}}_{[e]}}$).

Furthermore, from $\tilde{f} \upharpoonright (\tilde{I}' \cap \text{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}_{[e]})) = \tilde{f}' \upharpoonright \tilde{I}'$ we obtain that f' and f coincide on $\text{init}_{\mathsf{Obs}}(\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta})_{[(e,\hat{e})]}) \cap I$.

Suppose $(e''', \hat{e}''') \in \text{init}_{\mathsf{Obs}}(\underline{Ref}^e_A(\tilde{\mathcal{E}}', \tilde{\vartheta}')_{[(e',\hat{e}')]}) \cap f(I)$. Then from (4.1) and from the fact that $\tilde{f}^{-1} \upharpoonright (f(\tilde{I}') \cap \text{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}'_{[e']})) = \tilde{f}'^{-1} \upharpoonright f(\tilde{I}')$ we obtain the following equation $f'^{-1}(e''', \hat{e}''') = (\tilde{f}'^{-1}(e'''), F'(\tilde{f}'^{-1}(e'''))^{-1}(e''')) = (\tilde{f}^{-1}(e'''), F(\tilde{f}^{-1}(e'''))^{-1}(e''')) = f^{-1}(e''', \hat{e}''')$, which completes this case.

$\tilde{l}(e) \in A \wedge l_{\tilde{\vartheta}(e)}(\hat{e}) \neq \surd$: By Lemma 4.25 we get $\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta})_{[(e,\hat{e})]} = \underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta}[e \to \tilde{\vartheta}(e)_{[\hat{e}]}])$.

Define $I^e = \{\hat{e}'' \mid (e, \hat{e}'') \in I\}$. Since $I^e \in \mathcal{P}_{fin}(\text{init}_{\mathsf{Obs}}(\tilde{\vartheta}(e)))$ and $(\tilde{\vartheta}(e), \tilde{\vartheta}'(f(e)), F(e)) \in \mathcal{R}_{\tilde{l}(e)}$ there exists $\hat{e}' \in \text{init}(\tilde{\vartheta}'(\tilde{f}(e)))$ and $\hat{f}$ such that $(\tilde{\vartheta}(e)_{[\hat{e}]}, \tilde{\vartheta}'(\tilde{f}(e))_{[\hat{e}']}, \hat{f}) \in \mathcal{R}_{\tilde{l}(e)}$ and $l_{\tilde{\vartheta}(e)}(\hat{e}) = l_{\tilde{\vartheta}'(\tilde{f}(e))}(\hat{e}')$ and $l_{\tilde{\vartheta}(e)}(\hat{e}) \in \mathsf{Obs} \Rightarrow F(e)(\hat{e}) = \hat{e}'$ and $F(e) \upharpoonright (I^e \cap \text{init}_{\mathsf{Obs}}(\tilde{\vartheta}(e)_{[\hat{e}]})) = \hat{f} \upharpoonright I^e$ and $F(e)^{-1} \upharpoonright (F(e)(I^e) \cap \text{init}_{\mathsf{Obs}}(\tilde{\vartheta}'(\tilde{f}(e))_{[\hat{e}']})) = \hat{f}^{-1} \upharpoonright F(e)(I^e)$.

From Lemma 4.25 we obtain $\underline{Ref}^e_A(\tilde{\mathcal{E}}', \tilde{\vartheta}'[\tilde{f}(e) \to \tilde{\vartheta}'(\tilde{f}(e))_{[\hat{e}']}]) = \underline{Ref}^e_A(\tilde{\mathcal{E}}', \tilde{\vartheta}')_{[(\tilde{f}(e),\hat{e}')]}$. Furthermore, the labels of $(\tilde{f}(e), \hat{e}')$ and $(e, \hat{e})$ coincide and $l_{\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta})}(e, \hat{e}) \in \mathsf{Obs}$ implies $f(e, \hat{e}) = (\tilde{f}(e), F(e)(\hat{e})) = (\tilde{f}(e), \hat{e}')$.

Now define f' by

$$f'(e'', \hat{e}'') = \begin{cases} (\tilde{f}(e), \hat{f}(\hat{e}'')) & \text{if } (e'', \hat{e}'') \in \text{init}_{\mathsf{Obs}}(\underline{Ref}^e(\tilde{\mathcal{E}}_{[e]}, \tilde{\vartheta}) \wedge e'' = e \\ (\tilde{f}(e''), F(e'')(\hat{e}'')) & \text{if } (e'', \hat{e}'') \in \text{init}_{\mathsf{Obs}}(\underline{Ref}^e(\tilde{\mathcal{E}}_{[e]}, \tilde{\vartheta}) \wedge e'' \neq e \\ \text{undefined} & \text{otherwise} \end{cases} .$$

Then $(\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta}[e \to \tilde{\vartheta}(e)_{[\hat{e}]}]), (\underline{Ref}^e_A(\tilde{\mathcal{E}}', \tilde{\vartheta}'[\tilde{f}(e) \to \tilde{\vartheta}'(\tilde{f}(e))_{[\hat{e}']}]), f') \in \mathcal{R}_{Ref}$ where $\tilde{I} \cup \{e\}$ is its corresponding set and $\tilde{f}$,$F' = F[e \to \hat{f}]$ are its corresponding functions.

Furthermore, from $F(e) \upharpoonright (I^e \cap \text{init}_{\mathsf{Obs}}(\tilde{\vartheta}(e)_{[\hat{e}]})) = \hat{f} \upharpoonright I^e$ we obtain that f' and f coincide on $\text{init}_{\mathsf{Obs}}(\underline{Ref}^e_A(\tilde{\mathcal{E}}, \tilde{\vartheta})_{[(e,\hat{e})]}) \cap I$.

Suppose $(e''', \hat{e}''') \in \text{init}_{\mathsf{Obs}}(\underline{Ref}^e_A(\tilde{\mathcal{E}}', \tilde{\vartheta}')_{[(e',\hat{e}')]}) \cap f(I)$. Then from (4.1) and from the fact that $F(e)^{-1} \upharpoonright (F(e)(I^e) \cap \text{init}_{\mathsf{Obs}}(\tilde{\vartheta}'(\tilde{f}(e))_{[\hat{e}']})) = \hat{f}^{-1} \upharpoonright F(e)(I^e)$ we obtain $f'^{-1}(e''', \hat{e}''') = (\tilde{f}^{-1}(e'''), F'(\tilde{f}^{-1}(e'''))^{-1}(e''')) = (\tilde{f}^{-1}(e'''), F(\tilde{f}^{-1}(e'''))^{-1}(e''')) = f^{-1}(e''', \hat{e}''')$.

The last condition of the FUI-bisimulation can be derived by symmetrical arguments. Thus, we have proved that $\mathcal{R}_{Ref}$ is a FUI-bisimulation.

Obviously, $\underline{Ref}^e_A(\mathcal{E}, \vartheta) = Ref^e_A(\mathcal{E}, \theta)$ whenever $\forall e \in E : l(e) \in A \Rightarrow \vartheta(e) = \theta(l(e))$. Define $F(e) = \begin{cases} \text{cons}_{g(e)} & \text{if } l(e) \in A \wedge e'' \in \text{init}(\tilde{\mathcal{E}}) \\ g_{l(e)} & \text{if } l(e) \notin A \wedge e'' \in \text{init}(\tilde{\mathcal{E}}) \\ \bot & \text{otherwise} \end{cases}$ and $f(e, \hat{e}) = \begin{cases} (g(e), F(e)(\hat{e})) & \text{if } (e, \hat{e}) \in \text{init}_{\mathsf{Obs}}(Ref^e(\mathcal{E}, \theta)) \\ \text{undefined} & \text{otherwise} \end{cases}$. Then it is easy to check that $(Ref^e(\mathcal{E}, \theta), Ref^e(\mathcal{E}', \theta'), f) \in \mathcal{R}_{Ref}$, where $\emptyset$ is its corresponding set and g, F are its corresponding functions. □

4.4.2 Proof of Proposition 4.20

Before we present the proof of Proposition 4.20, we establish the following lemma.

Lemma 4.28 *For all $\mathcal{E}', \mathcal{E}_0, ..., \mathcal{E}_n, e_0, ..., e_{n-1}, f$ and for all $I \in \mathcal{P}_{fin}(\mathcal{U})$ and for all FUI-bisimulation $\mathcal{R}$ such that $\forall i \leq n-1 : \mathcal{E}_{i+1} = \mathcal{E}_{i[e_i]}$ and $(\mathcal{E}_0, \mathcal{E}', f) \in \mathcal{R}$ there is a function $g : (I \cap \bigcup_{i\leq n} \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_i)) \cup \bigcup_{i\leq n-1}\{e_i\}) \to E'$ such that g is injective, labeling-preserving and $\forall i \leq n : \exists g_i : (\mathcal{E}_i, \mathcal{E}'_{[g(e_0)]\cdots[g(e_{i-1})]}, g_i) \in \mathcal{R} \wedge g \upharpoonright (\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_i) \cap I) = g_i \upharpoonright (\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_i) \cap I)$.*

Proof: We use induction on n.

$n = 0$: The result follows immediately if we choose $g = f \upharpoonright I$.

$n+1$: Define $\tilde{I} = (I \cap \bigcup_{i\leq n} \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_i)) \cup \bigcup_{i\leq n}\{e_i\}$. Then by induction there exists $\tilde{g} : ((\tilde{I} \cap \bigcup_{i\leq n} \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_i)) \cup \bigcup_{i\leq n-1}\{e_i\}) \to E'$ such that $\tilde{g}$ is injective, labeling-preserving and $\forall i \leq n : \exists g_i : \tilde{g} \upharpoonright (\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_i) \cap \tilde{I}) = g_i \upharpoonright (\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_i) \cap \tilde{I}) \wedge (\mathcal{E}_i, \mathcal{E}'_i, g_i) \in \mathcal{R}$, where $\mathcal{E}'_i = \mathcal{E}'_{[\tilde{g}(e_0)]\cdots[\tilde{g}(e_{i-1})]}$.

From $(\mathcal{E}_n, \mathcal{E}'_n, g_n) \in \mathcal{R}$ and $\tilde{I}' \in \mathcal{P}_{fin}(\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_n))$, where $\tilde{I}' = \tilde{I} \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_n)$, we obtain the existence of e' and g_{n+1} such that $l_0(e_n) = l'(e')$ and $(\mathcal{E}_{n+1}, \mathcal{E}'_{n[e']}, g_{n+1}) \in \mathcal{R}$ and $l_0(e_n) \in \mathsf{Obs} \Rightarrow e' = g_n(e_n)$ and $g_n \upharpoonright (\tilde{I}' \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{n+1})) = g_{n+1} \upharpoonright \tilde{I}'$ and $g_n^{-1} \upharpoonright (g_n(\tilde{I}') \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}'_{n+1})) = g_{n+1}^{-1} \upharpoonright g_n(\tilde{I}')$. Define

$$g(e) \simeq \begin{cases} \tilde{g}(e) & \text{if } e \in \tilde{I} \\ g_{n+1}(e) & \text{if } e \in (I \cap (\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{n+1}))) \backslash \tilde{I} \end{cases} .$$

It is easily seen that g is labeling-preserving and that it coincides with g_i on $\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_i) \cap I$ for any $i \leq n+1$.

The injectivity of g can be seen as follows: Suppose there are e and $\tilde{e}$ such that $g(e) = g(\tilde{e}) \wedge e \neq \tilde{e}$. This is only possible if $\tilde{e} \in \tilde{I}$ and $e \in (I \cap (\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{n+1}))) \backslash \tilde{I}$, since otherwise a contradiction to the injectivity of $\tilde{g}$ or g_{n+1} follows. From $e \in \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{n+1})$ we obtain $e \in \mathsf{Obs}$, hence $\tilde{e} \in \mathsf{Obs}$ by the labeling preserving of $\tilde{g}$ and g_{n+1}. Furthermore, there exists $j \leq n$ such that $\tilde{e} \in \mathrm{init}(\mathcal{E}_j)$, thus $\tilde{g}(\tilde{e}) \in \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}'_j)$. Moreover, $\tilde{g}(\tilde{e}) = g_{n+1}(e) \in \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}'_{n+1})$. Hence, $\tilde{g}(\tilde{e}) \in \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}'_n)$ by the definition of the remainder. This implies that $\tilde{e} \in \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_n)$, hence $\tilde{g}(\tilde{e}) \in g_n(\tilde{I}') \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}'_{n+1})$. From the fact that $\tilde{g}(\tilde{e}) = g_n(\tilde{e})$ and that g_n^{-1} and g_{n+1}^{-1} coincide on $g_n(\tilde{I}') \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}'_{n+1})$ we obtain $e = \tilde{e}$, which is a contradiction. □

Proof of Proposition 4.20: The inclusion $\sim_{UI} \subset \sim_{FUI}$ is easily seen.

Suppose $\mathcal{E} \sim_{FUI} \mathcal{E}'$ and $(e_i, \gamma_i)_{i\leq n} \in T^{ic}(\mathcal{E})$. Define $\mathcal{E}_0 = \mathcal{E}$ and $\mathcal{E}_{i+1} = \mathcal{E}_{i[e_i]}$ for $i \leq n$, which is well defined since $(e_i, \gamma_i)_{i\leq n} \in T^{ic}(\mathcal{E})$. And let $\mathcal{R}$ be a FUI-bisimulation such that $(\mathcal{E}, \mathcal{E}', f) \in \mathcal{R}$. Furthermore, define $I = \bigcup_{i\leq n}(\gamma_i \cup \{e_i\})$. Then by Lemma 4.28 we obtain a function $g : I \to E'$ such that g is injective, labeling-preserving and $\forall i \leq n : \exists g_i : (\mathcal{E}_i, \mathcal{E}'_i, g_i) \in \mathcal{R} \wedge g \upharpoonright (\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_i) \cap I) = g_i \upharpoonright (\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_i) \cap I)$, where $\mathcal{E}'_i = \mathcal{E}'_{[g(e_0)]\cdots[g(e_{i-1})]}$.

From $(\mathcal{E}_i, \mathcal{E}'_i, g_i) \in \mathcal{R}$ and from $\gamma_i \subseteq \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_i)$ we obtain that $g_i(\gamma_i) \subseteq \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}'_i)$. Hence, $(g(e_i), g(\gamma_i))_{i\leq n} \in T^{ic}(\mathcal{E}')$, since $g(\gamma_i) = g_i(\gamma_i)$. □

4.4.3 Proof of Proposition 4.22

We use the following lemmas:

Lemma 4.29 *Suppose* $A \subseteq \mathsf{Obs}$ *and* $\theta_1, \theta_2 : A \to \mathbf{CBES}$ *such that* $\forall a \in A : \theta_1(a) \sim_b \theta_2(a)$. *Then* $Ref^e_A(\mathcal{E}, \theta_1) \sim_b Ref^e_A(\mathcal{E}, \theta_2)$ *for any* $\mathcal{E} \in \mathbf{CBES}$.

Proof: First, we show

$$(\forall e \in \mathcal{U} : \vartheta_1(e) \sim_b \vartheta_2(e)) \Rightarrow \underline{Ref}^e_A(\mathcal{E}, \vartheta_1) \sim_b \underline{Ref}^e_A(\mathcal{E}, \vartheta_2) \tag{4.4}$$

Let $\mathcal{R}_e$ be a corresponding bisimulation for $\vartheta_1(e) \sim_b \vartheta_2(e)$. Define

$$\mathcal{R} = \{(\underline{Ref}^e_A(\mathcal{E}', \vartheta_1'), \underline{Ref}^e_A(\mathcal{E}', \vartheta_2') \mid \mathcal{E}' \in \mathbf{CBES} \wedge \forall e \in \mathcal{U} : (\vartheta_1'(e), \vartheta_2'(e)) \in \mathcal{R}_e\}$$

By using Lemma 4.25, it is easy to check that $\mathcal{R}$ is a bisimulation, which establishes (4.4).

Moreover, ϑ_i can be easily derived from θ_i such that ϑ_i satisfies the requirement of (4.4) and that $Ref^e_A(\mathcal{E}, \theta_i) = \underline{Ref}^e_A(\mathcal{E}, \vartheta_i)$. □

Lemma 4.30 *Suppose* $A, A' \subseteq \mathsf{Obs}$, $\theta : A \to \mathbf{CBES}$, $\theta' : A' \to \mathbf{CBES}$ *and* $\theta'' : (A \cup A') \to \mathbf{CBES}$ *such that* $\theta''(a) = \begin{cases} Ref^e_A(\theta(a), \theta') & \text{if } a \in A \\ \theta'(a) & \text{if } a \in A' \backslash A \end{cases}$. *Then* $Ref^e_{A'}(Ref^e_A(\mathcal{E}, \theta), \theta')$ *is isomorphic to* $Ref^e_{A \cup A'}(\mathcal{E}, \theta'')$ *for any* $\mathcal{E} \in \mathbf{CBES}$.

Proof: The isomorphism $\kappa : E_{Ref^e_{A'}(Ref^e_A(\mathcal{E},\theta),\theta')} \to E_{Ref^e_{A\cup A'}(\mathcal{E},\theta'')}$ is given by

$$\kappa(((e, \hat{e}), \hat{e}')) = \begin{cases} (e, e) & \text{if } l(e) \notin A \cup A' \\ (e, \hat{e}') & \text{if } l(e) \in A' \backslash A \\ (e, (\hat{e}, \hat{e})) & \text{if } l(e) \in A \wedge \hat{e} \notin A' \\ (e, (\hat{e}, \hat{e}')) & \text{if } l(e) \in A \wedge \hat{e} \in A' \end{cases}.$$

The proof that κ is an isomorphism is straightforward. □

Proof of Proposition 4.22: We have $\sim_c \subseteq \sim_b$, since $\mathcal{E}$ is isomorphic to $Ref^e_\emptyset(\mathcal{E}, \theta)$. Therefore, it is only left to prove that $\sim_c$ is a congruence, since every congruence which is below $\sim_b$ has to satisfy the constraint of $\sim_c$.

Suppose $\mathcal{E}_1 \sim_c \mathcal{E}_2$, $A \subseteq \mathsf{Obs}$ and $\forall a \in A : \theta_1(a) \sim_c \theta_2(a)$. We have to verify $Ref^e_A(\mathcal{E}_1, \theta_1) \sim_c Ref^e_A(\mathcal{E}_2, \theta_2)$. Therefore, let $A' \subset \mathsf{Obs}$ and $\theta : A' \to \mathbf{CBES}$.

Define $\theta_i'(a) = \begin{cases} Ref^e_{A'}(\theta_i(a), \theta) & \text{if } a \in A \\ \theta(a) & \text{if } a \in A' \backslash A \end{cases}$ for $i \in \{1, 2\}$. From $\forall a \in A : \theta_1(a) \sim_c \theta_2(a)$ we get

$$\forall a \in A \cup A' : \theta_1'(a) \sim_b \theta_2'(a). \tag{4.5}$$

Therefore, we obtain that $Ref^e_{A'}(Ref^e_A(\mathcal{E}_1, \theta_1), \theta) \overset{Lemma\ 4.30}{\cong} Ref^e_{A\cup A'}(\mathcal{E}_1, \theta_1') \overset{(4.5)\ and\ Lemma\ 4.29}{\sim_b}$ $Ref^e_{A\cup A'}(\mathcal{E}_1, \theta_2') \overset{\mathcal{E}_1 \sim_c \mathcal{E}_2}{\sim_b} Ref^e_{A\cup A'}(\mathcal{E}_2, \theta_2') \overset{Lemma\ 4.30}{\cong} Ref^e_{A'}(Ref^e_A(\mathcal{E}_2, \theta_2), \theta)$, as required □

Chapter 5

Terminating by Action Execution

In this chapter, an alternative approach to process termination, where a process is considered to terminate by executing its 'final' action [14, 28], is considered. This approach is called *fa-approach* in the following. In addition to the transition relation a predicate for termination with respect to action names is used. This termination approach is also used in many event-based models, like those mentioned in [91], where termination is indicated by maximal configurations and not by termination events.

Two new kinds of event structures are introduced in order to give a true concurrency model for process algebras based on the fa-philosophy. One type of event structure models the 'non-disabling' of events instead of disabling by using a witness relation. The other type of event structure models disabling by indicating sets of precursor events. We show that both types of event structures have the same expressive power and are more expressive with respect to event traces than the standard event structures. A consistency result of an operational and a denotational semantics is shown.

5.1 Motivation

Disrupt mechanisms are important in order to model many realistic systems. Hence they have found their way into various process algebras [14, 15, 73, 75]. The disrupt operator of LOTOS [32], called disabling operator, is denoted by $B_1 [> B_2$. Here, any action executed by B_2 disables B_1, and the termination of B_1 disables B_2 (see also Chapter 3). Disrupt mechanisms are e.g. used to model timeouts, which represent an important concept for many applications.

In the definition of an operational semantics, the disrupt operator has to be described. Therefore it is necessary to specify when a process terminates. This can be achieved in two ways:

- By providing an additional syntactical expression **1** to indicate the process that may terminate immediately. For the operational description, an additional action $\surd$, which indicates termination, and a rule $\mathbf{1} \xrightarrow{\surd} \ldots$ are used. This approach is taken in Chapter 3 and in [47, 32] [1], for example.

[1] These papers only differ with respect to the handling of sequential composition. For example, the sequential operator removes action $\surd$ in [47], whereas it is replaced with the internal action in LOTOS [32].

Unfortunately, some interesting and important ways of using disrupt mechanisms are excluded by the approach described above, as it is argued by the following generic example.

Example 5.1 *Consider a nuclear power plant. Let P_{sd} be a process that controls the shutdown of the reactor and let P_{nr} be the process that describes the normal running behavior of the reactor. In process P_{nr} an action named $stop$ may be activated by the environment. If $stop$ is executed, the normal behavior P_{nr} is terminated and P_{nr} may only terminate in this way. The $stop$-action indicates the controlled shutdown demand of a worker and is in general not allowed at every execution step of P_{nr}, i.e. it can not be modeled by disruption. Then, a simple specification of a nuclear power plant is given by*

$$P_{nr}; P_{sd}$$

where ; denotes the sequential composition. A more realistic specification of a nuclear power plant may invoke the shutdown process P_{sd} for various other reason, e.g. if the temperature reaches a critical point. In addition, the shutdown may be combined with other activities, e.g. that of setting off an alarm.

Let us consider a nuclear power plant with the action named $stop$ and a shutdown which is triggered by temperature and that also invokes an alarm. In any system run either a normal termination of P_{nr} by $stop$ or a disruption of P_{nr} by a critical temperature message may happen, but it should be prevented that both occur. In particular, once the $stop$-action has been taken, no alarm should be set off. Let action t denote that the temperature of the reactor reaches the critical point and let action a denote that an alarm is set off. The natural representation of the reactor control in LOTOS is given by

$$((P_{nr}\,[>t); P_{sd})\|_{\{t\}}(t; a)$$

where $B_1\|_A B_2$ denotes the parallel execution of B_1 and B_2 with synchronization on the actions in A. However, according to the semantics of LOTOS, t can happen after $stop$. This originates in the fact that the $\surd$-action (and not the $stop$-action) terminates P_{nr}. Hence, an alarm with expensive consequences may be unnecessarily set off.

- An alternative approach to the problem of dealing with termination is to specify that a process terminates when it executes its 'final' action [14, 28]. For example, the process $a\|_\emptyset b$ terminates by executing a if b was executed before or it terminates by executing b if a was executed before. This approach, which is called *fa-approach* in the following, leads to the expected behavior of the process considered in the above example. The fa-approach is modeled by using a predicate for termination with respect to action names in the transition system. There is no need to extend the syntactical expressions by further expressions, like $\mathbf{1}$, in order to handle termination.

5.2 Syntax

Let τ, Obs and Var be defined as in Section 3.2. The set of all actions $\mathcal{A}ct$ is defined by $\mathcal{A}ct = \{\tau\} \cup \mathsf{Obs}$. A relabeling function f is a function from $\mathcal{A}ct$ to $\mathcal{A}ct$ such that $f(\tau) = \tau$. We denote the set of all labeling functions by $\mathcal{F}_L$.

The process algebra expressions $\mathsf{EXP_{st}}$ ($_\mathsf{s} \triangleq$ start-based, $_\mathsf{t} \triangleq$ fa-termination) are defined by the following BNF-grammar.

$$B \quad ::= \quad \mathbf{0} \mid a \mid B + B \mid B;B \mid B \,[\!> B \mid B \|_A B \mid B[f] \mid B \backslash\backslash A \mid x$$

where $f \in \mathcal{F}_L$, $x \in \mathsf{Var}$, $a \in \mathcal{A}ct$ and $A \subseteq \mathsf{Obs}$. A *process with respect to* $\mathsf{EXP_{st}}$ is a pair $\langle \mathrm{decl}, B \rangle$ consisting of a declaration $\mathrm{decl} : \mathsf{Var} \to \mathsf{EXP_{st}}$ and an expression $B \in \mathsf{EXP_{st}}$. Let $\mathsf{PA_{st}}$ denote the set of all processes. We sometimes call an expression $B \in \mathsf{EXP_{st}}$ also a process if decl is clear from the context.

The expressions have the following intuitive meaning: a is the process that executes a and terminates. $B_1 \,[\!> B_2$ is the disruption of B_1 by B_2, i.e. any action from B_2 disables B_1 as long as B_1 has not terminated. On the other hand, the termination of B_1 disables B_2. $B_1 \|_A B_2$ describes the parallel execution of B_1 and B_2, where both processes have to synchronize on actions of A. The process terminates if B_1 and B_2 terminate in the case of synchronization or if one terminates and the other one has already terminated. The relabeling process $B[f]$ executes action $f(a)$ if B executes action a. The restriction process $B \backslash\backslash A$ executes action a if B executes action a, provided a is not contained in A. The behavior of the inactive process $\mathbf{0}$, of the choice operator $+$, of the sequential composition $;$ and of variable x is described in Section 3.2.

Process algebras, like [32, 98] or the one used in Chapter 3, that are based on the synchronization we have just presented and which contain an expression for a termination process (denoted by $\mathbf{1}$) can model a restriction operator in terms of the parallel operator ($B \|_A \mathbf{1}$). Since we do not introduce an expression for a termination process, we include the restriction operator, as in [25, 138]. As it turns out, the restriction operator plays also a crucial role for the operational definition of the parallel operator in our setting.

5.3 Operational Semantics for $\mathsf{PA_{st}}$

As stated in the beginning of this chapter, we adopt the philosophy that the 'final' executed action terminates the process. Therefore, we have to distinguish between 'final' actions and 'non-final' actions. In transition systems, the fa-philosophy is usually modeled by using a predicate for every action a to determine that the process can terminate by executing a [14, 28]. The non-terminating action execution is modeled by the usual transition relation. We take a different approach using additional labels. More precisely, we allow transitions to be labeled with elements of $\mathcal{A}ct^T = \mathcal{A}ct \cup (\mathcal{A}ct \times \{\surd\})$, where label $(a, \surd)$ indicates that the process terminates by executing action a, i.e. a is a 'final' action of that process. Actions of $\mathcal{A}ct \times \{\surd\}$ are called *termination actions*. Our approach leads to a decrease in the number of transition rules and allows a simplification of the definition of bisimulation and of the related proofs. The transition rules of $\longrightarrow^t_{\mathrm{decl}} \subseteq \mathsf{EXP_{st}} \times \mathcal{A}ct^T \times \mathsf{EXP_{st}}$ with respect to $\mathrm{decl} : \mathsf{Var} \to \mathsf{EXP_{st}}$ are presented in Figure 5.1. We write $a\surd$ instead of $(a, \surd)$.

In the following, we explain the rules which deviate from the standard ones: Process a can execute a and terminates by executing this action. The process which results in a deadlock after executing a can be modeled by $a;\mathbf{0}$, for example. The transition rule for the choice operator is the standard CCS-rule [138]. Please note that no distinction between actions and termination actions is made. If the first process of the sequential composition terminates by executing a

In the following let γ be an element of $\mathcal{A}ct^T$ and a be an element of $\mathcal{A}ct$

$$A_1 : \frac{}{a \xrightarrow{a\surd} \mathbf{0}} \qquad C : \frac{B_1 \xrightarrow{\gamma} B'}{\begin{array}{c} B_1 + B_2 \xrightarrow{\gamma} B' \\ B_2 + B_1 \xrightarrow{\gamma} B' \end{array}}$$

$$S_1 : \frac{B_1 \xrightarrow{a} B_1'}{B_1; B_2 \xrightarrow{a} B_1'; B_2} \qquad S_2 : \frac{B_1 \xrightarrow{a\surd} B_1'}{B_1; B_2 \xrightarrow{a} B_2}$$

$$I_1 : \frac{B_1 \xrightarrow{a} B_1'}{B_1 [> B_2 \xrightarrow{a} B_1' [> B_2} \qquad I_2 : \frac{B_1 \xrightarrow{a\surd} B_1'}{B_1 [> B_2 \xrightarrow{a\surd} B_1'} \qquad I_3 : \frac{B_2 \xrightarrow{\gamma} B_2'}{B_1 [> B_2 \xrightarrow{\gamma} B_2'}$$

$$P_1 : \frac{B_1 \xrightarrow{a} B_1' \quad a \notin A}{\begin{array}{c} B_1 \|_A B_2 \xrightarrow{a} B_1' \|_A B_2 \\ B_2 \|_A B_1 \xrightarrow{a} B_2 \|_A B_1' \end{array}} \qquad P_2 : \frac{B_1 \xrightarrow{a\surd} B_1' \quad a \notin A}{\begin{array}{c} B_1 \|_A B_2 \xrightarrow{a} B_2 \backslash\backslash A \\ B_2 \|_A B_1 \xrightarrow{a} B_2 \backslash\backslash A \end{array}}$$

$$P_3 : \frac{B_1 \xrightarrow{a} B_1' \quad B_2 \xrightarrow{a} B_2' \quad a \in A}{B_1 \|_A B_2 \xrightarrow{a} B_1' \|_A B_2'} \qquad P_4 : \frac{B_1 \xrightarrow{a\surd} B_1' \quad B_2 \xrightarrow{a} B_2' \quad a \in A}{\begin{array}{c} B_1 \|_A B_2 \xrightarrow{a} B_2' \backslash\backslash A \\ B_2 \|_A B_1 \xrightarrow{a} B_2' \backslash\backslash A \end{array}}$$

$$P_5 : \frac{B_1 \xrightarrow{a\surd} B_1' \quad B_2 \xrightarrow{a\surd} B_2' \quad a \in A}{B_1 \|_A B_2 \xrightarrow{a\surd} \mathbf{0}}$$

$$Lab_1 : \frac{B \xrightarrow{a} B'}{B[f] \xrightarrow{f(a)} B'[f]} \qquad Lab_2 : \frac{B \xrightarrow{a\surd} B'}{B[f] \xrightarrow{f(a)\surd} B'[f]}$$

$$Res_1 : \frac{B \xrightarrow{a} B' \quad a \notin A}{B \backslash\backslash A \xrightarrow{a} B' \backslash\backslash A} \qquad Res_2 : \frac{B \xrightarrow{a\surd} B' \quad a \notin A}{B \backslash\backslash A \xrightarrow{a\surd} B' \backslash\backslash A}$$

$$Rec : \frac{\mathrm{decl}(x) \xrightarrow{\gamma} B'}{x \xrightarrow{\gamma} B'}$$

Table 5.1: Transition Rules for $\longrightarrow^t_{\mathrm{decl}}$

(rule S_2), then $B_1; B_2$ executes a without termination. The rules for the disrupt operator are as they were expected, in particular if B_1 terminates by executing a, so does $B_1 [> B_2$.

In the case of the parallel operator, we have to distinguish whether or not the actions that are executed by the subprocess are termination actions. The second rule states that if a subprocess executes a non-synchronizing termination action, then this process has to be removed and

all actions in the synchronization set have to be forbidden for the remaining process. If both processes execute a termination action of the synchronization set, the resulting process is $\mathbf{0}$, since after the termination of both processes the parallel process has terminated, and no further actions will be executed.

The rules for the relabeling operator and the restriction operator only depend on the action name and preserve termination, as expected.

Remark 5.2 *It is easily seen by induction on the depth of inference that after the execution of a termination action no further actions may be executed, i.e.*

$$\forall B, B', a : B \stackrel{a\surd}{\longrightarrow}{}^{t}{}_{\mathrm{decl}}\ B' \Rightarrow (\forall \gamma, B'' : \neg(B' \stackrel{\gamma}{\longrightarrow}{}^{t}{}_{\mathrm{decl}}\ B'')).$$

5.4 Denotational Semantics for $\mathsf{PA_{st}}$

A denotational semantics of $\mathsf{PA_{st}}$ that corresponds to the operational semantics can not be given in the standard event structures. This is argued as follows.

Prime event structures [145], *flow event structures* [36] and *stable event structures* [178] require a symmetric conflict relation which makes it hard to model disruption. In addition *configurations* [87, 177] do not provide a smooth way to model disruption. Consider, for example, process $\tilde{B}$, which consists of the disruption of $a; b$ by action c (i.e. $\tilde{B} = (a; b)\,[\!>\, c$). An intuitive approach is to assume that $\tilde{B}$ has three events denoted by a, b, c. The sets $\emptyset, \{a\}, \{c\}, \{a, b\}$ can be considered to be configurations, but what about $\{a, c\}$? Assuming it is not a configuration contradicts the existing execution $\stackrel{a}{\longrightarrow}\stackrel{c}{\longrightarrow}$. On the other hand, assuming that $\{a, c\}$ is also a configuration leads to the interpretation that the execution $\stackrel{c}{\longrightarrow}\stackrel{a}{\longrightarrow}$ is legal[2], which contradicts the branching structure of $\tilde{B}$, since after the disruption (by c) no further actions from the left-hand process may be executed.

Closed bundle event structures (Subsection 3.3.2), which can be used to give a denotational semantics to LOTOS, and *dual event structures* [116] (Remark 3.3) allow the modeling of disruption, since the symmetry condition for the conflict relation is dropped. If (and how) these types of event structures could be used to define a denotational semantics that also incorporates the fa-philosophy is highly questionable. Consider, for example, the process $(a \|_{\emptyset} b)\,[\!>\, c$: If we put c in conflict with a, then c can be disabled before b happens and by symmetry the same argument holds for b. But c has to be in conflict with some action, since otherwise c would remain enabled after the execution of a and b.

Therefore, a conflict relation that is based on a binary relation on events is not appropriate to model this kind of disrupt operator in the context of the fa-philosophy.

In this section, we present two new classes of event structures that are suitable to model the fa-philosophy and study their properties. The first one, which is called *extended termination bundle event structure*, moves from a conflict approach to a *witness* approach by introducing a relation ($\succ$) between sets of events and events, i.e. $\succ \subseteq \mathcal{P}(E) \times E$. A witness condition ($Z \succ e$) indicates that event e is not disabled in a system run if no event from the system run is contained

[2] by the common definition [177, 87]

in Z. In other words, a system run disables an event e if each witness-bundle Z of e ($Z \succ e$) contains an element of the system run. As long as there is a witness condition $Z \succ e$, where Z does not contain an element of the system run so far, e is not disabled. Z is considered to be a *witness* of this fact.

The extended termination bundle event structures follow the philosophy of bundle event structures [126] that each bundle either from causality or from the witness relation is interpreted existentially, i.e. the execution of any event of a bundle is sufficient to fulfill the requirement denoted by the bundle.

The other new class of event structures, which is called *extended termination precursor event structure*, follows the contrary approach that each bundle (called *precursor* in these event structures) is universally quantified, i.e. every event of a precursor-bundle has to be executed to fulfill the requirement. More precisely, the conflict relation ($\hat{\succ}$) is given as a relation between sets of events and events, i.e. $\hat{\succ} \subseteq \mathcal{P}(E) \times E$. An event e is disabled by a system run if and only if there is a precursor Z of e ($Z \hat{\succ} e$) such that all events of Z are contained in the system run. The causality relation of an extended termination precursor event structure is also defined with the universal quantification philosophy, as in Winskel's event structures [178].

The rest of this section is organized as follows: First we neglect disruption and only concentrate on the fa-philosophy. Therefore, termination bundle event structures, which represent a generalization of bundle event structures, are presented in Subsection 5.4.1. They allow the modeling of the fa-philosophy. In Subsection 5.4.2 the class of extended termination bundle event structures, which can also handle disruption, is introduced. This Subsection contains the result that this class of event structures is more expressive than the standard event structures with respect to event traces. Operators on extended termination bundle event structures are defined in Subsection 5.4.3. These operators are used in Subsection 5.4.4 to define the denotational semantics of $\mathsf{PA}_{\mathsf{st}}$. There the consistency between the denotational semantics and the operational semantics is also given. The class of extended termination precursor event structures is introduced in Subsection 5.4.5. In Subsection 5.4.6, it is shown that the class of extended termination bundle event structures and the class of extended termination precursor event structures have the same expressive power with respect to event traces.

5.4.1 Termination Bundle Event Structure (TBES)

Bundle event structures (Definition 3.1) indicate the termination of a process by additional events that are labeled with the termination symbol $\surd$. These events are maintained by sequential composition, where they are relabeled with the internal action. This is not appropriate for models of process algebras that are based on the fa-philosophy, since contrary to the semantics of LOTOS, no internal action is executed when the first process terminates in the sequential composition.

Therefore, we do not allow events labeled with the termination symbol. Consequently, we have to model termination in a different way. Our approach is to consider the termination event to be fictitious, and therefore we collect the bundles of the termination event without pointing directly to an event. This means that we add an additional component that consists of a collection of bundles, i.e. consists of a collection of subsets of events, to bundle event structures.

Definition 5.3 (Termination Bundle Event Structure) *A* termination bundle event structure, Tbes *for short,* $(E, \sharp, \mapsto, T, l)$ *is an element of* $\mathcal{P}(\mathcal{U}) \times \mathcal{P}(\mathcal{U} \times \mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U}) \times \mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U})) \times (\mathcal{U} \rightharpoonup \mathcal{A}ct)$ *such that*

- $\sharp \subseteq E \times E$ *and* $\sharp$ *is irreflexive and symmetric*
- $\mapsto \subseteq \mathcal{P}(E) \times E$
- $T \subseteq \mathcal{P}(E)$ *and* $T \neq \emptyset$
- $\mathrm{dom}(l) = E$
- $\forall e \in E : \_ \mapsto e$ *is approximation closed with respect to* E
- T *is approximation closed with respect to* E

Let $\mathbf{TBES}$ *denote the set of all termination bundle event structures.*

We call E the *set of events*, $\sharp$ the *conflict* relation, $\mapsto$ the *causality* relation, T the *termination set* and l the *action-labeling* function.

The intuitive meaning of a Tbes is the following: If two events e, e' are in conflict, i.e. $e \sharp e'$, then only one of them can appear in a system run. The meaning of $X \mapsto e$ is that before e may be executed, an event of X has to be executed. A system run of a Tbes is terminated if all bundles in the termination set are satisfied, i.e. every element of T contains an event of the system run. The constraint $T \neq \emptyset$ on the termination set ensures that an Tbes is not able to terminate immediately, i.e. it can only terminate by executing an action. However, T might consist of the empty set only. The labeling function indicates which action is observable when the event is executed. The two approximation closedness constraints are used to guarantee that the standard order yields a complete partial order.

$\mathbf{TBES}$ can be used as a denotational semantics for the processes of $\mathsf{PA}_{\mathsf{st}}$ that do not contain disrupt expressions. The operators on $\mathbf{TBES}$ have to vary from those of the original bundle event structure (Definition 3.17 or [125, 126]). This is necessary, since the original bundle event structure (Definition 3.1) allows any number of events for termination, whereas our approach has exactly one event (the fictive one) for termination. We do not present the operators for $\mathbf{TBES}$. They can be easily obtained by adapting the operators for extended termination bundle event structures presented in Definition 5.20.

Remark 5.4 *The original bundle event structures (Definition 3.1) contain an additional condition, called* bundle stability constraint *(compare with Remark 3.3). It can also be added to the condition of a Tbes, since it is an invariant for all operators needed in the denotational semantics. We omit this constraint, since it is not important for the theory presented here.*

5.4.2 Extended Termination Bundle Event Structure ($\mathbf{ETBES}$)

In *extended termination bundle event structures* the non-disabling of events rather than the disabling is modeled, which is done by a relation between sets of events and events. The non-disabling modeling follows the philosophy of bundle event structures, where each bundle (obtained from causality) is existentially quantified, i.e. the execution of any event of a bundle is

sufficient to fulfill the requirement denoted by the bundle. In Section 5.4.5 we present a different approach, where every bundle is universally quantified.

Definition 5.5 (Extended Termination Bundle Event Structure) *An* extended termination bundle event structure, eTbes *for short,* $\mathcal{E} = (E, \succ, \mapsto, T, l)$ *is an element of* $\mathcal{P}(\mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U}) \times \mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U}) \times \mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U})) \times (\mathcal{U} \rightharpoonup \mathcal{A}ct)$ *such that*

- $\succ \subseteq \mathcal{P}(E) \times E$ *and* $\forall e \in E : \exists Z : Z \succ e$ *and* $\forall (Z, e) \in \succ : e \in Z$
- $\mapsto \subseteq \mathcal{P}(E) \times E$
- $T \subseteq \mathcal{P}(E)$ *and* $T \neq \emptyset$
- $\mathrm{dom}(l) = E$
- $\forall e \in E : \_ \succ e$ *is approximation closed with respect to* E
- $\forall e \in E : \_ \mapsto e$ *is approximation closed with respect to* E
- T *is approximation closed with respect to* E

Let $\mathbf{ETBES}$ *denote the set of all extended termination bundle event structures.*

$\succ$ is called the *witness* relation. The attribute *extended* in the name of the event structures defined above is used to emphasize (as it is done for extended bundle event structures) that these event structures can model disruption.

The intuitive meaning of a witness-bundle Z of e ($Z \succ e$) is that event e is not disabled in a system run if no event from the system run is contained in Z. A system run disables an event e if all witness-bundles of e contain an element of the system run. The constraints imposed on the witness relation are: Firstly, every event e must have a witness-bundle, since otherwise e would never be enabled and hence, could be omitted. Secondly, every witness-bundle of e has to contain e, since the execution of an event disables itself, i.e. every event can be executed only once. Furthermore, the witness relation also has to satisfy approximation closedness constraints to guarantee that the standard order yields a complete partial order.

Example 5.6 *Some eTbes are shown in Figure 5.1. Here, the events are depicted as dots and their corresponding action names appears close by the dots (we do not name the events explicitly and we identify them with the action names if no confusion arises). The witness relation is illustrated by wavy lines. More precisely, a witness-bundle* $Z \succ e$ *is depicted by a wavy arrow from the elements of* $Z \backslash \{e\}$ *to* e*. In the special case when* Z *consists only of* e*, we use a wavy arrow from the empty-set to* e*. Sometimes, the same wavy lines are used in different witness-bundles, for example the witness-bundles in* $\mathcal{E}_6$ *are* $\{a, b, c\} \succ a$*,* $\{a, b, c\} \succ b$ *and* $\{a, b, c\} \succ c$*. The causality relation is depicted similarly to the witness relation, except that straight lines are used instead of wavy lines. A termination set* X *is displayed by surrounding its events by a closed line.*

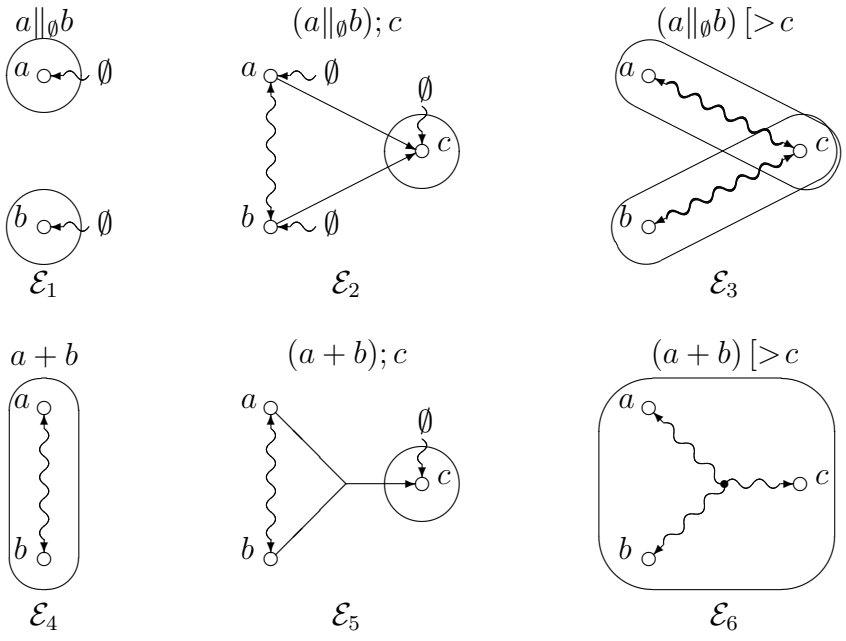


Figure 5.1: Some Extended Termination Bundle Event Structures

Hereafter, we consider $\mathcal{E}$ to be $(E, \succ, \mapsto, T, l)$, $\mathcal{E}_i$ to be $(E_i, \succ_i, \mapsto_i, T_i, l_i)$ and in general $\mathcal{E}$ to be $(E_\mathcal{E}, \succ_\mathcal{E}, \mapsto_\mathcal{E}, T_\mathcal{E}, l_\mathcal{E})$. Furthermore, $\mathrm{init}(\mathcal{E})$ denotes the set of events which are ready to execute and $\Upsilon(T, e)$ holds if and only if e is a *termination event* with respect to T, i.e. $\mathcal{E}$ terminates by executing e. Formally:

Definition 5.7 *Let $\mathcal{E}$ be an eTbes. The set of* initial events *of $\mathcal{E}$ is defined by*

$$\mathrm{init}(\mathcal{E}) = \{e \in E \mid \neg(\exists X : X \mapsto e)\}.$$

The termination predicate $\Upsilon \subseteq \mathcal{P}(\mathcal{P}(\mathcal{U})) \times \mathcal{U}$ *is defined by*

$$\Upsilon(T, e) \iff \forall X \in T : e \in X.$$

Remark 5.8 *Please note, that the events of an eTbes are labeled with elements of $\mathcal{A}ct$ and not of $\mathcal{A}ct^T$, i.e. events must not be labeled, for example, by $a\surd$. This is necessary, since we do not know a priori if an event is a termination event of a system run. See $\mathcal{E}_1$ in Figure 5.1, for example.*

Transition system from an eTbes.

Here, we describe how to obtain a transition system from an eTbes, which is later used to analyze the expressive power of **ETBES** and to establish a consistency result for the denotational and the operational semantics. In order to obtain a transition system from an eTbes, we define the remainder of an eTbes with respect to an initial event. The remainder with respect to event

e describes the system after the execution of e. Therefore, we remove all events which are disabled by e, i.e. we only keep those events that have a witness-bundle which does not contain e. Please remember that an event has to be an element of all its witness-bundles. Hence, it disables itself. Furthermore, all bundles (from causality, witness or termination) that contain e are removed, since the execution of e fulfills the requirements specified by those bundles, i.e. the bundle contains an element of the system run. In the definition of the termination set, we consider the case that an eTbes terminates by executing e separately in order to guarantee that the remainder of an eTbes is an eTbes. This distinction is necessary, since otherwise the termination set would become the empty set, which is not allowed for an eTbes.

Definition 5.9 (Remainder of an eTbes) *Let $\mathcal{E} \in \mathbf{ETBES}$ and $e \in \mathrm{init}(\mathcal{E})$. Then the* remainder $\mathcal{E}_{[e]}$ *of $\mathcal{E}$ is given by $(E', \succ', \mapsto', T', l')$ where*

$$\begin{array}{rcl} E' & = & \{e' \in E \mid \exists Z : Z \succ e' \wedge e \notin Z\} \\ \succ' & = & \{(Z \cap E', e') \mid e' \in E' \wedge Z \succ e' \wedge e \notin Z\} \\ \mapsto' & = & \{(X \cap E', e') \mid e' \in E' \wedge X \mapsto e' \wedge e \notin X\} \\ T' & = & \begin{cases} \{X \cap E' \mid X \in T \wedge e \notin X\} & \textit{if } \neg\Upsilon(T, e) \\ \{\emptyset\} & \textit{otherwise} \end{cases} \\ l' & = & l \restriction E' \end{array}$$

It can be shown that the remainder of an eTbes is also an eTbes.

Lemma 5.10 *Let $\mathcal{E} \in \mathbf{ETBES}$ and $e \in \mathrm{init}(\mathcal{E})$. Then $\mathcal{E}_{[e]} \in \mathbf{ETBES}$.*

Proof: The approximation closedness conditions are an immediate consequence of Corollary 2.18. The other conditions can be easily checked. □

The remainders are used in the following definition to obtain an interleaving semantics for $\mathbf{ETBES}$.

Definition 5.11 *The transition relation $\hookrightarrow \subseteq \mathbf{ETBES} \times \mathcal{A}ct^T \times \mathbf{ETBES}$ is defined by*

$$\hookrightarrow = \{(\mathcal{E}, \gamma, \mathcal{E}_{[e]}) \mid \mathcal{E} \in \mathbf{ETBES} \wedge e \in \mathrm{init}(\mathcal{E}) \wedge \gamma = \begin{cases} l(e)\surd & \textit{if } \Upsilon(T, e) \\ l(e) & \textit{if } \neg\Upsilon(T, e) \end{cases} \}.$$

The transition system obtained from $\mathcal{E}_3$ of Figure 5.1 is presented in Figure 5.2.

Remark 5.12 *According to the definition of the remainder, it is possible that further events may be executed after the execution of a termination event. This effect also arose in the original bundle event structures. It is possible to circumvent this effect by considering only those eTbes where every event set that leads to termination also disables every event. Formally, eTbes $\mathcal{E}$ has to satisfy (remember that $\_ \succ e = \{Z \mid Z \succ e\}$):*
$\forall e \in E : \forall E' \subseteq E : (\forall X \in T : E' \cap X \neq \emptyset) \Rightarrow (\forall Z \in \_ \succ e : Z \cap E' \neq \emptyset)$.

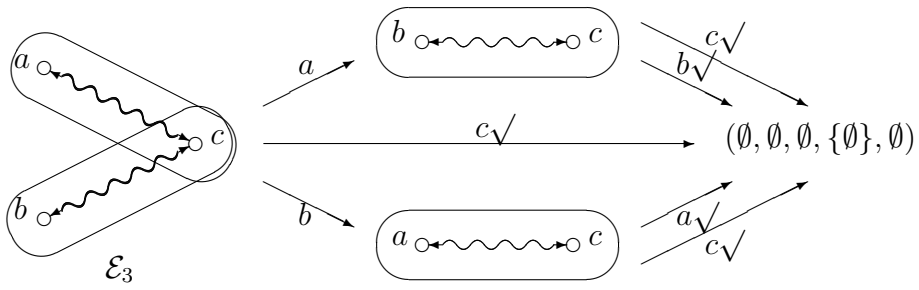


Figure 5.2: Transition System Derived from **ETBES**

Expressive Power.

The expressive power of event structures can be measured by comparing the set of event traces described by them. Here, we present the definition of event traces with respect to **ETBES**. The event traces for the other event structures are similarly defined.

Definition 5.13 *An* event trace *of $\mathcal{E}_1 \in$ **ETBES** is a finite sequence of events $(e_1, ..., e_n)$ such that there are eTbes $\mathcal{E}_2, ..., \mathcal{E}_{n+1}$ with $\mathcal{E}_{j[e_j]} = \mathcal{E}_{j+1}$ for all $j \leq n$. The* set of all event traces *of $\mathcal{E}_1$ is denoted by $\mathrm{Tr}^e(\mathcal{E}_1)$.*

A set M of finite sequences of events (set of event traces) is described by **ETBES** *if there exists $\mathcal{E} \in$ **ETBES** such that $M = \mathrm{Tr}^e(\mathcal{E})$.*

For simplicity, we neglect the termination information when we compare the expressive power. The termination information can be easily included by dividing the set of event traces into terminated and non-terminated traces.

Example 5.14 *The set of event traces of $\mathcal{E}_3$ from Figure 5.1 is*

$$\mathrm{Tr}^e(\mathcal{E}_3) = \{(a), (b), (c), (a, b), (b, a), (a, c), (b, c)\}.$$

Theorem 5.15 *Every set of event traces that is described by prime [145], flow [36], stable [178], bundle, extended bundle or dual event structures [116] is also described by extended termination bundle event structures, but not vice versa.*

Proof: From [116] we know that every set of event traces described by a cited class of event structures is also described by dual event structures. The inclusion of dual event structures in extended termination bundle event structures is shown by mapping the dual event structure $(E, \rightsquigarrow, \mapsto, l)$ to $(E, \succ, \mapsto, \{\emptyset\}, l)$ where $\succ = \{(\{e' \in E \mid e \rightsquigarrow e'\} \cup \{e\}, e) \mid e \in E\}$. Furthermore, the causality relation has to be extended such that it is approximation closed with respect to E. This extension does not change the set of event traces if the least extension is used.

On the other hand, the set of event traces obtained from $\mathcal{E}_3$ of Figure 5.1 can not be described by a dual event structure. □

Complete Partial Order.

In order to give a semantics to $\mathsf{PA}_{\mathsf{st}}$, we turn $\mathbf{ETBES}$ into an ω-complete partial order. First, we present the definition and results concerning the restriction of an eTbes, which is used to define an order on $\mathbf{ETBES}$.

Definition 5.16 (Restriction of an eTbes) *Suppose $\mathcal{E} \in \mathbf{ETBES}$ and $E' \subseteq E$. Then the* restriction of $\mathcal{E}$ to E'*, denoted by $\mathcal{E} \restriction E'$, is given by $(E', \succ', \mapsto', T', l')$ where*

$$\begin{array}{rcl} \succ' & = & \{(Z \cap E', e') \mid e' \in E' \wedge Z \succ e'\} \\ \mapsto' & = & \{(X \cap E', e') \mid e' \in E' \wedge X \mapsto e'\} \\ T' & = & \{X \cap E' \mid X \in T\} \\ l' & = & l \restriction E' \end{array}$$

Lemma 5.17 *Let $\mathcal{E} \in \mathbf{ETBES}$ and $E' \subseteq E$. Then $\mathcal{E} \restriction E' \in \mathbf{ETBES}$.*

Proof: Is an immediate consequence of Corollary 2.18. □

Definition 5.18 (Order on $\mathbf{ETBES}$) *Let $\mathcal{E}_i \in \mathbf{ETBES}$. Then $\mathcal{E}_1 \trianglelefteq \mathcal{E}_2$ if $E_1 \subseteq E_2$ and $\mathcal{E}_1 = \mathcal{E}_2 \restriction E_1$.*

Theorem 5.19 *The set of all eTbes ordered by $\trianglelefteq$ is an ω-complete partial order, where the least upper bound of an ω-chain $(\mathcal{E}_i)_{i\in\mathbb{N}}$ is given by $\bigsqcup_i \mathcal{E}_i = (\bigcup_i E_i, \succ, \mapsto, T, \bigcup_i l_i)$ with*

$$\begin{array}{rcl} \succ & = & \{(Z, e) \mid \forall k : e \in E_k \Rightarrow (Z \cap E_k) \succ_k e\} \\ \mapsto & = & \{(X, e) \mid \forall k : e \in E_k \Rightarrow (X \cap E_k) \mapsto_k e\} \ . \\ T & = & \{X \mid \forall k : X \cap E_k \in T_k\} \end{array}$$

Proof: It works analogously to the proof of Theorem 3.10. □

5.4.3 Operators on $\mathbf{ETBES}$

Here, we present the operators on $\mathbf{ETBES}$ that are later used to define the denotational semantics.

Definition 5.20 (Operators on $\mathbf{ETBES}$) *Let $A \subseteq \mathsf{Obs}$. Then define*

$\widehat{+} : \mathbf{ETBES} \times \mathbf{ETBES} \to \mathbf{ETBES}$ *with $\mathcal{E}_1 \widehat{+} \mathcal{E}_2 = (\tilde{E}, \tilde{\succ}, \tilde{\mapsto}, \tilde{T}, \tilde{l})$ where*

$$\begin{array}{rcl} \tilde{E} & = & (\{\star_1\} \times E_1) \cup (\{\star_2\} \times E_2) \\ \tilde{\succ} & = & \{((\{\star_i\} \times Z) \cup (\{\star_j\} \times \mathrm{init}(\mathcal{E}_j)), (\star_i, e)) \mid Z \succ_i e \wedge i \neq j\} \\ \tilde{\mapsto} & = & \{(\{\star_i\} \times X, (\star_i, e)) \mid X \mapsto_i e\} \\ \tilde{T} & = & \{(\{\star_1\} \times X_1) \cup (\{\star_2\} \times X_2) \mid X_1 \in T_1 \wedge X_2 \in T_2\} \end{array}$$

$\tilde{l}((\star_i, e)) = l_i(e)$

$\widehat{;} : \mathbf{ETBES} \times \mathbf{ETBES} \to \mathbf{ETBES}$ *with* $\mathcal{E}_1 \widehat{;} \mathcal{E}_2 = (\tilde{E}, \tilde{\succ}, \tilde{\mapsto}, \tilde{T}, \tilde{l})$ *where*

$$\begin{array}{rcl}
\tilde{E} & = & (\{\star_1\} \times E_1) \cup (\{\star_2\} \times E_2) \\
\tilde{\succ} & = & \{(\{\star_1\} \times (Z \cup X), (\star_1, e)) \mid Z \succ_1 e \wedge X \in T_1\} \cup \\
 & & \{(\{\star_2\} \times Z, (\star_2, e)) \mid Z \succ_2 e\} \\
\tilde{\mapsto} & = & \{(\{\star_i\} \times X, (\star_i, e)) \mid X \mapsto_i e\} \cup \\
 & & \{(\{\star_1\} \times X_1, (\star_2, e)) \mid e \in \mathrm{init}(\mathcal{E}_2) \wedge X_1 \in T_1\} \\
\tilde{T} & = & \{\{\star_2\} \times X_2 \mid X_2 \in T_2\} \\
\tilde{l}((\star_i, e)) = l_i(e) & &
\end{array}$$

$\widehat{[>} : \mathbf{ETBES} \times \mathbf{ETBES} \to \mathbf{ETBES}$ *with* $\mathcal{E}_1 \widehat{[>} \mathcal{E}_2 = (\tilde{E}, \tilde{\succ}, \tilde{\mapsto}, \tilde{T}, \tilde{l})$ *where*

$$\begin{array}{rcl}
\tilde{E} & = & (\{\star_1\} \times E_1) \cup (\{\star_2\} \times E_2) \\
\tilde{\succ} & = & \{((\{\star_1\} \times Z) \cup (\{\star_2\} \times \mathrm{init}(E_2)), (\star_1, e)) \mid Z \succ_1 e\} \cup \\
 & & \{(\{\star_1\} \times X)(\cup(\{\star_2\} \times Z), (\star_2, e)) \mid X \in T_1 \wedge Z \succ_2 e\} \\
\tilde{\mapsto} & = & \{(\{\star_i\} \times X, (\star_i, e)) \mid X \mapsto_i e\} \\
\tilde{T} & = & \{(\{\star_1\} \times X_1) \cup (\{\star_2\} \times X_2) \mid X_1 \in T_1 \wedge X_2 \in T_2\} \\
\tilde{l}((\star_i, e)) = l_i(e) & &
\end{array}$$

$\widehat{\|}_A : \mathbf{ETBES} \times \mathbf{ETBES} \to \mathbf{ETBES}$ *with* $\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2 = (\tilde{E}, \tilde{\succ}, \tilde{\mapsto}, \tilde{T}, \tilde{l})$ *where*

$$\begin{array}{rcl}
\tilde{E} & = & (E_1^f \times \{\star\}) \cup (\{\star\} \times E_2^f) \cup E^s \\
E_i^f & = & \{e \in E_i \mid l_i(e) \notin A\} \\
E^s & = & \{(e_1, e_2) \in E_1 \times E_2 \mid l_1(e_1) = l_2(e_2) \in A\} \\
\tilde{\succ} & = & \{(\{(e_1', e_2') \in \tilde{E} \mid e_1' \in Z_1 \cup X_1\}, (e_1, \star)) \mid Z_1 \succ_1 e_1 \wedge X_1 \in T_1\} \cup \\
 & & \{(\{(e_1', e_2') \in \tilde{E} \mid e_2' \in Z_2 \cup X_2\}, (\star, e_2)) \mid Z_2 \succ_2 e_2 \wedge X_2 \in T_2\} \cup \\
 & & \{(\{(e_1', e_2') \mid e_1' \in Z_1 \cup X_1 \vee e_2' \in Z_2 \cup X_2\}, (e_1, e_2)) \mid \\
 & & \qquad (e_1, e_2) \in E^s \wedge Z_1 \succ_1 e_1 \wedge X_1 \in T_1 \wedge Z_2 \succ_2 e_2 \wedge X_2 \in T_2\} \\
\tilde{\mapsto} & = & \{(\{(e_1', e_2') \in \tilde{E} \mid e_i' \in X_i\}, (e_1, e_2)) \mid X_i \mapsto_i e_i\} \\
\tilde{T} & = & \{\{(e_1, e_2) \in \tilde{E} \mid e_i \in X_i\} \mid X_i \in T_i\} \\
\tilde{l}((e_1, e_2)) = \begin{cases} l_1(e_1) & \textit{if } e_2 = \star \\ l_2(e_2) & \textit{otherwise} \end{cases} & &
\end{array}$$

$\widehat{\mathrm{Lab}} : (\mathbf{ETBES} \times \mathcal{F}_L) \to \mathbf{ETBES}$ *with* $\widehat{\mathrm{Lab}}(\mathcal{E}, f) = (E, \succ, \mapsto, T, f \circ l)$.

$\widehat{\backslash\backslash} A : \mathbf{ETBES} \to \mathbf{ETBES}$ *with* $\mathcal{E} \widehat{\backslash\backslash} A = \mathcal{E} \upharpoonright \{e \in E \mid l(e) \notin A\}$.

We will give some comments on the definition of these operators: The definitions of the set of events, the causality relation and the relabeling function are the standard ones [125], except that the disjoint union is explicitly used (see Subsection 3.3.3). An event $(\star_1, e)$ from $\mathcal{E}_1 \widehat{+} \mathcal{E}_2$ has to be disabled if an event corresponding to E_2 is executed, where the event of E_2 has to be necessarily an initial event of $\mathcal{E}_2$. Therefore, the witness-bundles of $(\star_1, e)$ are obtained by extending the original ones with $\mathrm{init}(E_2)$. And, of course, similarly defined for $(\star_2, e)$ events. The termination set of $\mathcal{E}_1 \widehat{+} \mathcal{E}_2$ is obtained by taking any combination of the elements of the two termination sets. This is intuitive, since an event is a termination event of $\mathcal{E}_1 \widehat{+} \mathcal{E}_2$ if and only if it is a termination event of $\mathcal{E}_1$ or $\mathcal{E}_2$.

An event $(\star_2, e)$ from $\mathcal{E}_1 \widehat{;} \mathcal{E}_2$ can not be disabled by events from $\mathcal{E}_1$, and therefore we only take the original witness-bundles. On the other hand, we additionally disable an event $(\star_1, e)$ by the termination of $\mathcal{E}_1$. This is done in order to achieve consistence with the operational semantics. $\mathcal{E}_1 \widehat{;} \mathcal{E}_2$ terminates if and only if a termination event from $\mathcal{E}_2$ is executed. Hence, the termination set of $\mathcal{E}_1 \widehat{;} \mathcal{E}_2$ is defined as the termination set of $\mathcal{E}_2$.

The witness-bundle relation of $\mathcal{E}_1 \widehat{[>} \mathcal{E}_2$ is a combination of the ideas of $\mathcal{E}_1 \widehat{+} \mathcal{E}_2$ and of $\mathcal{E}_1 \widehat{;} \mathcal{E}_2$, since the $(\star_1, e)$ events of $\mathcal{E}_1 \widehat{[>} \mathcal{E}_2$ are disabled by any $(\star_2, e)$-event and a $(\star_2, e)$-event is disabled by a termination event of $\mathcal{E}_1$. Furthermore, the set of termination is similar to $\mathcal{E}_1 \widehat{+} \mathcal{E}_2$.

An event $e = (e_1, e_2)$ of $\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2$ is disabled if e_1 or e_2 is disabled or if another event where one component is equal to a component of e, like (e_1, e_2'), is executed. Furthermore, e is also disabled if one side of the parallel operator terminates and e is a synchronization event, i.e. $e_1, e_2 \neq \star$. This is intuitive, since a process that terminates can not execute any further action. Therefore, no synchronization can take place. The disabling mechanism described above is obtained by taking any combination of the corresponding sets, as it can be seen in the definition. The set of termination for $\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2$ is obtained by taking the union of the termination sets of its components, where we have to guarantee that all corresponding synchronization events are contained in this union.

The witness-bundles and the termination are unaffected by the labeling operator. The restriction operator removes all forbidden events, i.e. those labeled with elements of A. This is also done in the witness-bundles and in the termination set.

Lemma 5.21 *All operators of Definition 5.20 are well defined, i.e. they really yield elements of* **ETBES**.

Proof: The well-definedness of $\widehat{\backslash\backslash} A$ follows from Lemma 5.17. The approximation closedness conditions of the witness relation of the sequential operator are a consequence of Corollary 2.19. The other conditions are easily seen except for the approximation closedness conditions of the parallel operator.

Therefore, let $\tilde{\mathcal{E}} = \mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2$.

$\tilde{\succ}$: Let $(e_1', e_2') \in E_{\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2}$. Define $M_i = \begin{cases} \{\emptyset\} & \text{if } e_i' = \star \\ \{Z_i \cup X_i \mid Z_i \succ_i e_i' \wedge X_i \in T_i\} & \text{otherwise} \end{cases}$.
Then M_i is approximation closed with respect to $E_i \cup \{\star\}$ by Corollary 2.19. From Corollary 2.20 we obtain that $M' = \{\{(e_1, e_2) \in (E_1 \cup \{\star\}) \times (E_2 \cup \{\star\}) \mid e_1 \in X_1 \vee e_2 \in X_2\} \mid X_1 \in M_1 \wedge X_2 \in M_2\}$ is approximation closed with respect to $(E_1 \cup \{\star\}) \times (E_2 \cup \{\star\})$. And so the approximation closedness of $\tilde{\succ}$ follows from Corollary 2.18, since $\{\tilde{X} \mid \tilde{X} \tilde{\succ} (e_1', e_2')\} = \{X' \cap \tilde{E} \mid X' \in M'\}$.

$\tilde{\mapsto}$: The proof is similar to the one of Lemma 3.18.

$\tilde{T}$: Analogous to $\tilde{\mapsto}$. □

Lemma 5.22 *All operators of Definition 5.20 are continuous with respect to* $\trianglelefteq$.

Proof: Analogous to the proof of Lemma 3.18. □

Remark 5.23 *The* bundle stability constraint *(see Remark 5.4) of an eTbes $\mathcal{E}$ can be formalized by*

$$\forall X \in T \cup \pi_1(\mapsto) : \forall e, e' \in X : \forall Z : Z \succ e \Rightarrow e' \in Z.$$

But contrary to **TBES***, this constraint is to restrictive, since it is not preserved by the disrupt operator. This can be seen as follows. Consider $\mathcal{E}_3$ of Figure 5.1. Then $\mathcal{E}_3$ does not satisfy the bundle stability constraint, since $\{a, c\} \in T_3$ but $\{b, c\} \succ c$. Furthermore, $\mathcal{E}_3 = \mathcal{E}_1 \widehat{[>}(\{\bullet\}, \{(\{\bullet\}, \bullet)\}, \emptyset, \{\{\bullet\}\}, \{(\bullet, c)\})$, where both components satisfy the bundle stability constraint.*

5.4.4 Denotational Meaning for $\mathsf{PA}_{\mathsf{st}}$

First, we define the denotational semantics of expressions ($\mathsf{EXP}_{\mathsf{st}}$) with respect to variable assignments, i.e. functions from Var to **ETBES**. Then variable assignments are derived from declarations, which are used to define the denotational semantics of processes ($\mathsf{PA}_{\mathsf{st}}$).

Definition 5.24 *Let $[\![\_]\!]_\_ : \mathsf{EXP}_{\mathsf{st}} \times (\mathsf{Var} \to \mathbf{ETBES}) \to \mathbf{ETBES}$ be defined as follows (where $\rho : \mathsf{Var} \to \mathbf{ETBES}$)*

$$\begin{array}{ll}
[\![\mathbf{0}]\!]_\rho = (\emptyset, \emptyset, \emptyset, \{\emptyset\}, \emptyset) & [\![a]\!]_\rho = (\{\bullet\}, \{(\{\bullet\}, \bullet)\}, \emptyset, \{\{\bullet\}\}, \{(\bullet, a)\}) \\
[\![B_1 + B_2]\!]_\rho = [\![B_1]\!]_\rho \widehat{+} [\![B_2]\!]_\rho & [\![B_1; B_2]\!]_\rho = [\![B_1]\!]_\rho \widehat{;} [\![B_2]\!]_\rho \\
[\![B_1 [> B_2]\!]_\rho = [\![B_1]\!]_\rho \widehat{[>} [\![B_2]\!]_\rho & [\![B_1 \|_A B_2]\!]_\rho = [\![B_1]\!]_\rho \widehat{\|}_A [\![B_2]\!]_\rho \\
[\![B[f]]\!]_\rho = \widehat{\mathrm{Lab}}([\![B]\!]_\rho, f) & [\![B \backslash\backslash A]\!]_\rho = [\![B]\!]_\rho \widehat{\backslash\backslash} A \\
[\![x]\!]_\rho = \rho(x) &
\end{array}$$

Remark 5.25 *$[\![B]\!]_\_$ is continuous with respect to $\trianglelefteq$ for every $B \in \mathsf{EXP}_{\mathsf{st}}$.*

Assume $\mathrm{decl} : \mathsf{Var} \to \mathsf{EXP}_{\mathsf{st}}$. Then define $\mathcal{F}_{\mathrm{decl}} : (\mathsf{Var} \to \mathbf{ETBES}) \to (\mathsf{Var} \to \mathbf{ETBES})$ with $\mathcal{F}_{\mathrm{decl}}(\rho)_{(x)} = [\![\mathrm{decl}(x)]\!]_\rho$. From Remark 5.25 it follows that $\mathcal{F}_{\mathrm{decl}}$ is continuous. Therefore, from the complete partial order theory we get $\{\![\_]\!\} : (\mathsf{Var} \to \mathsf{EXP}_{\mathsf{st}}) \to (\mathsf{Var} \to \mathbf{ETBES})$ with $\{\![\mathrm{decl}]\!\} = \mathrm{fix}(\mathcal{F}_{\mathrm{decl}}) = \bigsqcup_n \mathcal{F}^n_{\mathrm{decl}}(\bot)$ is well defined.

Definition 5.26 (Denotational Semantics)
Define $[\![\_]\!] : \mathsf{PA}_{\mathsf{st}} \to \mathbf{ETBES}$ by $[\![\langle \mathrm{decl}, B\rangle]\!] = [\![B]\!]_{\{\![\mathrm{decl}]\!\}}$.

Example 5.27 *The denotational semantics of some processes is illustrated in Figure 5.1.*

The denotational semantics is consistent with the operational semantics, since the transition system derived from the denotational semantics is bisimilar to the operational semantics.

Theorem 5.28 (Consistency) *Suppose $\langle \mathrm{decl}, B\rangle \in \mathsf{PA}_{\mathsf{st}}$. Then $(\mathsf{EXP}_{\mathsf{st}}, \mathcal{A}ct^T, \longrightarrow^t_{\mathrm{decl}}, B)$ and $(\mathbf{ETBES}, \mathcal{A}ct^T, \hookrightarrow, [\![\langle \mathrm{decl}, B\rangle]\!])$ are bisimilar.*

Proof: The proof is given in Subsection 5.6.1. □

5.4.5 Extended Termination Precursor Event Structures (ETPES)

An event e of an eTbes is caused (or disabled) in a system run if the run contains an element of every causality (respectively witness) bundle X of e (bundle approach). Another contrary way of modeling causality is Winkel's quantification approach [178]: An event e is caused if there exists a 'causality set' X of e such that all elements of X occur in the system run. Since this approach is a popular one, we introduce in this section another class of event structures, which again use a relation ($\hat{\succ}$) between sets of events and events in order to model disabling. But this time, the causality and the disabling relation are interpreted with Winskel's quantification approach. We show in Subsection 5.4.6 that these event structures and the extended termination bundle event structures are equivalent.

Definition 5.29 *M is* finitely determined *with respect to E if*

- *E is a countable set*
- *M is* upper closed *with respect to E, i.e.* $\forall X, X' : (X \subseteq X' \wedge X \in M) \Rightarrow X' \in M$
- $\forall X \in M : \exists X' \in M : X' \subseteq X \wedge |X'| < |\mathbb{N}|$

Definition 5.30 (Extended Termination Precursor Event Structure) *An* extended termination precursor event structure, eTpes *for short, $\mathfrak{E} = (\hat{E}, \hat{\succ}, \hat{\mapsto}, \hat{T}, \hat{l})$ is an element of $\mathcal{P}(\mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U}) \times \mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U}) \times \mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U})) \times (\mathcal{U} \rightharpoonup \mathcal{A}ct)$ such that*

- *$\hat{\succ} \subseteq \mathcal{P}(\hat{E}) \times \hat{E}$ and $\forall e \in \hat{E} : \neg(\emptyset \hat{\succ} e)$ and $\forall e \in \hat{E} : \{e\} \hat{\succ} e$*
- *$\hat{\mapsto} \subseteq \mathcal{P}(\hat{E}) \times \hat{E}$*
- *$\hat{T} \subseteq \mathcal{P}(\hat{E})$ and $\emptyset \notin T$*
- $\mathrm{dom}(\hat{l}) = \hat{E}$
- *$\forall e \in \hat{E} : \_ \hat{\succ} e$ is finitely determined with respect to $\hat{E}$*
- *$\forall e \in \hat{E} : \_ \hat{\mapsto} e$ is finitely determined with respect to $\hat{E}$*
- *$\hat{T}$ is finitely determined with respect to $\hat{E}$*

Let **ETPES** *denote the set of all extended termination precursor event structures.*

$\hat{\succ}$ is called the *precursor conflict* relation. The other components are called the same as those of the extended termination bundle event structures. The intuitive meaning of the precursor conflict relation is that event e is disabled in a system run if there is a conflict precursor Z of e ($Z \hat{\succ} e$) such that the system run contains all elements of Z. The intuitive meaning of the causality relation $\hat{\mapsto}$ and of the termination set $\hat{T}$ is similar to $\hat{\succ}$. For example, a system run of an eTpes is terminated if there is an element X of $\hat{T}$, where every element of X appears in the system run.

The constraints imposed on the precursor conflict relation are: no event is immediately disabled ($\neg(\emptyset \hat{\succ} e)$), since otherwise the event can be omitted. Furthermore, the execution of an event

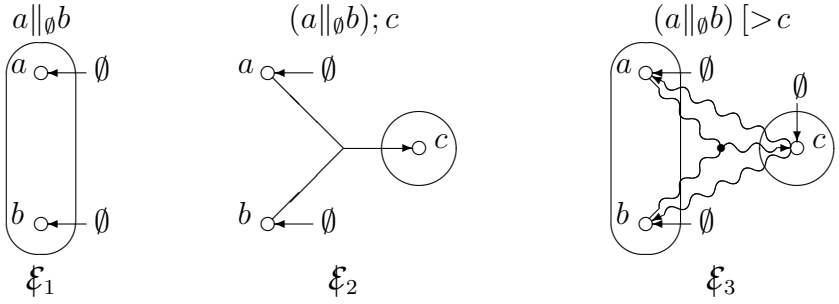


Figure 5.3: Some Extended Termination Precursor Event Structures

disables itself ($\{e\}\hat{\succ}e$), i.e. every event can happen only once. The constraint $\emptyset \notin \hat{T}$ on the termination set ensures that a precursor event structure may not terminate immediately, i.e. it can only terminate by executing an action. The three finitely determined constraints are used to guarantee that the order introduced later in this subsection yields a complete partial order.

Example 5.31 *Some eTpes are shown in Figure 5.3. The five components are displayed similarly to the components of an eTbes (see Example 5.6), i.e. the conflict relation is depicted as wavy lines, the causality as straight lines. We depict a termination precursor by surrounding its events by a closed line. Furthermore, we do not draw the conflict precursors of the form $\{e\}\hat{\succ}e$ and we omit the upper sets, e.g. in $\mathcal{E}_3$ we do not draw the termination precursors $\{a,c\}$, $\{b,c\}$ and $\{a,b,c\}$, which can be derived from the termination precursor $\{c\}$.*

Hereafter, $\mathcal{E}$ is considered to be $(\hat{E}, \hat{\succ}, \hat{\mapsto}, \hat{T}, \hat{l})$, $\mathcal{E}_i$ to be $(\hat{E}_i, \hat{\succ}_i, \hat{\mapsto}_i, \hat{T}_i, \hat{l}_i)$ and in general $\mathcal{E}$ is considered to be $(\hat{E}_\mathcal{E}, \hat{\succ}_\mathcal{E}, \hat{\mapsto}_\mathcal{E}, \hat{T}_\mathcal{E}, \hat{l}_\mathcal{E})$.

Definition 5.32 *Let $\mathcal{E}$ be an eTpes. The set of* initial events *of $\mathcal{E}$ is defined by*

$$\widehat{\mathrm{init}}(\mathcal{E}) = \{e \in \hat{E} \mid \emptyset \hat{\mapsto} e\}.$$

The termination predicate $\hat{\Upsilon} \subseteq \mathcal{P}(\mathcal{P}(\mathcal{U})) \times \mathcal{U}$ *is defined by*

$$\hat{\Upsilon}(\hat{T}, e) \iff \{e\} \in \hat{T}.$$

In the following two subsections, we derive a transition system from an eTpes and provide a complete partial order on $\mathbf{ETPES}$. These concepts will be used for the comparison of $\mathbf{ETBES}$ and $\mathbf{ETPES}$.

Transition Systems from an eTpes.

The remainder of an eTpes is given as follows.

Definition 5.33 (Remainder of an eTpes) *Let $\mathcal{E} \in \mathbf{ETPES}$ and $e \in \widehat{\mathrm{init}}(\mathcal{E})$. Then the* remainder $\mathcal{E}_{\widehat{[e]}}$ *of $\mathcal{E}$ is given by $(\hat{E}', \hat{\succ}', \hat{\mapsto}', \hat{T}', \hat{l}')$ where*

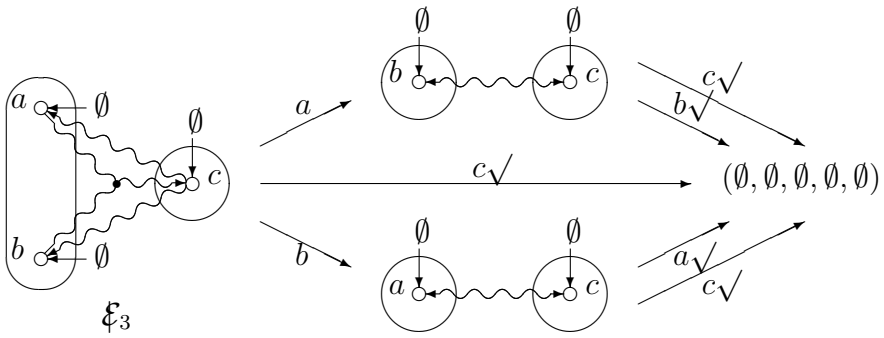


Figure 5.4: Transition System Derived from **ETPES**

$$
\begin{array}{rcl}
\hat{E}' & = & \{e' \in \hat{E} \mid \neg(\{e\}\hat{\succ}e')\} \\
\hat{\succ}' & = & \{(Z', e') \mid e' \in \hat{E}' \wedge Z' \subseteq \hat{E}' \wedge \exists Z : Z\hat{\succ}e' \wedge Z' = Z\backslash\{e\}\} \\
\hat{\mapsto}' & = & \{(X', e') \mid e' \in \hat{E}' \wedge X' \subseteq \hat{E}' \wedge \exists X : X\hat{\mapsto}e' \wedge X' = X\backslash\{e\}\} \\
\hat{T}' & = & \begin{cases} \{X' \mid X' \subseteq \hat{E}' \wedge \exists X \in \hat{T} : X' = X\backslash\{e\}\} & \textit{if } \neg\hat{\Upsilon}(\hat{T}, e) \\ \emptyset & \textit{otherwise} \end{cases} \\
\hat{l}' & = & \hat{l} \restriction \hat{E}'
\end{array}
$$

All events which are disabled by e are removed. Please remember that $\{e\}\hat{\succ}e$. Hence, e disables itself. After the execution of e, we keep exactly those precursors that are completely contained in $\hat{E}' \cup \{e\}$, since the other ones can not be contained in further system runs. In the definition of the termination set, we consider the case when an eTpes terminates by executing e separately in order to guarantee that the remainder of an eTpes is also an eTpes. This separation is necessary, since otherwise the empty set would be contained in the termination set, which is not allowed for an eTpes.

Lemma 5.34 *Let* $\mathcal{E} \in$ **ETPES** *and* $e \in \widehat{\mathrm{init}}(\mathcal{E})$. *Then* $\mathcal{E}_{\widehat{[e]}} \in$ **ETPES**.

Proof: Straightforward. □

Analogous to Definition 5.11, the remainder can be used to define an interleaving semantics for **ETPES**, which is omitted here. The transition system obtained from $\mathcal{E}_3$ of Figure 5.3 is presented in Figure 5.4.

Complete Partial Order.

We define the following order on **ETPES**.

Definition 5.35 *Let* $\mathcal{E}_i \in$ **ETPES**. *Then* $\mathcal{E}_1 \hat{\trianglelefteq} \mathcal{E}_2$ *if*

- $\hat{E}_1 \subseteq \hat{E}_2$
- $\hat{\succ}_1 = \{(X, e) \in \hat{\succ}_2 \mid X \subseteq \hat{E}_1 \wedge e \in \hat{E}_1\}$
- $\hat{\mapsto}_1 = \{(Z, e) \in \hat{\succ}_2 \mid Z \subseteq \hat{E}_1 \wedge e \in \hat{E}_1\}$

- $\hat{T}_1 = \{X \in \hat{T}_2 \mid X \subseteq \hat{E}_1\}$
- $\hat{l}_1 = \hat{l}_2 \upharpoonright \hat{E}_1$

Theorem 5.36 *The set of all eTpes ordered by $\hat{\trianglelefteq}$ is an ω-complete partial order, where the least upper bound of an ω-chain $(\hat{\mathcal{E}}_i)_{i\in\mathbb{N}}$ is given by $\hat{\bigsqcup}_i \hat{\mathcal{E}}_i = (\bigcup_i \hat{E}_i, \hat{\succ}, \hat{\mapsto}, \hat{T}, \bigcup_i \hat{l}_i)$ with*

$$\begin{array}{rcl} \hat{\succ} & = & \{(Z,e) \in \mathcal{P}(\bigcup_i \hat{E}_i) \times (\bigcup_i \hat{E}_i) \mid \exists j : (Z \cap \hat{E}_j) \hat{\succ}_j e\} \\ \hat{\mapsto} & = & \{(X,e) \in \mathcal{P}(\bigcup_i \hat{E}_i) \times (\bigcup_i \hat{E}_i) \mid \exists j : (X \cap \hat{E}_j) \hat{\mapsto}_j e\} \ . \\ \hat{T} & = & \{X \in \mathcal{P}(\bigcup_i \hat{E}_i) \mid \exists j : (X \cap \hat{E}_j) \in \hat{T}_j\} \end{array}$$

Proof: The proof is given in Subsection 5.6.2. □

5.4.6 Correspondence between ETBES and ETPES.

We show that there is a continuous function from **ETBES** ordered by $\trianglelefteq$ to **ETPES** ordered by $\hat{\trianglelefteq}$ and vice versa. This result is used to show that **ETBES** and **ETPES** have the same expressive power with respect to event traces.

Definition 5.37 *Let $F_E : \mathcal{P}(\mathcal{P}(E)) \to \mathcal{P}(\mathcal{P}(E))$ be defined by*

$$F_E(M) = \{\hat{X} \in \mathcal{P}(E) \mid \forall X \in M : X \cap \hat{X} \neq \emptyset\}.$$

Define $\mathcal{F} : \tilde{M} \to \tilde{M}$, where $\tilde{M} = \{(E, \succ, \mapsto, T, l) \mid T \subseteq \mathcal{P}(E) \wedge \mathrm{dom}(l) = E \wedge \ \succ, \mapsto \subseteq \mathcal{P}(E) \times E\}$, by

$$\mathcal{F}(E, \succ, \mapsto, T, l) = (E, \{(\hat{Z}, e) \mid \hat{Z} \in F_E(\_ \succ e)\}, \{(\hat{X}, e) \mid \hat{X} \in F_E(\_ \mapsto e)\}, F_E(T), l).$$

Example 5.38 *The transformation of $\mathcal{E}_3$ of Figure 5.1 is $\hat{\mathcal{E}}_3$ of Figure 5.3, i.e. $\mathcal{F}(\mathcal{E}_3) = \hat{\mathcal{E}}_3$. The transformation of $\hat{\mathcal{E}}_3$ yields $\mathcal{E}_3$ except that all upper sets are included in the conflict relation (respectively in the causality relation and the termination set), for example $\{a, b, c\}$ is contained in the termination set.*

Proposition 5.39 *Function $\mathcal{F} \upharpoonright$ **ETBES** is a continuous function from $(\mathbf{ETBES}, \trianglelefteq)$ into $(\mathbf{ETPES}, \hat{\trianglelefteq})$ and function $\mathcal{F} \upharpoonright$ **ETPES** is a continuous function from $(\mathbf{ETPES}, \hat{\trianglelefteq})$ into $(\mathbf{ETBES}, \trianglelefteq)$.*

Proof: The proof is given in Subsection 5.6.3. □

Theorem 5.40 *Every set of event traces that is described by* **ETBES** *is also described by* **ETPES** *and vice versa. More precisely, for all $\mathcal{E} \in$ **ETBES** it holds that $\mathcal{E}$ and $\mathcal{F}(\mathcal{E})$ describe the same set of event traces, and for all $\hat{\mathcal{E}} \in$ **ETPES** it holds that $\hat{\mathcal{E}}$ and $\mathcal{F}(\hat{\mathcal{E}})$ describe the same set of event traces.*

Proof: The proof is given in Subsection 5.6.4. □

Corollary 5.41 *Every set of event traces that is described by prime, flow, stable, bundle, extended bundle or dual event structures is also described by extended termination precursor event structures, but not vice versa.*

Proof: Is an immediate consequence of Theorem 5.15 and Theorem 5.40. □

Remark 5.42 *Of course it is possible to use* $\mathbf{ETPES}$ *instead of* $\mathbf{ETBES}$ *as a model of the denotational semantics for our process algebra. The necessary operators can be defined explicitly, as it is done in Subsection 5.4.3 for* $\mathbf{ETBES}$*. Another possibility is to define the operators on* $\mathbf{ETPES}$ *through the operators on* $\mathbf{ETBES}$*, i.e. define, for example, the parallel operator* $\widetilde{\|}_A$ *on* $\mathbf{ETPES}$ *by* $\mathcal{E}\widetilde{\|}_A\mathcal{E}' = \mathcal{F}(\mathcal{F}(\mathcal{E})\widehat{\|}_A\mathcal{F}(\mathcal{E}'))$*. These operators are continuous by Proposition 5.39 and Lemma 5.22. The denotational semantics that is obtained in this way is illustrated in Figure 5.3.*

5.5 Discussion

In this chapter, we have investigated new kinds of event structures in order to give denotational semantics to process algebras that are based on the fa-philosophy and that contain disruption. The motivation of such an approach results from the fact that it is more reasonable to have an fa-philosophy in end-based settings, since otherwise the intuitive equivalences fail to be the coarsest (Subsection 4.2.6). Note that it is possible to model disruption with action refinement in end-based settings.

Sets of events may disable events in both event structures presented in this chapter. One of these event structures is based on the bundle technique the other one is based on Winskel's approach. We have shown that these two event structures are equivalent approaches. Furthermore, we have used one of them to give a denotational semantics to a process algebra that is based on the fa-philosophy and that contains disruption. Moreover, we have shown that this denotational semantics is consistent with the standard operational semantics.

In the following chapter, we define the action refinement operator on $\mathbf{ETBES}$ with respect to the end-based view. In addition, we adapt two of the newly introduced equivalences of Chapter 4 to $\mathbf{ETBES}$ and show that they yield the coarsest equivalences with respect to trace (respectively bisimulation) equivalence for the end-based action refinement operator.

5.6 Proofs

5.6.1 Proof of Theorem 5.28.

The proof is analogous to the proof of Theorem 3.25, i.e. we introduce an event based transition relation. Then we show that this transition system is bisimilar to $(\mathsf{EXP}_{\mathsf{st}}, \mathcal{A}ct^T, \longrightarrow^t_{\mathrm{decl}}, B)$ and that it is, in addition, bisimilar to $(\mathbf{ETBES}, \mathcal{A}ct^T, \hookrightarrow, [\![\langle \mathrm{decl}, B\rangle]\!])$. Hence, Theorem 5.28 follows by the transitivity of bisimilarity.

Event Based Transition System.

Let $\mathsf{EXP}^{\mathsf{e}}_{\mathsf{st}}$ be the set that contains exactly those elements generated by

$$C \;::=\; B \mid C;B \mid C\,[>B \mid C\|_A C \mid C[f] \mid C\backslash\backslash A \mid \lceil C\rceil_i$$

where $B \in \mathsf{EXP}_{\mathsf{st}}$, $f \in \mathcal{F}_L$, $i \in \{1,2,l,r\}$ and $A \subseteq \mathsf{Obs}$. We do not need to extend the declaration, i.e. we define $\mathsf{PA}^{\mathsf{e}}_{\mathsf{st}} = (\mathsf{Var} \to \mathsf{EXP}_{\mathsf{st}}) \times \mathsf{EXP}^{\mathsf{e}}_{\mathsf{st}}$.

In Table 5.2, the event transition rules $\longrightarrow'_{\mathrm{decl}} \subseteq \mathsf{EXP}^{\mathsf{e}}_{\mathsf{st}} \times (\mathcal{A}ct^T \times \mathcal{U}) \times \mathsf{EXP}^{\mathsf{e}}_{\mathsf{st}}$ are presented.

The First Bisimilarity Result.

An expression C of $\mathsf{EXP}^{\mathsf{e}}_{\mathsf{st}}$ and an expression B of $\mathsf{EXP}_{\mathsf{st}}$ are related if we obtain B by removing all $\lceil\_\rceil$ expressions from C. This is formalized by the following function, where we also count the $\lceil\_\rceil$ symbols in C.

Definition 5.43 $\Xi : \mathbb{N} \times \mathsf{EXP}_{\mathsf{st}} \to \mathcal{P}(\mathsf{EXP}^{\mathsf{e}}_{\mathsf{st}})$ *is defined as follows, where* $I = \{1,2,l,r\}$.

$$\begin{array}{l}
\Xi(0,B) = \{B\} \\
\Xi(n+1,B) = \{\lceil\tilde{C}\rceil_i \mid i \in I \wedge \tilde{C} \in \Xi(n,B)\} \qquad \textit{if } B \in \{\mathbf{0}, a, B_1 + B_2, x\} \\
\Xi(n+1,B_1;B_2) = \{\lceil\tilde{C}\rceil_i \mid i \in I \wedge \tilde{C} \in \Xi(n,B_1;B_2)\} \cup \\
\quad \{C_1;B_2 \mid C_1 \in \Xi(n+1,B_1)\} \\
\Xi(n+1,B_1\,[>B_2) = \{\lceil\tilde{C}\rceil_i \mid i \in I \wedge \tilde{C} \in \Xi(n,B_1\,[>B_2)\} \cup \\
\quad \{C_1\,[>B_2 \mid C_1 \in \Xi(n+1,B_1)\} \\
\Xi(n+1,B_1\|_A B_2) = \{\lceil\tilde{C}\rceil_i \mid i \in I \wedge \tilde{C} \in \Xi(n,B_1\|_A B_2)\} \cup \\
\quad \{C_1\|_A C_2 \mid \exists m \in \mathbb{N} : m \le n+1 \wedge C_1 \in \Xi(m,B_1) \wedge C_2 \in \Xi(n+1-m,B_2)\} \\
\Xi(n+1,B[f]) = \{\lceil\tilde{C}\rceil_i \mid i \in I \wedge \tilde{C} \in \Xi(n,B[f])\} \cup \{C[f] \mid C \in \Xi(n+1,B)\} \\
\Xi(n+1,B\backslash\backslash A) = \{\lceil\tilde{C}\rceil_i \mid i \in I \wedge \tilde{C} \in \Xi(n,B\backslash\backslash A)\} \cup \{C\backslash\backslash A \mid C \in \Xi(n+1,B)\}
\end{array}$$

The well-definedness of Ξ is easily seen.

Lemma 5.44 *Let* $B \in \mathsf{EXP}_{\mathsf{st}}$, *then* $(\mathsf{EXP}_{\mathsf{st}}, \mathcal{A}ct^T, \longrightarrow^t_{\mathrm{decl}}, B)$ *and* $(\mathsf{EXP}^{\mathsf{e}}_{\mathsf{st}}, \mathcal{A}ct^T, \longrightarrow'', B)$ *are bisimilar, where* $C \overset{\gamma}{\longrightarrow}'' C' \Leftrightarrow \exists e \in \mathcal{U} : C \overset{\gamma}{\underset{e}{\longrightarrow}}'_{\mathrm{decl}} C'$.

Proof: Define $\mathcal{R} = \{(B,C) \in \mathsf{EXP}_{\mathsf{st}} \times \mathsf{EXP}^{\mathsf{e}}_{\mathsf{st}} \mid \exists n : C \in \Xi(n,B)\}$. In order to verify that $\mathcal{R}$ is a bisimulation, we show

$$(B \overset{\gamma}{\longrightarrow}^t B' \wedge C \in \Xi(n,B)) \Rightarrow \exists e, C', m : C \overset{\gamma}{\underset{e}{\longrightarrow}}'_{\mathrm{decl}} C' \wedge C' \in \Xi(m,B') \qquad (5.1)$$

The proof of (5.1) works by induction on the depth of inference of $B \overset{\gamma}{\longrightarrow}^t B'$ combined with the value of n. Then (5.1) can be easily checked through the following procedure:

- applying rule N_{12} or N_{rl} whenever $C = \lceil\tilde{C}\rceil_i$. In these cases, n is reduced by one and $B \overset{\gamma}{\longrightarrow}^t B'$ remains unaffected. Therefore, the hypothesis yields the result.

In the following let γ be an element of $\mathcal{A}ct^T$ and a be an element of $\mathcal{A}ct$

$$A_1 : \frac{}{a \xrightarrow{\frac{a\surd}{\bullet}} \mathbf{0}} \qquad C : \frac{B_1 \xrightarrow{\frac{\gamma}{e}} C'}{\begin{array}{l} B_1 + B_2 \xrightarrow{\frac{\gamma}{(\star_1,e)}} \lceil C' \rceil_1 \\ B_2 + B_1 \xrightarrow{\frac{\gamma}{(\star_2,e)}} \lceil C' \rceil_2 \end{array}}$$

$$S_1 : \frac{C \xrightarrow{\frac{a}{e}} C'}{C; B \xrightarrow{\frac{a}{(\star_1,e)}} C'; B} \qquad S_2 : \frac{C \xrightarrow{\frac{a\surd}{e}} C'}{C; B \xrightarrow{\frac{a}{(\star_1,e)}} \lceil B \rceil_2}$$

$$I_1 : \frac{C \xrightarrow{\frac{a}{e}} C'}{C \,[> B \xrightarrow{\frac{a}{(\star_1,e)}} C' \,[> B} \qquad I_2 : \frac{C \xrightarrow{\frac{a\surd}{e}} C'}{C \,[> B \xrightarrow{\frac{a\surd}{(\star_1,e)}} \lceil C' \rceil_1} \qquad I_3 : \frac{B \xrightarrow{\frac{\gamma}{e}} C'}{C \,[> B \xrightarrow{\frac{\gamma}{(\star_2,e)}} \lceil C' \rceil_2}$$

$$P_1 : \frac{C_1 \xrightarrow{\frac{a}{e}} C_1' \quad a \notin A}{\begin{array}{l} C_1 \|_A C_2 \xrightarrow{\frac{a}{(e,\star)}} C_1' \|_A C_2 \\ C_2 \|_A C_1 \xrightarrow{\frac{a}{(\star,e)}} C_2 \|_A C_1' \end{array}} \qquad P_2 : \frac{C_1 \xrightarrow{\frac{a\surd}{e}} C_1' \quad a \notin A}{\begin{array}{l} C_1 \|_A C_2 \xrightarrow{\frac{a}{(e,\star)}} (\lceil C_2 \rceil_r) \backslash\backslash A \\ C_2 \|_A C_1 \xrightarrow{\frac{a}{(\star,e)}} (\lceil C_2 \rceil_l) \backslash\backslash A \end{array}}$$

$$P_3 : \frac{C_1 \xrightarrow{\frac{a}{e_1}} C_1' \quad C_2 \xrightarrow{\frac{a}{e_2}} C_2' \quad a \in A}{C_1 \|_A C_2 \xrightarrow{\frac{a}{(e_1,e_2)}} C_1' \|_A C_2'} \qquad P_4 : \frac{C_1 \xrightarrow{\frac{a\surd}{e_1}} C_1' \quad C_2 \xrightarrow{\frac{a}{e_2}} C_2' \quad a \in A}{\begin{array}{l} C_1 \|_A C_2 \xrightarrow{\frac{a}{(e_1,e_2)}} (\lceil C_2' \rceil_r) \backslash\backslash A \\ C_2 \|_A C_1 \xrightarrow{\frac{a}{(e_1,e_2)}} (\lceil C_2' \rceil_l) \backslash\backslash A \end{array}}$$

$$P_5 : \frac{C_1 \xrightarrow{\frac{a\surd}{e_1}} C_1' \quad C_2 \xrightarrow{\frac{a\surd}{e_2}} C_2' \quad a \in A}{C_1 \|_A C_2 \xrightarrow{\frac{a\surd}{(e_1,e_2)}} \mathbf{0}}$$

$$Lab_1 : \frac{C \xrightarrow{\frac{a}{e}} C'}{C[f] \xrightarrow{\frac{f(a)}{e}} C'[f]} \qquad Lab_2 : \frac{C \xrightarrow{\frac{a\surd}{e}} C'}{C[f] \xrightarrow{\frac{f(a)\surd}{e}} C'[f]}$$

$$Res_1 : \frac{C \xrightarrow{\frac{a}{e}} C' \quad a \notin A}{C\backslash\backslash A \xrightarrow{\frac{a}{e}} C'\backslash\backslash A} \qquad Res_2 : \frac{C \xrightarrow{\frac{a\surd}{e}} C' \quad a \notin A}{C\backslash\backslash A \xrightarrow{\frac{a\surd}{e}} C'\backslash\backslash A}$$

$$Rec : \frac{\mathrm{decl}(x) \xrightarrow{\frac{\gamma}{e}} C'}{x \xrightarrow{\frac{\gamma}{e}} C'} \qquad N_{12} : \frac{C \xrightarrow{\frac{\gamma}{e}} C'}{\lceil C \rceil_i \xrightarrow{\frac{\gamma}{(\star_i,e)}} \lceil C' \rceil_i} \qquad N_{rl} : \frac{C \xrightarrow{\frac{\gamma}{e}} C'}{\begin{array}{l} \lceil C \rceil_r \xrightarrow{\frac{\gamma}{(\star,e)}} \lceil C' \rceil_r \\ \lceil C \rceil_l \xrightarrow{\frac{\gamma}{(e,\star)}} \lceil C' \rceil_l \end{array}}$$

Table 5.2: Event Based Transition Rules with respect to $\longrightarrow^t_{\mathrm{decl}}$

- applying the corresponding rules of $\xrightarrow{\gamma}{}^t\ B'$ whenever C is different to $\lceil \tilde{C} \rceil_i$. In these cases, the depth of inference is reduced and n gets not increased. Therefore, the hypothesis yields the result.

Another fact is

$$(C \xrightarrow[e]{\gamma}'_{\mathrm{decl}} C' \wedge C \in \Xi(n, B)) \Rightarrow \exists B', m : B \xrightarrow{\gamma}{}^{t} B' \wedge C' \in \Xi(m, B') \qquad (5.2)$$

This equation can be proved by induction on the depth of inference of $C \xrightarrow[e]{\gamma}'_{\mathrm{decl}} C'$.
Now we are ready to verify that $\mathcal{R}$ is a bisimulation:

- It is clear that $(B, B) \in \mathcal{R}$.
- Suppose $(B_1, C_1) \in \mathcal{R}$ and $B_1 \xrightarrow{\gamma}{}^{t} B_2$. Then $\exists e, C_2, m : C_1 \xrightarrow[e]{\gamma}'_{\mathrm{decl}} C_2 \wedge C_2 \in \Xi(m, B_2)$ by (5.1). Thus $C_1 \xrightarrow{\gamma}{}'' C_2$ and $(B_2, C_2) \in \mathcal{R}$, as required.
- Suppose $(B_1, C_1) \in \mathcal{R}$ and $C_1 \xrightarrow{\gamma}{}'' C_2$. Then $C_1 \xrightarrow[e]{\gamma}'_{\mathrm{decl}} C_2$ for some e. Hence, $\exists B_2, m : B_1 \xrightarrow{\gamma}{}^{t} B_2 \wedge C_2 \in \Xi(m, B_2)$ by (5.2). □

The Second Bisimilarity Result.

First, we show that the denotation of a variable is the same as the denotation of its corresponding expression.

Lemma 5.45 *Let* $\mathrm{decl} : \mathsf{Var} \to \mathsf{EXP}_{\mathsf{st}}$ *and* $x \in \mathsf{Var}$. *Then* $[\![\langle \mathrm{decl}, x\rangle]\!] = [\![\langle \mathrm{decl}, \mathrm{decl}(x)\rangle]\!]$.

Proof: Similar to the proof of Lemma 3.37. □

We extend the denotational semantics to $\mathsf{PA}^{\mathsf{e}}_{\mathsf{st}}$.

Definition 5.46 (Denotational semantics of $\mathsf{PA}^{\mathsf{e}}_{\mathsf{st}}$) *Let* $i \in \{1, 2\}$ *then*
$\widehat{Shift}_i : \mathbf{ETBES} \to \mathbf{ETBES}$ *with* $\widehat{Shift}_i(\mathcal{E}) = (\tilde{E}, \tilde{\succ}, \tilde{\mapsto}, \tilde{T}, \tilde{l})$ *where*

$$\begin{array}{rcl} \tilde{E} & = & \{\star_i\} \times E \\ \tilde{\succ} & = & \{(\{\star_i\} \times Z, (\star_i, e)) \mid Z \succ e\} \\ \tilde{\mapsto} & = & \{(\{\star_i\} \times X, (\star_i, e)) \mid X \mapsto_i e\} \\ \tilde{T} & = & \{\{\star_i\} \times X \mid X \in T\} \\ \tilde{l}(\star_i, e) = l(e) & & \end{array}$$

$\widehat{Shift}_r : \mathbf{ETBES} \to \mathbf{ETBES}$ *with* $\widehat{Shift}_r(\mathcal{E}) = (\tilde{E}, \tilde{\succ}, \tilde{\mapsto}, \tilde{T}, \tilde{l})$ *where*

$$\begin{array}{rcl} \tilde{E} & = & \{\star\} \times E \\ \tilde{\succ} & = & \{(\{\star\} \times Z, (\star, e)) \mid Z \succ e\} \\ \tilde{\mapsto} & = & \{(\{\star\} \times X, (\star, e)) \mid X \mapsto_i e\} \\ \tilde{T} & = & \{\{\star\} \times X \in T\} \\ \tilde{l}(\star, e) = l(e) & & \end{array}$$

$\widehat{Shift}_l : \mathbf{ETBES} \to \mathbf{ETBES}$ *with* $\widehat{Shift}_l(\mathcal{E}) = (\tilde{E}, \tilde{\succ}, \tilde{\mapsto}, \tilde{T}, \tilde{l})$ *where*

$$\begin{array}{rcl} \tilde{E} & = & E \times \{\star\} \\ \tilde{\succ} & = & \{(Z \times \{\star\}, (e, \star)) \mid Z \succ e\} \\ \tilde{\mapsto} & = & \{(X \times \{\star\}, (e, \star)) \mid X \mapsto_i e\} \\ \tilde{T} & = & \{X \times \{\star\} \mid X \in T\} \\ \tilde{l}(e, \star) = l(e) & & \end{array}$$

Furthermore, define $[\![\_]\!]' : \mathsf{PA}^{\mathsf{e}}_{\mathsf{st}} \to \mathbf{ETBES}$ *by*

$$[\![\langle \mathrm{decl}, B\rangle]\!]' = [\![\langle \mathrm{decl}, B\rangle]\!] \qquad [\![\langle \mathrm{decl}, C; B\rangle]\!]' = [\![\langle \mathrm{decl}, C\rangle]\!]' \widehat{;} [\![\langle \mathrm{decl}, B\rangle]\!]'$$
$$[\![\langle \mathrm{decl}, C\,[\!>B\rangle]\!]' = [\![\langle \mathrm{decl}, C\rangle]\!]' \widehat{[\!>} [\![\langle \mathrm{decl}, B\rangle]\!]'$$
$$[\![\langle \mathrm{decl}, C_1 \|_A C_2\rangle]\!]' = [\![\langle \mathrm{decl}, C_1\rangle]\!]' \widehat{\|}_A [\![\langle \mathrm{decl}, C_2\rangle]\!]'$$
$$[\![\langle \mathrm{decl}, C[f]\rangle]\!]' = \widehat{\mathrm{Lab}}([\![\langle \mathrm{decl}, C\rangle]\!]', f) \qquad [\![\langle \mathrm{decl}, C\backslash\backslash A\rangle]\!]' = [\![\langle \mathrm{decl}, C_2\rangle]\!]' \widehat{\backslash\backslash} A$$
$$[\![\langle \mathrm{decl}, \lceil C\rceil_i\rangle]\!]' = \widehat{Shift}_i([\![\langle \mathrm{decl}, C\rangle]\!]')$$

It is easy to check that $[\![\_]\!]'$ is well defined.

Lemma 5.47 *Suppose* $\mathcal{E}, \mathcal{E}_1, \mathcal{E}_2 \in \mathbf{ETBES}$. *Then*

$$(\mathcal{E}_1 \widehat{+} \mathcal{E}_2)_{[(\star_i, e)]} \simeq \widehat{Shift}_i(\mathcal{E}_{i[e]})$$

$$(\mathcal{E}_1 \widehat{;} \mathcal{E}_2)_{[(\star_1, e)]} \simeq \begin{cases} \widehat{Shift}_2(\mathcal{E}_2) & \text{if } e \in \mathrm{init}(\mathcal{E}_1) \wedge \Upsilon(T_1, e) \\ \mathcal{E}_{1[e]} \widehat{;} \mathcal{E}_2 & \text{otherwise} \end{cases}$$

$$(\mathcal{E}_1 \widehat{[\!>} \mathcal{E}_2)_{[(\star_i, e)]} \simeq \begin{cases} \widehat{Shift}_2(\mathcal{E}_{2[e]}) & \text{if } i = 2 \\ \widehat{Shift}_1(\mathcal{E}_{1[e]}) & \text{if } i = 1 \wedge \Upsilon(T_1, e) \\ \mathcal{E}_{1[e]} \widehat{[\!>} \mathcal{E}_2 & \text{otherwise} \end{cases}$$

$$(\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2)_{[(e_1, \star)]} \simeq \begin{cases} \mathcal{E}_{1[e_1]} \widehat{\|}_A \mathcal{E}_2 & \text{if } \neg\Upsilon(T_1, e_1) \wedge l_1(e_1) \notin A \\ \widehat{Shift}_r(\mathcal{E}_2) \widehat{\backslash\backslash} A & \text{if } e_1 \in \mathrm{init}(\mathcal{E}_1) \wedge \Upsilon(T_1, e_1) \wedge l_1(e_1) \notin A \end{cases}$$

$$(\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2)_{[(\star, e_2)]} \simeq \begin{cases} \mathcal{E}_1 \widehat{\|}_A \mathcal{E}_{2[e_2]} & \text{if } \neg\Upsilon(T_2, e_2) \wedge l_2(e_2) \notin A \\ \widehat{Shift}_l(\mathcal{E}_1) \widehat{\backslash\backslash} A & \text{if } e_2 \in \mathrm{init}(\mathcal{E}_2) \wedge \Upsilon(T_2, e_2) \wedge l_2(e_2) \notin A \end{cases}$$

$$(\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2)_{[(e_1, e_2)]} \simeq \begin{cases} \mathcal{E}_{1[e_1]} \widehat{\|}_A \mathcal{E}_{2[e_2]} & \text{if } \neg\Upsilon(T_1, e_1) \wedge \neg\Upsilon(T_2, e_2) \\ \widehat{Shift}_r(\mathcal{E}_{2[e_2]}) \widehat{\backslash\backslash} A & \text{if } e_1 \in \mathrm{init}(\mathcal{E}_1) \wedge \Upsilon(T_1, e_1) \wedge \neg\Upsilon(T_2, e_2) \\ \widehat{Shift}_l(\mathcal{E}_{1[e_1]}) \widehat{\backslash\backslash} A & \text{if } e_2 \in \mathrm{init}(\mathcal{E}_2) \wedge \Upsilon(T_2, e_2) \wedge \neg\Upsilon(T_1, e_1) \\ (\emptyset, \emptyset, \emptyset, \{\emptyset\}, \emptyset) & \text{if } \Upsilon(T_1, e_1) \wedge \Upsilon(T_2, e_2) \end{cases}$$
whenever $l_1(e_1) = l_2(e_2) \in A$

$$\widehat{\mathrm{Lab}}(\mathcal{E}, f)_{[e]} \simeq \widehat{\mathrm{Lab}}(\mathcal{E}_{[e]}, f)$$

$$(\mathcal{E} \widehat{\backslash\backslash} A)_{[e]} \simeq \begin{cases} \mathcal{E}_{[e]} \widehat{\backslash\backslash} A & \text{if } l(e) \notin A \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\widehat{Shift}_i(\mathcal{E})_{[(\star_i, e)]} \simeq \widehat{Shift}_i(\mathcal{E}_{[e]}) \quad \text{whenever } i \in \{1, 2\}$$

$$\widehat{Shift}_r(\mathcal{E})_{[(\star, e)]} \simeq \widehat{Shift}_r(\mathcal{E}_{[e]})$$

$$\widehat{Shift}_l(\mathcal{E})_{[(e, \star)]} \simeq \widehat{Shift}_l(\mathcal{E}_{[e]})$$

Proof: Straightforward. □

Lemma 5.48 *Suppose* $\langle \mathrm{decl}, C\rangle \in \mathsf{PA}^{\mathsf{e}}_{\mathsf{st}}$ *and* $C \xrightarrow{\gamma}{}'_{\mathrm{decl}} C'$. *Then*

$$e \in \mathrm{init}(\mathcal{E}) \wedge \mathcal{E}' = \mathcal{E}_{[e]} \wedge (\gamma \notin \mathcal{A}ct \Leftrightarrow \Upsilon(T, e)) \wedge l(e) = \begin{cases} \gamma & \text{if } \gamma \in \mathcal{A}ct \\ a & \text{if } \gamma = a\surd \end{cases}$$

with $\mathcal{E} = [\![\langle \mathrm{decl}, C\rangle]\!]'$ *and* $\mathcal{E}' = [\![\langle \mathrm{decl}, C'\rangle]\!]'$.

Proof: We use induction on the depth of inference of $C \xrightarrow[e]{\gamma}{}'_{\mathrm{decl}} C'$. Then the equation can be verified by case analysis on the derivation rules, where Lemma 5.47 is used. In the case of Rec' we make use of Lemma 5.45. □

Lemma 5.49 *Let* $\langle \mathrm{decl}, C\rangle \in \mathsf{PA}^{\mathsf{e}}_{\mathsf{st}}$, $\mathcal{E} = [\![\langle \mathrm{decl}, C\rangle]\!]'$ *and* $e \in \mathrm{init}(\mathcal{E})$. *Then*

$$\exists C' \in \mathsf{EXP}^{\mathsf{e}}_{\mathsf{st}} : C \xrightarrow[e]{\gamma}{}'_{\mathrm{decl}} C' \wedge \gamma = \begin{cases} l(e) & \text{if } \neg\Upsilon(T,e) \\ l(e)\surd & \text{if } \Upsilon(T,e) \end{cases}$$

Proof: First we show for any $\mathrm{decl} : \mathsf{Var} \to \mathsf{EXP}_{\mathsf{st}}$:

$$\begin{aligned} \forall n \in \mathbb{N} : \forall B \in \mathsf{EXP}_{\mathsf{st}} : e \in \mathrm{init}([\![B]\!]_{\mathcal{F}^n_{\mathrm{decl}}(\perp)}) \Rightarrow \Big(\exists C' \in \mathsf{EXP}^{\mathsf{e}}_{\mathsf{st}} : B \xrightarrow[e]{\gamma}{}' C' \wedge \\ \gamma = \begin{cases} \pi_5([\![B]\!]_{\mathcal{F}^n_{\mathrm{decl}}(\perp)})(e) & \text{if } \neg\Upsilon(\pi_4([\![B]\!]_{\mathcal{F}^n_{\mathrm{decl}}(\perp)}), e) \\ \pi_5([\![B]\!]_{\mathcal{F}^n_{\mathrm{decl}}(\perp)})(e)\surd & \text{if } \Upsilon(\pi_4([\![B]\!]_{\mathcal{F}^n_{\mathrm{decl}}(\perp)}), e) \end{cases} \Big) \end{aligned} \tag{5.3}$$

This is done by induction on n combined with the structure of B where the lexicographical order is used. Furthermore, a case analysis on the structure of B is used. We only present here the case $B = x$: $e \in \mathrm{init}([\![x]\!]_{\mathcal{F}^n_{\mathrm{decl}}(\perp)})$ implies that $n > 0$. Therefore, $[\![x]\!]_{\mathcal{F}^n_{\mathrm{decl}}(\perp)} = \mathcal{F}^n_{\mathrm{decl}}(\perp)(x) = [\![\mathrm{decl}(x)]\!]_{\mathcal{F}^{n-1}_{\mathrm{decl}}(\perp)}$. The rest follows by induction, since n is reduced. Thus (5.3) is established.

The main statement follows now by structural induction on C. We only present the case $C = B \in \mathsf{EXP}_{\mathsf{st}}$. By Remark 5.25 we get $[\![\langle \mathrm{decl}, B\rangle]\!] = \bigsqcup_n [\![B]\!]_{\mathcal{F}^n_{\mathrm{decl}}(\perp)}$. Then it is easily seen that there is m such that $e \in \mathrm{init}([\![B]\!]_{\mathcal{F}^m_{\mathrm{decl}}(\perp)})$ and $\gamma = \pi_5([\![B]\!]_{\mathcal{F}^m_{\mathrm{decl}}(\perp)})$. And so the result follows by (5.3). □

Lemma 5.50 *Let* $\langle \mathrm{decl}, C\rangle \in \mathsf{PA}^{\mathsf{e}}_{\mathsf{st}}$, *then the transition systems* $(\mathsf{EXP}^{\mathsf{e}}_{\mathsf{st}}, \mathcal{A}ct^T, \longrightarrow'', C)$ *and* $(\mathbf{ETBES}, \mathcal{A}ct^T, \hookrightarrow, [\![\langle \mathrm{decl}, C\rangle]\!]')$ *are bisimilar, where* $\longrightarrow''$ *is defined as in Lemma 5.44.*

Proof: Define $\mathcal{R} = \{(C', [\![\langle \mathrm{decl}, C'\rangle]\!]') \mid C' \in \mathsf{EXP}^{\mathsf{e}}_{\mathsf{st}}\}$. Then $(C, [\![\langle \mathrm{decl}, C\rangle]\!]') \in \mathcal{R}$ by definition.

Suppose $C_1 \in \mathsf{EXP}^{\mathsf{e}}_{\mathsf{st}}$ and $C_1 \xrightarrow{\gamma}{}'' C_2$. Then $C_1 \xrightarrow[e]{\gamma}{}'_{\mathrm{decl}} C_2$ for some e. Hence, by Lemma 5.48 we get $[\![\langle \mathrm{decl}, C_1\rangle]\!]' \xhookrightarrow{\gamma} [\![\langle \mathrm{decl}, C_2\rangle]\!]'$, as required.

Suppose $C_1 \in \mathsf{EXP}^{\mathsf{e}}_{\mathsf{st}}$ and $[\![\langle \mathrm{decl}, C_1\rangle]\!]' \xhookrightarrow{\gamma} \mathcal{E}_2$. Then there is $e \in \mathrm{init}([\![\langle \mathrm{decl}, C_1\rangle]\!]')$ such that $\mathcal{E}_2 = [\![\langle \mathrm{decl}, C_1\rangle]\!]'_{[e]}$ and $\gamma = \begin{cases} l(e) & \text{if } \neg\Upsilon(T,e) \\ l(e)\surd & \text{if } \Upsilon(T,e) \end{cases}$. From Lemma 5.49 we get the existence of $C_2 \in \mathsf{EXP}^{\mathsf{e}}_{\mathsf{st}}$ such that $C_1 \xrightarrow[e]{\gamma}{}'_{\mathrm{decl}} C_2$. Moreover, $[\![\langle \mathrm{decl}, C_1\rangle]\!]'_{[e]} = [\![\langle \mathrm{decl}, C_2\rangle]\!]'$ by Lemma 5.48, which concludes the proof. □

5.6.2 Proof of Theorem 5.36.

It is easily seen that $\hat{\trianglelefteq}$ is a partial order with $(\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$ as its least element. Furthermore, $\hat{\bigsqcup}_i \hat{\mathcal{E}}_i$ is an eTpes. In the following, we only consider $\hat{T}$. The cases $\hat{\succ}$ and $\hat{\mapsto}$ follow analogously.

Upper bound: Obviously, $\hat{T}_j \subseteq \{X \in \hat{T} \mid X \subseteq \hat{E}_j\}$.

Let $X \subseteq \bigcup_i \hat{E}_i$ such that $X \subseteq \hat{E}_j$ and $\exists i : (X \cap \hat{E}_i) \in \hat{T}_i$.

If $i \geq j$ then $\hat{\mathcal{E}}_j \hat{\trianglelefteq} \hat{\mathcal{E}}_i$. Thus $X \cap \hat{E}_i = X$, hence $X \in \hat{T}_i$. Moreover, $X \in \hat{T}_j$, since $\hat{\mathcal{E}}_j \hat{\trianglelefteq} \hat{\mathcal{E}}_i$.

If $i < j$ then $\hat{\mathcal{E}}_i \hat{\trianglelefteq} \hat{\mathcal{E}}_j$. Thus $(X \cap \hat{E}_i) \in \hat{T}_j$. Since $\hat{T}_j$ is finitely determined, we get $X = (X \cap \hat{E}_j) \in \hat{T}_j$.

Least upper bound: Let $\hat{\mathfrak{E}}'$ be an eTpes such that $\hat{\mathfrak{E}}_i \hat{\trianglelefteq} \hat{\mathfrak{E}}'$ for all $i \in \mathbb{N}$. Then $\bigcup_i \hat{E}_i \subseteq \hat{E}'$.

Let $X \in \hat{T}$. Then $\exists j : (X \cap \hat{E}_j) \in \hat{T}_j$. Hence, $(X \cap \hat{E}_j) \in \hat{T}'$, since $\hat{\mathfrak{E}}_j \hat{\trianglelefteq} \hat{\mathfrak{E}}'$. Thus $X \in \hat{T}'$, since $\hat{T}'$ is finitely determined.

Let $X \in \hat{T}'$ such that $X \subseteq \bigcup_i \hat{E}_i$. Since $\hat{T}'$ is finitely determined, there is $X' \in \hat{T}'$ such that $X' \subseteq X \wedge |X'| < |\mathbb{N}|$. Therefore, there exists $j \in \mathbb{N}$ such that $X' \subseteq \hat{E}_j$. Hence, $X' \in \hat{T}_j$. Then by definition $X' \in \hat{T}$. Thus $X \in \hat{T}$, since $\hat{T}$ is finitely determined.

Hence, Theorem 5.36 is established.

5.6.3 Proof of Proposition 5.39.

The following lemmas show that the constraints on eTbes are transformed into the constraints on eTpes and vice versa.

Lemma 5.51 *We have*

(i) $M \neq \emptyset \iff \emptyset \notin F_E(M)$

(ii) $(\forall X \in M : e \in X) \iff \{e\} \in F_E(M)$

(iii) $\emptyset \notin \hat{M} \iff F_E(\hat{M}) \neq \emptyset$

(iv) $\{e\} \in \hat{M} \Rightarrow (\forall X \in F_e(\hat{M}) : e \in X)$

(v) $(\{e\} \notin \hat{M} \wedge \emptyset \notin \hat{M}) \Rightarrow (\exists X \in F_e(\hat{M}) : e \notin X)$

Proof: Straightforward. □

Lemma 5.52 *If M is approximation closed with respect to E then $F_E(M)$ is finitely determined with respect to E.*

Proof: Suppose $\hat{X} \in F_E(M)$. From the definition of F_E it follows that all upper sets of $\hat{X}$ are in $F_E(M)$. Now suppose $|\hat{X}| = |\mathbb{N}|$. Let $\hat{X} = \{e_i \mid i \in \mathbb{N}\}$. Define $\hat{X}' = \{e_i \mid \exists X \in M : e_i \in X \wedge \forall j < i : e_j \notin X\}$. Then $\hat{X}' \subseteq \hat{X}$ and $\hat{X}' \in F_E(M)$.

Assume $|\hat{X}'| = |\mathbb{N}|$. Then for all $e_i \in \hat{X}'$ there exists $X_i \in M$ such that $e_i \in X_i$ and $\{e_1, ..., e_{i-1}\} \cap X_i = \emptyset$. Let $\dot{X} = \mathcal{X}((X_i)_{e_i \in \hat{X}'}, \kappa, (X_i)_{e_i \in \hat{X}'}, \kappa)$, where $\kappa : \mathbb{N} \to E$ be bijective. Then $\dot{X}$ only contains elements that are contained infinitely often in $(X_i)_{e_i \in \hat{X}'}$ by the definition of $\mathcal{X}$ (Definition 2.16. Furthermore, $\dot{X} \in M$ by Proposition 2.17. Therefore, $\forall i : e_i \notin \dot{X}$, since e_i only appears finitely often in $(X_i)_{e_i \in \hat{X}'}$. Hence, $\dot{X} \cap \hat{X} = \emptyset$, which contradicts $\hat{X} \in F_E(M)$. Thus $|\hat{X}'| < |\mathbb{N}|$ as required. □

Lemma 5.53 *If $\hat{M}$ is finitely determined with respect to E then $F_E(\hat{M})$ is approximation closed with respect to E.*

Proof: Suppose $X \subseteq E$ and $(E_i)_{i\in\mathbb{N}}$ is a finite, monotone approximation of E such that

$$\forall k \in \mathbb{N} : \exists X_k : X_k \cap E_k = X \cap E_k \wedge (\forall \hat{X} \in \hat{M} : X_k \cap \hat{X} \neq \emptyset). \tag{5.4}$$

Let $\hat{X} \in \hat{M}$. Then there is $\hat{X}' \in \hat{M}$ such that $\hat{X}' \subseteq \hat{X} \wedge |\hat{X}'| < |\mathbb{N}|$, since $\hat{M}$ is finitely determined. Thus, there is $n \in \mathbb{N}$ such that $\hat{X}' \subseteq E_n$. From (5.4) we obtain $\emptyset \neq X_n \cap \hat{X}' \overset{\hat{X}'\subseteq E_n}{\subseteq} X_n \cap E_n \cap \hat{X}' = X \cap E_n \cap \hat{X}' \subseteq X \cap \hat{X}' \subseteq X \cap \hat{X}$. Hence, $\forall \hat{X} \in \hat{M} : X \cap \hat{X} \neq \emptyset$, as required. □

Proof of Proposition 5.39: That $\mathcal{F}(\mathbf{ETBES}) \subseteq \mathbf{ETPES}$ and $\mathcal{F}(\mathbf{ETPES}) \subseteq \mathbf{ETBES}$ is an immediate consequence of Lemma 5.51, Lemma 5.52 and Lemma 5.53.

It is left to show the continuity of $\mathcal{F} \restriction \mathbf{ETBES}$ and $\mathcal{F} \restriction \mathbf{ETPES}$, i.e. $\hat{\bigsqcup}_i \mathcal{F}(\mathcal{E}_i) = \mathcal{F}(\bigsqcup_i \mathcal{E}_i)$ for every ω-chain $(\mathcal{E}_i)_{i\in\mathbb{N}}$ with respect to $\trianglelefteq$ and $\bigsqcup_i \mathcal{F}(\mathfrak{E}_i) = \mathcal{F}(\hat{\bigsqcup}_i \mathfrak{E}_i)$ for every ω-chain $(\mathfrak{E}_i)_{i\in\mathbb{N}}$ with respect to $\hat{\trianglelefteq}$. The coincidence of the sets of events and the labeling function is easily seen. In the following, we only consider the termination set, since the conflict and the causality relation follow analogously. We have

$$T_{\hat{\bigsqcup}_i \mathcal{F}(\mathcal{E}_i)} = \{\hat{X} \mid \exists j : \forall X \in T_j : \hat{X} \cap E_j \cap X \neq \emptyset\} \tag{5.5}$$

$$T_{\mathcal{F}(\bigsqcup_i \mathcal{E}_i)} = \{\hat{X} \mid \forall X \in \mathcal{P}(\bigcup_i E_i) : (\forall k : X \cap E_k \in T_k) \Rightarrow \hat{X} \cap X \neq \emptyset\} \tag{5.6}$$

$$T_{\bigsqcup_i \mathcal{F}(\mathfrak{E}_i)} = \{X \mid \forall k : \forall \hat{X} \in \hat{T}_k : X \cap \hat{E}_k \cap \hat{X} \neq \emptyset\} \tag{5.7}$$

$$T_{\mathcal{F}(\hat{\bigsqcup}_i \mathfrak{E}_i)} = \{X \mid \forall \hat{X} \in \mathcal{P}(\bigcup_i \hat{E}_i) : (\exists j : (\hat{X} \cap \hat{E}_j) \in \hat{T}_j) \Rightarrow X \cap \hat{X} \neq \emptyset\} \tag{5.8}$$

$T_{\hat{\bigsqcup}_i \mathcal{F}(\mathcal{E}_i)} \subseteq T_{\mathcal{F}(\bigsqcup_i \mathcal{E}_i)}$: Suppose $\hat{X} \in T_{\hat{\bigsqcup}_i \mathcal{F}(\mathcal{E}_i)}$. Let $X \in \mathcal{P}(\bigcup_i E_i)$ such that $\forall k : X \cap E_k \in T_k$. Then $\hat{X} \cap X \supseteq \hat{X} \cap E_j \cap (E_j \cap X) \overset{(5.5)}{\neq} \emptyset$.

$T_{\hat{\bigsqcup}_i \mathcal{F}(\mathcal{E}_i)} \supseteq T_{\mathcal{F}(\bigsqcup_i \mathcal{E}_i)}$: Suppose $\hat{X} \in T_{\mathcal{F}(\bigsqcup_i \mathcal{E}_i)}$. Since $\mathcal{F}(\bigsqcup_i \mathcal{E}_i)$ is an eTpes, there is $\hat{X}' \in T_{\mathcal{F}(\bigsqcup_i \mathcal{E}_i)}$ such that $\hat{X}' \subseteq \hat{X} \wedge |\hat{X}'| < |\mathbb{N}|$. Hence, there is j such that $\hat{X}' \subseteq E_j$.

Let $X \in T_j$, then by Theorem 5.19 there is $X' \in \bigsqcup_i \mathcal{E}_i$ (i.e. $\forall k : X' \cap E_k \in T_k$) such that $X' \cap E_j = X$. Thus $\hat{X} \cap E_j \cap X \supseteq \hat{X}' \cap E_j \cap X = \hat{X}' \cap E_j \cap X' \overset{\hat{X}'\subseteq E_j}{=} \hat{X}' \cap X' \overset{(5.6)}{\neq} \emptyset$.

$T_{\bigsqcup_i \mathcal{F}(\mathfrak{E}_i)} \subseteq T_{\mathcal{F}(\hat{\bigsqcup}_i \mathfrak{E}_i)}$: Suppose $X \in T_{\bigsqcup_i \mathcal{F}(\mathfrak{E}_i)}$. Let $\hat{X} \in \mathcal{P}(\bigcup_i \hat{E}_i)$ such that $\exists j : (\hat{X} \cap \hat{E}_j) \in \hat{T}_j$. Then $X \cap \hat{X} \supseteq X \cap \hat{E}_j \cap (\hat{E}_j \cap \hat{X}) \overset{(5.7)}{\neq} \emptyset$.

$T_{\bigsqcup_i \mathcal{F}(\mathfrak{E}_i)} \supseteq T_{\mathcal{F}(\hat{\bigsqcup}_i \mathfrak{E}_i)}$: Suppose $X \in T_{\mathcal{F}(\hat{\bigsqcup}_i \mathfrak{E}_i)}$. Let $\hat{X} \in \hat{T}_k$, then $\hat{X} = \hat{X} \cap \hat{E}_k$. Hence, $X \cap \hat{E}_k \cap \hat{X} = X \cap \hat{X} \overset{(5.8)}{\neq} \emptyset$. □

5.6.4 Proof of Theorem 5.40.

In order to prove that **ETBES** and **ETPES** have the same expressive power with respect to event traces, we define remainders with respect to $\mathcal{P}(\mathcal{P}(E))$, as follows.

Definition 5.54 *Suppose $E' \subseteq E$. Define $\mathrm{Rem}_{E,E'} : \mathcal{P}(\mathcal{P}(E)) \times E \to \mathcal{P}(\mathcal{P}(E'))$ by*
$\mathrm{Rem}_{E,E'}(M,e) = \{X' \subseteq E' \mid \exists X \in M : X' = X \cap E' \wedge e \notin X\}$
and $\widehat{\mathrm{Rem}}_{E,E'} : \mathcal{P}(\mathcal{P}(E)) \times E \to \mathcal{P}(\mathcal{P}(E'))$ by
$\widehat{\mathrm{Rem}}_{E,E'}(\hat{M},e) = \{\hat{X}' \subseteq E' \mid \exists \hat{X} \in \hat{M} : \hat{X}' = \hat{X}\backslash\{e\}\}$.

The remainders defined above coincide with respect to F_E:

Lemma 5.55 *Suppose $e \notin E'$ and $E' \subseteq E$. Then*

(i) $F_{E'}(\mathrm{Rem}_{E,E'}(M,e)) = \widehat{\mathrm{Rem}}_{E,E'}(F_E(M),e)$

(ii) $F_{E'}(\widehat{\mathrm{Rem}}_{E,E'}(\hat{M},e)) = \mathrm{Rem}_{E,E'}(F_E(\hat{M}),e)$.

Proof:

(i): We have $\hat{X}' \in F_{E'}(\mathrm{Rem}_{E,E'}(M,e))$ if and only if

$$\hat{X}' \subseteq E' \wedge \forall X' \in \mathcal{P}(E') : (\exists X \in M : X' = X \cap E' \wedge e \notin X) \Rightarrow X' \cap \hat{X}' \neq \emptyset \qquad (5.9)$$

and $\hat{X}' \in \widehat{\mathrm{Rem}}_{E,E'}(F_E(M),e)$ if and only if

$$\hat{X}' \subseteq E' \wedge \exists \hat{X} \in \mathcal{P}(E) : (\forall \tilde{X} \in M : \tilde{X} \cap \hat{X} \neq \emptyset) \wedge \hat{X}' = \hat{X}\backslash\{e\} \qquad (5.10)$$

$\subseteq$: Suppose $\hat{X}' \in F_{E'}(\mathrm{Rem}_{E,E'}(M,e))$. Define $\hat{X} = \hat{X}' \cup \{e\}$. Let $\tilde{X} \in M$. If $e \in \tilde{X}$ then $\tilde{X} \cap \hat{X} \neq \emptyset$. Therefore, suppose $e \notin \tilde{X}$. From (5.9) we get $\emptyset \neq (\tilde{X} \cap E') \cap \hat{X}' \overset{\hat{X}' \subseteq E'}{=} \tilde{X} \cap \hat{X}' \overset{e \notin \tilde{X}}{=} \tilde{X} \cap \hat{X}$, which establishes (5.10).

$\supseteq$: Suppose $\hat{X}' \in \widehat{\mathrm{Rem}}_{E,E'}(F_E(M),e)$. Then by (5.10) there is $\hat{X} \in \mathcal{P}(E)$ such that $\forall \tilde{X} \in M : \tilde{X} \cap \hat{X} \neq \emptyset$ and $\hat{X}' = \hat{X}\backslash\{e\}$. Let $X' \in \mathcal{P}(E')$ such that $\exists X \in M : X' = X \cap E' \wedge e \notin X$. Hence, $X' \cap \hat{X}' = (X \cap E') \cap \hat{X}' \overset{\hat{X}' \subseteq E'}{=} X \cap \hat{X}' = X \cap (\hat{X}\backslash\{e\}) \overset{e \notin X}{=} X \cap \hat{X}$ which is non-empty by (5.10). Hence, (5.9) is concluded.

(ii): We have $X' \in F_{E'}(\widehat{\mathrm{Rem}}_{E,E'}(\hat{M},e))$ if and only if

$$X' \subseteq E' \wedge \forall \hat{X}' \in \mathcal{P}(E') : (\exists \hat{X} \in \hat{M} : \hat{X}' = \hat{X}\backslash\{e\}) \Rightarrow X' \cap \hat{X}' \neq \emptyset \qquad (5.11)$$

and $X' \in \mathrm{Rem}_{E,E'}(F_E(\hat{M}),e)$ if and only if

$$X' \subseteq E' \wedge \exists X \in \mathcal{P}(E) : (\forall \tilde{X} \in \hat{M} : X \cap \tilde{X} \neq \emptyset) \wedge X' = X \cap E' \wedge e \notin X \qquad (5.12)$$

$\subseteq$: Suppose $X' \in F_{E'}(\widehat{\mathrm{Rem}}_{E,E'}(\hat{M}, e))$ Define $X = X' \cup (E \backslash (E' \cup \{e\}))$. Then $X \cap E' = X'$ and $e \notin X$, since $X' \subseteq E'$ and $e \notin E'$. Let $\tilde{X} \in \hat{M}$.

If $\tilde{X} \subseteq E' \cup \{e\}$ then $X' \cap (\tilde{X} \backslash \{e\}) \neq \emptyset$ by (5.11). Furthermore, $X \cap \tilde{X} \supseteq X' \cap \tilde{X} = X' \cap (\tilde{X} \backslash \{e\})$, since $X' \subseteq E'$ and $e \notin E'$.

If $\tilde{X} \not\subseteq E' \cup \{e\}$ then there exists $\tilde{e} \in \tilde{X}$ such that $\tilde{e} \notin E' \cup \{e\}$. Hence $\tilde{e} \in \tilde{X} \cap (E \backslash (E' \cup \{e\})) \subseteq \tilde{X} \cap X$.

Thus (5.12) is established.

$\supseteq$: Suppose $X' \in \mathrm{Rem}_{E,E'}(F_E(\hat{M}), e)$. Then by (5.12) there is $X \in \mathcal{P}(E)$ such that $\forall \tilde{X} \in \hat{M} : X \cap \tilde{X} \neq \emptyset$ and $X' = X \cap E'$ and $e \notin X$.

Let $\hat{X}' \in \mathcal{P}(E')$ such that $\exists \hat{X} \in \hat{M} : \hat{X}' = \hat{X} \backslash \{e\}$. Then $X' \cap \hat{X}' = X \cap E' \cap \hat{X}' \overset{\hat{X}' \subseteq E'}{=} X \cap \hat{X}' = X \cap (\hat{X} \backslash \{e\}) \overset{e \notin X}{=} X \cap \hat{X}$ which is non-empty, since $\forall \tilde{X} \in \hat{M} : X \cap \tilde{X} \neq \emptyset$. Hence, (5.11) is concluded. □

Before we continue, we give a modified version of the remainder on $\mathbf{ETBES}$. There we guarantee in the case of termination that T is upper closed.

Definition 5.56 *Let $\mathcal{E} \in \mathbf{ETBES}$ and $e \in \mathrm{init}(\mathcal{E})$. Then $\mathcal{E}_{/[e]'}$ is given by $(E', \succ', \mapsto', T', l')$, where E', $\succ'$, $\mapsto'$ and l' are defined as in Definition 5.9 and*

$$T' = \begin{cases} \{X \cap E' \mid X \in T \wedge e \notin X\} & \text{if } \neg \Upsilon(T, e) \\ \mathcal{P}(E') & \text{otherwise} \end{cases}$$

Lemma 5.57 *Let $\mathcal{E} \in \mathbf{ETBES}$. Then the event traces obtained by Definition 5.9 and by Definition 5.56 are identical. Moreover the corresponding labels of the event executions coincide.*

Proof: This follows from the fact that T does not influence the event traces. It only determines the fact when an event becomes a termination event. After termination no further termination may happen by both remainders. This holds, since $\emptyset$ is in the termination set. □

The remainders on $\mathbf{ETBES}$ and $\mathbf{ETPES}$ coincide:

Proposition 5.58 *Let $\mathcal{E} \in \mathbf{ETBES}$ and $\mathfrak{E} \in \mathbf{ETPES}$. Then $\mathcal{F}(\mathcal{E}_{/[e]'}) \simeq \mathcal{F}(\mathcal{E})_{\widehat{[e]}}$ and $\mathcal{F}(\mathfrak{E}_{\widehat{[e]}}) \simeq \mathcal{F}(\mathfrak{E})_{/[e]'}$.*

Proof: Let $\mathcal{E}' = \mathcal{E}_{/[e]'}$ and $\mathfrak{E}' = \mathfrak{E}_{\widehat{[e]}}$. Then it is easy to check that

$$\begin{aligned} \succ' &= \{(Z', e') \mid e' \in E' \wedge Z' \in \mathrm{Rem}_{E,E'}(\_ \succ e', e)\} \\ \mapsto' &= \{(X', e') \mid e' \in E' \wedge X' \in \mathrm{Rem}_{E,E'}(\_ \mapsto e', e)\} \\ T' &= \begin{cases} \mathrm{Rem}_{E,E'}(T, e) & \text{if } \neg \Upsilon(T, e) \\ \mathcal{P}(E') & \text{otherwise} \end{cases} \end{aligned}$$

and

$$\begin{aligned} \hat{\succ}' &= \{(Z', e') \mid e' \in \hat{E}' \wedge Z' \in \widehat{\mathrm{Rem}}_{\hat{E},\hat{E}'}(\_ \hat{\succ} e', e)\} \\ \hat{\mapsto}' &= \{(X', e') \mid e' \in \hat{E}' \wedge X' \in \widehat{\mathrm{Rem}}_{\hat{E},\hat{E}'}(\_ \hat{\mapsto} e', e)\} \\ \hat{T}' &= \begin{cases} \widehat{\mathrm{Rem}}_{\hat{E},\hat{E}'}(\hat{T}, e) & \text{if } \neg \hat{\Upsilon}(\hat{T}, e) \\ \emptyset & \text{otherwise} \end{cases} \end{aligned}$$

Furthermore, $e \in \mathrm{init}(\mathcal{E}) \iff \neg(\exists X : X \mapsto e) \overset{\text{Lem. 5.51}}{\iff} \emptyset \hat{\mapsto}_{\mathcal{F}(\mathcal{E})} e \iff e \in \widehat{\mathrm{init}}(\mathcal{F}(\mathcal{E}))$ and $e \in \widehat{\mathrm{init}}(\mathfrak{E}) \iff \emptyset \hat{\mapsto} e \overset{\text{Lem. 5.51}}{\iff} \neg(\exists X : X \mapsto_{\mathcal{F}(\mathfrak{E})} e) \iff e \in \mathrm{init}(\mathcal{F}(\mathfrak{E}))$. Hence, $\mathcal{F}(\mathcal{E}')$ is defined if and only if $\mathcal{F}(\mathcal{E})_{\widehat{[e]}}$ is defined, and $\mathcal{F}(\mathfrak{E}')$ is defined if and only if $\mathcal{F}(\mathfrak{E})_{'[e]'}$ is defined.

We have $e' \in \hat{E}_{\mathcal{F}(\mathcal{E}')} \iff e' \in E' \iff (\exists Z : Z \succ e' \wedge e \notin Z) \overset{\text{Lem. 5.51}}{\iff} \neg(\{e\} \hat{\succ}_{\mathcal{F}(\mathcal{E})} e') \iff e' \in \hat{E}_{\mathcal{F}(\mathcal{E})_{\widehat{[e]}}}$ and $e' \in E_{\mathcal{F}(\mathfrak{E}')} \iff e' \in \hat{E}' \iff \neg(\{e\} \hat{\succ} e') \overset{\text{Lem. 5.51}}{\iff} \exists Z : Z \succ_{\mathcal{F}(\mathfrak{E})} e' \wedge e \notin Z \iff e' \in E_{\mathcal{F}(\mathfrak{E})_{'[e]'}}$.

The rest follows from Lemma 5.55 and the fact that $F_{E'}(\emptyset) = \mathcal{P}(E')$ and $F_{E'}(\mathcal{P}(E')) = \emptyset$. □

Theorem 5.40 is an immediate consequence of Lemma 5.57 and Proposition 5.58.

Chapter 6

End-Based View in $\mathbf{ETBES}$

As in Chapter 4, we consider the view that a choice is determined by the ending of actions (end-based view), which is contrary to the usual approach, where the start of an action triggers the choice. A motivation for an end-based approach is given in Section 1.3.

In this chapter, we apply the end-based approach to $\mathbf{ETBES}$ and not to $\mathbf{CBES}$, as it is done in Chapter 4. More precisely, we define an end-based refinement operator on $\mathbf{ETBES}$ such that the refinement terminates by its 'final' executed event (action) and not with an additional event, as it is done in Chapter 4. The end-based approach is adjusted to $\mathbf{ETBES}$, because the intuitive equivalences (ICT- and FUI) fail to be the coarsest for the end-based refinement operator in the $\mathbf{CBES}$-setting (Subsection 4.2.6).

We adjust the definition of the ICT- and the FUI-equivalence (Section 4.2) to the $\mathbf{ETBES}$ setting. The UI-equivalence can also be adjusted to the $\mathbf{ETBES}$ setting in a straightforward way, which is omitted here. We show that the ICT- (and the FUI-) equivalence is indeed the coarsest congruence for the end-based refinement operator with respect to trace (respectively bisimulation) equivalence in the $\mathbf{ETBES}$ setting. This circumstance underpins the fact that extended termination bundle event structures represent a reasonable extension of the standard event structures. Furthermore, we show that the hierarchy of the equivalences considered in the $\mathbf{ETBES}$ setting is the same as in the $\mathbf{CBES}$ setting.

6.1 An End-Based Refinement Operator on $\mathbf{ETBES}$

The differences between refinement operators in start-based and in end-based settings is illustrated in Section 4.1. There, we also argue that an event structure suitable for an end-based refinement operator has to allow the modeling of disruption. This is true for extended termination bundle event structures (eTbes), which are introduced in Subsection 5.4.2. Hence, they represent a suitable model for introducing an end-based refinement operator. This operator is given in the following definition.

Let τ, Obs and Var be defined as in Section 3.2 and let $\mathcal{A}ct$ be defined as in Section 5.2.

Definition 6.1 *Let $A \subseteq \mathsf{Obs}$. Then define $Ref_A^{eT} : \mathbf{ETBES} \times (A \to \mathbf{ETBES}) \to \mathbf{ETBES}$ by $Ref_A^{eT}(\mathcal{E}, \theta) = (\tilde{E}, \tilde{\succ}, \tilde{\mapsto}, \tilde{T}, \tilde{l})$ where*

$$
\begin{array}{rcl}
\tilde{E} & = & \{(e,\hat{e}) \mid e \in E \wedge l(e) \in A \wedge \hat{e} \in E_{\theta(l(e))}\} \cup \\
 & & \{(e,e) \in E \times E \mid l(e) \notin A\} \\
\tilde{\succ} & = & \{(\tilde{Z},(e,\hat{e})) \mid \exists Z : Z \succ e \wedge \exists f : Z \to \mathcal{P}(\mathcal{U}) : \\
 & & (\forall e' \in Z : (l(e') \notin A \wedge f(e') = \{e'\}) \vee (l(e') \in A \wedge e' \neq e \wedge f(e') \in T_{\theta(l(e'))}) \vee \\
 & & \quad (l(e') \in A \wedge e' = e \wedge \exists \hat{X} \in T_{\theta(l(e'))}, \hat{Z} : f(e') = \hat{Z} \cup \hat{X} \wedge \hat{Z} \succ_{\theta(l(e'))} \hat{e})) \wedge \\
 & & \tilde{Z} = \{(e',\hat{e}') \in \tilde{E} \mid e' \in Z \wedge \hat{e}' \in f(e')\}\} \\
\tilde{\mapsto} & = & \{(\{e\} \times X', (e,\hat{e})) \mid l(e) \in A \wedge X' \mapsto_{\theta(l(e))} \hat{e}\} \cup \\
 & & \{(\tilde{X},(e,\hat{e})) \mid (l(e) \in A \Rightarrow \hat{e} \in \mathrm{init}(\theta(l(e)))) \wedge \exists X : X \mapsto e \wedge \exists f : X \to \mathcal{P}(\mathcal{U}) : \\
 & & (\forall e' \in X : (l(e') \notin A \wedge f(e') = \{e'\}) \vee (l(e') \in A \wedge f(e') \in T_{\theta(l(e'))})) \wedge \\
 & & \tilde{X} = \{(e',\hat{e}') \in \tilde{E} \mid e' \in X \wedge \hat{e}' \in f(e')\}\} \\
\tilde{T} & = & \{\tilde{X} \mid \exists X \in T \wedge \exists f : X \to \mathcal{P}(\mathcal{U}) : (\forall e' \in X : (l(e') \notin A \wedge f(e') = \{e'\}) \vee \\
 & & (l(e') \in A \wedge f(e') \in T_{\theta(l(e'))})) \wedge \tilde{X} = \{(e',\hat{e}') \in \tilde{E} \mid e' \in X \wedge \hat{e}' \in f(e')\}\}
\end{array}
$$

$$
\tilde{l}(e,\hat{e}) = \begin{cases} l(e) & \text{if } l(e) \notin A \\ l_{\theta(l(e))}(\hat{e}) & \text{if } l(e) \in A \end{cases}
$$

We give some comments on the definition. A witness-bundle derived from the witness-bundle Z of e (i.e. $Z \succ e$) of event $(e,\hat{e})$ has to contain all events of a termination-bundle of the refinement of an event e' (different to e) that is in Z. This is done in order to guarantee that bundles derived from Z can not be used as a witness of e when e' terminates. If e' is equal to e, then the witness-bundles (instead of the termination bundles) of the refinement are considered. This guarantees that the events of the refinement become disabled, as specified by the refinement. A termination-bundle of the refinement of e' is used in the causality relation whenever e' appears in the causality bundle X of the unrefined event structures, since the refinement has to terminate before the constraint specified by X is fulfilled. This is also the case in the definition of the termination set.

Lemma 6.2 *The refinement operator Ref^{eT} is well defined, i.e. it really yields elements of* **ETBES**.

Proof: The proof is given in Subsection 6.4.1. □

An example that illustrates how the refinement operator Ref^{eT} behaves is given in Figure 6.1. For a better understanding, we augment the examples by process term descriptions of the systems (see Section 5.2). Furthermore, $(a \to \mathcal{E}_{12})$ denotes the function from $\{a\}$ to **ETBES** that maps a to $\mathcal{E}_{12}$. Moreover, if an event e' is a necessary causality of e, i.e. e can only be executed after the execution of e', we sometimes omit e in the witness bundles of e', since it has no consequence for the behavior. For example, in $Ref^{eT}_{\{a\}}(\mathcal{E}^{+a}, (a \to \mathcal{E}_{12}))$ of Figure 6.1, the witness bundles to events labeled with a_1 have to contain both events labeled with a_2.

As in Remark 4.4, the refinement operator Ref^{eT} allows the modeling of the disrupt operator $\widehat{[>}$ of Definition 5.20, i.e. $\mathcal{E}_1 \widehat{[>} \mathcal{E}_2$ and $Ref^{e}_{\{a\}}(\mathcal{E}_1 + [\![a]\!], (a \to \mathcal{E}_2))$ have the same behavior if label a does not appear in $\mathcal{E}_1$.

6.2 Equivalences for ETBES

As in Section 4.2, we examine congruence equivalences for Ref^{eT}. We are particularly interested in the coarsest congruence with respect to trace / strong bisimulation equivalences, where

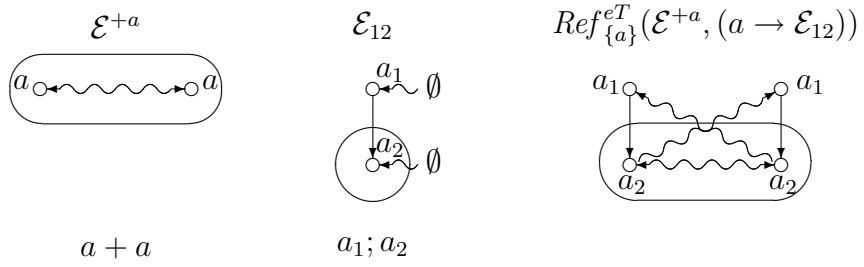


Figure 6.1: End-Based Refinement in $\mathbf{ETBES}$

trace and strong bisimulation equivalences is obtained, as usual, from the derived transition system introduced in Definition 5.11. Formally:

Definition 6.3 (Trace Equivalence) *Two $\mathcal{E}, \mathcal{E}' \in \mathbf{ETBES}$ are* trace equivalent*, denoted by $\mathcal{E} \sim_t \mathcal{E}'$, if and only if the transition systems $(\mathbf{ETBES}, \mathcal{A}ct, \hookrightarrow, \mathcal{E})$ and $(\mathbf{ETBES}, \mathcal{A}ct, \hookrightarrow, \mathcal{E}')$, where $\hookrightarrow$ is defined in Definition 5.11, are trace equivalent (Definition 2.4).*

Definition 6.4 (Strong Bisimulation Equivalence) *Two $\mathcal{E}, \mathcal{E}' \in \mathbf{ETBES}$ are* strong bisimilar *(or strong bisimulation equivalent), denoted by $\mathcal{E} \sim_b \mathcal{E}'$, if and only if the transition systems $(\mathbf{ETBES}, \mathcal{A}ct, \hookrightarrow, \mathcal{E})$ and $(\mathbf{ETBES}, \mathcal{A}ct, \hookrightarrow, \mathcal{E}')$ are bisimilar (Definition 2.5).*

6.2.1 ICT-Equivalence on $\mathbf{ETBES}$

ICT-equivalence from Subsection 4.2.2 is adapted to $\mathbf{ETBES}$ as follows.

Definition 6.5 (ICT-equivalence) *Let $\mathcal{E} \in \mathbf{ETBES}$. Then the* initial event traces *of $\mathcal{E}$ are defined by $T^{ic}(\mathcal{E}) = \{((e_i, \nu_i), \gamma_i)_{i \leq n} \mid n \in \mathbb{N} \wedge \exists \mathcal{E}_0, \cdots, \mathcal{E}_{n+1} : \mathcal{E}_0 = \mathcal{E} \wedge \forall i \leq n : \mathcal{E}_{i[e_i]} = \mathcal{E}_{i+1} \wedge (\Upsilon(\mathcal{E}_i, e_i) \Leftrightarrow \nu_i = \surd) \wedge \gamma_i \in \mathcal{P}_{fin}(\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_i))\}$.*

Two $\mathcal{E}, \mathcal{E}' \in \mathbf{ETBES}$ are initial corresponding trace equivalent *(ICT-equivalent), denoted by $\mathcal{E} \sim_{ICT} \mathcal{E}'$, if*

- *for every $((e_i, \nu_i), \gamma_i)_{i \leq n} \in T^{ic}(\mathcal{E})$ there is an injective, labeling preserving function $f : (\bigcup_{i \leq n}(\gamma_i \cup \{e_i\})) \to E'$ such that $((f(e_i), \nu_i), f(\gamma_i))_{i \leq n} \in T^{ic}(\mathcal{E}')$ and*
- *for every $((e'_i, \nu_i), \gamma'_i)_{i \leq n} \in T^{ic}(\mathcal{E}')$ there is an injective, labeling preserving function $f' : (\bigcup_{i \leq n}(\gamma'_i \cup \{e'_i\})) \to E$ such that $((f'(e'_i), \nu_i), f'(\gamma'_i))_{i \leq n} \in T^{ic}(\mathcal{E})$*

The eTbes from Figure 6.2 are not ICT equivalences, whereas the eTbes from Figure 6.3 are ICT equivalent.

Proposition 6.6 *Two ICT-equivalent eTbes are also trace equivalent, i.e. $\sim_{ICT} \subset \sim_t$.*

Proof: It follows from the fact that every trace is also an initial trace, where the second component is always empty. □

$b + a$ $\qquad$ $a; b + b; a$

Figure 6.2: Non ICT-Equivalent eTbes

$a + b; a$ $\qquad$ $a; b + a; b$

Figure 6.3: ICT-Equivalent eTbes

Theorem 6.7 *ICT-equivalence is a congruence for the refinement operator Ref^{eT}, i.e. $\mathcal{E} \sim_{ICT} \mathcal{E}' \wedge \forall a \in A : \theta(a) \sim_{ICT} \theta'(a)$ implies that $Ref^{eT}_A(\mathcal{E}, \theta) \sim_{ICT} Ref^{eT}_A(\mathcal{E}', \theta')$.*

Proof: It works analogously to the proof of Theorem 4.10. □

Contrary to the **CBES** setting, ICT-equivalence is the coarsest congruence for Ref^{eT} with respect to trace equivalence.

Theorem 6.8 *ICT-equivalence is the coarsest congruence for Ref^{eT} with respect to trace equivalence. Moreover, if $\forall A, \theta : Ref^{eT}_A(\mathcal{E}, \theta) \sim_t Ref^{eT}_A(\mathcal{E}', \theta)$ then $\mathcal{E} \sim_{ICT} \mathcal{E}'$.*

Proof: The proof is given in Subsection 6.4.2. □

6.2.2 FUI-Equivalence on ETBES

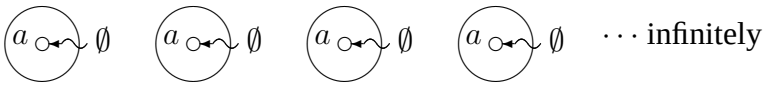
FUI-equivalence from Subsection 4.2.4 is adapted to **ETBES** as follows.

Definition 6.9 (FUI-Bisimulation) *A* finite unique initial bisimulation *(FUI-bisimulation) $\mathcal{R}$ is a subset of* $\mathbf{ETBES} \times \mathbf{ETBES} \times (\mathcal{U} \rightharpoonup \mathcal{U})$ *such that whenever $(\mathcal{E}_1, \mathcal{E}_2, f) \in \mathcal{R}$, then*

- $\mathrm{dom}(f) = \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_1)$,
- *f is a labeling preserving isomorphism between* $\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_1)$ *and* $\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_2)$
- *$e_1 \in \mathrm{init}(\mathcal{E}_1) \wedge I \in \mathcal{P}_{fin}(\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_1))$ implies that there exist e_2 and f' such that $l_1(e_1) = l_2(e_2)$ and $\Upsilon(T_1, e_1) \Leftrightarrow \Upsilon(T_2, e_2)$ and $l_1(e_1) \in \mathsf{Obs} \Rightarrow e_2 = f(e_1)$ and $(\mathcal{E}_{1[e_1]}, \mathcal{E}_{2[e_2]}, f') \in \mathcal{R}$ and $f \upharpoonright (I \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{1[e_1]})) = f' \upharpoonright I$ and $f^{-1} \upharpoonright (f(I) \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{2[e_2]})) = f'^{-1} \upharpoonright f(I)$*

Figure 6.4: FUI-Equivalent eTbes (1)

$$\mathcal{E}_1^{inf} = [\![\langle (x \mapsto a \|_\emptyset x), x \rangle]\!]$$



$$\mathcal{E}_2^{inf} = [\![\langle (x \mapsto a \|_\emptyset x), (a;a) \|_\emptyset x \rangle]\!]$$

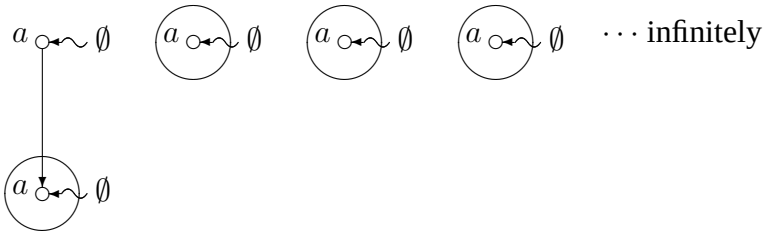


Figure 6.5: FUI-Equivalent eTbes (2)

- $e_2 \in \mathrm{init}(\mathcal{E}_2) \wedge I \in \mathcal{P}_{fin}(\mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_2))$ *implies that there exist* e_1 *and* f' *such that* $l_1(e_1) = l_2(e_2)$ *and* $\Upsilon(T_1, e_1) \Leftrightarrow \Upsilon(T_2, e_2)$ *and* $l_1(e_1) \in \mathsf{Obs} \Rightarrow e_2 = f(e_1)$ *and* $(\mathcal{E}_{1[e_1]}, \mathcal{E}_{2[e_2]}, f') \in \mathcal{R}$ *and* $f \upharpoonright (I \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{1[e_1]})) = f' \upharpoonright I$ *and* $f^{-1} \upharpoonright (f(I) \cap \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}_{2[e_2]})) = f'^{-1} \upharpoonright f(I)$

We say that $\mathcal{E}_1, \mathcal{E}_2$ *are* FUI-bisimilar *(or* FUI-equivalent*), denoted by* $\mathcal{E}_1 \sim_{FUI} \mathcal{E}_2$*, if and only if there is a FUI-bisimulation* $\mathcal{R}$ *and an* $f : \mathcal{U} \rightharpoonup \mathcal{U}$ *such that* $(\mathcal{E}_1, \mathcal{E}_2, f) \in \mathcal{R}$*.*

The eTbes from Figure 6.3 are not FUI-equivalent, whereas the eTbes from Figure 6.4 are FUI-equivalent. Moreover, the eTbes from Figure 6.5 are also FUI-equivalent.

FUI-equivalence yields a congruence.

Theorem 6.10 *FUI-equivalence is a congruence for* Ref^{eT}*, i.e.* $\mathcal{E} \sim_{FUI} \mathcal{E}' \wedge \forall a \in A : \theta(a) \sim_{FUI} \theta'(a)$ *implies that* $Ref_A^{eT}(\mathcal{E}, \theta) \sim_{FUI} Ref_A^{eT}(\mathcal{E}', \theta')$*.*

Proof: It works analogously to the proof of Theorem 4.18.

Contrary to the **CBES** setting, FUI-equivalence is the coarsest congruence for Ref^{eT} with respect to bisimulation equivalence.

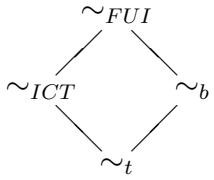


Figure 6.6: Relations Between the Equivalences

Theorem 6.11 *FUI-equivalence is the coarsest congruence for Ref^{eT} with respect to bisimulation equivalence. Moreover, if $\forall A, \theta : Ref^{eT}_A(\mathcal{E}, \theta) \sim_b Ref^{eT}_A(\mathcal{E}', \theta)$ then $\mathcal{E} \sim_{FUI} \mathcal{E}'$.*

Proof: The proof is given in Subsection 6.4.2. □

6.2.3 Comparison of Equivalences

Theorem 6.12 *All valid relations between the equivalences $\sim_t, \sim_{ICT}, \sim_b, \sim_{FUI}$ are presented in Figure 6.6: If two equivalences are connected via a line, then the lower one identifies more elements than the upper one.*

Proof: $\sim_{FUI} \subseteq \sim_b \subseteq \sim_t$ and $\sim_{ICT} \subseteq \sim_t$ is obvious.

Suppose $\mathcal{E} \sim_{FUI} \mathcal{E}'$, then by Theorem 6.10 we have $\forall A, \theta : Ref^{eT}_A(\mathcal{E}, \theta) \sim_{FUI} Ref^{eT}_A(\mathcal{E}', \theta)$. Since $\sim_{FUI} \subseteq \sim_b \subseteq \sim_t$, we obtain $\forall A, \theta : Ref^{eT}_A(\mathcal{E}, \theta) \sim_t Ref^{eT}_A(\mathcal{E}', \theta)$. Thus by Theorem 6.8 it follows that $\mathcal{E} \sim_{ICT} \mathcal{E}'$.

The strictness of $\sim_{FUI} \subset \sim_{ICT}$ follows from the event structure depicted in Figure 6.5. The strictness of $\sim_b \subset \sim_t$ is well known. And the other strictness follows from the fact that the event structures corresponding to a and $a + a$ are bisimilar but not ICT-equivalent. □

6.3 Discussion

It is now straightforward to give a denotational semantics to process algebras that contain end-based choice operators together with action refinement operators as long as no parallel operator with action synchronization is contained in the process algebra.

For process algebras that also contain a parallel operator with action synchronization, it is reasonable that some start-based choices are modeled. For example, we expect that the expression $(a + a)\|_{\{a\}} a$ may only start one a-action, since the process on the right hand side can only start one a-action. Hence, a start-based choice is modeled for the process on the left hand side. Such a circumstance arises, for example, if the left hand side demands processor resources and the right hand side specifies the administration of the processor resource, where only one a can be executed.

Therefore, it is more reasonable to consider process algebras that contain both choice operators, i.e. end-based and start-based choice operators. Such a process algebra is intensively examined in the following chapter.

6.4 Proofs

6.4.1 Proof of Lemma 6.2

The constraints which are different to the approximation closedness are easy to check. Let $\tilde{E} = Ref_A^{eT}(\mathcal{E}, \theta)$ and $E_e = \begin{cases} E_{\theta(l(e))} & \text{if } l(e) \in A \\ \{e\} & \text{otherwise} \end{cases}$.

$\tilde{\succ}$: Suppose $(e', \hat{e}') \in \tilde{E}$. Define $M = \{Z \mid Z \succ e'\}$. Furthermore, let

$$M_e = \begin{cases} \{\{e\}\} & \text{if } l(e) \notin A \\ T_{\theta(l(e))} & \text{if } l(e) \in A \wedge e' \neq e \\ \{\hat{Z} \cup \hat{X} \mid \hat{Z} \succ_{\theta(l(e))} \hat{e}' \wedge \hat{X} \in T_{\theta(l(e))}\} & \text{if } l(e) \in A \wedge e' = e \end{cases}.$$

From Corollary 2.19 we obtain that M_e is approximation closed with respect to E_e.

Furthermore, $\tilde{M} = \{\{(e, \hat{e}) \mid e \in X \wedge \hat{e} \in X_e\} \mid X \in M \wedge X_e \in M_e\}$ is approximation closed with respect to $\tilde{E}$ by Corollary 2.21. Hence, $\tilde{\mapsto}$ is approximation closed with respect to $\tilde{E}$, since $\{\tilde{Z} \mid \tilde{Z} \tilde{\mapsto} (e', \hat{e}')\} = \tilde{M}$.

$\tilde{\mapsto}$: Suppose $(e', \hat{e}') \in \tilde{E}$. Define $M = \{Z \mid Z \mapsto e'\}$ and

$$M_1 = \begin{cases} \{\{e'\} \times \hat{X} \mid \hat{X} \mapsto_{\theta(l(e'))} \hat{e}'\} & \text{if } l(e') \in A \\ \emptyset & \text{otherwise} \end{cases}.$$

Then M_1 is approximation closed with respect to $\tilde{E}$. Furthermore, let

$$M_e = \begin{cases} \{\{e\}\} & \text{if } l(e) \notin A \\ T_{\theta(l(e))} & \text{if } l(e) \in A \end{cases}.$$

Obviously, M_e is approximation closed with respect to E_e.

From Corollary 2.21 we obtain that

$$M_2 = \begin{cases} \emptyset & \text{if } l(e') \in A \wedge \hat{e}' \notin \text{init}(\theta(l(e'))) \\ \{\{(e, \hat{e}) \mid e \in X \wedge \hat{e} \in X_e\} \mid X \in M \wedge X_e \in M_e\} & \text{otherwise} \end{cases}$$

is approximation closed with respect to $\tilde{E}$. And so the approximation closedness of $\tilde{\mapsto}$ follows from Proposition 2.15, since $\{\tilde{X} \mid \tilde{X} \tilde{\mapsto} (e', \hat{e}')\} = M_1 \cup M_2$.

$\tilde{T}$: Let

$$M_e = \begin{cases} \{\{e\}\} & l(e) \notin A \\ T_{\theta(l(e))} & \text{if } l(e) \in A \end{cases}.$$

Obviously, M_e is approximation closed with respect to E_e.

Furthermore, $\tilde{M} = \{\{(e, \hat{e}) \mid e \in X \wedge \hat{e} \in X_e\} \mid X \in M \wedge X_e \in M_e\}$ is approximation closed with respect to $\tilde{E}$ by Corollary 2.21. Hence, $\tilde{T}$ is approximation closed with respect to $\tilde{E}$, since $\tilde{T} = \tilde{M}$.

Thus Lemma 6.2 is established.

6.4.2 Proofs of the Coarsest Congruence Results

We introduce an event-based refinement. This refinement differs from Ref^{eT} by assigning event structures to each event and not only to action names.

Definition 6.13 $\underline{Ref}_A^{eT} : \mathbf{ETBES} \times (\mathcal{U} \to \mathbf{ETBES}) \to \mathbf{ETBES}$ *with*

$\underline{Ref}_A^{eT}(\mathcal{E},\vartheta) = (\tilde{E},\tilde{\succ},\tilde{\mapsto},\tilde{T},\tilde{l})$ *where*

$$
\begin{array}{rcl}
\tilde{E} & = & \{(e,\hat{e}) \mid e \in E \wedge l(e) \in A \wedge \hat{e} \in E_{\vartheta(e)}\} \cup \\
 & & \{(e,e) \in E \times E \mid l(e) \notin A\} \\
\tilde{\succ} & = & \{(\tilde{Z},(e,\hat{e})) \mid \exists Z : Z \succ e \wedge \exists f : Z \to \mathcal{P}(\mathcal{U}) : \\
 & & (\forall e' \in Z : (l(e') \notin A \wedge f(e') = \{e'\}) \vee (l(e') \in A \wedge e' \neq e \wedge f(e') \in T_{\vartheta(e')}) \vee \\
 & & \quad (l(e') \in A \wedge e' = e \wedge \exists \hat{X} \in T_{\vartheta(e')}, \hat{Z} : f(e') = \hat{Z} \cup \hat{X} \wedge \hat{Z} \succ_{\vartheta(e')} \hat{e})) \wedge \\
 & & \tilde{Z} = \{(e',\hat{e}') \in \tilde{E} \mid e' \in Z \wedge \hat{e}' \in f(e')\}\} \\
\tilde{\mapsto} & = & \{(\{e\} \times X',(e,\hat{e})) \mid l(e) \in A \wedge X' \mapsto_{\vartheta(e)} \hat{e}\} \cup \\
 & & \{(\tilde{X},(e,\hat{e})) \mid (l(e) \in A \Rightarrow \hat{e} \in \mathrm{init}(\vartheta(e))) \wedge \exists X : X \mapsto e \wedge \exists f : X \to \mathcal{P}(\mathcal{U}) : \\
 & & (\forall e' \in X : (l(e') \notin A \wedge f(e') = \{e'\}) \vee (l(e') \in A \wedge f(e') \in T_{\vartheta(e')})) \wedge \\
 & & \tilde{X} = \{(e',\hat{e}') \in \tilde{E} \mid e' \in X \wedge \hat{e}' \in f(e')\}\} \\
\tilde{T} & = & \{\tilde{X} \mid \exists X \in T \wedge \exists f : X \to \mathcal{P}(\mathcal{U}) : (\forall e' \in X : (l(e') \notin A \wedge f(e') = \{e'\}) \vee \\
 & & (l(e') \in A \wedge f(e') \in T_{\vartheta(e')})) \wedge \tilde{X} = \{(e',\hat{e}') \in \tilde{E} \mid e' \in X \wedge \hat{e}' \in f(e')\}\}
\end{array}
$$

$$
\tilde{l}(e,\hat{e}) = \begin{cases} l(e) & \text{if } l(e) \notin A \\ l_{\vartheta(e)}(\hat{e}) & \text{if } l(e) \in A \end{cases}
$$

The advantage of $\underline{Ref}_A^{eT}$ is that the event execution of $\underline{Ref}_A^{eT}(\mathcal{E},\vartheta)$ can be reduced to the event execution of $\mathcal{E}$ and ϑ, as it is shown in the following lemma.

Lemma 6.14 *Suppose* $\mathcal{E} \in \mathbf{ETBES}$, $\vartheta : \mathcal{U} \to \mathbf{ETBES}$. *Then*

$$
\underline{Ref}_A^{eT}(\mathcal{E},\vartheta)_{[(e,\hat{e})]} \simeq \begin{cases} \underline{Ref}_A^{eT}(\mathcal{E}_{[e]},\vartheta) & \text{if } l(e) \notin A \wedge e = \hat{e} \\ \underline{Ref}_A^{eT}(\mathcal{E}_{[e]},\vartheta[e \to \vartheta(e)_{[\hat{e}]}]) & \text{if } l(e) \in A \wedge \Upsilon(T_{\vartheta(e)},\hat{e}) \\ \underline{Ref}_A^{eT}(\mathcal{E},\vartheta[e \to \vartheta(e)_{[\hat{e}]}]) \} & \text{if } l(e) \in A \wedge \neg\Upsilon(T_{\vartheta(e)},\hat{e}) \end{cases} .
$$

Furthermore, $\underline{Ref}_A^{eT}(\mathcal{E}_{[e]},\vartheta[e \to \mathcal{E}']) \simeq \underline{Ref}_A^{eT}(\mathcal{E}_{[e]},\vartheta)$ *holds for any* $\mathcal{E}' \in \mathbf{ETBES}$.

Moreover,

$$
\Upsilon(T_{\underline{Ref}_A^{eT}(\mathcal{E},\vartheta)},(e,\hat{e})) \Leftrightarrow \begin{cases} \Upsilon(T,e) & \text{if } l(e) \notin A \wedge e = \hat{e} \\ \Upsilon(T,e) \wedge \Upsilon(T_{\vartheta(e)},\hat{e}) & \text{if } l(e) \in A \end{cases}
$$

Proof: Straightforward and left to the reader. □

Proof of Theorem 6.8: Suppose $((e_i,\nu_i),\gamma_i)_{i\leq n} \in T^{ic}(\mathcal{E})$. We define a refinement θ' which is used to construct a corresponding trace.

Therefore, let $\kappa : \mathcal{U} \to \mathbb{N}$ be an isomorphism. Furthermore, define $\hat{E} = \bigcup_{i\leq n}(\gamma_i \cup \{e_i\})$ and $\hat{E}_a = \{e \in \hat{E} \mid l(e) = a\}$. Additionally, define $\delta : E \to \mathbb{N}$ by $\delta(e) = 1 + |\{i \mid e \in \gamma_i\}|$. Moreover, let $A \subset \mathcal{A}ct$ be the set of all action-names occurring in $\mathcal{E}$ or in $\mathcal{E}'$, i.e. $A = \{l(e) | e \in E\} \cup \{l'(e') | e' \in E'\}$. And let $\mu : E \times \mathbb{N} \to \mathsf{Obs}\backslash A$ be an injective function. Such a function exists.

Our idea of θ' is that we replace any event e of $\hat{E}$ by the sequential composition of actions $\mu(e,1),...,\mu(e,\delta(e)+1)$. Since θ' only maps action-names instead of events, we take the sum

of all the corresponding events, i.e. $\theta'(a) = \mathcal{E}'_a$, where

$$\begin{aligned}\mathcal{E}'_a \;= (\;& \{ \star_2^{\kappa(e)} \star_1^j \bullet \mid e \in \hat{E}_a \wedge 1 \leq j \leq \delta(e)\},\\ & \{(\{\star_2^k \star_1 \bullet \mid k \neq \kappa(e)\} \cup \{\star_2^{\kappa(e)} \star_1^j \bullet\}, \star_2^{\kappa(e)} \star_1^j \bullet) \mid e \in \hat{E}_a \wedge 1 \leq j \leq \delta(e)\},\\ & \{(\{\star_2^{\kappa(e)} \star_1^j \bullet\}, \star_2^{\kappa(e)} \star_1^{j+1} \bullet) \mid e \in \hat{E}_a \wedge 1 \leq j < \delta(e)\},\\ & \{\{\star_2^{\kappa(e)} \star_1^{\delta(e)} \bullet \mid e \in \hat{E}_a\}\},\\ & \{(\star_2^{\kappa(e)} \star_1^j \bullet, \mu(e,j)) \mid e \in \hat{E}_a \wedge 1 \leq j \leq \delta(e)\}).\end{aligned}$$

The sequences $\star_2^k \star_1^j \bullet$ are considered to be right bracketed and therefore to be elements of $\mathcal{U}$. Let $m = n + \sum_{i=0}^{n} |\gamma_i|$ and define $I_{-1} = \gamma_0$ and $s_{-1} = 0$ and for $i \in \{0, ..., m-1\}$

- If $I_{i-1} \neq \emptyset$, then $s_i = s_{i-1}$, $I_i = I_{i-1} \backslash \{e\}$ and $\tilde{e}_i = (e, \star_2^{\kappa(e)} \star_1^{|\{j \mid j \leq s_{i-1} \wedge e \in \gamma_j\}|} \bullet)$, where e is an element of I_{i-1}
- If $I_{i-1} = \emptyset$, then $s_i = s_{i-1} + 1$, $I_i = \gamma_{s_i}$ and
$$\tilde{e}_i = \begin{cases} (e_{s_{i-1}}, e_{s_{i-1}}) & \text{if } l(e_{s_{i-1}}) = \tau \\ (e_{s_{i-1}}, \star_2^{\kappa(e_{s_{i-1}})} \star_1^{\delta(e_{s_{i-1}})} \bullet) & \text{otherwise} \end{cases}$$

and $\tilde{e}_m = \begin{cases} (e_n, e_n) & \text{if } l(e_n) = \tau \\ (e_n, \star_2^{\kappa(e_n)} \star_1^{\delta(e_n)} \bullet) & \text{otherwise} \end{cases}$.

It is easily seen that $\{\pi_1(\tilde{e}_i) \mid i \in \{0, ..., m\}\} = \hat{E}$. Define $\tilde{\mathcal{E}}_0 = Ref_A^{eT}(\mathcal{E}, \theta')$ and $\tilde{\mathcal{E}}_{i+1} = \tilde{\mathcal{E}}_{i[\tilde{e}_i]}$ for $i \leq m$. The $\tilde{\mathcal{E}}_i$ are well defined which, can be seen by induction, as follows. Obviously for $i = 0$. Suppose $\tilde{e}_i = (e, \star_2^{\kappa(e)} \star_1^j \bullet)$. By Lemma 6.14 we obtain that $\tilde{\mathcal{E}}_i = \underline{Ref}_A^{eT}(\mathcal{E}_{[e_0]..[e_{s_{i-1}-1}]}, \vartheta'_i)$, where $\vartheta'_i(e) = \theta'(l(e))_{[\star_2^{\kappa(e)} \star_1^1 \bullet], ..., [\star_2^{\kappa(e)} \star_1^q \bullet]}$ with $q = |\{j \mid e = \pi_1(\tilde{e}_j) \wedge l(e) \neq \tau \wedge j < i\}|$. It is easily seen that $(e, \star_2^{\kappa(e)} \star_1^1 \bullet), ..., (e, \star_2^{\kappa(e)} \star_1^{j-1} \bullet)$ appears in the sequence before $\tilde{e}_i$. Thus $\star_2^{\kappa(e)} \star_1^j \bullet \in \text{init}(\vartheta'_i(e))$. Furthermore, $e_{s_{i-1}} \in \text{init}(\mathcal{E}_{[e_0]..[e_{s_{i-1}-1}]})$, since $((e_j, \nu_j), \gamma_j)_{j \leq n} \in T^{ic}(\mathcal{E})$. Hence, $\tilde{e}_i \in \tilde{\mathcal{E}}_i$. Furthermore, by Lemma 6.14 we obtain

$$\Upsilon(\tilde{\mathcal{E}}_i, \tilde{e}_i) \Leftrightarrow \exists j, \hat{e} : \tilde{e}_i = (e_j, \hat{e}) \wedge \nu_j = \surd \wedge (l(e_j)) = \tau \vee \hat{e} = \star_2^{\kappa(e_j)} \star_1^{\delta(e_j)} \bullet). \tag{6.1}$$

From the definition of $\tilde{\mathcal{E}}_i$ it follows that $(\alpha_i)_{(i \leq m)} \in T(Ref_A^{eT}(\mathcal{E}, \theta'))$, where α_i is defined by $\alpha_i = \begin{cases} l_{Ref_A^{eT}(\mathcal{E}, \theta')}(\tilde{e}_i) & \text{if } \neg\Upsilon(\tilde{\mathcal{E}}_i, \tilde{e}_i) \\ l_{Ref_A^{eT}(\mathcal{E}, \theta')}(\tilde{e}_i)\surd & \text{if } \Upsilon(\tilde{\mathcal{E}}_i, \tilde{e}_i) \end{cases}$. Therefore, we get $(\alpha_i)_{(i \leq m)} \in T(Ref_A^{eT}(\mathcal{E}', \theta'))$, since $Ref_A^{eT}(\mathcal{E}, \theta') \sim_t Ref_A^{eT}(\mathcal{E}', \theta')$. Hence, there exists $(\tilde{e}'_i)_{(i \leq m)}$ such that $\tilde{\mathcal{E}}'_0 = Ref_A^{eT}(\mathcal{E}', \theta')$ and $\tilde{\mathcal{E}}'_{i+1} = \tilde{\mathcal{E}}'_{i[\tilde{e}'_i]}$ are well defined and $\alpha_i = \begin{cases} l_{Ref_A^{eT}(\mathcal{E}', \theta')}(\tilde{e}'_i) & \text{if } \neg\Upsilon(\tilde{\mathcal{E}}'_i, \tilde{e}'_i) \\ l_{Ref_A^{eT}(\mathcal{E}', \theta')}(\tilde{e}'_i)\surd & \text{if } \Upsilon(\tilde{\mathcal{E}}'_i, \tilde{e}'_i) \end{cases}$.

From the injectivity of μ we get $\forall i \leq m : \pi_1(\tilde{e}_i) \neq \tau \Rightarrow \pi_2(\tilde{e}_i) = \pi_2(\tilde{e}'_i)$. Define $e'_j = \pi_1(\tilde{e}'_i)$ where i is chosen such that $(\tilde{e}_i = (e_j, \star_2^{\kappa(e_j)} \star_1^{\delta(e_j)} \bullet)) \vee (l(e_j) = \tau \wedge \tilde{e}_i = (e_j, e_j))$. Now, we verify by induction that

$$\tilde{\mathcal{E}}'_i = \underline{Ref}_A^{eT}(\mathcal{E}'_{[e'_0]..[e'_{s_{i-1}-1}]}, \vartheta''_i), \tag{6.2}$$

where $\vartheta''_i(e') = \begin{cases} \vartheta'_i(\pi_1(\tilde{e}_j)) & \text{if } (e', \star_2^{\kappa(e)} \star_1^1 \bullet) = \tilde{e}'_j \\ \theta'(l'(e')) & \text{otherwise} \end{cases}$.

Obviously for $i = 0$. We proceed by making a case analysis:

$l(\pi_1(\tilde{e}_i)) = \tau$: By induction and Lemma 6.14 we get $\tilde{e}_{i+1} = \underline{Ref}_A^{eT}(\mathcal{E}'_{[e'_0]..[e'_{s_{i-1}-1}][\pi_1(\tilde{e}'_i)]}, \vartheta''_i)$, which is equal to $\underline{Ref}_A^{eT}(\mathcal{E}'_{[e'_0]..[e'_{s_{i-1}}]}, \vartheta''_{i+1})$.

$\tilde{e}_i = (e_j, \star_2^{\kappa(e_j)} \star_1^{\delta(e_j)} \bullet)$: Then $\Upsilon(\vartheta'_i, \pi_2(\tilde{e}'_i))$, since $\tilde{e}_i$ and $\tilde{e}'_i$ have the same label and μ is injective. Thus by induction and Lemma 6.14 we get
$\tilde{\mathcal{E}}'_{i+1} = \underline{Ref}_A^{eT}(\mathcal{E}'_{[e'_0]..[e'_{s_{i-1}-1}][\pi_1(\tilde{e}'_i)]}, \vartheta''_i[\pi_1(\tilde{e}'_i) \to \vartheta''_i(\pi_1(\tilde{e}'_i))_{[\pi_2(\tilde{e}'_i)]}])$.

Furthermore, there is $k < i$ such that $\tilde{e}'_k = (\pi_1(\tilde{e}'_i), \star_2^{\kappa(e)} \star_1^1 \bullet)$, since otherwise $\tilde{e}'_i \notin \mathrm{init}(\tilde{\mathcal{E}}'_i)$. From the injectivity of μ we obtain that $\pi_1(\tilde{e}_i) = \pi_1(\tilde{e}_k)$, since $\tilde{e}_k$ and $\tilde{e}'_k$ have the same label. Hence, $\tilde{\mathcal{E}}'_{i+1} = \underline{Ref}_A^{eT}(\mathcal{E}'_{[e'_0]..[e'_{s_i-1}]}, \vartheta''_{i+1})$, as required.

Otherwise: Then $\neg\Upsilon(\vartheta'_i, \pi_2(\tilde{e}'_i))$. Thus by induction and Lemma 6.14 it follows that $\tilde{\mathcal{E}}'_{i+1} = \underline{Ref}_A^{eT}(\mathcal{E}'_{[e'_0]..[e'_{s_{i-1}-1}]}, \vartheta''_i[\pi_1(\tilde{e}'_i) \to \vartheta''_i(\pi_1(\tilde{e}'_i))_{[\pi_2(\tilde{e}'_i)]}])$. Furthermore, there is $k \leq i$ such that $\tilde{e}'_k = (\pi_1(\tilde{e}'_i), \star_2^{\kappa(e)} \star_1^1 \bullet)$, since otherwise $\tilde{e}_i \notin \mathrm{init}(\tilde{\mathcal{E}}'_i)$. From the injectivity of μ we obtain that $\pi_1(\tilde{e}_i) = \pi_1(\tilde{e}_k)$. Hence, $\tilde{\mathcal{E}}'_{i+1} = \underline{Ref}_A^{eT}(\mathcal{E}'_{[e'_0]..[e'_{s_i-1}]}, \vartheta''_{i+1})$, as required.

From (6.2) we obtain that $\pi_1(\tilde{e}_i) = \pi_1(\tilde{e}_j) \Leftrightarrow \pi_1(\tilde{e}'_i) = \pi_1(\tilde{e}'_j)$. Hence, the function $f : \hat{E} \to E'$ with $f(e) = \pi_1(\tilde{e}'_i)$ whenever $e = \pi_1(\tilde{e}_i)$ is well defined, labeling preserving and injective.

Additionally, (6.2) becomes $\tilde{\mathcal{E}}'_i = \underline{Ref}_A^{eT}(\mathcal{E}'_{[f(e_0)]..[f(e_{s_{i-1}-1})]}, \vartheta''_i)$, where ϑ'' is defined by $\vartheta''_i(e') = \begin{cases} \vartheta'_i(f^{-1}(e')) & \text{if } f^{-1}(e') \text{ is defined} \\ \theta'(l'(e')) & \text{otherwise} \end{cases}$. Furthermore, by Lemma 6.14 we get

$$\Upsilon(\mathcal{E}'_{[f(e_0)]..[f(e_{s_{i-1}-1})]}, f(e_{s_{i-1}})) \Leftrightarrow \begin{array}{l} \exists j : \pi_1(\tilde{e}_j) = e_{s_{i-1}} \wedge \Upsilon(\tilde{\mathcal{E}}'_j, \tilde{e}'_j) \wedge \\ (l(\pi_1(\tilde{e}_j)) = \tau \vee \pi_2(\tilde{e}_j) = \star_2^{\kappa(e_{s_{i-1}})} \star_1^{\delta(e_{s_{i-1}})} \bullet) \end{array}.$$

Therefore, by (6.1) we obtain $\Upsilon(\mathcal{E}'_{[f(e_0)]..[f(e_{s_{i-1}-1})]}, f(e_{s_{i-1}})) \Leftrightarrow \nu_i = \surd$. From this and (6.2) it follows that $((f(e_i), \nu_i), f(\gamma_i))_{i \leq n} \in T^{ic}(\mathcal{E})$.

The other case can be shown by symmetrical arguments. □

Proof of Theorem 6.11: We verify the stronger statement claiming that there is a refinement function θ' such that $Ref_A^{eT}(\mathcal{E}, \theta') \sim_b Ref_A^{eT}(\mathcal{E}', \theta')$ implies $\mathcal{E} \sim_{FUI} \mathcal{E}'$.

Define $A \subseteq \mathcal{A}ct$ to be the set of all action-names occurring in $\mathcal{E}$ or in $\mathcal{E}'$, i.e. $A = \{l(e) | e \in E\} \cup \{l'(e') | e' \in E'\}$. Let $\mu : \{1,2\} \times A \times \mathbb{N} \to \mathcal{A}ct \backslash A$ be an injective function. Such a function exists. We define for all $a \in A$ an eTbes $\mathcal{E}_a$, which corresponds to the process algebra term $X = \mu(1,a,0); \mu(2,a,0) + X[f]$, where $f(\mu(i,a,n)) = \mu(i,a,n+1)$. In the definition the sequences $\star_2^n \star_1 \star_i \bullet$ are considered to be right bracketed and therefore to be elements of $\mathcal{U}$.

$$\begin{array}{rl} \mathcal{E}_a = (& \{ \star_2^n \star_1 \star_i \bullet \mid n \in \mathbb{N} \wedge i \in \{1,2\}\}, \\ & \{(\{\star_2^n \star_1 \star_1 \bullet \mid n \in \mathbb{N} \backslash \{j\}\} \cup \{\star_2^j \star_1 \star_i \bullet\}, \star_2^j \star_1 \star_i \bullet) \mid j \in \mathbb{N} \wedge i \in \{1,2\}\}, \\ & \{(\{\star_2^n \star_1 \star_1 \bullet\}, \star_2^n \star_1 \star_2 \bullet) \mid n \in \mathbb{N}\}, \\ & \{\{\star_2^n \star_1 \star_2 \bullet \mid n \in \mathbb{N}\}\}, \\ & \{(\star_2^n \star_1 \star_i \bullet, \mu(i,a,n)) \mid n \in \mathbb{N} \wedge i \in \{1,2\}\}) \end{array}$$

Define $\theta' : A \to \mathbf{ETBES}$ by $\theta'(a) = \mathcal{E}_a$. Furthermore, define $\mathcal{E}^{(a,n)}$ by

$$\mathcal{E}^{(a,n)} = (\{\star_2^n \star_1 \star_2 \bullet\}, \{(\{\star_2^n \star_1 \star_2 \bullet\}, \star_2^n \star_1 \star_2 \bullet)\}, \emptyset, \{\{\star_2^n \star_1 \star_2 \bullet\}\}, \{(\star_2^n \star_1 \star_2 \bullet, \mu(2,a,n))\}).$$

Let $\mathcal{R}_b$ be a strong bisimulation such that $(Ref_A^{eT}(\mathcal{E},\theta'), Ref_A^{eT}(\mathcal{E}',\theta')) \in \mathcal{R}_b$. Without loss of generality, R_b contains only elements which can be derived from $Ref_A^{eT}(\mathcal{E},\theta'), Ref_A^{eT}(\mathcal{E}',\theta')$. Furthermore, let $\kappa : \mathcal{U} \to \mathbb{N}$ be an isomorphism. We define the relation Ref_{FUI} by

$$\begin{aligned}\mathcal{R}_{FUI} = \{(\tilde{\mathcal{E}},\tilde{\mathcal{E}}',\tilde{f}) \mid \tilde{f} : \mathrm{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}) \to \mathrm{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}') \text{ is a labeling preserving isomorphism} \wedge \\ \forall \tilde{I} \in \mathcal{P}_{fin}(\mathrm{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}})) : \exists \tilde{J} \in \mathcal{P}_{fin}(\mathrm{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}})) : \tilde{I} \subseteq \tilde{J} \wedge \exists \tilde{\vartheta},\tilde{\vartheta}' : \\ \left(\forall e \in \tilde{E} : \tilde{\vartheta}(e) = \begin{cases} \mathcal{E}_{\tilde{l}(e)} & \text{if } e \in \tilde{E}\backslash\tilde{J} \\ \mathcal{E}^{(\tilde{l}(e),\kappa(\tilde{f}(e)))} & \text{if } e \in \tilde{J} \end{cases}\right) \wedge \\ \left(\forall e' \in \tilde{E}' : \tilde{\vartheta}'(e') = \begin{cases} \mathcal{E}_{\tilde{l}'(e')} & \text{if } e' \in \tilde{E}'\backslash\tilde{f}(\tilde{J}) \\ \mathcal{E}^{(\tilde{l}'(e'),\kappa(e'))} & \text{if } e' \in \tilde{f}(\tilde{J}) \end{cases}\right) \wedge \\ (\underline{Ref}_A^{eT}(\tilde{\mathcal{E}},\tilde{\vartheta}), \underline{Ref}_A^{eT}(\tilde{\mathcal{E}}',\tilde{\vartheta}')) \in \mathcal{R}_b\}\end{aligned}$$

In the following we show that Ref_{FUI} is a FUI-bisimulation. Therefore, suppose $(\tilde{\mathcal{E}},\tilde{\mathcal{E}}',\tilde{f}) \in Ref_{FUI}$.

Then $\tilde{f}$ is such a required isomorphism by definition. Now suppose $\tilde{e} \in \mathrm{init}(\tilde{\mathcal{E}})$ and $\tilde{I} \in \mathcal{P}_{fin}(\mathrm{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}))$. Then there is $\tilde{J}$ such that $\tilde{I} \cup \{\tilde{e}\} \subseteq \tilde{J}$, and $(\underline{Ref}_A^{eT}(\tilde{\mathcal{E}},\tilde{\vartheta}), \underline{Ref}_A^{eT}(\tilde{\mathcal{E}}',\tilde{\vartheta}')) \in \mathcal{R}_b$, where $\tilde{\vartheta},\tilde{\vartheta}'$ are the corresponding functions. We proceed by making a case analysis:

$\tilde{l}(e) \in \mathsf{Obs} \wedge \neg\Upsilon(\tilde{\mathcal{E}},\tilde{e})$: Then $\underline{Ref}_A^{eT}(\tilde{\mathcal{E}},\tilde{\vartheta}) \overset{\mu(2,\tilde{l}(\tilde{e}),\kappa(\tilde{f}(\tilde{e})))}{\longrightarrow} \underline{Ref}_A^{eT}(\tilde{\mathcal{E}}_{[\tilde{e}]},\tilde{\vartheta})$ by Lemma 6.14, since $\tilde{l}(e) \in A$. Furthermore, because of the fact that $\mathcal{R}_b$ is a strong bisimulation, there exists e' such that $(\underline{Ref}_A^{eT}(\tilde{\mathcal{E}}_{[\tilde{e}]},\tilde{\vartheta}), \underline{Ref}_A^{eT}(\tilde{\mathcal{E}}',\tilde{\vartheta}')_{[e']}) \in \mathcal{R}_b$ and $l_{\underline{Ref}_A^{eT}(\tilde{\mathcal{E}}',\tilde{\vartheta}')}(e') = \mu(2,\tilde{l}(\tilde{e}),\kappa(\tilde{f}(e)))$. Thus, $\pi_1(e') = f(\tilde{e})$ and $\tilde{l}'(\pi_1(e')) = \tilde{l}(\tilde{e})$ by the injectivity of κ. Moreover, we have $\underline{Ref}_A^{eT}(\tilde{\mathcal{E}}',\tilde{\vartheta}')_{[e']} = \underline{Ref}_A^{eT}(\tilde{\mathcal{E}}'_{[\tilde{e}']},\tilde{\vartheta}')$, where $\tilde{e}' = \pi_1(e')$.

$\tilde{l}(e) = \tau \wedge \neg\Upsilon(\tilde{\mathcal{E}},\tilde{e})$: Then $\underline{Ref}_A^{eT}(\tilde{\mathcal{E}},\tilde{\vartheta}) \xrightarrow{\tau} \underline{Ref}_A^{eT}(\tilde{\mathcal{E}}_{[\tilde{e}]},\tilde{\vartheta})$ by Lemma 6.14. From the fact that $\mathcal{R}_b$ is a strong bisimulation, there exists e' such that $(\underline{Ref}_A^{eT}(\tilde{\mathcal{E}}_{[\tilde{e}]},\tilde{\vartheta}), \underline{Ref}_A^{eT}(\tilde{\mathcal{E}}',\tilde{\vartheta}')_{[e']}) \in \mathcal{R}_b$ and $l_{\underline{Ref}_A^{eT}(\tilde{\mathcal{E}}',\tilde{\vartheta}')}(e') = \tau$. Thus, $\tilde{l}'(\tilde{e}') = \tau$ and $\underline{Ref}_A^{eT}(\tilde{\mathcal{E}}',\tilde{\vartheta}')_{[e']} = \underline{Ref}_A^{eT}(\tilde{\mathcal{E}}'_{[\tilde{e}']},\tilde{\vartheta}')$, where $\tilde{e}' = \pi_1(e')$.

The cases when $\Upsilon(\tilde{\mathcal{E}},\tilde{e})$ holds are carried out analogously. Furthermore, we have $\Upsilon(\tilde{\mathcal{E}},\tilde{e}) \Leftrightarrow \Upsilon(\tilde{\mathcal{E}}',\tilde{e}')$ by Lemma 6.14.

So it remains to find a function $\hat{f} : \mathrm{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}_{[\tilde{e}]}) = \mathrm{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}'_{[\tilde{e}']})$ that satisfies the necessary constraints. Therefore, define functions $\hat{f}_i$ and eTbes $\tilde{\mathcal{E}}_i, \tilde{\mathcal{E}}_i$ with $i \in \mathbb{N}$ as follows: $\hat{f}_0 = \tilde{f} \cap (\tilde{J} \cap \mathrm{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}_{[\tilde{e}]}))$, $\tilde{\mathcal{E}}_0 = \underline{Ref}_A^{eT}(\tilde{\mathcal{E}}_{[\tilde{e}]},\tilde{\vartheta})$ and $\tilde{\mathcal{E}}'_0 = \underline{Ref}_A^{eT}(\tilde{\mathcal{E}}'_{[\tilde{e}']},\tilde{\vartheta}')$.

for $2n+1$: If $\kappa^{-1}(n) \notin \mathrm{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}_{[\tilde{e}]})$ or $\hat{f}_{2n}(\kappa^{-1}(n))$ is defined, then $\hat{f}_{2n+1} = \hat{f}_{2n}$, $\tilde{\mathcal{E}}_{2n+1} = \tilde{\mathcal{E}}_{2n}$ and $\tilde{\mathcal{E}}'_{2n+1} = \tilde{\mathcal{E}}'_{2n}$.

If $\kappa^{-1}(n) \in \mathrm{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}_{[\tilde{e}]}) \wedge \hat{f}_{2n}(\kappa^{-1}(n))$ is undefined, then $(\kappa^{-1}(n), \star_2^n \star_1 \star_1 \bullet) \in \mathrm{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}_{2n})$. Hence, there is $\tilde{e}'_{2n}$ with the same label such that $(\tilde{\mathcal{E}}_{2n[\tilde{e}_{2n}]}, \tilde{\mathcal{E}}'_{2n[\tilde{e}'_{2n}]}) \in \mathcal{R}_b$, where $\tilde{e}_{2n} = (\kappa^{-1}(n), \star_2^n \star_1 \star_1 \bullet)$. Define $\hat{f}_{2n+1} = \hat{f}_{2n} \cup \{(\kappa^{-1}(n), \pi_1(\tilde{e}'_{2n})\}$, $\tilde{\mathcal{E}}_{2n+1} = \tilde{\mathcal{E}}_{2n[\tilde{e}_{2n}]}$ and $\tilde{\mathcal{E}}'_{2n+1} = \tilde{\mathcal{E}}'_{2n[\tilde{e}'_{2n}]}$.

for $2n+2$: If $\kappa^{-1}(n) \notin \mathrm{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}'_{[\tilde{e}']})$ or $\hat{f}_{2n}^{-1}(\kappa^{-1}(n))$ is defined, then $\hat{f}_{2n+1} = \hat{f}_{2n}$, $\tilde{\mathcal{E}}_{2n+1} = \tilde{\mathcal{E}}_{2n}$ and $\tilde{\mathcal{E}}'_{2n+1} = \tilde{\mathcal{E}}'_{2n}$.

If $\kappa^{-1}(n) \in \text{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}'_{[\tilde{e}']}) \wedge \hat{f}_{2n}^{-1}(\kappa^{-1}(n))$ is undefined, then $(\kappa^{-1}(n), \star_2^n \star_1 \star_1 \bullet) \in \text{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}'_{2n})$. Hence, there is $\tilde{e}_{2n}$ with the same label such that $(\tilde{\mathcal{E}}_{2n[\tilde{e}_{2n}]}, \tilde{\mathcal{E}}'_{2n[\tilde{e}'_{2n}]}) \in \mathcal{R}_b$, where $\tilde{e}'_{2n} = (\kappa^{-1}(n), \star_2^n \star_1 \star_1 \bullet)$. Define $\hat{f}_{2n+1} = \hat{f}_{2n} \cup \{(\pi_1(\tilde{e}_{2n}), \kappa^{-1}(n))\}$, $\tilde{\mathcal{E}}_{2n+1} = \tilde{\mathcal{E}}_{2n[\tilde{e}_{2n}]}$ and $\tilde{\mathcal{E}}'_{2n+1} = \tilde{\mathcal{E}}'_{2n[\tilde{e}'_{2n}]}$.

$\hat{f}$ is defined by $\hat{f} = \bigcup_{n \in \mathbb{N}} \hat{f}_n$.

$\hat{f}$ is a partial function, since $\hat{f}_i(\kappa^{-1}(n))$ is defined implies that its corresponding $\mathcal{E}_i$ does not possess events labeled by $\mu(1, \tilde{l}(\kappa^{-1}(n)), n)$. Hence, it is not defined twice. Moreover, $\hat{f}$ is a labeling preserving isomorphism between $\text{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}_{[\tilde{e}]}) \rightarrow \text{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}'_{[\tilde{e}']})$, since every event of both sides will be considered in the definition.

By definition, the restriction of $\hat{f}$ to $\tilde{J} \cap \text{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}_{[\tilde{e}]})$ is equal to f if it is restricted to this set. This restriction constraint also holds for $\hat{f}^{-1}$, since otherwise $\underline{Ref}_A^{eT}(\tilde{\mathcal{E}}_{[\tilde{e}]}, \tilde{\vartheta})$ and $\underline{Ref}_A^{eT}(\tilde{\mathcal{E}}'_{[\tilde{e}']}, \tilde{\vartheta}')$ can not be bisimilar.

It remains to prove that $(\tilde{\mathcal{E}}_{[\tilde{e}]}, \tilde{\mathcal{E}}'_{[\tilde{e}']}, \hat{f}) \in \mathcal{R}_{FUI}$. Therefore, let $\hat{I} \in \mathcal{P}_{fin}(\text{init}_{\mathsf{Obs}}(\tilde{\mathcal{E}}_{[\tilde{e}]}))$. Define $m = \max(\{2n+1 \in \mathbb{N} \mid \kappa^{-1}(n) \in \hat{I}\})$ and $\hat{J} = \text{dom}(\hat{f}_m)$. Furthermore, $(\tilde{\mathcal{E}}_m, \tilde{\mathcal{E}}'_m) \in \mathcal{R}_b$ and $\tilde{\mathcal{E}}_m, \tilde{\mathcal{E}}'_m$ satisfies the requirements. Hence, $(\tilde{\mathcal{E}}_{[\tilde{e}]}, \tilde{\mathcal{E}}'_{[\tilde{e}']}, \hat{f}) \in \mathcal{R}_{FUI}$.

The third requirement of the FUI-bisimulation follows by symmetrical arguments. Thus we have proved that Ref_{FUI} is a FUI-bisimulation.

The construction of a function f such that $(\mathcal{E}, \mathcal{E}', f) \in Ref_{FUI}$ is analogous to the construction of $\hat{f}$. Hence, $\mathcal{E} \sim_{FUI} \mathcal{E}'$. □

Chapter 7

Start-Based Choice together with End-Based Choice

In this chapter, a process algebra that contains action refinement together with three choice operators is introduced. Similar to Chapter 5, termination is determined by the 'final' executed action.

In order to give a true concurrency denotational semantics, event structures with two different conflict relations corresponding to the start-based and respectively to the end-based determination are introduced. An operational semantics, which corresponds to the denotational semantics, is presented.

Furthermore, the coarsest congruence with respect to bisimilarity is given. An axiom system that is sound and complete for finite state processes is investigated for this equivalence.

7.1 Motivation

Motivations of an end-based choice ($\oplus$), i.e. a choice that is triggered when an action finishes, have already been given in Section 1.3. It is reasonable that process algebras which contain an end-based choice operator and a parallel operator with action synchronization lead to some start-based choices, as it is discussed in Section 6.3.

A start-based choice ($+$), i.e. a choice that is triggered as soon as an action starts, is the usual kind of choice in process algebras, e.g. as in [10, 99, 133, 141, 174]. An example of a start-based choice is the following: a person standing in front of a fork has to decide immediately which direction she should take, i.e. she does not follow both directions at the same time and then make a decision depending on where she arrives.

Sometimes, it is also useful to have a choice ($\dot{+}$, called end-start choice) that is end-based and start-based triggered. More precisely, it is a choice that is triggered when its right process starts an action and it is triggered when its left process finishes an action. This is for example useful to model some special kinds of disruption, described as follows. Consider the process that executes a followed by b ($a;b$). This process should be allowed to be disrupted by the start of action c as long as action a runs, i.e. after the ending of a no disruption by c is allowed. This disruption is modeled by $(a;b)\dot{+}c$.

The usefulness of these three kinds of choices is illustrated in the following example.

Example 7.1 *Let us consider a nuclear power plant that consists of two reactors which can burn uranium or plutonium fuel rods. When the power plant gets the instructions to produce electricity, it warms up its two reactors and starts to burn either uranium or plutonium in the reactor that warms up first. The process can be disrupted during the warm-up phase. The phase during which the fuel rods are carried into the reactor is critical and may not be disrupted. This process is specified as follows in a process algebra setting. The actions considered are:*
$w_i \triangleq$ *'warm up the* i*-th reactor',*
$c \triangleq$ *'cancel the warm-up phase',*
$u_i \triangleq$ *'carry a uranium fuel rod to the* i*-th reactor', and*
$p_i \triangleq$ *'carry a plutonium fuel rod to the* i*-th reactor'.*
All actions are considered as abstractions of more concrete processes, i.e. they are refined by more concrete processes in a next specification level. Furthermore, N_i *denotes the specification of the working behavior of reactor* i*, which includes, for example, disruption possibilities. Then the nuclear power plant is specified by the process algebra expression*

$$P = \Big((w_1;(u_1+p_1);N_1) \oplus (w_2;(u_2+p_2);N_2)\Big) \dot{+} c.$$

7.2 Syntax

Let τ, Obs and Var be defined as in Section 3.2 and let $\mathcal{A}ct$ be defined as in Section 5.2.

The process algebra expressions $\mathsf{EXP}_{\mathsf{se}}$ ($_{\mathsf{s}} \triangleq$ start-based, $_{\mathsf{e}} \triangleq$ end-based) are defined by the following BNF-grammar.

$$B \;::=\; \mathbf{0} \mid a \mid B + B \mid B \dot{+} B \mid B \oplus B \mid B;B \mid B\|_A B \mid B\backslash\backslash A \mid B[(a \to B)^{a\in A}] \mid x$$

where $x \in \mathsf{Var}$, $a \in \mathcal{A}ct$ and $A \subseteq \mathsf{Obs}$. A *process with respect to* $\mathsf{EXP}_{\mathsf{se}}$ is a pair $\langle \mathrm{decl}, B\rangle$ consisting of a declaration $\mathrm{decl} : \mathsf{Var} \to \mathsf{EXP}_{\mathsf{se}}$ and an expression $B \in \mathsf{EXP}_{\mathsf{se}}$. Let $\mathsf{PA}_{\mathsf{se}}$ denote the set of all processes. We sometimes call an expression $B \in \mathsf{EXP}_{\mathsf{se}}$ also a process if decl is clear from the context.

The intuitive meaning of the end-based choice ($\oplus$) and of the end-start choice ($\dot{+}$) is given in Section 7.1. The intuitive meaning of the refinement expression $B[(a \to B_a)^{a\in A}]$ is that it behaves like process B except that every execution of action a in A is substituted by the behavior of B_a. The other operators are explained in Section 5.2.

7.3 Denotational Semantics for $\mathsf{PA}_{\mathsf{se}}$

7.3.1 Start-End Bundle Event Structures (SEBES)

Event structures that are used as denotational models of $\mathsf{PA}_{\mathsf{se}}$ have to handle a start-based and an end-based choice. Therefore, we introduce two relations for conflicts, one for the start-based

and another for the end-based conflict. We use the witness approach (Subsection 5.4.2) for the end-based conflict, since we follow the fa-approach, i.e. termination is determined by the action that is finally executed (see Chapter 5). The start-based conflict is modeled in the classical way, i.e. by a binary relation between events. Therefore, the *start-end bundle event structures* are a combination of closed bundle event structures (Definition 3.9) and extended termination bundle event structures (Definition 5.5):

Definition 7.2 (Start-End Bundle Event Structure) *A* start-end bundle event structure, sebes *for short, $\mathcal{E} = (E, \rightsquigarrow, \succ, \mapsto, T, l)$ is an element of $\mathcal{P}(\mathcal{U}) \times \mathcal{P}(\mathcal{U} \times \mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U}) \times \mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U}) \times \mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U})) \times (\mathcal{U} \rightharpoonup \mathcal{A}ct)$ such that*

- $\rightsquigarrow \subseteq E \times E$ *and* $\forall e \in E : \neg(e \rightsquigarrow e)$
- $\succ \subseteq \mathcal{P}(E) \times E$ *and* $\forall e \in E : \exists Z : Z \succ e$ *and* $\forall (Z, e) \in \succ : e \in Z$
- $\mapsto \subseteq \mathcal{P}(E) \times E$
- $T \subseteq \mathcal{P}(E)$ *and* $T \neq \emptyset$
- $\mathrm{dom}(l) = E$
- $\forall e \in E : \_ \succ e$ *is approximation closed with respect to* E
- $\forall e \in E : \_ \mapsto e$ *is approximation closed with respect to* E
- T *is approximation closed with respect to* E

Let $\mathbf{SEBES}$ *denote the set of all start-end bundle event structures.*

We call E the set of events, $\rightsquigarrow$ the (irreflexive) *(start) conflict* relation, $\succ$ the *(end) witness* relation, $\mapsto$ the *causality* relation, T the *termination set* and l the *action-labeling* function.

The intuitive meaning of the components of a sebes is given in Section 5.4 and in Section 3.3.

Remark 7.3 *A tuple $(E, \rightsquigarrow, \succ, \mapsto, T, l)$ is a sebes if and only if $(E, \succ, \mapsto, T, l)$ is an eTbes (Definition 5.5) and $\rightsquigarrow \subseteq E \times E$ and $\forall e \in E : \neg(e \rightsquigarrow e)$.*

Example 7.4 *Some sebes are depicted in Figure 7.1. The different components of a sebes is depicted as described in Example 5.6 and Subsection 3.3.1.*

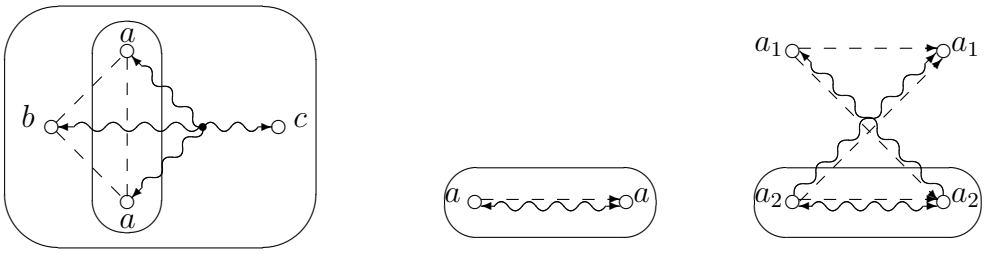
Hereafter, we consider $\mathcal{E}$ to be $(E, \rightsquigarrow, \succ, \mapsto, T, l)$, $\mathcal{E}_i$ to be $(E_i, \rightsquigarrow_i, \succ_i, \mapsto_i, T_i, l_i)$ and in general $\mathcal{E}$ to be $(E_\mathcal{E}, \rightsquigarrow_\mathcal{E}, \succ_\mathcal{E}, \mapsto_\mathcal{E}, T_\mathcal{E}, l_\mathcal{E})$. Furthermore, $\mathrm{init}(\mathcal{E})$ denotes the set of events which are ready to be executed and $\Upsilon(T, e)$ holds if and only if e is a termination event with respect to T, i.e. $\mathcal{E}$ terminates by executing e. Formally:

Definition 7.5 *Let $\mathcal{E}$ be a sebes. The set of* initial events *of $\mathcal{E}$ is defined by*

$$\mathrm{init}(\mathcal{E}) = \{e \in E \mid \neg(\exists X : X \mapsto e)\}.$$

The termination predicate $\Upsilon \subseteq \mathcal{P}(\mathcal{P}(\mathcal{U})) \times \mathcal{U}$ *is defined by*

$$\Upsilon(T, e) \iff \forall X \in T : e \in X.$$



$(a \oplus a)\|_{\{a\}}((a + b) \oplus c)$ $\quad a \dot{+} a$ $\quad (a \dot{+} a)[a \to a_1; a_2]$

Figure 7.1: Some Start-End Bundle Event Structures

Complete Partial Order

First, we present the definition and the properties of the restriction of a sebes, which are used to define an order on **SEBES**.

Definition 7.6 (Restriction of a sebes) *Suppose $\mathcal{E} \in \mathbf{SEBES}$ and $E' \subseteq E$. Then the* restriction of $\mathcal{E}$ to E'*, denoted by $\mathcal{E} \restriction E'$, is $(E', \rightsquigarrow', \succ', \mapsto', T', l')$ where*

$$\begin{array}{rcl} \rightsquigarrow' & = & \rightsquigarrow \cap (E' \times E') \\ \succ' & = & \{(Z \cap E', e') \mid e' \in E' \wedge Z \succ e'\} \\ \mapsto' & = & \{(X \cap E', e') \mid e' \in E' \wedge X \mapsto e'\} \\ T' & = & \{X \cap E' \mid X \in T\} \\ l' & = & l \restriction E' \end{array}$$

Lemma 7.7 *Let $\mathcal{E} \in \mathbf{SEBES}$ and $E' \subseteq E$. Then $\mathcal{E} \restriction E' \in \mathbf{SEBES}$.*

Proof: Is an immediate consequence of Corollary 2.18. □

Definition 7.8 (Order on SEBES) *Let $\mathcal{E}_i \in \mathbf{SEBES}$. Then $\mathcal{E}_1 \trianglelefteq \mathcal{E}_2$ if and only if $E_1 \subseteq E_2$ and $\mathcal{E}_1 = \mathcal{E}_2 \restriction E_1$.*

Remark 7.9 *Suppose $\mathcal{E}_1, \mathcal{E}_2 \in \mathbf{SEBES}$, then $\mathcal{E}_1 \trianglelefteq \mathcal{E}_2$ if and only if $\rightsquigarrow_1 = \rightsquigarrow_2 \cap (E_1 \times E_1)$ and $(E_1, \succ_1, \mapsto_1, T_1, l_1)$ is less than or equal to $(E_2, \succ_2, \mapsto_2, T_2, l_2)$ with respect to the order defined in Definition 5.18.*

Theorem 7.10 *The set of all sebes ordered by $\trianglelefteq$ is an ω-complete partial order, where the least upper bound of an ω-chain $(\mathcal{E}_i)_{i \in \mathbb{N}}$ is $\bigsqcup_i \mathcal{E}_i = (\bigcup_i E_i, \bigcup_i \rightsquigarrow_i, \succ, \mapsto, T, \bigcup_i l_i)$ with*

$$\begin{array}{rcl} \succ & = & \{(Z, e) \mid \forall k : e \in E_k \Rightarrow (Z \cap E_k) \succ_k e\} \\ \mapsto & = & \{(X, e) \mid \forall k : e \in E_k \Rightarrow (X \cap E_k) \mapsto_k e\} \\ T & = & \{X \mid \forall k : X \cap E_k \in T_k\} \end{array}$$

Proof: Is an immediate consequence of Theorem 5.19, Remark 7.3 and Remark 7.9. □

7.3.2 Operators on SEBES

Here, we present the operators on **SEBES** that will be used later to define the denotational semantics.

Definition 7.11 (Operators on SEBES) *Let $A \subseteq \mathsf{Obs}$. Then define*

$\widehat{+} : \mathbf{SEBES} \times \mathbf{SEBES} \to \mathbf{SEBES}$ *with* $\mathcal{E}_1 \widehat{+} \mathcal{E}_2 = (\tilde{E}, \tilde{\rightsquigarrow}, \tilde{\succ}, \tilde{\mapsto}, \tilde{T}, \tilde{l})$ *where*

$$
\begin{array}{rcl}
\tilde{E} & = & (\{\star_1\} \times E_1) \cup (\{\star_2\} \times E_2) \\
\tilde{\rightsquigarrow} & = & \{((\star_i, e_i), (\star_j, e_j)) \mid i \neq j \wedge e_i \in \mathrm{init}(E_i)\} \cup \{((\star_i, e), (\star_i, e')) \mid e \rightsquigarrow_i e'\} \\
\tilde{\succ} & = & \{((\{\star_i\} \times Z) \cup (\{\star_j\} \times \mathrm{init}(E_j)), (\star_i, e)) \mid Z \succ_i e \wedge i \neq j\} \\
\tilde{\mapsto} & = & \{(\{\star_i\} \times X, (\star_i, e)) \mid X \mapsto_i e\} \\
\tilde{T} & = & \{(\{\star_1\} \times X_1) \cup (\{\star_2\} \times X_2) \mid X_1 \in T_1 \wedge X_2 \in T_2\} \\
\tilde{l}((\star_i, e)) = l_i(e) &&
\end{array}
$$

$\widehat{+} : \mathbf{SEBES} \times \mathbf{SEBES} \to \mathbf{SEBES}$ *with* $\mathcal{E}_1 \widehat{+} \mathcal{E}_2 = (\tilde{E}, \tilde{\rightsquigarrow}, \tilde{\succ}, \tilde{\mapsto}, \tilde{T}, \tilde{l})$ *where*

$$
\begin{array}{rcl}
\tilde{E} & = & (\{\star_1\} \times E_1) \cup (\{\star_2\} \times E_2) \\
\tilde{\rightsquigarrow} & = & \{((\star_1, e_1), (\star_2, e_2)) \mid e_i \in \mathrm{init}(E_i)\} \cup \{((\star_i, e), (\star_i, e')) \mid e \rightsquigarrow_i e')\} \\
\tilde{\succ} & = & \{((\{\star_i\} \times Z) \cup (\{\star_j\} \times \mathrm{init}(E_j)), (\star_i, e)) \mid Z \succ_i e \wedge i \neq j\} \\
\tilde{\mapsto} & = & \{(\{\star_i\} \times X, (\star_i, e)) \mid X \mapsto_i e\} \\
\tilde{T} & = & \{(\{\star_1\} \times X_1) \cup (\{\star_2\} \times X_2) \mid X_1 \in T_1 \wedge X_2 \in T_2\} \\
\tilde{l}((\star_i, e)) = l_i(e) &&
\end{array}
$$

$\widehat{\oplus} : \mathbf{SEBES} \times \mathbf{SEBES} \to \mathbf{SEBES}$ *with* $\mathcal{E}_1 \widehat{\oplus} \mathcal{E}_2 = (\tilde{E}, \tilde{\rightsquigarrow}, \tilde{\succ}, \tilde{\mapsto}, \tilde{T}, \tilde{l})$ *where*

$$
\begin{array}{rcl}
\tilde{E} & = & (\{\star_1\} \times E_1) \cup (\{\star_2\} \times E_2) \\
\tilde{\rightsquigarrow} & = & \{((\star_i, e), (\star_i, e')) \mid e \rightsquigarrow_i e')\} \\
\tilde{\succ} & = & \{((\{\star_i\} \times Z) \cup (\{\star_j\} \times \mathrm{init}(E_j)), (\star_i, e)) \mid Z \succ_i e \wedge i \neq j\} \\
\tilde{\mapsto} & = & \{(\{\star_i\} \times X, (\star_i, e)) \mid X \mapsto_i e\} \\
\tilde{T} & = & \{(\{\star_1\} \times X_1) \cup (\{\star_2\} \times X_2) \mid X_1 \in T_1 \wedge X_2 \in T_2\} \\
\tilde{l}((\star_i, e)) = l_i(e) &&
\end{array}
$$

$\widehat{;} : \mathbf{SEBES} \times \mathbf{SEBES} \to \mathbf{SEBES}$ *with* $\mathcal{E}_1 \widehat{;} \mathcal{E}_2 = (\tilde{E}, \tilde{\rightsquigarrow}, \tilde{\succ}, \tilde{\mapsto}, \tilde{T}, \tilde{l})$ *where*

$$
\begin{array}{rcl}
\tilde{E} & = & (\{\star_1\} \times E_1) \cup (\{\star_2\} \times E_2) \\
\tilde{\rightsquigarrow} & = & \{((\star_i, e), (\star_i, e')) \mid e \rightsquigarrow_i e'\} \\
\tilde{\succ} & = & \{(\{\star_1\} \times (Z \cup X), (\star_1, e)) \mid Z \succ_1 e \wedge X \in T_1\} \cup \\
 & & \{(\{\star_2\} \times Z, (\star_2, e)) \mid Z \succ_2 e\} \\
\tilde{\mapsto} & = & \{(\{\star_i\} \times X, (\star_i, e)) \mid X \mapsto_i e\} \cup \\
 & & \{(\{\star_1\} \times X_1, (\star_2, e)) \mid e \in \mathrm{init}(\mathcal{E}_2) \wedge X_1 \in T_1\} \\
\tilde{T} & = & \{\{\star_2\} \times X_2 \mid X_2 \in T_2\} \\
\tilde{l}((\star_i, e)) = l_i(e) &&
\end{array}
$$

$\widehat{\|}_A : \mathbf{SEBES} \times \mathbf{SEBES} \to \mathbf{SEBES}$ *with* $\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2 = (\tilde{E}, \tilde{\leadsto}, \tilde{\succ}, \tilde{\mapsto}, \tilde{T}, \tilde{l})$ *where*

$$
\begin{array}{rcl}
\tilde{E} & = & (E_1^f \times \{\star\}) \cup (\{\star\} \times E_2^f) \cup E^s \\
E_i^f & = & \{e \in E_i \mid l_i(e) \notin A\} \\
E^s & = & \{(e_1, e_2) \in E_1 \times E_2 \mid l_1(e_1) = l_2(e_2) \in A\} \\
\tilde{\leadsto} & = & \{((e_1, e_2), (e_1', e_2')) \mid e_1 \leadsto_1 e_1' \vee e_2 \leadsto_2 e_2' \vee \\
 & & (e_1 = e_1' \neq \star \wedge e_2 \neq e_2') \vee (e_2 = e_2' \neq \star \wedge e_1 \neq e_1')\} \\
\tilde{\succ} & = & \{(\{(e_1', e_2') \in \tilde{E} \mid e_1' \in Z_1 \cup X_1\}, (e_1, \star)) \mid Z_1 \succ_1 e_1 \wedge X_1 \in T_1\} \cup \\
 & & \{(\{(e_1', e_2') \in \tilde{E} \mid e_2' \in Z_2 \cup X_2\}, (\star, e_2)) \mid Z_2 \succ_2 e_2 \wedge X_2 \in T_2\} \cup \\
 & & \{(\{(e_1', e_2') \mid e_1' \in Z_1 \cup X_1 \vee e_2' \in Z_2 \cup X_2\}, (e_1, e_2)) \mid \\
 & & \quad (e_1, e_2) \in E^s \wedge Z_1 \succ_1 e_1 \wedge X_1 \in T_1 \wedge Z_2 \succ_2 e_2 \wedge X_2 \in T_2\} \\
\tilde{\mapsto} & = & \{(\{(e_1', e_2') \in \tilde{E} \mid e_i' \in X_i\}, (e_1, e_2)) \mid X_i \mapsto_i e_i\} \\
\tilde{T} & = & \{\{(e_1, e_2) \in \tilde{E} \mid e_i \in X_i\} \mid X_i \in T_i\}
\end{array}
$$

$$
\tilde{l}((e_1, e_2)) = \begin{cases} l_1(e_1) & \textit{if } e_2 = \star \\ l_2(e_2) & \textit{otherwise} \end{cases}
$$

$\widehat{\backslash\backslash} A : \mathbf{SEBES} \to \mathbf{SEBES}$ *with* $\mathcal{E} \widehat{\backslash\backslash} A = \mathcal{E} \upharpoonright \{e \in E \mid l(e) \notin A\}$

$Ref_A^{se} : \mathbf{SEBES} \times (A \to \mathbf{SEBES}) \to \mathbf{SEBES}$ *by* $Ref_A^{eT}(\mathcal{E}, \theta) = (\tilde{E}, \tilde{\leadsto}, \tilde{\succ}, \tilde{\mapsto}, \tilde{T}, \tilde{l})$ *where*

$$
\begin{array}{rcl}
\tilde{E} & = & \{(e, \hat{e}) \mid e \in E \wedge l(e) \in A \wedge \hat{e} \in E_{\theta(l(e))}\} \cup \\
 & & \{(e, e) \in E \times E \mid l(e) \notin A\} \\
\tilde{\leadsto} & = & \{((e, \hat{e}), (e', \hat{e}')) \mid e \leadsto e' \vee (e = e' \wedge l(e) \in A \wedge \hat{e} \leadsto_{\theta(l(e))} \hat{e}')\} \\
\tilde{\succ} & = & \{(\tilde{Z}, (e, \hat{e})) \mid \exists Z : Z \succ e \wedge \exists f : Z \to \mathcal{P}(\mathcal{U}) : \\
 & & \quad (\forall e' \in Z : (l(e') \notin A \wedge f(e') = \{e'\}) \vee (l(e') \in A \wedge e' \neq e \wedge f(e') \in T_{\theta(l(e'))}) \vee \\
 & & \qquad (l(e') \in A \wedge e' = e \wedge \exists \hat{X} \in T_{\theta(l(e'))}, \hat{Z} : f(e') = \hat{Z} \cup \hat{X} \wedge \hat{Z} \succ_{\theta(l(e'))} \hat{e})) \wedge \\
 & & \quad \tilde{Z} = \{(e', \hat{e}') \in \tilde{E} \mid e' \in Z \wedge \hat{e}' \in f(e')\}\} \\
\tilde{\mapsto} & = & \{(\{e\} \times X', (e, \hat{e})) \mid l(e) \in A \wedge X' \mapsto_{\theta(l(e))} \hat{e}\} \cup \\
 & & \{(\tilde{X}, (e, \hat{e})) \mid (l(e) \in A \Rightarrow \hat{e} \in \mathrm{init}(\theta(l(e)))) \wedge \exists X : X \mapsto e \wedge \exists f : X \to \mathcal{P}(\mathcal{U}) : \\
 & & \quad (\forall e' \in X : (l(e') \notin A \wedge f(e') = \{e'\}) \vee (l(e') \in A \wedge f(e') \in T_{\theta(l(e'))})) \wedge \\
 & & \quad \tilde{X} = \{(e', \hat{e}') \in \tilde{E} \mid e' \in X \wedge \hat{e}' \in f(e')\}\} \\
\tilde{T} & = & \{\tilde{X} \mid \exists X \in T \wedge \exists f : X \to \mathcal{P}(\mathcal{U}) : (\forall e' \in X : (l(e') \notin A \wedge f(e') = \{e'\}) \vee \\
 & & \quad (l(e') \in A \wedge f(e') \in T_{\theta(l(e'))})) \wedge \tilde{X} = \{(e', \hat{e}') \in \tilde{E} \mid e' \in X \wedge \hat{e}' \in f(e')\}\}
\end{array}
$$

$$
\tilde{l}(e, \hat{e}) = \begin{cases} l(e) & \textit{if } l(e) \notin A \\ l_{\theta(l(e))}(\hat{e}) & \textit{if } l(e) \in A \end{cases}
$$

The definitions of these operators are similar to those presented in Subsection 5.4.3, Subsection 3.3.3 and Section 6.1. For comments on these operators, please consult the subsections mentioned.

Lemma 7.12 *All operators of Definition 7.11 are well defined, i.e. they really yield elements of* **SEBES**.

Proof: The conditions of the start conflict relation are easy to check. The rest is an immediate consequence of Lemma 5.21, Lemma 6.2 and Remark 7.3 □

Lemma 7.13 *All operators of Definition 7.11 are continuous with respect to* $\trianglelefteq$.

Proof: Analogous to the proof of Lemma 3.18. □

7.3.3 Denotational Meaning for $\mathsf{PA}_{\mathsf{se}}$

As in Subsection 3.3.4, we define the denotational semantics of expressions ($\mathsf{EXP}_{\mathsf{se}}$) relatively to variable assignments, i.e. functions from Var to $\mathbf{SEBES}$. Variable assignments are derived from declarations, which are used to define the denotational semantics of processes ($\mathsf{PA}_{\mathsf{se}}$).

Definition 7.14 *Let* $[\![\_]\!]_\_ : \mathsf{EXP}_{\mathsf{se}} \times (\mathsf{Var} \to \mathbf{SEBES}) \to \mathbf{SEBES}$ *be defined as follows (where* $\rho : \mathsf{Var} \to \mathbf{SEBES}$*)*

$$
\begin{array}{ll}
[\![\mathbf{0}]\!]_\rho = (\emptyset,\emptyset,\emptyset,\emptyset,\{\emptyset\},\emptyset) & [\![a]\!]_\rho = (\{\bullet\},\emptyset,\{(\{\bullet\},\bullet)\},\emptyset,\{\{\bullet\}\},\{(\bullet,a)\}) \\
[\![B_1 + B_2]\!]_\rho = [\![B_1]\!]_\rho \widehat{+} [\![B_2]\!]_\rho & [\![B_1 \dot{+} B_2]\!]_\rho = [\![B_1]\!]_\rho \widehat{\dot{+}} [\![B_2]\!]_\rho \\
[\![B_1 \oplus B_2]\!]_\rho = [\![B_1]\!]_\rho \widehat{\oplus} [\![B_2]\!]_\rho & [\![B_1; B_2]\!]_\rho = [\![B_1]\!]_\rho \widehat{;} [\![B_2]\!]_\rho \\
[\![B_1 \|_A B_2]\!]_\rho = [\![B_1]\!]_\rho \widehat{\|}_A [\![B_2]\!]_\rho & [\![B\backslash\backslash A]\!]_\rho = [\![B]\!]_\rho \widehat{\backslash\backslash} A \\
[\![B[(a \to B_a)^{a\in A}]]\!]_\rho = Ref_A^{eT}([\![B]\!]_\rho, (a \to [\![B_a]\!]_\rho)^{a\in A}) & \\
[\![x]\!]_\rho = \rho(x) &
\end{array}
$$

Remark 7.15 $[\![B]\!]_\_$ *is continuous for every* $B \in \mathsf{EXP}_{\mathsf{se}}$. *This follows analogously to Lemma 3.20, where Lemma 7.13 is used.*

Assume $\mathrm{decl} : \mathsf{Var} \to \mathsf{EXP}_{\mathsf{se}}$. Then define $\mathcal{F}_{\mathrm{decl}} : (\mathsf{Var} \to \mathbf{SEBES}) \to (\mathsf{Var} \to \mathbf{SEBES})$ with $\mathcal{F}_{\mathrm{decl}}(\rho)_{(x)} = [\![\mathrm{decl}(x)]\!]_\rho$. From Remark 7.15 it follows that $\mathcal{F}_{\mathrm{decl}}$ is continuous. Therefore, we get $\{\![\_]\!\} : (\mathsf{Var} \to \mathsf{EXP}_{\mathsf{se}}) \to (\mathsf{Var} \to \mathbf{SEBES})$ with $\{\![\mathrm{decl}]\!\} = \mathrm{fix}(\mathcal{F}_{\mathrm{decl}}) = \bigsqcup_n \mathcal{F}^n_{\mathrm{decl}}(\bot)$ is well defined from the cpo theory (Section 2.3).

Definition 7.16 (Denotational Semantics)

Define $[\![\_]\!] : \mathsf{PA}_{\mathsf{se}} \to \mathbf{SEBES}$ *by* $[\![\langle \mathrm{decl}, B\rangle]\!] = [\![B]\!]_{\{\![\mathrm{decl}]\!\}}$.

Example 7.17 *The denotational semantics of some processes is illustrated in Figure 7.1.*

7.4 Operational Semantics for $\mathsf{PA}_{\mathsf{se}}$

Similarly to the ST semantics, we distinguish between the start and the ending of actions and relate the ending uniquely to the start of the corresponding action. There are different techniques of encoding the history (executed events or started events) in operational semantics: via *static names* [6, 44], via *pointers* [49, 61, 95, 96], via *dynamic names* [44, 140] and via the *stack technique* [44]. We adapt the stack technique to our process algebra, since it has the following advantages:

- it produces finite transition systems for a wide class of processes. Hence bisimulation equivalence is decidable for this class of processes. Moreover, the transition system derived from the stack technique needs less states than the transition system derived from the other techniques.

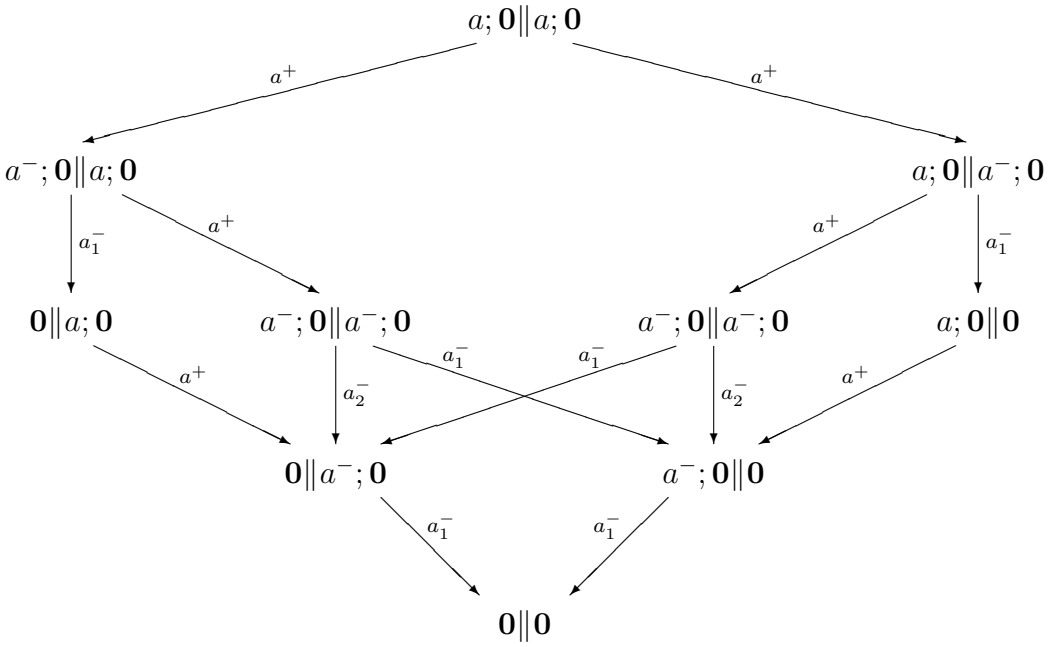


Figure 7.2: Illustration of the Stack Technique

- it is compositional, i.e. the transition system of a process can be derived from the transition system of its components. This has the advantage of simplifying the derivation of an axiomatization (see Section 7.7). More precisely, the standard axiom set developed by Milner [138] can be extended in order to obtain an axiomatization.
- it yields an appropriate method to handle refinement operators, as it can be seen in [98].

The intuitive idea behind the stack techniques is the following: the start of an action $a \in \mathsf{Obs}$ is denoted in the transition system by a^+; and the termination of an action $a \in \mathsf{Obs}$ is denoted by a_n^-, where the natural number n indicates that exactly $n-1$ many a-actions that are started after the start of the corresponding a_n^- action are still active. In other words, if an a-action starts at position t_s[1] and finishes with a_n^- at position t_f, then the number of the a-actions that are started after position t_s and that are not finished before position t_f is exactly $n-1$. An illustration that may help to understand this approach is given in Figure 7.2. The number n is called the *relative active number* of the action corresponding to a_n^-. We do not split internal actions (τ), i.e. they execute τ as usual. This is different to [44], where an internal action is split into two internal actions.

In order to define transition rules, we have to encode the information stating when the active actions were started in the expressions. For example, we have to know whether the left a-action started before the right a in $a^-;\mathbf{0}\|a^-;\mathbf{0}$ or not (compare with Figure 7.2). This can be encoded by extending each operator that allows more than one process to be active by the information indicating to which subcomponent (and also to which relative position of the subcomponents)

[1] In this context, positions are considered with respect to the execution order.

the n-th active a-action corresponds. In other words, the operator has to be extended by a function from $\mathsf{Obs} \times \mathbb{N}$ to the natural numbers combined with the possible subcomponents. For example, the parallel operator has to be extended by a function $\mathsf{Obs} \times \mathbb{N} \to \{l, r\}$, where l indicates the left process and r indicates the right process of the parallel operator. Such functions have to be changed dynamically after every execution. In order to simplify the dynamical changes, these functions are encoded by strings, i.e. by a function from Obs to the set of strings over the possible subcomponents. For example, the parallel operator is extended by a function $M : \mathsf{Obs} \to \{l, r\}^\star$. Then $a^-; \mathbf{0} \|_M a^-; \mathbf{0}$ where $M(a) = lr$ indicates that the a-action on the right hand side has been started before the a-action on the left hand side.

The process algebra expressions $\mathsf{EXP}^{\mathsf{O}}_{\mathsf{se}}$ for the operational semantics are defined by the following BNF-grammar.

$$\begin{aligned} C &::= B \mid b^- \mid C \dot{+} B \mid C \oplus_M C \mid C; B \mid C \|_{A,M} C \mid C \backslash\backslash_M A \mid \\ &\quad\; C[(a \to B)^{a \in A}, (a \to \vec{C})^{a \in \tilde{A}}]_{M_A} \\ \vec{C} &::= C \mid C \cdot \vec{C} \end{aligned}$$

where $B \in \mathsf{EXP}_{\mathsf{se}}$, $b \in \mathsf{Obs}$, $A \subseteq \mathsf{Obs}$, $\tilde{A} \in \mathcal{P}_{fin}(A)$, $M : \mathsf{Obs} \rightharpoonup \{l, r\}$ and $M_A : \mathsf{Obs} \rightharpoonup ((A \times \mathbb{N}^+) \cup \{0, \flat\})$, where $\rightharpoonup$ is defined in Subsection 2.1.2. We consider function $(a \to \vec{C})^{a \in \tilde{A}}$ to be the function $(a \to (\vec{C} \cup \varepsilon))^{a \in A}$ where a maps onto the empty string (ε) if and only if $a \notin \tilde{A}$. The symbols in the definition of $\mathsf{EXP}^{\mathsf{O}}_{\mathsf{se}}$, e.g. $\dot{+}$, are overloaded, since they are also used in the definition of $\mathsf{EXP}_{\mathsf{se}}$. Hence, the unique derivation of an expression of $\mathsf{EXP}^{\mathsf{O}}_{\mathsf{se}}$ is contradicted. Nevertheless, it does not harm our theory (both have the same transition rule) and therefore, we use the same symbols, especially to reduce the number of transition rules.

The intuitive meaning that differs from those given in Section 7.2 is the following: b^- indicates that action b is active, i.e. it has been started, but it has not been finished yet. It is able to execute b_1^-. The end-based choice operator is extended by M, since both components may contain active actions. This is not the case for the end-start choice, where only the left hand side may be an active process. The parallel operator is also extended by M. Actions of the synchronization set are not relevant in M, since they have to be uniquely executed on both sides. The restriction operator $C \backslash\backslash_M A$ is also extended by M, since it is used in some cases to maintain the M information of the parallel and the end-based choice expressions. The refinement operator $C[(a \to B_a)^{a \in A}, (a \to \vec{C}_a)^{a \in A}]_{M_A}$ contains additional strings of active processes ($\vec{C}_a$) for each $a \in A$ to encode the execution state of each active action in C. Furthermore, the refinement operator has to be extended by M_A to encode the corresponding position of the active actions, where (a, i) refers to the i-th position in $\vec{C}_a$, 0 refers to C and $\flat$ is a default value used when active actions are disrupted.

As mentioned in the beginning of this section, we have to adapt M and M_A after every action execution. Therefore, the following functions are defined, where $w \cdot \sigma$ and $\sigma \backslash i$ are defined in Subsection 2.1.1.

$$\_ \cdot \_ : (\mathsf{Obs} \times W) \times (\mathsf{Obs} \to W^\star) \to (\mathsf{Obs} \to W^\star) \text{ with}$$
$$([a, w] \cdot M)(b) = \begin{cases} w \cdot M(a) & \text{if } a = b \\ M(b) & \text{otherwise} \end{cases}$$

$$\_ \backslash \_ : (\mathsf{Obs} \to W^\star) \times (\mathsf{Obs} \times \mathbb{N}) \to (\mathsf{Obs} \to W^\star) \text{ with}$$
$$(M \backslash (a, i))(b) \simeq \begin{cases} M(a) \backslash i & \text{if } a = b \\ M(b) & \text{otherwise} \end{cases}$$

Function $\_\cdot\_$ puts value w in front of string $M(a)$ and is used when an action is started. Function $\_\backslash\_$ removes the i-th position in string $M(a)$ and is used when an action finishes.

In the case of the refinement operator $C[(a \to B_a)^{a\in A}, (a \to \vec{C}_a)^{a\in A}]_{M_A}$, further functions are needed. They are necessary, since the relative pointers have to be changed when the length of $\vec{C}_a$ changes, which is the case as soon as an action of A is started in C or a process in $\vec{C}_a$ terminates. These functions are given as follows, where the i-th element of a string σ is denoted by $\sigma[i]$.

$$\_\diamond\_ : (\mathsf{Obs} \to ((A \times \mathbb{N}^+) \cup \{0, \natural\})^\star) \times A \to (\mathsf{Obs} \to ((A \times \mathbb{N}^+) \cup \{0, \natural\})^\star) \text{ with}$$
$$(M_A \diamond a)(b)[k] \simeq \begin{cases} (a, i+1) & \text{if } M_A(b)[k] = (a, i) \\ M_A(b)[k] & \text{otherwise} \end{cases}$$

$$\_\dagger\_ : (\mathsf{Obs} \to ((A \times \mathbb{N}^+) \cup \{0, \natural\})^\star) \times (A \times \mathbb{N}^+) \to (\mathsf{Obs} \to ((A \times \mathbb{N}) \cup \{0, \natural\})^\star) \text{ with}$$
$$(M_A \dagger (a, j))(b)[k] \simeq \begin{cases} \natural & \text{if } M_A(b)[k] = (a, j) \\ (a, i-1) & \text{if } M_A(b)[k] = (a, i) \wedge j < i \\ M_A(b)[k] & \text{otherwise} \end{cases}$$

Function $\_\diamond a$ is used to shift the relative pointers of a by one when a process is added to $\vec{C}_a$. Function $\_\dagger(a, j)$ is used when the j-th process of $\vec{C}_a$ is removed. It reduces the relative pointers to $\vec{C}_a$ by 1 if it is greater than j.

It is also necessary to obtain the active number of an action a from M when the active number i of its subcomponent p is given. This is done by counting the elements that are in front of the i-th occurrence of p in $M(a)$. Formally:

Suppose $\sigma \in W^\star$ then define

$$\widehat{\sigma} : W \times \mathbb{N}^+ \rightharpoonup \mathbb{N}^+ \text{ with}$$
$$\widehat{\sigma}(w, i) \simeq \begin{cases} \min\{j \mid \sigma[j] = w\} & \text{if } i = 1 \\ \min\{j \mid \sigma[j] = w \wedge j > \widehat{\sigma}(w, i-1)\} & \text{otherwise} \end{cases}$$

We will also use the function that permutes r and l in M:

$$\overline{\ } : (\mathsf{Obs} \to \{l, r\}^\star) \to (\mathsf{Obs} \to \{l, r\}^\star) \text{ with}$$
$$\overline{M}(a)[i] = \begin{cases} l & \text{if } M(a)[i] = r \\ r & \text{if } M(a)[i] = l \end{cases}$$

Furthermore, $\vec{C}$ is considered to be a string. Hence, $\vec{C}[i]$ determines the i-th component if it exists, $\vec{C}\backslash i$ removes the i-th component and $\vec{C} \pm (i, C)$ replaces the i-th component by C if it exists.

The operational semantics for a process with respect to $\mathsf{EXP}_{\mathsf{se}}$ is given by a transition system where the set of states consists of the elements of $\mathsf{EXP}^{\mathsf{O}}_{\mathsf{se}}$, and the transition labels are $\mathcal{L}_{\mathsf{se}} = \{\tau, (\tau, \surd)\} \cup (\mathsf{Obs} \times (\{+\} \cup \mathbb{N} \cup (\mathbb{N} \times \{\surd\})))$.

Remark 7.18 *In [44] the internal action τ is split into its start and into its end. This is not necessary, since the internal action can not be refined and therefore the observer can not detect the start and the ending of this action.*

The transition rules $\longrightarrow^{c}_{\mathrm{decl}}$ with respect to $\mathrm{decl} : \mathsf{Var} \to \mathsf{EXP_{se}}$ are presented in Table 7.1, except for the transition rules of the parallel operator and those of the restriction operator, which are presented in Table 7.2, and except for the transition rules of the refinement operator, which are presented in Table 7.3. In these tables, γ denotes an element of $\mathcal{L}_{\mathsf{se}}$, $\perp$ denotes the function that maps every action to the empty string and D is either an element of $\mathsf{EXP}^{\mathsf{O}}_{\mathsf{se}}$ or of $\mathsf{EXP_{se}}$. Furthermore, we write a^+ instead of $(a,+)$, a_n^- instead of (a,n), $a_n^-\surd$ instead of $(a,n,\surd)$ and $\tau\surd$ instead of $(\tau,\surd)$. To reduce the number of rules, we use the notation where $\surd$ is in brackets. It means that either all bracketed $\surd$s are considered as $\surd$ or all bracketed $\surd$s are ignored. For example, rule Ch_3^e of Table 7.1 encodes the following two rules

$$\frac{C_1 \xrightarrow{\tau} C_1'}{\begin{array}{l} C_1 \oplus_M C_2 \xrightarrow{\tau} C_1' \backslash\backslash_M \emptyset \\ C_2 \oplus_M C_1 \xrightarrow{\tau} C_1' \backslash\backslash_{\overline{M}} \emptyset \end{array}} \qquad \frac{C_1 \xrightarrow{\tau\surd} C_1'}{\begin{array}{l} C_1 \oplus_M C_2 \xrightarrow{\tau\surd} C_1' \backslash\backslash_M \emptyset \\ C_2 \oplus_M C_1 \xrightarrow{\tau\surd} C_1' \backslash\backslash_{\overline{M}} \emptyset \end{array}}.$$

We give some comments on the transition rules: In Ac_1 an observable action starts by executing a^+ and results into the process that can finish this a by executing $a_1^-\surd$, which is described in Ac_2. The internal action is handled in rule Ac_3.

The transition rule for the start-based choice (Ch^s) is the standard one. Any action from the right term triggers the end-start choice (Ch_1^h), whereas only the non-starting actions of the left term trigger the end-start choice (Ch_2^h, Ch_3^h).

String $M(a)$ of the end-based choice expression is extended whenever an a-action starts (Ch_1^e). The case when an action finishes is described in Ch_2^e, where the corresponding relative active number is calculated by $\widehat{M(a)}(l,i)$. Furthermore, the relative active numbers have to be kept after this execution, which triggers the choice. This is done by using the restriction operator, where the information of the executed action is removed in M and no action is forbidden. In the case where the right process triggers the choice, M has to be transposed, i.e. $\overline{M}$ is taken, since the restriction operator considers the encoding by left to be active. The internal action is handled in a similar way (Ch_3^e). Rule Ch_1^e embeds $\mathsf{EXP_{se}}$ expressions by adding the information that no action is active.

The transition rules of the sequential operator (S_1, S_2) are the standard ones, i.e. the execution is given to the second process if and only if the first one terminates. The rules of the parallel operator have to change M, as it is done in the end-based choice rules. Furthermore, if one side terminates, it is removed (compare with the transition rules presented in Section 5.3). In this case, the relative active numbers are kept as in the case of the end-based choice, except that the actions of the synchronization set is forbidden. The restriction operator deals with M like the end-based and the parallel operator.

Rules R_1, R_2 and R_3 consider the cases when the execution of the action is independent of the refinement. The case when an action that is not in A starts, is considered in R_1. There, M_A is extended by the information that the started action results from the process that gets refined. In R_2 the ending of an action is considered, where M_A is modified in the usual way.

The case when an action that gets refined starts, is considered in R_4, R_5 and R_6. Rule R_6 considers the special case when the refinement terminates because of the execution of the internal action. In this case, also C has to finish the started action, which is described by $C' \xrightarrow{a_1^-(\surd)} C''$.

$$Ac_1 : \frac{a \in \mathsf{Obs}}{a \xrightarrow{a^+} a^-} \qquad Ac_2 : \frac{}{b^- \xrightarrow{b_1^-\surd} \mathbf{0}} \qquad Ac_3 : \frac{}{\tau \xrightarrow{\tau\surd} \mathbf{0}}$$

$$Ch^s : \frac{B_1 \xrightarrow{\gamma} C_1'}{\begin{array}{l} B_1 + B_2 \xrightarrow{\gamma} C_1' \\ B_2 + B_1 \xrightarrow{\gamma} C_1' \end{array}} \qquad Ch_1^h : \frac{B_2 \xrightarrow{\gamma} C_2'}{D_1 \dagger B_2 \xrightarrow{\gamma} C_2'}$$

$$Ch_2^h : \frac{D_1 \xrightarrow{a^+} C_1'}{D_1 \dagger B_2 \xrightarrow{a^+} C_1' \dagger B_2} \qquad Ch_3^h : \frac{D_1 \xrightarrow{\gamma} C_1' \quad \gamma \notin \mathsf{Obs} \times \{+\}}{D_1 \dagger B_2 \xrightarrow{\gamma} C_1'}$$

$$Ch_0^e : \frac{B_1 \oplus_\perp B_2 \xrightarrow{\gamma} C'}{B_1 \oplus B_2 \xrightarrow{\gamma} C'} \qquad Ch_1^e : \frac{C_1 \xrightarrow{a^+} C_1'}{\begin{array}{l} C_1 \oplus_M C_2 \xrightarrow{a^+} C_1' \oplus_{[a,l]\cdot M} C_2 \\ C_2 \oplus_M C_1 \xrightarrow{a^+} C_2 \oplus_{[a,r]\cdot M} C_1' \end{array}}$$

$$Ch_2^e : \frac{C_1 \xrightarrow{a_i^-(\surd)} C_1'}{\begin{array}{l} C_1 \oplus_M C_2 \xrightarrow{a^-_{\widehat{M(a)}(l,i)}(\surd)} C_1' \backslash\backslash_{M \setminus (a,\widehat{M(a)}(l,i))} \emptyset \\ C_2 \oplus_M C_1 \xrightarrow{a^-_{\widehat{M(a)}(r,i)}(\surd)} C_1' \backslash\backslash_{\overline{M} \setminus (a,\widehat{M(a)}(r,i))} \emptyset \end{array}} \qquad Ch_3^e : \frac{C_1 \xrightarrow{\tau(\surd)} C_1'}{\begin{array}{l} C_1 \oplus_M C_2 \xrightarrow{\tau(\surd)} C_1' \backslash\backslash_M \emptyset \\ C_2 \oplus_M C_1 \xrightarrow{\tau(\surd)} C_1' \backslash\backslash_{\overline{M}} \emptyset \end{array}}$$

$$S_1 : \frac{D_1 \xrightarrow{\alpha} C_1' \quad \alpha \in \{\tau\} \cup (\mathsf{Obs} \times \{+\}) \cup (\mathsf{Obs} \times \mathbb{N})}{D_1; B_2 \xrightarrow{\alpha} C_1'; B_2}$$

$$S_2 : \frac{D_1 \xrightarrow{\alpha\surd} C_1' \quad \alpha\surd \in (\{\tau\} \times \{\surd\}) \cup (\mathsf{Obs} \times \mathbb{N} \times \{\surd\})}{D_1; B_2 \xrightarrow{\alpha} B_2}$$

$$Rec : \frac{\mathrm{decl}(x) \xrightarrow{\gamma} C'}{x \xrightarrow{\gamma} C'}$$

Table 7.1: Transition Rules for $\longrightarrow^c_{\mathrm{decl}}$ (1)

Furthermore, φ does not change, since the started process is terminated. In the other cases (R_4 and R_5) the process that corresponds to a (B_a) is activated and therefore attached to φ. Furthermore, the pointers corresponding to a have to be increased by one in M_A, since their positions in φ is changed by one. This is done by $M_A \diamond a$. Furthermore, we only consider here starting actions and internal actions for B_a, since an expression of $\mathsf{EXP}_{\mathsf{se}}$ has no active actions. The case when an observable action of the refinement is started ($B_a \xrightarrow{b^+} C_a'$), is considered in R_4. The active function is adapted as usual.

Rules R_7, R_8 and R_9 consider the case when the active refinement (φ) executes an action that is different to a termination action, i.e. the process remains active. Before the active process may

$$P_0 : \frac{B_1 \|_{A,\perp} B_2 \xrightarrow{\gamma} C'}{B_1 \|_A B_2 \xrightarrow{\gamma} C'} \qquad P_1 : \frac{C_1 \xrightarrow{a^+} C_1' \quad a \notin A}{\begin{array}{l} C_1 \|_{A,M} C_2 \xrightarrow{a^+} C_1' \|_{A,[a,l]\cdot M} C_2 \\ C_2 \|_{A,M} C_1 \xrightarrow{a^+} C_2 \|_{A,[a,r]\cdot M} C_1' \end{array}}$$

$$P_2 : \frac{C_1 \xrightarrow{a_i^-} C_1' \quad a \notin A}{\begin{array}{l} C_1 \|_{A,M} C_2 \xrightarrow{a^-_{\widehat{M(a)}(l,i)}} C_1' \|_{A,M\setminus(a,\widehat{M(a)}(l,i))} C_2 \\ C_2 \|_{A,M} C_1 \xrightarrow{a^-_{\widehat{M(a)}(r,i)}} C_2 \|_{A,M\setminus(a,\widehat{M(a)}(r,i))} C_1' \end{array}}$$

$$P_3 : \frac{C_1 \xrightarrow{a_i^-\surd} C_1' \quad a \notin A}{\begin{array}{l} C_1 \|_{A,M} C_2 \xrightarrow{a^-_{\widehat{M(a)}(l,i)}} C_2 \backslash\backslash_{\overline{M}\setminus(a,\widehat{M(a)}(l,i))} A \\ C_2 \|_{A,M} C_1 \xrightarrow{a^-_{\widehat{M(a)}(r,i)}} C_2 \backslash\backslash_{M\setminus(a,\widehat{M(a)}(r,i))} A \end{array}}$$

$$P_4 : \frac{C_1 \xrightarrow{a^+} C_1' \quad C_2 \xrightarrow{a^+} C_2' \quad a \in A}{C_1 \|_{A,M} C_2 \xrightarrow{a^+} C_1' \|_{A,M} C_2'} \qquad P_5 : \frac{C_1 \xrightarrow{a_i^-\surd} C_1' \quad C_2 \xrightarrow{a_i^-\surd} C_2' \quad a \in A}{C_1 \|_{A,M} C_2 \xrightarrow{a_i^-\surd} \mathbf{0}}$$

$$P_6 : \frac{C_1 \xrightarrow{a_i^-} C_1' \quad C_2 \xrightarrow{a_i^-} C_2' \quad a \in A}{C_1 \|_{A,M} C_2 \xrightarrow{a_i^-} C_1' \|_{A,M} C_2'} \qquad P_7 : \frac{C_1 \xrightarrow{a_i^-\surd} C_1' \quad C_2 \xrightarrow{a_i^-} C_2' \quad a \in A}{\begin{array}{l} C_1 \|_{A,M} C_2 \xrightarrow{a_i^-} C_2' \backslash\backslash_{\overline{M}} A \\ C_2 \|_{A,M} C_1 \xrightarrow{a_i^-} C_2' \backslash\backslash_{M} A \end{array}}$$

$$P_8 : \frac{C_1 \xrightarrow{\tau} C_1'}{\begin{array}{l} C_1 \|_{A,M} C_2 \xrightarrow{\tau} C_1' \|_{A,M} C_2 \\ C_2 \|_{A,M} C_1 \xrightarrow{\tau} C_2 \|_{A,M} C_1' \end{array}} \qquad P_9 : \frac{C_1 \xrightarrow{\tau\surd} C_1'}{\begin{array}{l} C_1 \|_{A,M} C_2 \xrightarrow{\tau} C_2 \backslash\backslash_{\overline{M}} A \\ C_2 \|_{A,M} C_1 \xrightarrow{\tau} C_2 \backslash\backslash_{M} A \end{array}}$$

$$Res_0 : \frac{B \backslash\backslash_{\perp} A \xrightarrow{\gamma} C'}{B \backslash\backslash A \xrightarrow{\gamma} C'} \qquad Res_1 : \frac{C \xrightarrow{a^+} C' \quad a \notin A}{C \backslash\backslash_M A \xrightarrow{a^+} C' \backslash\backslash_{[a,l]\cdot M} A}$$

$$Res_2 : \frac{C \xrightarrow{a_i^-(\surd)} C' \quad a \notin A}{C \backslash\backslash_M A \xrightarrow{a^-_{\widehat{M(a)}(l,i)}(\surd)} C' \backslash\backslash_{M\setminus(a,\widehat{M(a)}(l,i))} A} \qquad Res_3 : \frac{C \xrightarrow{\tau(\surd)} C'}{C \backslash\backslash_M A \xrightarrow{\tau(\surd)} C' \backslash\backslash_M A}$$

Table 7.2: Transition Rules for $\longrightarrow^{c}_{\mathrm{decl}}$ (2)

For simplicity, let $\phi = (a \to B_a)^{a\in A}$ and $\varphi = (a \to \vec{C}_a)^{a\in A}$

$$R_0 : \frac{B[\phi, (a \to \varepsilon)^{a\in A})]_\perp \xrightarrow{\gamma} C'}{B[\phi] \xrightarrow{\gamma} C'} \qquad R_1 : \frac{C \xrightarrow{a^+} C' \quad a \notin A}{C[\phi,\varphi]_{M_A} \xrightarrow{a^+} C'[\phi,\varphi]_{[a,0]\cdot M_A}}$$

$$R_2 : \frac{C \xrightarrow{a_i^-(\surd)} C' \quad a \notin A \quad m = \widehat{M_A(a)}(0,i)}{C[\phi,\varphi]_{M_A} \xrightarrow{a_m^-(\surd)} C'[\phi,\varphi]_{M_A\setminus(a,m)}} \qquad R_3 : \frac{C \xrightarrow{\tau(\surd)} C'}{C[\phi,\varphi]_{M_A} \xrightarrow{\tau(\surd)} C'[\phi,\varphi]_{M_A}}$$

$$R_4 : \frac{C \xrightarrow{a^+} C' \quad a \in A \quad B_a \xrightarrow{b^+} C'_a}{C[\phi,\varphi]_{M_A} \xrightarrow{b^+} C'[\phi,\varphi[a \to C'_a \cdot \vec{C}_a]]_{[b,(a,1)]\cdot(M_A\diamond a)}}$$

$$R_5 : \frac{C \xrightarrow{a^+} C' \quad a \in A \quad B_a \xrightarrow{\tau} C'_a}{C[\phi,\varphi]_{M_A} \xrightarrow{\tau} C'[\phi,\varphi[a \to C'_a \cdot \vec{C}_a]]_{M_A\diamond a}}$$

$$R_6 : \frac{C \xrightarrow{a^+} C' \quad a \in A \quad B_a \xrightarrow{\tau\surd} C'_a \quad C' \xrightarrow{a_1^-(\surd)} C''}{C[\phi,\varphi]_{M_A} \xrightarrow{\tau(\surd)} C''[\phi,\varphi]_{M_A}}$$

$$R_7 : \frac{C \xrightarrow{a_i^-(\surd)} C' \quad a \in A \quad \vec{C}_a[i] \xrightarrow{b^+} C'_a}{C[\phi,\varphi]_{M_A} \xrightarrow{b^+} C[\phi,\varphi[a \to \vec{C}_a \pm (i, C'_a)]]_{[b,(a,i)]\cdot M_A}}$$

$$R_8 : \frac{C \xrightarrow{a_i^-(\surd)} C' \quad a \in A \quad \vec{C}_a[i] \xrightarrow{b_j^-} C'_a \quad m = \widehat{M_A(b)}((a,i),j)}{C[\phi,\varphi]_{M_A} \xrightarrow{b_m^-} C[\phi,\varphi[a \to \vec{C}_a \pm (i, C'_a)]]_{M_A\setminus(b,m)}}$$

$$R_9 : \frac{C \xrightarrow{a_i^-(\surd)} C' \quad a \in A \quad \vec{C}_a[i] \xrightarrow{\tau} C'_a}{C[\phi,\varphi]_{M_A} \xrightarrow{\tau} C[\phi,\varphi[a \to \vec{C}_a \pm (i, C'_a)]]_{M_A}}$$

$$R_{10} : \frac{C \xrightarrow{a_i^-(\surd)} C' \quad a \in A \quad \vec{C}_a[i] \xrightarrow{b_j^-\surd} C'_a \quad m = \widehat{M_A(b)}((a,i),j)}{C[\phi,\varphi]_{M_A} \xrightarrow{b_m^-(\surd)} C'[\phi,\varphi[a \to \vec{C}_a\backslash i]]_{(M_A\dagger(a,i))\setminus(b,m)}}$$

$$R_{11} : \frac{C \xrightarrow{a_i^-(\surd)} C' \quad a \in A \quad \vec{C}_a[i] \xrightarrow{\tau\surd} C'_a}{C[\phi,\varphi]_{M_A} \xrightarrow{\tau(\surd)} C'[\phi,\varphi[a \to \vec{C}_a\backslash i]]_{M_A\dagger(a,i)}}$$

Table 7.3: Transition Rules for $\longrightarrow^c_{\text{decl}}$ (3)

execute an action, one has to check that it is really an active action, i.e. that the corresponding action in C is not disrupted. This is done by $C \stackrel{a_i^-(\surd)}{\longrightarrow} C'$. Please note that C rather than C' is used in the resulting expression, since a_i^- has to remain active. The rest is handled as expected.

Rule R_{10} and R_{11} consider the case when the active process terminates. Similar as above it has to be checked that the corresponding action is still active. Furthermore, the active process is removed. As a result, some pointers corresponding to a in M_A have to be decreased by one, since their positions in φ are changed. Only those pointers which are greater than i have to be changed, since the other ones point to the correct position. This modification in M_A is done by $M_A \dagger (a, i)$. The handling of the relative active numbers is as usual. Please note that C' rather than C is used in the resulting process, since action a_i^- finishes when its active process terminates.

The refinement process $C[\phi, \varphi]_{M_A}$ terminates if and only if a termination action that is not refined is executed in C (rule R_2 and R_3) or if the refinement terminates and its corresponding action in C is a termination action (rule R_6, R_{10} and R_{11}).

Remark 7.19 *To reduce the state space, i.e. the expressions which are derived from a process, it is possible to modify the transition rules in such a way that every expression results in* $\mathbf{0}$ *if it executes a termination action. This is reasonable, since every expressions that results from the execution of a termination action is equivalent to the inactive process.*

7.5 Consistency of the Operational and the 3Denotational Semantics for $\mathsf{PA_{se}}$

In this section, we show that a transition system can be derived from a sebes such that the denotational and the operational semantics yield bisimilar transition systems.

We define the remainder of a sebes similarly to Definition 5.9 and Definition 3.14.

Definition 7.20 (Remainder of a sebes) *Let* $\mathcal{E} \in \mathbf{SEBES}$ *and* $e \in \mathrm{init}(\mathcal{E})$. *Then the remainder* $\mathcal{E}_{[e]}$ *of* $\mathcal{E}$ *is given by* $(E', \rightsquigarrow', \succ', \mapsto', T', l')$ *where*

$$
\begin{array}{rcl}
E' & = & \{e' \in E \mid \neg(e' \rightsquigarrow e) \wedge \exists Z : Z \succ e' \wedge e \notin Z\} \\
\rightsquigarrow' & = & \rightsquigarrow \cap (E' \times E') \\
\succ' & = & \{(Z \cap E', e') \mid e' \in E' \wedge Z \succ e' \wedge e \notin Z\} \\
\mapsto' & = & \{(X \cap E', e') \mid e' \in E' \wedge X \mapsto e' \wedge e \notin X\} \\
T' & = & \begin{cases} \{X \cap E' \mid X \in T \wedge e \notin X\} & \textit{if } \neg\Upsilon(T, e) \\ \{\emptyset\} & \textit{otherwise} \end{cases} \\
l' & = & l \restriction E'
\end{array}
$$

Lemma 7.21 *Let* $\mathcal{E} \in \mathbf{SEBES}$ *and* $e \in \mathrm{init}(\mathcal{E})$. *Then* $\mathcal{E}_{[e]} \in \mathbf{SEBES}$.

Proof: It is an immediate consequence of Remark 7.3 and Lemma 5.10, since the remainder correspondence to the other remainder on all relevant components. □

The remainder is used to obtain an interleaving semantics for **SEBES**:

Definition 7.22 *The transition relation* $\hookrightarrow \subseteq \mathbf{SEBES} \times \mathcal{A}ct \times \mathbf{SEBES}$ *is defined by* $\hookrightarrow = \{(\mathcal{E}, \gamma, \mathcal{E}_{[e]}) \mid \mathcal{E} \in \mathbf{SEBES} \wedge e \in \mathrm{init}(\mathcal{E}) \wedge (\Upsilon(T, e) \Rightarrow \gamma = l(e)\surd) \wedge (\neg\Upsilon(T, e) \Rightarrow \gamma = l(e))\}$.

The transition system derived from the denotational semantics is bisimilar to the operational semantics that is restricted to pure action execution. Formally:

Theorem 7.23 (Consistency) *Suppose* $\langle \mathrm{decl}, B\rangle \in \mathsf{PA}_{\mathsf{se}}$. *Then the two transition systems* $(\mathsf{EXP}^{\mathsf{O}}_{\mathsf{se}}, \mathcal{A}ct \cup (\mathcal{A}ct \times \{\surd\}), \twoheadrightarrow_{\mathrm{decl}}, B)$ *and* $(\mathbf{SEBES}, \mathcal{A}ct \cup (\mathcal{A}ct \times \{\surd\}), \hookrightarrow, [\![\langle \mathrm{decl}, B\rangle]\!])$ *are bisimilar,*

where $C \overset{a(\surd)}{\twoheadrightarrow}_{\mathrm{decl}} C' \Leftrightarrow \begin{cases} C \overset{a^+}{\longrightarrow}^c_{\mathrm{decl}} \overset{a_1^-(\surd)}{\longrightarrow}^c_{\mathrm{decl}} C' & \text{if } a \in \mathsf{Obs} \\ C \overset{\tau(\surd)}{\longrightarrow}^c_{\mathrm{decl}} C' & \text{if } a = \tau \end{cases}$.

Before we verify Theorem 7.23, which is done in Subsection 7.8.1, we show the stronger bisimilarity-result which says that the operational semantics is bisimilar to a $\mathcal{L}_{\mathsf{se}}$ labeled transition system derived from the denotational semantics. The transition system from $\mathbf{SEBES}$ that has labels from $\mathcal{L}_{\mathsf{se}}$ is defined as follows. We define the *start-remainder* with respect to event e in order to describe the system that remains after the start of event e.

Definition 7.24 (Start-remainder of a sebes) *Let* $\mathcal{E} \in \mathbf{SEBES}$ *and* $e \in \mathrm{init}(\mathcal{E})$. *Then define* $\mathcal{E}_{\langle e\rangle}$ *by* $\mathcal{E}_{\langle e\rangle} = \mathcal{E} \upharpoonright \{e' \in E \mid \neg(e' \rightsquigarrow e)\}$.

For the $\mathcal{L}_{\mathsf{se}}$ labeled transition system derived from $\mathbf{SEBES}$, it is necessary to save the information of the relative start of the events, i.e. we have to determine for each started event e how many active events labeled with the same action as e started after e. This is done by taking a $\mathbf{SEBES}$ combined with the set of partial functions from $\mathcal{U}$ to $\mathbb{N}^+$. The partial functions are also used to encode which actions are active, i.e. have already been started. Moreover, we restrict this set further to guarantee that:

- only a finite number of events may be active,
- every started event can not be start-based in conflict with another event, i.e. each start-based choice is triggered and
- each started event has a unique relative active number, i.e. for all a, n there exists at most one event e labeled with a that has exactly n active events labeled with a that started after e.

This is formalized by defining

$$\mathbf{SEBES}_{\mathcal{M}} = \{(\mathcal{E}, \mathcal{M}) \in \mathbf{SEBES} \times (\mathcal{U} \rightharpoonup \mathbb{N}^+) \mid \mathrm{dom}(\mathcal{M}) \in \mathcal{P}_{fin}(\mathrm{init}_{\mathsf{Obs}}(\mathcal{E})) \wedge \rightsquigarrow \cap(\mathcal{U} \times \mathrm{dom}(\mathcal{M})) = \emptyset \wedge \forall a \in \mathsf{Obs} : \mathcal{M} \upharpoonright \mathrm{init}_a(\mathcal{E})) \text{ is injective}\},$$

where $\mathrm{init}_A(\mathcal{E}) = \{e \in \mathrm{init}(\mathcal{E}) \mid l(e) \in A\}$ and $\mathrm{init}_a(\mathcal{E})$ is a short hand for $\mathrm{init}_{\{a\}}(\mathcal{E})$.

Furthermore, we need the following function to define the $\mathcal{L}_{\mathsf{se}}$ labeled transition system from $\mathbf{SEBES}$. Function $\_\widehat{\diamond}\_$ moves the relative active number of active actions by one. This function

is needed when a new event starts. Function $\_\widehat{\dagger}\_$ is used to reduce the relative active number of the affected active actions by one if an event finishes.

$$\begin{array}{l}\_\widehat{\diamond}\_ : (\mathcal{U} \rightharpoonup \mathbb{N}^+) \times \mathcal{P}(\mathcal{U}) \rightarrow (\mathcal{U} \rightharpoonup \mathbb{N}^+) \text{ with} \\ (\mathcal{M}\widehat{\diamond}E)(e) \simeq \left\{ \begin{array}{ll} \mathcal{M}(e)+1 & \text{if } e \in E \\ \mathcal{M}(e) & \text{otherwise} \end{array} \right.\end{array}$$

$$\begin{array}{l}\_\widehat{\dagger}\_ : (\mathcal{U} \rightharpoonup \mathbb{N}^+) \times (\mathcal{P}(\mathcal{U}) \times \mathbb{N}^+) \rightarrow (\mathcal{U} \rightharpoonup \mathbb{N}^+) \text{ with} \\ (\mathcal{M}\widehat{\dagger}(E,j))(e) \simeq \left\{ \begin{array}{ll} \mathcal{M}(e)-1 & \text{if } e \in E \wedge \mathcal{M}(e) > j \\ \mathcal{M}(e) & \text{otherwise} \end{array} \right.\end{array}$$

Now we are ready to present the $\mathcal{L}_{\mathsf{se}}$ labeled transition system from $\mathbf{SEBES}$.

Definition 7.25 *Suppose $\mathcal{E} = (\mathcal{E}, \mathcal{M}) \in \mathbf{SEBES}_{\mathcal{M}}$ then define $\mathcal{E}_{\rangle e \langle}$, which is an element of $\mathbf{SEBES}_{\mathcal{M}}$, by*

$$\mathcal{E}_{\rangle e \langle} \simeq \left\{ \begin{array}{ll} (\mathcal{E}_{[e]}, \mathcal{M} \restriction E_{\mathcal{E}_{[e]}}) & \textit{if } e \in \mathrm{init}_{\tau}(\mathcal{E}) \\ (\mathcal{E}_{[e]}, (\mathcal{M} \dagger (\mathrm{init}_{l(e)}(\mathcal{E}), \mathcal{M}(e))) \restriction E_{\mathcal{E}_{[e]}}) & \textit{if } e \in \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}) \wedge e \in \mathrm{dom}(\mathcal{M}) \\ (\mathcal{E}_{\langle e \rangle}, (\{(e,1)\} \cup (\mathcal{M} \diamond \mathrm{init}_{l(e)}(\mathcal{E}))) \restriction E_{\mathcal{E}_{\langle e \rangle}}) & \textit{if } e \in \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}) \wedge e \notin \mathrm{dom}(\mathcal{M}) \end{array} \right.$$

The transition relation $\hookrightarrow^c \subseteq \mathbf{SEBES}_{\mathcal{M}} \times \mathcal{L}_{\mathsf{se}} \times \mathbf{SEBES}_{\mathcal{M}}$ is defined by

$(\mathcal{E}, \mathcal{M}) \stackrel{\gamma}{\hookrightarrow^c} (\mathcal{E}', \mathcal{M}')$ if and only if

$$(\mathcal{E}', \mathcal{M}') = (\mathcal{E}, \mathcal{M})_{\rangle e \langle} \wedge \gamma = \left\{ \begin{array}{ll} \tau & \textit{if } e \in \mathrm{init}_{\tau}(\mathcal{E}) \wedge \neg \Upsilon(T, e) \\ \tau\surd & \textit{if } e \in \mathrm{init}_{\tau}(\mathcal{E}) \wedge \Upsilon(T, e) \\ l(e)^+ & \textit{if } e \in \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}) \wedge e \notin \mathrm{dom}(\mathcal{M}) \\ l(e)^-_{\mathcal{M}(e)} & \textit{if } e \in \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}) \wedge e \in \mathrm{dom}(\mathcal{M}) \wedge \neg \Upsilon(T, e) \\ l(e)^-_{\mathcal{M}(e)}\surd & \textit{if } e \in \mathrm{init}_{\mathsf{Obs}}(\mathcal{E}) \wedge e \in \mathrm{dom}(\mathcal{M}) \wedge \Upsilon(T, e) \end{array} \right. .$$

We have that the operational semantics is bisimilar to the $\mathcal{L}_{\mathsf{se}}$ labeled transition system derived from the denotational semantics.

Theorem 7.26 *Suppose $\langle \mathrm{decl}, B \rangle \in \mathsf{PA}_{\mathsf{se}}$. Then $(\mathbf{SEBES}_{\mathcal{M}}, \mathcal{L}_{\mathsf{se}}, \hookrightarrow^c, ([\![\langle \mathrm{decl}, B \rangle]\!], \bot))$ is bisimilar to $(\mathsf{EXP}^{\mathsf{O}}_{\mathsf{se}}, \mathcal{L}_{\mathsf{se}}, \longrightarrow^c_{\mathrm{decl}}, B)$.*

Proof: The proof is given in Subsection 7.8.1. □

7.6 Equivalence

We introduce an equivalence relation where differences between the start and the ending of an event is made. Furthermore, the ending of an event has to be related uniquely to its start. Therefore, we call this equivalence ST-equivalence (compare with Subsection 4.2.1). In its definition, we make use of the $\mathcal{L}_{\mathsf{se}}$ labeled transition systems defined in Section 7.4 and in Section 7.5.

Definition 7.27 *$\langle \mathrm{decl}, B\rangle$ and $\langle \mathrm{decl}', B'\rangle$ of $\mathsf{PA_{se}}$ are* ST-bisimilar *(or* ST-equivalent*), denoted $\langle \mathrm{decl}, B\rangle \sim_{ST} \langle \mathrm{decl}', B'\rangle$, if $(\mathsf{EXP}^{\mathsf{O}}_{\mathsf{se}}, \mathcal{L}_{\mathsf{se}}, \longrightarrow^{c}_{\mathrm{decl}}, B)$ and $(\mathsf{EXP}^{\mathsf{O}}_{\mathsf{se}}, \mathcal{L}_{\mathsf{se}}, \longrightarrow^{c}_{\mathrm{decl}'}, B')$ are bisimilar (Definition 2.5).*

Two sebes $\mathcal{E}$ and $\mathcal{E}'$ are ST-bisimilar*, denoted $\mathcal{E} \sim_{ST} \mathcal{E}'$, if $(\mathbf{SEBES}_{\mathcal{M}}, \mathcal{L}_{\mathsf{se}}, \hookrightarrow^{c}, (\mathcal{E}, \bot))$ and $(\mathbf{SEBES}_{\mathcal{M}}, \mathcal{L}_{\mathsf{se}}, \hookrightarrow^{c}, (\mathcal{E}', \bot))$ are bisimilar.*

From Theorem 7.26 we obtain the correspondence of the ST-equivalences on $\mathsf{PA_{se}}$ and $\mathbf{SEBES}$, i.e. for all $\langle \mathrm{decl}, B\rangle, \langle \mathrm{decl}', B'\rangle \in \mathsf{PA_{se}}$ we have $\langle \mathrm{decl}, B\rangle \sim_{ST} \langle \mathrm{decl}', B'\rangle$ if and only if $[\![\langle \mathrm{decl}, B\rangle]\!] \sim_{ST} [\![\langle \mathrm{decl}', B'\rangle]\!]$.

As in the standard event structures settings, ST-equivalence is the coarsest congruence for our refinement operator with respect to bisimulation equivalence.

Theorem 7.28 *ST-equivalence is a congruence for the refinement operator Ref^{se}, i.e. $\mathcal{E} \sim_{ST} \mathcal{E}' \wedge \forall a \in A : \theta(a) \sim_{ST} \theta'(a)$ implies that $Ref^{se}_A(\mathcal{E}, \theta) \sim_{ST} Ref^{se}_A(\mathcal{E}', \theta')$. Moreover, it is also a congruence for the operators $\widehat{+}$, $\widehat{+\!\!\!\cdot}$, $\widehat{\oplus}$, $\widehat{;}$, $\widehat{\|}_A$ and $\widehat{\backslash\backslash A}$, which are defined in Definition 7.11.*

Proof: The proof is given in Subsection 7.8.2. □

Theorem 7.29 *ST-equivalence is the coarsest congruence for Ref^{se} with respect to bisimulation equivalence, i.e. if $\forall A, \theta : Ref^{se}_A(\mathcal{E}, \theta) \sim_b Ref^{se}_A(\mathcal{E}', \theta)$ then $\mathcal{E} \sim_{ST} \mathcal{E}'$.*

Proof: The proof is given in Subsection 7.8.2. □

Remark 7.30 *All transition rules for $\longrightarrow^{c}_{\mathrm{decl}}$ are in* panth*-format [171]. The transition rules are also complete [9], since no negative literates are used. Hence, ST-equivalence is a congruence for all expression constructions of $\mathsf{PA_{se}}$ [9].*

This fact does not immediately follow from Theorem 7.28, since processes can contain recursion.

7.7 Axiomatization

In this section, we present an axiom system for ST-equivalence with respect to $\mathsf{PA_{se}}$. We follow the idea of [44], where some modifications are made, since the declaration technique is used. Further modifications are necessary, since termination is determined by the final executed action and since the internal action τ is not split into its start and end. We also use a different definition of the refinement operator taken from [98].

Following the approach mentioned, we extend the syntax by further expressions, for example by the left merge ($\|^{-}$) and the synchronization merge ($|$) operators [8, 25]. We also add expressions considering the start and ending of actions. The equality of $\mathsf{PA_{se}}$ processes can be concluded from the equality of the newly introduced processes, since they are a conservative extension [9] of $\mathsf{PA_{se}}$.

The process algebra expressions $\mathsf{EXP}^{\mathsf{Ax}}_{\mathsf{se}}$ are defined by the following BNF-grammar.

$$\begin{aligned} H \;::=\; & \mathbf{0} \mid a \mid b^+; H \mid b^- \mid b^-_q \mid b^-_q\surd; H \mid \tau\surd; H \mid H + H \mid H \dot{+} H \mid H \dot{+}^- H \mid \\ & H \oplus_M H \mid H \oplus^-_M H \mid H; H \mid H[(a \to H)^{a\in A}, (a \to \vec{H})^{a\in\tilde{A}}]_{M_A} \mid \\ & H[(a \to H)^{a\in A}, (a \to H)^{a\in A}, (a \to \vec{H})^{a\in\tilde{A}}]^-_{M_A} H \mid \\ & H \|_{A,M} H \mid H \|^-_{A,M} H \mid H|_{A,M} H \mid H \backslash\backslash_M A \mid x \\ \vec{H} \;::=\; & H \mid H \cdot \vec{H} \end{aligned}$$

where $x \in \mathsf{Var}$, $a \in \mathcal{A}ct$, $b \in \mathsf{Obs}$, $A \subseteq \mathsf{Obs}$, $\tilde{A} \in \mathcal{P}_{fin}(A)$, $M : \mathsf{Obs} \rightharpoonup \{l, r\}$ and $M_A : \mathsf{Obs} \rightharpoonup ((A \times \mathbb{N}^+) \cup \{0, b\})$. As in Section 7.4, we consider function $(a \to \vec{H})^{a\in\tilde{A}}$ to be function $(a \to (\vec{H} \cup \varepsilon))^{a\in A}$ where a maps to ε if and only if $a \notin \tilde{A}$. A *process with respect to* $\mathsf{EXP}^{\mathsf{Ax}}_{\mathsf{se}}$ is a pair $\langle \mathrm{decl}, H\rangle$ consisting of a declaration $\mathrm{decl} : \mathsf{Var} \to \mathsf{EXP}^{\mathsf{Ax}}_{\mathsf{se}}$ and an expression $H \in \mathsf{EXP}^{\mathsf{Ax}}_{\mathsf{se}}$. Let $\mathsf{PA}^{\mathsf{Ax}}_{\mathsf{se}}$ denote the set of all processes. We sometimes call an expression $H \in \mathsf{EXP}^{\mathsf{Ax}}_{\mathsf{se}}$ also a process if decl is clear from the context.

The intuitive meaning of the new expressions are as follows: $b^+; H$ is the process that evolves into H by starting a b-action. Process b^-_q terminates the active b-action that started before the last $q-1$ active b-actions. $b^-_q\surd; H$ is similar to b^-_q, except that also the whole process is terminated, i.e. it executes $b^-_q\surd$ and evolves into $\mathbf{0}$ (and not into H).

The end-start ($\dot{+}$) and the end choice ($\oplus$) together with the parallel operator are extended with left merge ones ($\dot{+}^-$, $\oplus^-_M$, $\|^-_{A,M}$). In these operators the left process has to execute the next action. A synchronization merge operator for the parallel operator is also introduced ($|_{A,M}$). Here, the next action that is executed has to be obtained from communication.

Process $H_1[(a \to H_a)^{a\in A}, (a \to H'_a)^{a\in A}, (a \to \vec{H}_a)^{a\in\tilde{A}}]^-_{M_A} H_2$ has the same behavior as $H_2[(a \to H'_a)^{a\in A}, (a \to \vec{H}_a)^{a\in\tilde{A}}]_{M_A}$ except that H_2 and $(a \to H'_a)^{a\in A}$ are replaced by H_1 (respectively by $(a \to H_a)^{a\in A}$) in the evolved process. That means, H_2 and $(a \to H'_a)^{a\in A}$ are used to determine the next actions and H_1 and $(a \to H_a)^{a\in A}$ are used to determine the future behavior of the process. Please note that H_1 is only used for the future behavior if H_2 does not execute the action. This new refinement expression is introduced for the axiom system, since we have to expand the refinement to determine the next action of the refinement (expand $U[\phi[a \to T + R], \varphi]_{M_A}$ to $U[\phi[a \to T], \varphi]_{M_A} + U[\phi[a \to R], \varphi]_{M_A}$). But if we use this expansion, we forget the original refinement function, which is essential for the future behavior, for example if U is equal to $a; a$. Therefore, the original refinement function is kept in the expression and only the copy of this function may expand further. Similar arguments hold for H_1, since we have to check, whether an action is still active, which destroys the original process.

The operational semantics ($\longrightarrow^z$) for $\mathsf{PA}^{\mathsf{Ax}}_{\mathsf{se}}$ is given as follows. The transition rules for the operators that exist in $\mathsf{EXP}_{\mathsf{se}}$ are the old ones presented in Table 7.1, Table 7.2 and Table 7.3. The transition rules for the newly introduced operators are given in Table 7.4 and Table 7.5.

We give some comments on the transition rules. One may expect that the processes in rule Ac_6 and Ac_7 should evolve into H. This is not reasonable, since the execution of a termination action has to lead to an inactive process. This is also respected in the axioms $b^-_q = b^-_q\surd; T$ and $\tau = \tau\surd; T$ from Table 7.6, where the processes evolve to $\mathbf{0}$ for all these cases. The other rules are just as expected.

ST-equivalence is adapted to $\mathsf{PA}^{\mathsf{Ax}}_{\mathsf{se}}$ in the straightforward way, i.e. $\langle \mathrm{decl}, H\rangle$ and $\langle \mathrm{decl}', H'\rangle$ are *ST-bisimilar* (or *ST-equivalent*), denoted $\langle \mathrm{decl}, H\rangle \sim_{ST} \langle \mathrm{decl}', H'\rangle$, if $(\mathsf{EXP}^{\mathsf{Ax}}_{\mathsf{se}}, \mathcal{L}_{\mathsf{se}}, \longrightarrow^z_{\mathrm{decl}}, H)$

$$Ac_4 : \frac{}{b^+; H \xrightarrow{b^+} H} \qquad Ac_5 : \frac{}{b_q^- \xrightarrow{b_q^-\surd} \mathbf{0}} \qquad Ac_6 : \frac{}{b_q^-\surd; H \xrightarrow{b_q^-\surd} \mathbf{0}}$$

$$Ac_7 : \frac{}{\tau\surd; H \xrightarrow{\tau\surd} \mathbf{0}}$$

$$Ch_1^{Lh} : \frac{H_1 \xrightarrow{a^+} H_1'}{H_1 \frown^- H_2 \xrightarrow{a^+} H_1' \frown H_2} \qquad Ch_2^{Lh} : \frac{H_1 \xrightarrow{\gamma} H_1' \quad \gamma \notin \mathsf{Obs} \times \{+\}}{H_1 \frown^- H_2 \xrightarrow{\gamma} H_1'}$$

$$Ch_1^{Le} : \frac{H_1 \xrightarrow{a^+} H_1'}{H_1 \oplus_M^- H_2 \xrightarrow{a^+} H_1' \oplus_{[a,l]\cdot M} H_2}$$

$$Ch_2^{Le} : \frac{H_1 \xrightarrow{a_i^-(\surd)} H_1'}{H_1 \oplus_M^- H_2 \xrightarrow{a^-_{\widehat{M(a)}(l,i)}(\surd)} H_1' \backslash\backslash_{M\setminus(a,\widehat{M(a)}(l,i))} \emptyset} \qquad Ch_3^{Le} : \frac{H_1 \xrightarrow{\tau(\surd)} H_1'}{H_1 \oplus_M^- H_2 \xrightarrow{\tau(\surd)} H_1' \backslash\backslash_M \emptyset}$$

$$P_1^L : \frac{H_1 \xrightarrow{a^+} H_1' \quad a \notin A}{H_1 \|_{A,M}^- H_2 \xrightarrow{a^+} H_1' \|_{A,[a,l]\cdot M} H_2}$$

$$P_2^L : \frac{H_1 \xrightarrow{a_i^-} H_1' \quad a \notin A}{H_1 \|_{A,M}^- H_2 \xrightarrow{a^-_{\widehat{M(a)}(l,i)}} H_1' \|_{A,M\setminus(a,\widehat{M(a)}(l,i))} H_2}$$

$$P_3^L : \frac{H_1 \xrightarrow{a_i^-\surd} H_1' \quad a \notin A}{H_1 \|_{A,M}^- H_2 \xrightarrow{a^-_{\widehat{M(a)}(l,i)}} H_2 \backslash\backslash_{\overline{M}\setminus(a,\widehat{M(a)}(l,i))} A}$$

$$P_4^L : \frac{H_1 \xrightarrow{\tau} H_1'}{H_1 \|_{A,M}^- H_2 \xrightarrow{\tau} H_1' \|_{A,M} H_2} \qquad P_5^L : \frac{H_1 \xrightarrow{\tau\surd} H_1'}{H_1 \|_{A,M}^- H_2 \xrightarrow{\tau} H_2 \backslash\backslash_{\overline{M}} A}$$

$$P_1^S : \frac{H_1 \xrightarrow{a^+} H_1' \quad H_2 \xrightarrow{a^+} H_2' \quad a \in A}{H_1|_{A,M} H_2 \xrightarrow{a^+} H_1' \|_{A,M} H_2'} \qquad P_2^S : \frac{H_1 \xrightarrow{a_i^-\surd} H_1' \quad H_2 \xrightarrow{a_i^-\surd} H_2' \quad a \in A}{H_1|_{A,M} H_2 \xrightarrow{a_i^-\surd} \mathbf{0}}$$

$$P_3^S : \frac{H_1 \xrightarrow{a_i^-} H_1' \quad H_2 \xrightarrow{a_i^-} H_2' \quad a \in A}{H_1|_{A,M} H_2 \xrightarrow{a_i^-} H_1' \|_{A,M} H_2'} \qquad P_4^S : \frac{H_1 \xrightarrow{a_i^-\surd} H_1' \quad H_2 \xrightarrow{a_i^-} H_2' \quad a \in A}{\begin{array}{c} H_1|_{A,M} H_2 \xrightarrow{a_i^-} H_2' \backslash\backslash_{\overline{M}} A \\ H_2|_{A,M} H_1 \xrightarrow{a_i^-} H_2' \backslash\backslash_M A \end{array}}$$

Table 7.4: Transition Rules for $\longrightarrow^z_{\mathrm{decl}}$ (1)

For simplicity, let $\phi' = (a \to H_a)^{a \in A}$ and $\varphi = (a \to \vec{H}_a)^{a \in A}$

$$R'_1 : \frac{H \xrightarrow{a^+} H' \quad a \notin A}{D[\phi, \phi', \varphi]^-_{M_A} H \xrightarrow{a^+} H'[\phi, \varphi]_{[a,0]\cdot M_A}}$$

$$R'_2 : \frac{H \xrightarrow{a_i^-(\surd)} H' \quad a \notin A \quad m = \widehat{M_A(a)}(0,i)}{D[\phi, \phi', \varphi]^-_{M_A} H \xrightarrow{a_m^-(\surd)} H'[\phi, \varphi]_{M_A \backslash (a,m)}}$$

$$R'_3 : \frac{H \xrightarrow{\tau(\surd)} H'}{D[\phi, \phi', \varphi]^-_{M_A} H \xrightarrow{\tau(\surd)} H'[\phi, \varphi]_{M_A}}$$

$$R'_4 : \frac{H \xrightarrow{a^+} H' \quad a \in A \quad H_a \xrightarrow{b^+} H'_a}{D[\phi, \phi', \varphi]^-_{M_A} H \xrightarrow{b^+} H'[\phi, \varphi[a \to H'_a \cdot \vec{H}_a]]_{[b,(a,1)]\cdot(M_A \diamond a)}}$$

$$R'_5 : \frac{H \xrightarrow{a^+} H' \quad a \in A \quad H_a \xrightarrow{\tau} H'_a}{D[\phi, \phi', \varphi]^-_{M_A} H \xrightarrow{\tau} H'[\phi, \varphi[a \to H'_a \cdot \vec{H}_a]]_{M_A \diamond a}}$$

$$R'_6 : \frac{H \xrightarrow{a^+} H' \quad a \in A \quad H_a \xrightarrow{\tau\surd} H'_a \quad H' \xrightarrow{a_1^-(\surd)} H''}{D[\phi, \phi', \varphi]^-_{M_A} H \xrightarrow{\tau(\surd)} H''[\phi, \varphi]_{M_A}}$$

$$R'_7 : \frac{H \xrightarrow{a_i^-(\surd)} H' \quad a \in A \quad \vec{H}_a[i] \xrightarrow{b^+} H'_a}{D[\phi, \phi', \varphi]^-_{M_A} H \xrightarrow{b^+} D[\phi, \varphi[a \to \vec{H}_a \pm (i, H'_a)]]_{[b,(a,i)]\cdot M_A}}$$

$$R'_8 : \frac{H \xrightarrow{a_i^-(\surd)} H' \quad a \in A \quad \vec{H}_a[i] \xrightarrow{b_j^-} H'_a \quad m = \widehat{M_A(b)}((a,i),j)}{D[\phi, \phi', \varphi]^-_{M_A} H \xrightarrow{b_m^-} D[\phi, \varphi[a \to \vec{H}_a \pm (i, H'_a)]]_{M_A \backslash (b,m)}}$$

$$R'_9 : \frac{H \xrightarrow{a_i^-(\surd)} H' \quad a \in A \quad \vec{H}_a[i] \xrightarrow{\tau} H'_a}{D[\phi, \phi', \varphi]^-_{M_A} H \xrightarrow{\tau} D[\phi, \varphi[a \to \vec{H}_a \pm (i, H'_a)]]_{M_A}}$$

$$R'_{10} : \frac{H \xrightarrow{a_i^-(\surd)} H' \quad a \in A \quad \vec{H}_a[i] \xrightarrow{b_j^-\surd} H'_a \quad m = \widehat{M_A(b)}((a,i),j)}{D[\phi, \phi', \varphi]^-_{M_A} H \xrightarrow{b_m^-(\surd)} H'[\phi, \varphi[a \to \vec{H}_a \backslash i]]_{(M_A \dagger (a,i)) \backslash (b,m)}}$$

$$R'_{11} : \frac{H \xrightarrow{a_i^-(\surd)} H' \quad a \in A \quad \vec{H}_a[i] \xrightarrow{\tau\surd} H'_a}{D[\phi, \phi', \varphi]^-_{M_A} H \xrightarrow{\tau(\surd)} H'[\phi, \varphi[a \to \vec{H}_a \backslash i]]_{M_A \dagger (a,i)}}$$

Table 7.5: Transition Rules for $\longrightarrow^z_{\text{decl}}$ (2)

and $(\mathsf{EXP}_{\mathsf{se}}^{\mathsf{Ax}}, \mathcal{L}_{\mathsf{se}}, \longrightarrow^{z}_{\mathrm{decl}'}, H')$ are bisimilar. Furthermore, ST-equivalence remains a congruence for $\mathsf{PA}_{\mathsf{se}}^{\mathsf{Ax}}$, since the transition rules are in panth format and complete (compare with Remark 7.30).

As in [44], we restrict our axioms for recursion to *sequential guarded* processes. The definition of sequential guarded has to be adjusted to process algebras that are based on the declaration technique. This adjustment is done with respect to a subset of Var, since we restrict our axioms to those processes that use only a finite number of variables. For example, we do not allow processes like $\langle \mathrm{decl}, x_1 \rangle$ where $\mathrm{decl}(x_i) = x_{i+1}$. This restriction is done, since we are only interested in derivation rules with a finite number of pre-conditions.

Definition 7.31 *A declaration* $\mathrm{decl} : \mathsf{Var} \to \mathsf{EXP}_{\mathsf{se}}^{\mathsf{Ax}}$ *is* sequentially guarded *with respect to* $V \in \mathcal{P}(\mathsf{Var})$ *if* $\forall x \in V : \exists n_x \in \mathbb{N}, u_x : \mathbb{N} \to \mathcal{A}ct, f_x : \mathbb{N} \to V : \mathrm{decl}(x) = \sum_{j=1}^{n_x} u_x(j); f_x(j)$.

SeqG *denotes the set of all declarations that are sequentially guarded by some* V*, i.e.* $\mathsf{SeqG} = \{(\mathrm{decl}, V) \mid \mathrm{decl}$ *is sequentially guarded with respect to* $V\}$.

We introduce the following axiom systems for $\mathsf{PA}_{\mathsf{se}}^{\mathsf{Ax}}$. This is done by presenting axioms (Table 7.6 and Table 7.7) to conclude equality with respect to the same declaration. Additionally, we present a derivation rule that relates processes with possibly different declarations.

Definition 7.32 *Let* $\mathrm{decl} : \mathsf{Var} \to \mathsf{EXP}_{\mathsf{se}}^{\mathsf{Ax}}$ *and let* $H, H' \in \mathsf{EXP}_{\mathsf{se}}^{\mathsf{Ax}}$*. Then we write* $\vdash_{\mathrm{decl}} H = H'$ *if* $H = H'$ *can be derived from the axioms presented in Table 7.6 and Table 7.7.*

Furthermore, the equality deduction of $\mathsf{PA}_{\mathsf{se}}^{\mathsf{Ax}}$ *is given by the following rule*

$$\frac{(\mathrm{decl}', \{x_0, ..., x_n\}) \in \mathsf{SeqG} \qquad \forall i \leq n : \vdash_{\mathrm{decl}} H_i = \mathrm{decl}'(x_i)\{(H_j/x_j)^{j \in \{0,..,n\}}\}}{\vdash \langle \mathrm{decl}, H_0 \rangle = \langle \mathrm{decl}', x_0 \rangle} \tag{7.1}$$

where $H\{(H_j/x_j)^{j \in \{0,..,n\}}\}$ *denotes the simultaneous replacement (substitution) of every occurrence of* x_j *by* H_j *in expression* H.

7.7.1 Soundness

In this subsection, we show that the deductive system presented is sound with respect to ST-equivalence. First, we consider the soundness of $\vdash_{\mathrm{decl}}$:

Lemma 7.33 *If* $\vdash_{\mathrm{decl}} H = H'$ *then* $\langle \mathrm{decl}, H \rangle$ *and* $\langle \mathrm{decl}, H' \rangle$ *are ST-equivalent.*

Proof: It can be straightforwardly checked that all axioms are sound. The rest follows from the fact that ST-equivalence is a congruence. □

Now we are ready to verify the soundness of our equality deduction:

Theorem 7.34 (Soundness) *If* $\vdash \langle \mathrm{decl}, H \rangle = \langle \mathrm{decl}', H' \rangle$ *then* $\langle \mathrm{decl}, H \rangle$ *and* $\langle \mathrm{decl}', H' \rangle$ *are ST-equivalent.*

Proof: The proof is given in Subsection 7.8.3. □

Let $a \in A$, $b \in \mathsf{Obs}$, $c \in \mathsf{Obs}\backslash A$, $R, T, U \in \mathsf{EXP}^{\mathsf{Ax}}_{\mathsf{se}}$ and $q, k \in \mathbb{N}$ with $q \neq k$

$b = b^+; b^- \qquad b^- = b^-_1 \qquad b^-_q = b^-_q\surd; T \qquad \tau = \tau\surd; T$

$T + U = U + T \qquad (T + U) + R = T + (U + R) \qquad T + T = T \qquad T + \mathbf{0} = T$

$T \leftplus U = (T \leftplus^- U) + U \qquad (T + U) \leftplus^- R = (T \leftplus^- R) + (U \leftplus^- R)$
$(b^+; T) \leftplus^- U = b^+; (T \leftplus U) \qquad (b^-_q(\surd); T) \leftplus^- U = b^-_q(\surd); T$
$(\tau(\surd); T) \leftplus^- U = \tau(\surd); T \qquad \mathbf{0} \leftplus^- T = \mathbf{0}$

$T \oplus_M U = (T \oplus^-_M U) + U \oplus^-_{\overline{M}} T \qquad (T + U) \oplus^-_M R = (T \oplus^-_M R) + (U \oplus^-_M R)$
$(b^+; T) \oplus^-_M R = b^+; (T \oplus_{[b,l]\cdot M} R) \qquad (\tau(\surd); T) \oplus^-_M R = \tau(\surd); (T \backslash\backslash_M \emptyset)$
$(b^-_q(\surd); T) \oplus^-_M R = b^-_{\widehat{M(b)}(l,q)}(\surd); (T \backslash\backslash_{M\backslash(b,\widehat{M(b)}(l,q))} \emptyset) \qquad \mathbf{0} \oplus^-_M R = \mathbf{0}$

$(T + U); R = (T; R) + (U; R) \qquad \mathbf{0}; R = \mathbf{0}$
$(b^+; T); R = b^+; (T; R) \qquad (b^-_q; T); R = b^-_q; (T; R) \qquad (b^-_q\surd; T); R = b^-_q; R$
$(\tau; T); R = \tau; (T; R) \qquad (\tau\surd; T); R = \tau; R$

$T \|_{A,M} U = (T \|^-_{A,M} U) + (U \|^-_{A,\overline{M}} T) + (T|_{A,M} U)$
$(T + U) \|^-_{A,M} R = (T \|^-_{A,M} R) + (U \|^-_{A,M} R) \qquad (c^+; T) \|^-_{A,M} R = c^+; (T \|_{A,[c,l]\cdot M} R)$
$(c^-_q; T) \|^-_{A,M} R = c^-_{\widehat{M(c)}(l,q)}; (T \|_{A,M\backslash(c,\widehat{M(c)}(l,q))} R) \qquad (\tau; T) \|^-_{A,M} R = \tau; (T \|_{A,M} R)$
$(c^-_q\surd; T) \|^-_{A,M} R = c^-_{\widehat{M(c)}(l,q)}; (R \backslash\backslash_{\overline{M}\backslash(c,\widehat{M(c)}(l,q))} A) \qquad (\tau\surd; T) \|^-_{A,M} R = \tau; (R \backslash\backslash_{\overline{M}} A)$
$(a^+; T) \|^-_{A,M} R = \mathbf{0} \qquad (a^-_q(\surd); T) \|^-_{A,M} R = \mathbf{0} \qquad \mathbf{0} \|^-_{A,M} R = \mathbf{0}$
$T|_{A,M} R = R|_{A,\overline{M}} T \qquad (T + U)|_{A,M} R = (T|_{A,M} R) + (U|_{A,M} R)$
$(c^+; T)|_{A,M} R = \mathbf{0} \qquad (c^-_q(\surd); T)|_{A,M} R = \mathbf{0} \qquad (\tau(\surd); T)|_{A,M} R = \mathbf{0}$
$(a^+; T)|_{A,M} (a^+; R) = a^+; (T|_{A,M} R) \qquad (a^+; T)|_{A,M} (a^-_q(\surd); R) = \mathbf{0}$
$(a^-_q; T)|_{A,M} (a^-_k(\surd); R) = \mathbf{0} \qquad (a^-_q\surd; T)|_{A,M} (a^-_k\surd; R) = \mathbf{0}$
$(a^-_q; T)|_{A,M} (a^-_q; R) = a^-_q; (T|_{A,M} R) \qquad (a^-_q; T)|_{A,M} (a^-_q\surd; R) = a^-_q; (T \backslash\backslash_M A)$
$(a^-_q\surd; T)|_{A,M} (a^-_q\surd; R) = a^-_q\surd; \mathbf{0} \qquad \mathbf{0}|_{A,M} R = \mathbf{0}$

$(T + U) \backslash\backslash_M A = (T \backslash\backslash_M A) + (U \backslash\backslash_M A)$
$(c^+; T) \backslash\backslash_M A = c^+; (T \backslash\backslash_{[c,l]\cdot M} A) \qquad (a^+; T) \backslash\backslash_M A = \mathbf{0}$
$(c^-_q(\surd); T) \backslash\backslash_M A = c^-_{\widehat{M(c)}(l,q)}(\surd); (T \backslash\backslash_{M\backslash(c,\widehat{M(c)}(l,q))} A) \qquad (a^-_q(\surd); T) \backslash\backslash_M A = \mathbf{0}$
$(\tau(\surd); T) \backslash\backslash_M A = \tau(\surd); (T \backslash\backslash_M A) \qquad \mathbf{0} \backslash\backslash_M A = \mathbf{0}$

$x = \mathrm{decl}(x)$

Table 7.6: Axioms for the Non-Refinement Operators

Let $\phi = (z \to G_z)^{z\in A}$, $\varphi = (z \to \vec{G}_z)^{z\in A}$, $a \in A$, $c \in \mathsf{Obs}\backslash A$, $R, T, U \in \mathsf{EXP}^{\mathsf{Ax}}_{\mathsf{se}}$ and $q, k \in \mathbb{N}$

$U[\phi, \varphi]_{M_A} = U[\phi, \phi, \varphi]^-_{M_A} U$

$U[\phi, \phi', \varphi]^-_{M_A}(T + R) = U[\phi, \phi', \varphi]^-_{M_A} T + U[\phi, \phi', \varphi]^-_{M_A} R$
$U[\phi, \phi'[a \to T + R], \varphi]^-_{M_A} V = U[\phi, \phi'[a \to T], \varphi]^-_{M_A} V + U[\phi, \phi'[a \to R], \varphi]^-_{M_A} V$

$U[\phi, \phi', \varphi]^-_{M_A}(c^+; R) = c^+; R[\phi, \varphi]_{[c,0]\cdot M_A}$
$U[\phi, \phi', \varphi]^-_{M_A}(c^-_q(\surd); R) = c^-_m(\surd); R[\phi, \varphi]_{M_A\backslash(a,m)}$ if $m = \widehat{M_A(c)}(0, q)$
$U[\phi, \phi', \varphi]^-_{M_A}(c^-_q(\surd); R) = \mathbf{0}$ if $\widehat{M_A(c)}(0, q)$ is undefined
$U[\phi, \phi', \varphi]^-_{M_A}(\tau(\surd); R) = \tau(\surd); R[\phi, \varphi]_{M_A}$

$U[\phi, \phi'[a \to b^+; T], \varphi]^-_{M_A}(a^+; R) = b^+; R[\phi, \varphi[a \to T \cdot \varphi(a)]]_{[b,(a,1)]\cdot(M_A \diamond a)}$
$U[\phi, \phi'[a \to b^-_q(\surd); T], \varphi]^-_{M_A}(a^+; R) = \mathbf{0}$
$U[\phi, \phi'[a \to \tau; T], \varphi]^-_{M_A}(a^+; R) = \tau; R[\phi, \varphi[a \to T \cdot \varphi(a)]]_{M_A \diamond a}$
$U[\phi, \phi'[a \to \tau\surd; T], \varphi]^-_{M_A}(a^+; (R + V)) =$
$\quad U[\phi, \phi'[a \to \tau\surd; T], \varphi]^-_{M_A}(a^+; R) + U[\phi, \phi'[a \to \tau\surd; T], \varphi]^-_{M_A}(a^+; V)$
$U[\phi, \phi'[a \to \tau\surd; T], \varphi]^-_{M_A}(a^+; (a^-_1(\surd); R)) = \tau(\surd); R[\phi, \varphi]_{M_A}$
$U[\phi, \phi'[a \to \tau\surd; T], \varphi]^-_{M_A}(a^+; (\gamma; R)) = \mathbf{0}$ if $\gamma \notin \{a^-_1, a^-_1\surd\}$
$U[\phi, \phi'[a \to \mathbf{0}], \varphi]^-_{M_A}(a^+; R) = \mathbf{0}$

$U[\phi, \phi', \varphi]^-_{M_A}(a^-_q(\surd); V) = \mathbf{0}$ if $|\varphi(a)| < q$
$U[\phi, \phi', \varphi[a \to \varphi(a) \pm (q, T + R)]]^-_{M_A}(a^-_q(\surd); V) =$
$\quad U[\phi, \phi', \varphi[a \to \varphi(a) \pm (q, T)]]^-_{M_A}(a^-_q(\surd); V) +$
$\quad U[\phi, \phi', \varphi[a \to \varphi(a) \pm (q, R)]]^-_{M_A}(a^-_q(\surd); V)$
$U[\phi, \phi', \varphi[a \to \varphi(a) \pm (q, b^+; T)]]^-_{M_A}(a^-_q(\surd); R) =$
$\quad b^+; U[\phi, \varphi[a \to \varphi(a) \pm (q, T)]]_{[b,(a,q)]\cdot M_A}$
$U[\phi, \phi', \varphi[a \to \varphi(a) \pm (q, b^-_k; T)]]^-_{M_A}(a^-_q(\surd); R) =$
$\quad b^-_m; U[\phi, \varphi[a \to \varphi(a) \pm (q, T)]]_{M_A\backslash(b,m)}$ if $m = \widehat{M_A(b)}((a, q), k)$
$U[\phi, \phi', \varphi[a \to \varphi(a) \pm (q, b^-_k; T)]]^-_{M_A}(a^-_q(\surd); R) = \mathbf{0}$ if $\widehat{M_A(b)}((a, q), k)$ is undefined
$U[\phi, \phi', \varphi[a \to \varphi(a) \pm (q, b^-_k\surd; T)]]^-_{M_A}(a^-_q(\surd); R) =$
$\quad b^-_m(\surd); R[\phi, \varphi[a \to \varphi(a)\backslash q]]_{(M_A \dagger (a,q))\backslash(b,m)}$ if $m = \widehat{M_A(b)}((a, q), k)$
$U[\phi, \phi', \varphi[a \to \varphi(a) \pm (q, b^-_k\surd; T)]]^-_{M_A}(a^-_q(\surd); R) = \mathbf{0}$ if $\widehat{M_A(b)}((a, q), k)$ is undefined
$U[\phi, \phi', \varphi[a \to \varphi(a) \pm (q, \tau; T)]]^-_{M_A}(a^-_q(\surd); R) = \tau; U[\phi, \varphi[a \to \varphi(a) \pm (q, T)]]_{M_A}$
$U[\phi, \phi', \varphi[a \to \varphi(a) \pm (q, \tau\surd; T)]]^-_{M_A}(a^-_q(\surd); R) = \tau(\surd); R[\phi, \varphi[a \to \varphi(a)\backslash q]]_{M_A \dagger (a,q)}$
$U[\phi, \phi', \varphi[a \to \varphi(a) \pm (q, \mathbf{0})]]^-_{M_A}(a^-_q(\surd); R) = \mathbf{0}$

$U[\phi, \phi', \varphi]^-_{M_A}\mathbf{0} = \mathbf{0}$

Table 7.7: Axioms for the Refinement Operators

7.7.2 Completeness

We will show that our axioms are complete with respect to guarded processes that are finite state. Before we present the definition of guarded and finite state, we restrict the processes to those that use only a finite number of variables.

Definition 7.35 *A process* $\langle \mathrm{decl}, H\rangle \in \mathsf{PA}_{\mathsf{se}}^{\mathsf{Ax}}$ *is specified by* $V \in \mathcal{P}(\mathsf{Var})$ *if* $|V| < |\mathbb{N}|$ *and every variable occurrence in* H *is an element of* V *and every variable occurrence in* $\mathrm{decl}(x)$ *is an element of* V *for any* $x \in V$. VarSp *denotes the set of all processes that are specified by some* V*, i.e.* $\mathsf{VarSp} = \{(\langle \mathrm{decl}, H\rangle, V) \mid \langle \mathrm{decl}, H\rangle$ *is specified by* $V\}$.

A process is guarded if every variable x used by the process is behind an action (a, $\tau\surd; H$) or behind a started action (b^-, b_q^-, $b_q^-\surd; H$) in the scope of $\mathrm{decl}(x)$. It is not sufficient that x is behind a starting action (b^+), since otherwise it would be possible for infinitely many actions to start without any of them finishing. For example, $\langle \mathrm{decl}, x\rangle$ with $\mathrm{decl}(x) = b^+; x$ starts infinitely many b-actions. The possibility to start infinitely many actions without finishing one is problematic for our verifications. We define guardedness only for processes that use only finitely many variables, i.e. processes that are in VarSp.

Definition 7.36 (Guarded) *Define* $\mathcal{G} \subseteq \mathsf{VarSp} \times \mathcal{P}(\mathsf{Var})$ *by*

$$
\begin{array}{lcl}
\mathcal{G}((\langle \mathrm{decl}, H\rangle, V), \tilde{V}) & & \text{if } H \in \{\mathbf{0}, a, b^-, b_q^-, b_q^-\surd; H', \tau\surd; H'\} \\
\mathcal{G}((\langle \mathrm{decl}, H\rangle, V), \tilde{V}) & \Leftrightarrow & \mathcal{G}((\langle \mathrm{decl}, H'\rangle, V), \tilde{V}) \quad \text{if } H \in \{b^+; H', H'; H'', H'\backslash\backslash_M A\} \\
\mathcal{G}((\langle \mathrm{decl}, H\rangle, V), \tilde{V}) & \Leftrightarrow & \mathcal{G}((\langle \mathrm{decl}, H_1\rangle, V), \tilde{V}) \wedge \mathcal{G}((\langle \mathrm{decl}, H_2\rangle, V), \tilde{V}) \\
 & & \quad \text{if } H \in \{H_1 + H_2, H_1 \between H_2, H_1 \between^- H_2, H_1 \oplus_M H_2, H_1 \oplus_M^- H_2, \\
 & & \qquad H_1 \|_{A,M} H_2, H_1 \|_{A,M}^- H_2, H_1|_{A,M} H_2\} \\
\mathcal{G}((\langle \mathrm{decl}, H[(a \to H_a)^{a\in A}, (a \to \vec{H}_a)^{a\in\tilde{A}}]_{M_A}\rangle, V), \tilde{V}) \Leftrightarrow & & \\
 & & \mathcal{G}((\langle \mathrm{decl}, H\rangle, V), \tilde{V}) \wedge \\
 & & \forall a \in A : \mathcal{G}((\langle \mathrm{decl}, H_a\rangle, V), \tilde{V}) \wedge \\
 & & \forall a \in \tilde{A} : \forall i \le |\vec{H}_a| : \mathcal{G}((\langle \mathrm{decl}, \vec{H}_a[i]\rangle, V), \tilde{V}) \\
\mathcal{G}((\langle \mathrm{decl}, H[(a \to H_a)^{a\in A}, (a \to H'_a)^{a\in A}, (a \to \vec{H}_a)^{a\in\tilde{A}}]_{M_A}^- H'\rangle, V), \tilde{V}) \Leftrightarrow & & \\
 & & \mathcal{G}((\langle \mathrm{decl}, H\rangle, V), \tilde{V}) \wedge \mathcal{G}((\langle \mathrm{decl}, H'\rangle, V), \tilde{V}) \wedge \\
 & & \forall a \in A : (\mathcal{G}((\langle \mathrm{decl}, H_a\rangle, V), \tilde{V}) \wedge \mathcal{G}((\langle \mathrm{decl}, H'_a\rangle, V), \tilde{V})) \wedge \\
 & & \forall a \in \tilde{A} : \forall i \le |\vec{H}_a| : \mathcal{G}((\langle \mathrm{decl}, \vec{H}_a[i]\rangle, V), \tilde{V}) \\
\mathcal{G}((\langle \mathrm{decl}, x\rangle, V), \tilde{V}) & \Leftrightarrow & \mathcal{G}((\langle \mathrm{decl}, \mathrm{decl}(x)\rangle, V), \tilde{V} \cup \{x\}) \wedge x \notin \tilde{V}
\end{array}
$$

A process $\langle \mathrm{decl}, H\rangle \in \mathsf{EXP}_{\mathsf{se}}^{\mathsf{Ax}}$ *is* guarded *if there is* $V \in \mathcal{P}_{fin}(\mathsf{Var})$ *such that* $(\langle \mathrm{decl}, H\rangle, V) \in \mathsf{VarSp}$ *and for all* $x \in V : \mathcal{G}((\langle \mathrm{decl}, x\rangle, V), \emptyset)$.

The set of all guarded processes is denoted by $\mathsf{PA}_{\mathsf{se}}^{\mathsf{Gu}}$.

The guarded predicate ($\mathcal{G}$) can be applied directly to the expression of a guarded process, as it is illustrated by the following lemma.

Lemma 7.37 *Suppose* $\langle \mathrm{decl}, H\rangle \in \mathsf{EXP}_{\mathsf{se}}^{\mathsf{Ax}}$ *is* guarded *then there is* $V \in \mathcal{P}_{fin}(\mathsf{Var})$ *such that* $(\langle \mathrm{decl}, H\rangle, V) \in \mathsf{VarSp}$ *and* $\mathcal{G}((\langle \mathrm{decl}, H\rangle, V), \emptyset)$.

Proof: By definition, there exists $V \in \mathcal{P}_{fin}(\mathsf{Var})$ such that $(\langle \mathrm{decl}, H\rangle, V) \in \mathsf{VarSp}$ and for all $x \in V : \mathcal{G}((\langle \mathrm{decl}, x\rangle, V), \emptyset)$. The rest can be easily checked by structural induction on H. □

Guardedness is preserved by the transition rules, i.e. if $\langle \mathrm{decl}, H\rangle \in \mathsf{PA}_{\mathsf{se}}^{\mathsf{Gu}}$ and $H \xrightarrow{\gamma}{}^{z}{}_{\mathrm{decl}} H'$ then $\langle \mathrm{decl}, H'\rangle \in \mathsf{PA}_{\mathsf{se}}^{\mathsf{Gu}}$. This just results from the fact that no new variables can be used by H' and guardedness only depends on decl.

Finite state processes are introduced as follows:

Definition 7.38 *An element of* $\mathsf{PA}_{\mathsf{se}}^{\mathsf{Ax}}$ *is* finite state *if its corresponding transition system has a finite set of states.*

We have the following completeness result:

Theorem 7.39 (Completeness) *Suppose* $\langle \mathrm{decl}, H\rangle, \langle \mathrm{decl}', H'\rangle \in \mathsf{PA}_{\mathsf{se}}^{\mathsf{Gu}}$ *are finite state and ST-equivalent. Then* $\vdash \langle \mathrm{decl}, H\rangle = \langle \mathrm{decl}', H'\rangle$.

Proof: The proof is given in Subsection 7.8.4. □

7.8 Proofs

7.8.1 Proof of the Consistency Results

The proof of Theorem 7.26 is analogous to the proof of Theorem 3.25, i.e. we introduce an event based transition relation. Then we show that this transition system is bisimilar to $(\mathsf{EXP}_{\mathsf{se}}^{\mathsf{O}}, \mathcal{L}_{\mathsf{se}}, \longrightarrow^{c}_{\mathrm{decl}}, B)$ and that it is in addition bisimilar to $(\mathbf{SEBES}, \mathcal{L}_{\mathsf{se}}, \hookrightarrow^{c}, [\![\langle \mathrm{decl}, B\rangle]\!])$. And so Theorem 7.26 follows by the transitivity of bisimilarity. Finally, we conclude Theorem 7.23 from Theorem 7.26.

Event Based Transition System for $\mathsf{EXP}_{\mathsf{se}}$.

The process algebra expressions $\mathsf{EXP}_{\mathsf{se}}^{\mathsf{O}'}$ are defined by the following BNF-grammar.

$$\begin{array}{rcl} G & ::= & B \mid b^{-} \mid G \dot{+} B \mid G \oplus_M G \mid G; B \mid G \|_{A,M} G \mid G \backslash\backslash_M A \mid \\ & & G[(a \to B)^{a\in A}, (a \to \vec{G})^{a\in A}]_{M_A} \mid \lceil G \rceil_q \\ \vec{G} & ::= & \varepsilon \mid G \cdot \vec{G} \end{array}$$

where $B \in \mathsf{EXP}_{\mathsf{se}}$, $b \in \mathsf{Obs}$, $A \subseteq \mathsf{Obs}$, $M : \mathsf{Obs} \to \{l, r\}^{\star}$, $M_A : \mathsf{Obs} \to ((A \times \mathbb{N}) \cup \{0\})^{\star}$ and $q \in \{1, 2, l, r\}$. The event based transition rules $\longrightarrow'_{\mathrm{decl}}$ with respect to $\mathrm{decl} : \mathsf{Var} \to \mathsf{EXP}_{\mathsf{se}}$ are presented in Table 7.8, Table 7.9 and Table 7.10.

$$Ac'_1 : \frac{a \in \mathsf{Obs}}{a \xrightarrow[\bullet]{a^+} a^-} \qquad Ac'_2 : \frac{}{b^- \xrightarrow[\bullet]{b_1^-\surd} \mathbf{0}} \qquad Ac'_3 : \frac{}{\tau \xrightarrow[\bullet]{\tau\surd} \mathbf{0}}$$

$$Ch_1^{s\prime} : \frac{B_1 \xrightarrow[e]{\gamma} G'_1}{\begin{array}{l} B_1 + B_2 \xrightarrow[(\star_1,e)]{\gamma} \lceil G'_1 \rceil_1 \\ B_2 + B_1 \xrightarrow[(\star_2,e)]{\gamma} \lceil G'_1 \rceil_2 \end{array}} \qquad Ch_1^{h\prime} : \frac{B_2 \xrightarrow[e]{\gamma} G'_2}{D_1 \dagger B_2 \xrightarrow[(\star_2,e)]{\gamma} \lceil G'_2 \rceil_2}$$

$$Ch_2^{h\prime} : \frac{D_1 \xrightarrow[e]{a^+} G'_1}{D_1 \dagger B_2 \xrightarrow[(\star_1,e)]{a^+} G'_1 \dagger B_2} \qquad Ch_3^{h\prime} : \frac{D_1 \xrightarrow[e]{\gamma} G'_1 \quad \gamma \notin \mathsf{Obs} \times \{+\}}{D_1 \dagger B_2 \xrightarrow[(\star_1,e)]{\gamma} \lceil G'_1 \rceil_1}$$

$$Ch_0^{e\prime} : \frac{B_1 \oplus_\perp B_2 \xrightarrow[e]{\gamma} G'}{B_1 \oplus B_2 \xrightarrow[e]{\gamma} G'} \qquad Ch_1^{e\prime} : \frac{G_1 \xrightarrow[e]{a^+} G'_1}{\begin{array}{l} G_1 \oplus_M G_2 \xrightarrow[(\star_1,e)]{a^+} G'_1 \oplus_{[a,l]\cdot M} G_2 \\ G_2 \oplus_M G_1 \xrightarrow[(\star_2,e)]{a^+} G_2 \oplus_{[a,r]\cdot M} G'_1 \end{array}}$$

$$Ch_2^{e\prime} : \frac{G_1 \xrightarrow[e]{a_i^-(\surd)} G'_1}{\begin{array}{l} G_1 \oplus_M G_2 \xrightarrow[(\star_1,e)]{a^-_{\widehat{M(a)}(l,i)}(\surd)} \lceil G'_1 \rceil_1 \backslash\backslash_{M \setminus (a,\widehat{M(a)}(l,i))} \emptyset \\ G_2 \oplus_M G_1 \xrightarrow[(\star_2,e)]{a^-_{\widehat{M(a)}(r,i)}(\surd)} \lceil G'_1 \rceil_2 \backslash\backslash_{\overline{M} \setminus (a,\widehat{M(a)}(r,i))} \emptyset \end{array}}$$

$$Ch_3^{e\prime} : \frac{G_1 \xrightarrow[e]{\tau(\surd)} G'_1}{\begin{array}{l} G_1 \oplus_M G_2 \xrightarrow[(\star_1,e)]{\tau(\surd)} \lceil G'_1 \rceil_1 \backslash\backslash_M \emptyset \\ G_2 \oplus_M G_1 \xrightarrow[(\star_2,e)]{\tau(\surd)} \lceil G'_1 \rceil_2 \backslash\backslash_{\overline{M}} \emptyset \end{array}}$$

$$S'_1 : \frac{D_1 \xrightarrow[e]{\alpha} G'_1 \quad \alpha \in \{\tau\} \cup (\mathsf{Obs} \times \{+\}) \cup (\mathsf{Obs} \times \mathbb{N})}{D_1; B_2 \xrightarrow[(\star_1,e)]{\alpha} G'_1; B_2}$$

$$S'_2 : \frac{D_1 \xrightarrow[e]{\alpha\surd} G'_1 \quad \alpha\surd \in (\{\tau\} \times \{\surd\}) \cup (\mathsf{Obs} \times \mathbb{N} \times \{\surd\})}{D_1; B_2 \xrightarrow[(\star_1,e)]{\alpha} \lceil B_2 \rceil_2}$$

$$Rec' : \frac{\mathrm{decl}(x) \xrightarrow[e]{\gamma} G'}{x \xrightarrow[e]{\gamma} G'}$$

$$N'_1 : \frac{G \xrightarrow[e]{\gamma} G' \quad i \in \{1,2\}}{\lceil G \rceil_i \xrightarrow[(\star_i,e)]{\gamma} \lceil G' \rceil_i} \qquad N'_2 : \frac{G \xrightarrow[e]{\gamma} G'}{\lceil G \rceil_l \xrightarrow[(e,\star)]{\gamma} \lceil G' \rceil_l} \qquad N'_3 : \frac{G \xrightarrow[e]{\gamma} G'}{\lceil G \rceil_r \xrightarrow[(\star,e)]{\gamma} \lceil G' \rceil_r}$$

Table 7.8: Event Based Transition Rules with respect to $\longrightarrow^{c}_{\mathrm{decl}}$ (1)

$$P'_0 : \frac{B_1 \|_{A,\perp} B_2 \xrightarrow{\gamma}_{e} G'}{B_1 \|_A B_2 \xrightarrow{\gamma}_{e} G'} \qquad P'_1 : \frac{G_1 \xrightarrow{a^+}_{e} G'_1 \quad a \notin A}{\begin{array}{l} G_1 \|_{A,M} G_2 \xrightarrow{a^+}_{(e,\star)} G'_1 \|_{A,[a,l]\cdot M} G_2 \\ G_2 \|_{A,M} G_1 \xrightarrow{a^+}_{(\star,e)} G_2 \|_{A,[a,r]\cdot M} G'_1 \end{array}}$$

$$P'_2 : \frac{G_1 \xrightarrow{a_i^-}_{e} G'_1 \quad a \notin A}{\begin{array}{l} G_1 \|_{A,M} G_2 \xrightarrow[(e,\star)]{a^-_{\widehat{M(a)}(l,i)}} G'_1 \|_{A,M\setminus(a,\widehat{M(a)}(l,i))} G_2 \\ G_2 \|_{A,M} G_1 \xrightarrow[(\star,e)]{a^-_{\widehat{M(a)}(r,i)}} G_2 \|_{A,M\setminus(a,\widehat{M(a)}(r,i))} G'_1 \end{array}}$$

$$P'_3 : \frac{G_1 \xrightarrow{a_i^-\surd}_{e} G'_1 \quad a \notin A}{\begin{array}{l} G_1 \|_{A,M} G_2 \xrightarrow[(e,\star)]{a^-_{\widehat{M(a)}(l,i)}} \lceil G_2 \rceil_r \backslash\backslash_{\overline{M}\setminus(a,\widehat{M(a)}(l,i))} A \\ G_2 \|_{A,M} G_1 \xrightarrow[(\star,e)]{a^-_{\widehat{M(a)}(r,i)}} \lceil G_2 \rceil_l \backslash\backslash_{M\setminus(a,\widehat{M(a)}(r,i))} A \end{array}}$$

$$P'_4 : \frac{G_1 \xrightarrow{a^+}_{e_1} G'_1 \quad G_2 \xrightarrow{a^+}_{e_2} G'_2 \quad a \in A}{G_1 \|_{A,M} G_2 \xrightarrow{a^+}_{(e_1,e_2)} G'_1 \|_{A,M} G'_2} \qquad P'_5 : \frac{G_1 \xrightarrow{a_i^-\surd}_{e_1} G'_1 \quad G_2 \xrightarrow{a_i^-\surd}_{e_2} G'_2 \quad a \in A}{G_1 \|_{A,M} G_2 \xrightarrow{a_i^-\surd}_{(e_1,e_2)} \mathbf{0}}$$

$$P'_6 : \frac{G_1 \xrightarrow{a_i^-}_{e_1} G'_1 \quad G_2 \xrightarrow{a_i^-}_{e_2} G'_2 \quad a \in A}{G_1 \|_{A,M} G_2 \xrightarrow{a_i^-}_{(e_1,e_2)} G'_1 \|_{A,M} G'_2} \qquad P'_7 : \frac{G_1 \xrightarrow{a_i^-\surd}_{e_1} G'_1 \quad G_2 \xrightarrow{a_i^-}_{e_2} G'_2 \quad a \in A}{\begin{array}{l} G_1 \|_{A,M} G_2 \xrightarrow{a_i^-}_{(e_1,e_2)} \lceil G'_2 \rceil_r \backslash\backslash_{\overline{M}} A \\ G_2 \|_{A,M} G_1 \xrightarrow{a_i^-}_{(e_2,e_1)} \lceil G'_2 \rceil_l \backslash\backslash_{M} A \end{array}}$$

$$P'_8 : \frac{G_1 \xrightarrow{\tau}_{e} G'_1}{\begin{array}{l} G_1 \|_{A,M} G_2 \xrightarrow{\tau}_{(e,\star)} G'_1 \|_{A,M} G_2 \\ G_2 \|_{A,M} G_1 \xrightarrow{\tau}_{(\star,e)} G_2 \|_{A,M} G'_1 \end{array}} \qquad P'_9 : \frac{G_1 \xrightarrow{\tau\surd}_{e} G'_1}{\begin{array}{l} G_1 \|_{A,M} G_2 \xrightarrow{\tau}_{(e,\star)} \lceil G_2 \rceil_r \backslash\backslash_{\overline{M}} A \\ G_2 \|_{A,M} G_1 \xrightarrow{\tau}_{(\star,e)} \lceil G_2 \rceil_l \backslash\backslash_{M} A \end{array}}$$

$$Res'_0 : \frac{B \backslash\backslash_{\perp} A \xrightarrow{\gamma}_{e} G'}{B \backslash\backslash A \xrightarrow{\gamma}_{e} G'} \qquad Res'_1 : \frac{G \xrightarrow{a^+}_{e} G' \quad a \notin A}{G \backslash\backslash_M A \xrightarrow{a^+}_{e} G' \backslash\backslash_{[a,l]\cdot M} A}$$

$$Res'_2 : \frac{G \xrightarrow{a_i^-(\surd)}_{e} G' \quad a \notin A}{G \backslash\backslash_M A \xrightarrow[e]{a^-_{\widehat{M(a)}(l,i)}(\surd)} G' \backslash\backslash_{M\setminus(a,\widehat{M(a)}(l,i))} A} \qquad Res'_3 : \frac{G \xrightarrow{\tau(\surd)}_{e} G'}{G \backslash\backslash_M A \xrightarrow{\tau(\surd)}_{e} G' \backslash\backslash_M A}$$

Table 7.9: Event Based Transition Rules with respect to $\longrightarrow^{c}_{\text{decl}}$ (2)

For simplicity, let $\phi = (a \to B_a)^{a\in A}$ and $\varphi = (a \to \vec{G}_a)^{a\in A}$

$$R'_0 : \frac{B[\phi, (a \to \varepsilon)^{a\in A})]_\perp \xrightarrow[e]{\gamma} G'}{B[\phi] \xrightarrow[e]{\gamma} G'} \qquad R'_1 : \frac{G \xrightarrow[e]{a^+} G' \quad a \notin A}{G[\phi,\varphi]_{M_A} \xrightarrow[(e,e)]{a^+} G'[\phi,\varphi]_{[a,0]\cdot M_A}}$$

$$R'_2 : \frac{G \xrightarrow[e]{a_i^-(\surd)} G' \quad a \notin A \quad m = \widehat{M_A(a)}(0,i)}{G[\phi,\varphi]_{M_A} \xrightarrow[(e,e)]{a_m^-(\surd)} G'[\phi,\varphi]_{M_A\setminus(a,m)}} \qquad R'_3 : \frac{G \xrightarrow[e]{\tau(\surd)} G'}{G[\phi,\varphi]_{M_A} \xrightarrow[(e,e)]{\tau(\surd)} G'[\phi,\varphi]_{M_A}}$$

$$R'_4 : \frac{G \xrightarrow[e]{a^+} G' \quad a \in A \quad B_a \xrightarrow[\hat{e}]{b^+} G'_a}{G[\phi,\varphi]_{M_A} \xrightarrow[(e,\hat{e})]{b^+} G'[\phi,\varphi[a \to G'_a\vec{G}_a]]_{[b,(a,1)]\cdot(M_A\diamond a)}}$$

$$R'_5 : \frac{G \xrightarrow[e]{a^+} G' \quad a \in A \quad B_a \xrightarrow[\hat{e}]{\tau} G'_a}{G[\phi,\varphi]_{M_A} \xrightarrow[(e,\hat{e})]{\tau} G'[\phi,\varphi[a \to G'_a\vec{G}_a]]_{M_A\diamond a}}$$

$$R'_6 : \frac{G \xrightarrow[e]{a^+} G' \quad a \in A \quad B_a \xrightarrow[\hat{e}]{\tau\surd} G'_a \quad G' \xrightarrow[e]{a_1^-(\surd)} G''}{G[\phi,\varphi]_{M_A} \xrightarrow[(e,\hat{e})]{\tau(\surd)} G''[\phi,\varphi]_{M_A}}$$

$$R'_7 : \frac{G \xrightarrow[e]{a_i^-(\surd)} G' \quad a \in A \quad \vec{G}_a[i] \xrightarrow[\hat{e}]{b^+} G'_a}{G[\phi,\varphi]_{M_A} \xrightarrow[(e,\hat{e})]{b^+} G[\phi,\varphi[a \to \vec{G}_a \pm (i, G'_a)]]_{[b,(a,i)]\cdot M_A}}$$

$$R'_8 : \frac{G \xrightarrow[e]{a_i^-(\surd)} G' \quad a \in A \quad \vec{G}_a[i] \xrightarrow[\hat{e}]{b_j^-} G'_a \quad m = \widehat{M_A(b)}((a,i),j)}{G[\phi,\varphi]_{M_A} \xrightarrow[(e,\hat{e})]{b_m^-} G[\phi,\varphi[a \to \vec{G}_a \pm (i, G'_a)]]_{M_A\setminus(b,m)}}$$

$$R'_9 : \frac{G \xrightarrow[e]{a_i^-(\surd)} G' \quad a \in A \quad \vec{G}_a[i] \xrightarrow[\hat{e}]{\tau} G'_a}{G[\phi,\varphi]_{M_A} \xrightarrow[(e,\hat{e})]{\tau} G[\phi,\varphi[a \to \vec{G}_a \pm (i, G'_a)]]_{M_A}}$$

$$R'_{10} : \frac{G \xrightarrow[e]{a_i^-(\surd)} G' \quad a \in A \quad \vec{G}_a[i] \xrightarrow[\hat{e}]{b_j^-\surd} G'_a \quad m = \widehat{M_A(b)}((a,i),j)}{G[\phi,\varphi]_{M_A} \xrightarrow[(e,\hat{e})]{b_m^-(\surd)} G'[\phi,\varphi[a \to \vec{G}_a\backslash i]]_{(M_A\dagger(a,i))\setminus(b,m)}}$$

$$R'_{11} : \frac{G \xrightarrow[e]{a_i^-(\surd)} G' \quad a \in A \quad \vec{G}_a[i] \xrightarrow[\hat{e}]{\tau\surd} G'_a}{G[\phi,\varphi]_{M_A} \xrightarrow[(e,\hat{e})]{\tau(\surd)} G'[\phi,\varphi[a \to \vec{G}_a\backslash i]]_{M_A\dagger(a,i)}}$$

Table 7.10: Event Based Transition Rules with respect to $\longrightarrow^c_{\text{decl}}$ (3)

First Bisimulation Result.

An expression G of $\mathsf{EXP}_{\mathsf{se}}^{\mathsf{O}'}$ and an expression C of $\mathsf{EXP}_{\mathsf{se}}^{\mathsf{O}}$ are related if G results in C through the removal of all $\lceil\_\rceil$ expressions. This is formalized by the following function, where we also count the $\lceil\_\rceil$ symbols in G.

Definition 7.40 $\Xi : \mathbb{N} \times \mathsf{EXP}_{\mathsf{se}}^{\mathsf{O}} \to \mathcal{P}(\mathsf{EXP}_{\mathsf{se}}^{\mathsf{O}'})$ *is defined as follows, where* $Q = \{1, 2, l, r\}$.

$$
\begin{array}{l}
\Xi(0, C) = \{C\} \\
\Xi(n+1, B) = \{\lceil\tilde{G}\rceil_i \mid i \in Q \wedge \tilde{G} \in \Xi(n, B)\} \\
\Xi(n+1, b^-) = \{\lceil\tilde{G}\rceil_i \mid i \in Q \wedge \tilde{G} \in \Xi(n, b^-)\} \\
\Xi(n+1, C \dotplus B) = \{\lceil\tilde{G}\rceil_i \mid i \in Q \wedge \tilde{G} \in \Xi(n, C \dotplus B)\} \cup \{G\,[\!>\, B \mid G \in \Xi(n+1, C)\} \\
\Xi(n+1, C_1 \oplus_M C_2) = \{\lceil\tilde{G}\rceil_i \mid i \in Q \wedge \tilde{G} \in \Xi(n, C_1 \oplus_M C_2)\} \cup \\
\quad \{G_1 \oplus_M G_2 \mid \exists m \in \mathbb{N} : m \leq n+1 \wedge G_1 \in \Xi(m, C_1) \wedge G_2 \in \Xi(n+1-m, C_2)\} \\
\Xi(n+1, C; B) = \{\lceil\tilde{G}\rceil_i \mid i \in Q \wedge \tilde{G} \in \Xi(n, C; B)\} \cup \{G; B \mid G \in \Xi(n+1, C)\} \\
\Xi(n+1, C_1 \|_{A,M} C_2) = \{\lceil\tilde{G}\rceil_i \mid i \in Q \wedge \tilde{G} \in \Xi(n, C_1 \|_{A,M} C_2)\} \cup \\
\quad \{G_1 \|_{A,M} G_2 \mid \exists m \in \mathbb{N} : m \leq n+1 \wedge G_1 \in \Xi(m, C_1) \wedge G_2 \in \Xi(n+1-m, C_2)\} \\
\Xi(n+1, C \backslash\backslash_M A) = \{\lceil\tilde{G}\rceil_i \mid i \in Q \wedge \tilde{G} \in \Xi(n, C \backslash\backslash_M A)\} \cup \\
\quad \{G \backslash\backslash_M A \mid G \in \Xi(n+1, C)\} \\
\Xi(n+1, C[(a \to B_a)^{a \in A}, (a \to \vec{C}_a)^{a \in A}]_{M_A}) = \{G[(a \to B_a)^{a \in A}, (a \to \vec{G}_a)^{a \in A}] \mid \\
\qquad \exists m \in \mathbb{N}, (m_a^j)_{a \in A} \in \mathbb{N}^A : m + \textstyle\sum_{a \in A \wedge j \in \mathbb{N}} m_a^j = n+1 \wedge G \in \Xi(m, C) \wedge \\
\qquad (\vec{G}_a[j] \in \Xi(m_a^j, \vec{C}_a[j]) \vee (\vec{G}_a[j] \textit{ is undefined } \wedge \vec{C}_a[j] \textit{ is undefined}))\} \cup \\
\quad \{\lceil\tilde{G}\rceil_i \mid i \in Q \wedge \tilde{G} \in \Xi(n, C[(a \to B_a)^{a \in A}, (a \to \vec{C}_a)^{a \in A}]_{M_A})\}
\end{array}
$$

The well-definedness of Ξ is easily seen.

Lemma 7.41 *Let* $B \in \mathsf{EXP}_{\mathsf{se}}$ *then* $(\mathsf{EXP}_{\mathsf{se}}^{\mathsf{O}}, \mathcal{L}_{\mathsf{se}}, \longrightarrow_{\mathrm{decl}}^{c}, B)$ *and* $(\mathsf{EXP}_{\mathsf{se}}^{\mathsf{O}'}, \mathcal{L}_{\mathsf{se}}, \longrightarrow_{\mathrm{decl}}^{\prime\prime}, B)$ *are bisimilar, where* $G \overset{\gamma}{\longrightarrow}{}''_{\mathrm{decl}}\, G' \Leftrightarrow \exists e \in \mathcal{U} : G \overset{\gamma}{\underset{e}{\longrightarrow}}{}'_{\mathrm{decl}}\, G'$.

Proof: Define $\mathcal{R} = \{(C, G) \in \mathsf{EXP}_{\mathsf{se}}^{\mathsf{O}} \times \mathsf{EXP}_{\mathsf{se}}^{\mathsf{O}'} \mid \exists n : G \in \Xi(n, C)\}$. In order to verify that $\mathcal{R}$ is a bisimulation, we show

$$(C \overset{\gamma}{\longrightarrow}{}^{c} C' \wedge G \in \Xi(n, C)) \Rightarrow \exists e, G', m : G \overset{\gamma}{\underset{e}{\longrightarrow}}{}' G' \wedge G' \in \Xi(m, C') \tag{7.2}$$

The proof of (7.2) works by induction on the depth of inference of $C \overset{\gamma}{\longrightarrow}{}^{c} C'$ combined with the value of n. Then, (7.2) can be easily checked with the following procedure:

- applying rule N_j whenever $C = \lceil\tilde{C}\rceil_q$. In these cases n is reduced by one and $C \overset{\gamma}{\longrightarrow}{}^{c} C'$ keeps unaffected. Therefore, the result follows by induction.
- applying the correspondent rules of $C \overset{\gamma}{\longrightarrow}{}^{c} C'$ whenever C is different to $\lceil\tilde{C}\rceil_q$. In these cases the depth of inference is reduced and n gets not increased. Therefore the result follows by induction.

Another fact is

$$(G \xrightarrow[e]{\gamma}{}' G' \wedge G \in \Xi(n, C)) \Rightarrow \exists C', m : C \xrightarrow{\gamma}{}^{c} C' \wedge G' \in \Xi(m, C') \tag{7.3}$$

This equation can be seen by induction on the depth of inference of $G \xrightarrow[e]{\gamma}{}' G'$.
Now we are ready to verify that $\mathcal{R}$ is a bisimulation:

- It is clear that $(B, B) \in \mathcal{R}$.
- Suppose $(C_1, G_1) \in \mathcal{R}$ and $C_1 \xrightarrow{\gamma}{}^{c} C_2$. Then $\exists e, G_2, m : G_1 \xrightarrow[e]{\gamma}{}' G_2 \wedge G_2 \in \Xi(m, C_2)$ by (7.2). Thus $G_1 \xrightarrow{\gamma}{}'' G_2$ and $(C_2, G_2) \in \mathcal{R}$, as required.
- Suppose $(C_1, G_1) \in \mathcal{R}$ and $G_1 \xrightarrow{\gamma}{}'' G_2$. Then $G_1 \xrightarrow[e]{\gamma}{}' G_2$ for some e. Hence, $\exists C_2, m : C_1 \xrightarrow{\gamma}{}^{c} C_2 \wedge G_2 \in \Xi(m, C_2)$ by (7.3). □

Second Bisimulation Result.

First, we show that the denotation of a variable is the same as the denotation of its corresponding expression.

Lemma 7.42 *Let* $\mathrm{decl} : \mathsf{Var} \to \mathsf{EXP}_{\mathsf{se}}$ *and* $x \in \mathsf{Var}$. *Then* $[\![\langle \mathrm{decl}, x\rangle]\!] = [\![\langle \mathrm{decl}, \mathrm{decl}(x)\rangle]\!]$.

Proof: Similar to the proof of Lemma 3.37. □

We extend the denotational semantics to $\mathsf{EXP}^{\mathsf{O}'}_{\mathsf{se}}$. We define an event based refinement operator, which refines events rather than actions.

Definition 7.43 *Define* $\underline{Ref}^{se}_{A} : \mathbf{SEBES} \times (\mathcal{U} \to \mathbf{SEBES}) \to \mathbf{SEBES}$ *by* $\underline{Ref}^{se}_{A}(\mathcal{E}, \vartheta) = (\tilde{E}, \tilde{\rightsquigarrow}, \tilde{\succ}, \tilde{\mapsto}, \tilde{T}, \tilde{l})$ *where*

$$\begin{array}{rcl}
\tilde{E} & = & \{(e, \hat{e}) \mid e \in E \wedge l(e) \in A \wedge \hat{e} \in E_{\vartheta(e)}\} \cup \\
 & & \{(e, e) \in E \times E \mid l(e) \notin A\} \\
\tilde{\rightsquigarrow} & = & \{((e, \hat{e}), (e', \hat{e}')) \mid e \rightsquigarrow e' \vee (e' = e' \wedge l(e) \in A \wedge \hat{e} \rightsquigarrow_{\vartheta(e)} \hat{e}')\} \\
\tilde{\succ} & = & \{(\tilde{Z}, (e, \hat{e})) \mid \exists Z : Z \succ e \wedge \exists f : Z \to \mathcal{P}(\mathcal{U}) : \\
 & & (\forall e' \in Z : (l(e') \notin A \wedge f(e') = \{e'\}) \vee (l(e') \in A \wedge e' \neq e \wedge f(e') \in T_{\vartheta(e')}) \vee \\
 & & \quad (l(e') \in A \wedge e' = e \wedge \exists \hat{X} \in T_{\vartheta(e')}, \hat{Z} : f(e') = \hat{Z} \cup \hat{X} \wedge \hat{Z} \succ_{\vartheta(e')} \hat{e})) \wedge \\
 & & \tilde{Z} = \{(e', \hat{e}') \in \tilde{E} \mid e' \in Z \wedge \hat{e}' \in f(e')\}\} \\
\tilde{\mapsto} & = & \{(\{e\} \times X', (e, \hat{e})) \mid l(e) \in A \wedge X' \mapsto_{\vartheta(e)} \hat{e}\} \cup \\
 & & \{(\tilde{X}, (e, \hat{e})) \mid (l(e) \in A \Rightarrow \hat{e} \in \mathrm{init}(\vartheta(e))) \wedge \exists X : X \mapsto e \wedge \exists f : Z \to \mathcal{P}(\mathcal{U}) : \\
 & & (\forall e' : (l(e') \notin A \wedge f(e') = \{e'\}) \vee (l(e') \in A \wedge f(e') \in T_{\vartheta(e')})) \wedge \\
 & & \tilde{X} = \{(e', \hat{e}') \in \tilde{E} \mid e' \in X \wedge \hat{e}' \in f(e')\}\} \\
\tilde{T} & = & \{\tilde{X} \mid \exists X \in T \wedge \exists f : X \to \mathcal{P}(\mathcal{U}) : (\forall e' : (l(e') \notin A \wedge f(e') = \{e'\}) \vee \\
 & & (l(e') \in A \wedge f(e') \in T_{\vartheta(e')})) \wedge \tilde{X} = \{(e', \hat{e}') \in \tilde{E} \mid e' \in X \wedge \hat{e}' \in f(e')\}\}
\end{array}$$

$$\tilde{l}(e, \hat{e}) = \begin{cases} l(e) & \text{if } l(e) \notin A \\ l_{\vartheta(e)}(\hat{e}) & \text{if } l(e) \in A \end{cases}$$

For the operators on $\mathsf{EXP}^{\mathsf{O}'}_{\mathsf{se}}$ we need the following function, which adapts function $\mathcal{M}$ when the events are renamed.

Define $\Psi : (\mathcal{U} \rightharpoonup \mathbb{N}) \times \{1, 2, l, r\} \to (\mathcal{U} \rightharpoonup \mathbb{N})$ by

$$\Psi(\mathcal{M}, q)(e) \simeq \begin{cases} \mathcal{M}(e') & \text{if } q \in \{1,2\} \wedge e = (\star_q, e') \\ \mathcal{M}(e') & \text{if } q = l \wedge e = (\star, e') \\ \mathcal{M}(e') & \text{if } q = r \wedge e = (e', \star) \end{cases}$$

The operators on $\mathsf{EXP}^{\mathsf{O}'}_{\mathsf{se}}$ are given as follows:

Definition 7.44 (Operators on $\mathsf{EXP}^{\mathsf{O}'}_{\mathsf{se}}$) *Let $A \subseteq \mathsf{Obs}$, $M : \mathsf{Obs} \to \{l, r\}^\star$, $M_A : \mathsf{Obs} \to ((A \times \mathbb{N}) \cup \{0\})^\star$ and $q \in \{1, 2, l, r\}$ Furthermore, suppose $\mathfrak{E} = (\mathcal{E}, \mathcal{M})$ and $\mathfrak{E}_j = (\mathcal{E}_j, \mathcal{M}_j)$. Then define*

$\widetilde{+} : \mathbf{SEBES}_{\mathcal{M}} \times \mathbf{SEBES} \to \mathbf{SEBES}_{\mathcal{M}}$ *with* $\mathfrak{E}_1 \widetilde{+} \mathcal{E}_2 = (\mathcal{E}_1 \widehat{+} \mathcal{E}_2, \Psi(\mathcal{M}_1, 1))$

$\widetilde{\oplus}_M : \mathbf{SEBES}_{\mathcal{M}} \times \mathbf{SEBES}_{\mathcal{M}} \to \mathbf{SEBES}_{\mathcal{M}}$ *with* $\mathfrak{E}_1 \widetilde{\oplus}_M \mathfrak{E}_2 = (\mathcal{E}_1 \widehat{\oplus} \mathcal{E}_2 \upharpoonright E', \mathcal{M}')$

where $E' = \{(\star_1, e) \mid e \in \mathrm{dom}(\mathcal{M}_1) \Rightarrow \widehat{M(l_1(e))}(l, \mathcal{M}_1(e))$ *is defined*$\} \cup$
$\{(\star_2, e) \mid e \in \mathrm{dom}(\mathcal{M}_2) \Rightarrow \widehat{M(l_2(e))}(r, \mathcal{M}_2(e))$ *is defined*$\}$.

and $\mathcal{M}'(e') \simeq \begin{cases} \widehat{M(l_1(e))}(l, \mathcal{M}_1(e)) & \text{if } e' = (\star_1, e) \\ \widehat{M(l_2(e))}(r, \mathcal{M}_2(e)) & \text{if } e' = (\star_2, e) \end{cases}$

$\widetilde{;}_M : \mathbf{SEBES}_{\mathcal{M}} \times \mathbf{SEBES} \to \mathbf{SEBES}_{\mathcal{M}}$ *with* $\mathfrak{E}_1 \widetilde{;} \mathcal{E}_2 = (\mathcal{E}_1 \widehat{;} \mathcal{E}_2, \Psi(\mathcal{M}_1, 1))$

$\widetilde{\|}_{A,M} : \mathbf{SEBES}_{\mathcal{M}} \times \mathbf{SEBES}_{\mathcal{M}} \to \mathbf{SEBES}_{\mathcal{M}}$ *with*
$\mathfrak{E}_1 \widetilde{\|}_{A,M} \mathfrak{E}_2 = (\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2 \upharpoonright E', \mathcal{M}')$

where $E' = \{(e, \star) \mid l_1(e) \notin A \wedge (e \in \mathrm{dom}(\mathcal{M}_1) \Rightarrow \widehat{M(l_1(e))}(l, \mathcal{M}_1(e))$ *is defined*$)\} \cup$
$\{(\star_2, e) \mid l_2(e) \notin A \wedge (e \in \mathrm{dom}(\mathcal{M}_2) \Rightarrow \widehat{M(l_2(e))}(r, \mathcal{M}_2(e))$ *is defined*$)\} \cup$
$\{(e_1, e_2) \mid l_1(e_1) = l(e_2) \wedge \mathcal{M}_1(e_1) \simeq \mathcal{M}_2(e_2)\}$.

and $\mathcal{M}'(e') \simeq \begin{cases} \widehat{M(l_1(e))}(l, \mathcal{M}_1(e)) & \text{if } e' = (e, \star) \\ \widehat{M(l_2(e))}(r, \mathcal{M}_2(e)) & \text{if } e' = (\star, e) \\ \mathcal{M}_1(e_1) & \text{if } e = (e_1, e_2) \wedge l_1(e_1) = l(e_2) \wedge \mathcal{M}_1(e_1) \simeq \mathcal{M}_2(e_2) \end{cases}$

$\widetilde{\backslash\backslash}_M A : \mathbf{SEBES}_{\mathcal{M}} \to \mathbf{SEBES}_{\mathcal{M}}$ *with* $\mathfrak{E} \widetilde{\backslash\backslash}_M A = (\mathcal{E} \widehat{\backslash\backslash} A \upharpoonright E', \mathcal{M}' \upharpoonright E')$
where $E' = \{e \in E \mid e \in \mathrm{dom}(\mathcal{M}) \Rightarrow \widehat{M(l(e))}(l, \mathcal{M}(e))$ *is defined*$\}$
and $\mathcal{M}'(e) \simeq \widehat{M(l(e))}(l, \mathcal{M}(e))$

$\widetilde{Ref}^{se}_{A,M_A} : \mathbf{SEBES}_{\mathcal{M}} \times (A \to \mathbf{SEBES}) \times (A \to \mathbf{SEBES}^\star_{\mathcal{M}}) \to \mathbf{SEBES}_{\mathcal{M}}$
with $\widetilde{Ref}^{se}_{A,M_A}(\mathfrak{E}, \theta, \vec{\theta}) = (\underline{Ref}^{se}(\mathcal{E}, \vartheta) \upharpoonright E', \mathcal{M}')$

where $\vartheta(e) = \begin{cases} \theta(l(e)) & \text{if } l(e) \in A \wedge e \notin \mathrm{dom}(\mathcal{M}) \\ \pi_1(\vec{\theta}(l(e))[\mathcal{M}(e)]) & \text{if } l(e) \in A \wedge e \in \mathrm{dom}(\mathcal{M}) \\ (\emptyset, \emptyset, \emptyset, \emptyset, \{\emptyset\}, \emptyset) & \text{otherwise} \end{cases}$

and $E' = \{(e, e) \mid l(e) \notin A \wedge (e \in \mathrm{dom}(\mathcal{M}) \Rightarrow \widehat{M_A(l(e))}(0, \mathcal{M}(e))$ *is defined*$)\} \cup$
$\{(e, \hat{e}) \mid l(e) \in A \wedge (e \in \mathrm{dom}(\mathcal{M}) \Rightarrow (\exists \mathcal{E}'', \mathcal{M}'' : \vec{\theta}(l(e))[\mathcal{M}(e)] = (\mathcal{E}'', \mathcal{M}'') \wedge$
$(\hat{e} \in \mathrm{dom}(\mathcal{M}'') \Rightarrow \widehat{M_A(l''(\hat{e}))}((l(e), \mathcal{M}(e)), \mathcal{M}''(\hat{e}))$ *is defined*$)))\}$.

$$\textit{and } \mathcal{M}'(e') \simeq \begin{cases} \widehat{M_A(l(e))}(0, \mathcal{M}(e)) & \textit{if } e' = (e,e) \wedge l(e) \notin A \\ \widehat{M_A(l''(\hat{e}))}((l(e), \mathcal{M}(e)), \mathcal{M}''(\hat{e})) & \textit{if } e' = (e,\hat{e}) \wedge l(e) \in A \wedge \\ & \quad \vec{\theta}(l(e))[\mathcal{M}(e)] = (\mathcal{E}'', \mathcal{M}'') \end{cases}$$

$\widetilde{Shift}_q : \mathbf{SEBES}_{\mathcal{M}} \to \mathbf{SEBES}_{\mathcal{M}}$ *with* $\widetilde{Shift}_q(\mathfrak{E}) = (\widehat{Shift}_q(\mathcal{E}), \Psi(\mathcal{M}, q))$

These operators are used to define the following denotational semantics for $\mathsf{EXP}^{\mathsf{O}'}_{\mathsf{se}}$.

Definition 7.45 (Denotational semantics of $\mathsf{EXP}^{\mathsf{O}'}_{\mathsf{se}}$)
Define $[\![\_]\!]' : (\mathsf{Var} \to \mathsf{EXP}_{\mathsf{se}}) \times \mathsf{EXP}^{\mathsf{O}'}_{\mathsf{se}} \to \mathbf{SEBES}_{\mathcal{M}}$ *as follows*

$$\begin{array}{l} [\![\mathrm{decl}, B]\!]' = ([\![\langle \mathrm{decl}, B\rangle]\!], \bot) \qquad [\![\mathrm{decl}, b^-]\!]' = ([\![\langle \mathrm{decl}, b\rangle]\!], \{(\bullet, 1)\}) \\ [\![\mathrm{decl}, G_1 \dot{+} B_2]\!]' = [\![\mathrm{decl}, G_1]\!]' \widetilde{\dot{+}} [\![\langle \mathrm{decl}, B_2\rangle]\!] \\ [\![\mathrm{decl}, G_1 \oplus_M G_2]\!]' = [\![\mathrm{decl}, G_1]\!]' \widetilde{\oplus}_M [\![\mathrm{decl}, G_2]\!]' \\ [\![\mathrm{decl}, G; B]\!]' = [\![\mathrm{decl}, G]\!]' \,\widetilde{;}\, [\![\langle \mathrm{decl}, B\rangle]\!] \\ [\![\mathrm{decl}, G_1 \|_{A,M} G_2]\!]' = [\![\mathrm{decl}, G_1]\!]' \widetilde{\|}_{A,M} [\![\mathrm{decl}, G_2]\!]' \\ [\![\mathrm{decl}, G \backslash\backslash_M A]\!]' = [\![\mathrm{decl}, G]\!]' \,\widetilde{\backslash\backslash}_M A \\ [\![\mathrm{decl}, G[(a \to B_a)^{a\in A}, (a \to \vec{G}_a)^{a\in A}]_{M_A}]\!]' = \widetilde{Ref}^{se}_{A,M_A}([\![\mathrm{decl}, G]\!]', \theta, \vec{\theta}) \\ \quad \textit{where } \theta(a) = [\![\langle \mathrm{decl}, B_a\rangle]\!] \textit{ and } \vec{\theta}(a)[i] \simeq [\![\mathrm{decl}, \vec{G}_a[i]]\!]' \\ [\![\mathrm{decl}, \lceil G \rceil_q]\!]' = \widetilde{Shift}_q([\![\mathrm{decl}, G]\!]') \end{array}$$

It is easy to check that $[\![\_]\!]'$ is well defined.

The following lemma states how the remainder (respectively the start-remainder) of a sebes is determined with respect to the different operators of Definition 7.11. Furthermore, the termination predicate is reduced to the termination predicate of the components of the operator.

Lemma 7.46 *Suppose* $\mathcal{E}, \mathcal{E}_1, \mathcal{E}_2 \in \mathbf{SEBES}$ *and* $\vartheta : \mathcal{U} \to \mathbf{SEBES}$. *Then*

$$\begin{array}{rcl} (\mathcal{E}_1 \widehat{\dot{+}} \mathcal{E}_2)_{[(\star_i, e)]} & \simeq & \widehat{Shift}_i(\mathcal{E}_{i[e]}) \\ (\mathcal{E}_1 \widehat{\dot{+}} \mathcal{E}_2)_{[(\star_i, e)]} & \simeq & \widehat{Shift}_i(\mathcal{E}_{i[e]}) \\ (\mathcal{E}_1 \widehat{\oplus} \mathcal{E}_2)_{[(\star_i, e)]} & \simeq & \widehat{Shift}_i(\mathcal{E}_{i[e]}) \\ (\mathcal{E}_1 \,\widehat{;}\, \mathcal{E}_2)_{[(\star_1, e)]} & \simeq & \begin{cases} \widehat{Shift}_2(\mathcal{E}_2) & \textit{if } e \in \mathrm{init}(\mathcal{E}_1) \wedge \Upsilon(T_1, e) \\ \mathcal{E}_{1[e]} \,\widehat{;}\, \mathcal{E}_2 & \textit{otherwise} \end{cases} \\ (\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2)_{[(e_1, \star)]} & \simeq & \begin{cases} \mathcal{E}_{1[e_1]} \widehat{\|}_A \mathcal{E}_2 & \textit{if } \neg\Upsilon(T_1, e_1) \wedge l_1(e_1) \notin A \\ \widehat{Shift}_r(\mathcal{E}_2) \,\widehat{\backslash\backslash} A & \textit{if } e_1 \in \mathrm{init}(\mathcal{E}_1) \wedge \Upsilon(T_1, e_1) \wedge l_1(e_1) \notin A \end{cases} \\ (\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2)_{[(\star, e_2)]} & \simeq & \begin{cases} \mathcal{E}_1 \widehat{\|}_A \mathcal{E}_{2[e_2]} & \textit{if } \neg\Upsilon(T_2, e_2) \wedge l_2(e_2) \notin A \\ \widehat{Shift}_l(\mathcal{E}_1) \,\widehat{\backslash\backslash} A & \textit{if } e_2 \in \mathrm{init}(\mathcal{E}_2) \wedge \Upsilon(T_2, e_2) \wedge l_2(e_2) \notin A \end{cases} \\ (\mathcal{E}_1 \widehat{\|}_A \mathcal{E}_2)_{[(e_1, e_2)]} & \simeq & \begin{cases} \mathcal{E}_{1[e_1]} \widehat{\|}_A \mathcal{E}_{2[e_2]} & \textit{if } \neg\Upsilon(T_1, e_1) \wedge \neg\Upsilon(T_2, e_2) \\ \widehat{Shift}_r(\mathcal{E}_{2[e_2]}) \,\widehat{\backslash\backslash} A & \textit{if } e_1 \in \mathrm{init}(\mathcal{E}_1) \wedge \Upsilon(T_1, e_1) \wedge \neg\Upsilon(T_2, e_2) \\ \widehat{Shift}_l(\mathcal{E}_{1[e_1]}) \,\widehat{\backslash\backslash} A & \textit{if } e_2 \in \mathrm{init}(\mathcal{E}_2) \wedge \Upsilon(T_2, e_2) \wedge \neg\Upsilon(T_1, e_1) \\ (\emptyset, \emptyset, \emptyset, \emptyset, \{\emptyset\}, \emptyset) & \textit{if } \Upsilon(T_1, e_1) \wedge \Upsilon(T_2, e_2) \end{cases} \end{array}$$

whenever $l_1(e_1) = l_2(e_2) \in A$

$$
\begin{array}{rcl}
(\mathcal{E}\,\widehat{\backslash\backslash} A)_{[e]} & \simeq & \begin{cases} \mathcal{E}_{[e]}\,\widehat{\backslash\backslash} A & \textit{if } l(e) \notin A \\ \textit{undefined} & \textit{otherwise} \end{cases} \\
\underline{Ref}^{se}_{A}(\mathcal{E},\vartheta)_{[(e,\hat{e})]} & \simeq & \begin{cases} \underline{Ref}^{se}_{A}(\mathcal{E}_{[e]},\vartheta) & \textit{if } l(e) \notin A \wedge e = \hat{e} \\ \underline{Ref}^{se}_{A}(\mathcal{E}_{[e]},\vartheta[e \to \vartheta(e)_{[\hat{e}]}]) & \textit{if } l(e) \in A \wedge \Upsilon(T_{\vartheta(e)},\hat{e}) \\ \underline{Ref}^{se}_{A}(\mathcal{E}_{\langle e\rangle},\vartheta[e \to \vartheta(e)_{[\hat{e}]}])\} & \textit{if } l(e) \in A \wedge \neg\Upsilon(T_{\vartheta(e)},\hat{e}) \end{cases} \\
\widehat{Shift}_i(\mathcal{E})_{[(\star_i,e)]} & \simeq & \widehat{Shift}_i(\mathcal{E}_{[e]}) \qquad \textit{whenever } i \in \{1,2\} \\
\widehat{Shift}_r(\mathcal{E})_{[(\star,e)]} & \simeq & \widehat{Shift}_r(\mathcal{E}_{[e]}) \\
\widehat{Shift}_l(\mathcal{E})_{[(e,\star)]} & \simeq & \widehat{Shift}_l(\mathcal{E}_{[e]})
\end{array}
$$

and $\underline{Ref}^{se}_{A}(\mathcal{E}_{[e]},\vartheta[e \to \mathcal{E}']) \simeq \underline{Ref}^{se}_{A}(\mathcal{E}_{[e]},\vartheta)$ *holds for any* $\mathcal{E}' \in \mathbf{SEBES}$.

Furthermore

$$
\begin{array}{rcl}
(\mathcal{E}_1\,\widehat{+}\,\mathcal{E}_2)_{\langle(\star_i,e)\rangle} & \simeq & \widehat{Shift}_i(\mathcal{E}_{i\langle e\rangle}) \\
(\mathcal{E}_1\,\widehat{\dagger}\,\mathcal{E}_2)_{\langle(\star_i,e)\rangle} & \simeq & \begin{cases} \mathcal{E}_{1\langle e\rangle}\,\widehat{\dagger}\,\mathcal{E}_2 & \textit{if } i = 1 \\ \widehat{Shift}_2(\mathcal{E}_{2\langle e\rangle}) & \textit{if } i = 2 \end{cases} \\
(\mathcal{E}_1\,\widehat{\oplus}\,\mathcal{E}_2)_{\langle(\star_i,e)\rangle} & \simeq & \begin{cases} \mathcal{E}_{1\langle e\rangle}\,\widehat{\oplus}\,\mathcal{E}_2 & \textit{if } i = 1 \\ \mathcal{E}_1\,\widehat{\oplus}\,\mathcal{E}_{2\langle e\rangle} & \textit{if } i = 2 \end{cases} \\
(\mathcal{E}_1\,\widehat{;}\,\mathcal{E}_2)_{\langle(\star_1,e)\rangle} & \simeq & \mathcal{E}_{1\langle e\rangle}\,\widehat{;}\,\mathcal{E}_2 \\
(\mathcal{E}_1\,\widehat{\|}_A\mathcal{E}_2)_{\langle(e_1,e_2)\rangle} & \simeq & \begin{cases} \mathcal{E}_{1\langle e_1\rangle}\,\widehat{\|}_A\mathcal{E}_2 & \textit{if } l_1(e_1) \notin A \wedge e_2 = \star \\ \mathcal{E}_1\,\widehat{\|}_A\mathcal{E}_{2\langle e_2\rangle} & \textit{if } l_2(e_2) \notin A \wedge e_1 = \star \\ (\mathcal{E}_{1\langle e_1\rangle}\,\widehat{\|}_A\mathcal{E}_{2\langle e_2\rangle}) \upharpoonright E' & \textit{if } l_1(e_1) = l_2(e_2) \in A \wedge E' = \{(e_1',e_2') \mid \\ & \qquad (e_1' \neq e_1 \wedge e_2' \neq e_2) \vee (e_1',e_2') = (e_1,e_2)\} \end{cases} \\
(\mathcal{E}\,\widehat{\backslash\backslash} A)_{\langle e\rangle} & \simeq & \begin{cases} \mathcal{E}_{\langle e\rangle}\,\widehat{\backslash\backslash} A & \textit{if } l(e) \notin A \\ \textit{undefined} & \textit{otherwise} \end{cases} \\
\underline{Ref}^{se}_{A}(\mathcal{E},\vartheta)_{\langle(e,\hat{e})\rangle} & \simeq & \begin{cases} \underline{Ref}^{se}_{A}(\mathcal{E}_{\langle e\rangle},\vartheta) & \textit{if } l(e) \notin A \wedge e = \hat{e} \\ \underline{Ref}^{se}_{A}(\mathcal{E}_{\langle e\rangle},\vartheta[e \to \vartheta(e)_{\langle\hat{e}\rangle}]) & \textit{if } l(e) \in A \end{cases} \\
\widehat{Shift}_i(\mathcal{E})_{\langle(\star_i,e)\rangle} & \simeq & \widehat{Shift}_i(\mathcal{E}_{\langle e\rangle}) \qquad \textit{whenever } i \in \{1,2\} \\
\widehat{Shift}_r(\mathcal{E})_{\langle(\star,e)\rangle} & \simeq & \widehat{Shift}_r(\mathcal{E}_{\langle e\rangle}) \\
\widehat{Shift}_l(\mathcal{E})_{\langle(e,\star)\rangle} & \simeq & \widehat{Shift}_l(\mathcal{E}_{\langle e\rangle})
\end{array}
$$

Moreover,

$$
\begin{array}{rcl}
\Upsilon(T_{\mathcal{E}_1\,\widehat{+}\,\mathcal{E}_2},(\star_i,e)) & \Leftrightarrow & \Upsilon(T_i,e) \\
\Upsilon(T_{\mathcal{E}_1\,\widehat{\dagger}\,\mathcal{E}_2},(\star_i,e)) & \Leftrightarrow & \Upsilon(T_i,e) \\
\Upsilon(T_{\mathcal{E}_1\,\widehat{\oplus}\,\mathcal{E}_2},(\star_i,e)) & \Leftrightarrow & \Upsilon(T_i,e) \\
\Upsilon(T_{\mathcal{E}_1\,\widehat{;}\,\mathcal{E}_2},(\star_i,e)) & \Leftrightarrow & \Upsilon(T_2,e) \wedge i = 2 \\
\Upsilon(T_{\mathcal{E}_1\,\widehat{\|}_A\mathcal{E}_2},(e_1,e_2)) & \Leftrightarrow & (\Upsilon(T_1,e_1) \wedge \Upsilon(T_2,e_2) \wedge l_1(e_1) = l_2(e_2) \in A) \\
\Upsilon(T_{\mathcal{E}\,\widehat{\backslash\backslash} A},e) & \Leftrightarrow & (\Upsilon(T,e) \wedge l(e) \notin A)
\end{array}
$$

$$\begin{array}{rcl}
\Upsilon(T_{\underline{Ref}_A^{se}(\mathcal{E},\vartheta)},(e,\hat{e})) & \Leftrightarrow & \left\{\begin{array}{ll} \Upsilon(T,e) & \textit{if } l(e)\notin A \wedge e=\hat{e} \\ \Upsilon(T,e)\wedge\Upsilon(T_{\vartheta(e)},\hat{e}) & \textit{if } l(e)\in A \end{array}\right. \\
\Upsilon(T_{\widehat{Shift}_i(\mathcal{E})},(\star_i,e)) & \Leftrightarrow & \Upsilon(T,e) \textit{ whenever } i\in\{1,2\} \\
\Upsilon(T_{\widehat{Shift}_l(\mathcal{E})},(e,\star)) & \Leftrightarrow & \Upsilon(T,e) \\
\Upsilon(T_{\widehat{Shift}_r(\mathcal{E})},(\star,e)) & \Leftrightarrow & \Upsilon(T,e)
\end{array}$$

Proof: Straightforward. □

Now we are ready to show how the remainder on $\mathbf{SEBES}_{\mathcal{M}}$ defined in Definition 7.25 is determined with respect to the different operators of Definition 7.44.

Lemma 7.47 *Let $A \subseteq \mathsf{Obs}$, $M : \mathsf{Obs} \to \{l,r\}^\star$, $M_A : \mathsf{Obs} \to ((A\times\mathbb{N})\cup\{0\})^\star$ and $q\in\{1,2,l,r\}$. Furthermore, suppose $\mathcal{E}=(\mathcal{E},\mathcal{M})$ and $\mathcal{E}_j=(\mathcal{E}_j,\mathcal{M}_j)$. Then*

$$(\mathcal{E}_1\widetilde{+}\mathcal{E}_2)_{\rangle(\star_i,e)\langle} \simeq \left\{\begin{array}{ll} \mathcal{E}_{1\rangle e\langle}\widetilde{+}\mathcal{E}_2 & \textit{if } i=1\wedge e\notin\mathrm{dom}(\mathcal{M}_1)\wedge l(e)\in\mathsf{Obs} \\ \widetilde{Shift}_1(\mathcal{E}_{1\rangle e\langle}) & \textit{if } i=1\wedge(e\in\mathrm{dom}(\mathcal{M}_1)\vee l(e)=\tau) \\ \widetilde{Shift}_2((\mathcal{E}_2,\bot)_{\rangle e\langle}) & \textit{if } i=2 \end{array}\right.$$

Let $\mathcal{E}_c=(\mathcal{E}_1\widetilde{\oplus}_M\mathcal{E}_2)_{\rangle(\star_i,e)\langle}$ then

$$\mathcal{E}_c \simeq \left\{\begin{array}{ll}
\mathcal{E}_{1\rangle e\langle}\widetilde{\oplus}_{[l_1(e),l]\cdot M}\mathcal{E}_2 & \textit{if } i=1\wedge l_1(e)\in\mathsf{Obs}\wedge e\notin\mathrm{dom}(\mathcal{M}_1) \\
\mathcal{E}_1\widetilde{\oplus}_{[l_2(e),r]\cdot M}\mathcal{E}_{2\rangle e\langle} & \textit{if } i=2\wedge l_2(e)\in\mathsf{Obs}\wedge e\notin\mathrm{dom}(\mathcal{M}_2) \\
\widetilde{Shift}_1(\mathcal{E}_{1\rangle e\langle})\,\widetilde{\backslash\!\backslash}_{M\setminus(l_1(e),m)}\emptyset & \textit{if } i=1\wedge l_1(e)\in\mathsf{Obs}\wedge m=\widehat{M(l_1(e))}(l,\mathcal{M}_1(e)) \\
\widetilde{Shift}_2(\mathcal{E}_{2\rangle e\langle})\,\widetilde{\backslash\!\backslash}_{\overline{M}\setminus(l_2(e),m)}\emptyset & \textit{if } i=2\wedge l_2(e)\in\mathsf{Obs}\wedge m=\widehat{M(l_2(e))}(r,\mathcal{M}_2(e)) \\
\widetilde{Shift}_1(\mathcal{E}_{1\rangle e\langle})\,\widetilde{\backslash\!\backslash}_M\emptyset & \textit{if } i=1\wedge l_1(e)=\tau \\
\widetilde{Shift}_2(\mathcal{E}_{2\rangle e\langle})\,\widetilde{\backslash\!\backslash}_{\overline{M}}\emptyset & \textit{if } i=2\wedge l_2(e)=\tau
\end{array}\right.$$

$$(\mathcal{E}_1\widetilde{;}\mathcal{E}_2)_{\rangle(\star_1,e)\langle} \simeq \left\{\begin{array}{ll} \widetilde{Shift}_2((\mathcal{E}_2,\bot)) & \textit{if } (e\in\mathrm{dom}(\mathcal{M}_1)\vee e\in\mathrm{init}_\tau(\mathcal{E}_1))\wedge\Upsilon(T_1,e) \\ \mathcal{E}_{1\rangle e\langle}\widehat{;}\mathcal{E}_2 & \textit{otherwise} \end{array}\right.$$

Let $\mathcal{E}_p=(\mathcal{E}_1\widetilde{\|}_{A,M}\mathcal{E}_2)_{\rangle(e_1,e_2)\langle}$ then

$$\mathcal{E}_p \simeq \left\{\begin{array}{ll}
\mathcal{E}_{1\rangle e\langle}\widetilde{\|}_{A,[l_1(e),l]\cdot M}\mathcal{E}_2 & \textit{if } l_1(e)\in\mathsf{Obs}\backslash A\wedge e\notin\mathrm{dom}(\mathcal{M}_1) \\
\mathcal{E}_{1\rangle e\langle}\widetilde{\|}_{A,M\setminus(l_1(e),\widehat{M(l_1(e))}(l,\mathcal{M}_1(e)))}\mathcal{E}_2 & \textit{if } l_1(e)\in\mathsf{Obs}\backslash A\wedge\neg\Upsilon(T_1,e)\wedge \\
 & \quad e\in\mathrm{dom}(\mathcal{M}_1) \\
\mathcal{E}_{1\rangle e\langle}\widetilde{\|}_{A,M}\mathcal{E}_2 & \textit{if } l_1(e)=\tau\wedge\neg\Upsilon(T_1,e) \\
\widetilde{Shift}_r(\mathcal{E}_2)\,\widetilde{\backslash\!\backslash}_{\overline{M}\setminus(l_1(e),\widehat{M(l_1(e))}(l,\mathcal{M}_1(e)))}A & \textit{if } e\in\mathrm{init}_{\mathsf{Obs}\backslash A}(\mathcal{E}_1)\wedge\Upsilon(T_1,e)\wedge \\
 & \quad e\in\mathrm{dom}(\mathcal{M}_1) \\
\widetilde{Shift}_r(\mathcal{E}_2)\,\widetilde{\backslash\!\backslash}_{\overline{M}}A & \textit{if } e\in\mathrm{init}_\tau(\mathcal{E}_1)\wedge\Upsilon(T_1,e)
\end{array}\right.$$

whenever $e_1=e\wedge e_2=\star$

$$\mathcal{E}_p \simeq \begin{cases} \mathcal{E}_1 \widetilde{\|}_{A,[l_2(e),r]\cdot M} \mathcal{E}_{2\rangle e\langle} & \textit{if } l_2(e) \in \mathsf{Obs} \backslash A \wedge e \notin \mathrm{dom}(\mathcal{M}_2) \\ \mathcal{E}_1 \widetilde{\|}_{A,M\backslash(l_2(e),\widehat{M(l_2(e))}(r,\mathcal{M}_2(e)))} \mathcal{E}_{2\rangle e\langle} & \textit{if } l_2(e) \in \mathsf{Obs} \backslash A \wedge \neg\Upsilon(T_2,e) \wedge \\ & \quad e \in \mathrm{dom}(\mathcal{M}_2) \\ \mathcal{E}_1 \widetilde{\|}_{A,M} \mathcal{E}_{2\rangle e\langle} & \textit{if } l_2(e) = \tau \wedge \neg\Upsilon(T_2,e) \\ \widetilde{Shift}_l(\mathcal{E}_1) \widetilde{\backslash\backslash}_{M\backslash(l_2(e),\widehat{M(l_2(e))}(r,\mathcal{M}_2(e)))} A & \textit{if } e \in \mathrm{init}_{\mathsf{Obs}\backslash A}(\mathcal{E}_2) \wedge \Upsilon(T_2,e) \wedge \\ & \quad e \in \mathrm{dom}(\mathcal{M}_2) \\ \widetilde{Shift}_l(\mathcal{E}_1) \widetilde{\backslash\backslash}_M A & \textit{if } e \in \mathrm{init}_\tau(\mathcal{E}_2) \wedge \Upsilon(T_2,e) \end{cases}$$

whenever $e_1 = \star \wedge e_2 = e$

$$\mathcal{E}_p \simeq \begin{cases} \mathcal{E}_{1\rangle e_1\langle} \widetilde{\|}_{A,M} \mathcal{E}_{2\rangle e_2\langle} & \textit{if } \mathcal{M}_1(e_1) \simeq \mathcal{M}_2(e_2) \wedge \neg\Upsilon(T_1,e_1) \wedge \neg\Upsilon(T_2,e_2) \\ \widetilde{Shift}_r(\mathcal{E}_{2\rangle e_2\langle}) \widetilde{\backslash\backslash}_{\overline{M}} A & \textit{if } \mathcal{M}_1(e_1) = \mathcal{M}_2(e_2) \wedge \Upsilon(T_1,e_1) \wedge \neg\Upsilon(T_2,e_2) \\ \widetilde{Shift}_l(\mathcal{E}_{1\rangle e_1\langle}) \widetilde{\backslash\backslash}_M A & \textit{if } \mathcal{M}_1(e_1) = \mathcal{M}_2(e_2) \wedge \Upsilon(T_2,e_2) \wedge \neg\Upsilon(T_1,e_1) \\ (\emptyset,\emptyset,\emptyset,\emptyset,\{\emptyset\},\emptyset) & \textit{if } \Upsilon(T_1,e_1) \wedge \Upsilon(T_2,e_2) \end{cases}$$

whenever $l_1(e_1) = l_2(e_2) \in A$ *(please note that* $\mathcal{M}_1(e_1) = \mathcal{M}_2(e_2) \Rightarrow e_i \in \mathrm{dom}(\mathcal{M}_i)$*)*

$$(\mathcal{E} \widetilde{\backslash\backslash}_M A)_{\rangle e\langle} \simeq \begin{cases} \mathcal{E}_{\rangle e\langle} \widehat{\backslash\backslash}_{[l(e),l]\cdot M} A & \textit{if } l(e) \notin A \wedge e \notin \mathrm{dom}(\mathcal{M}) \\ \mathcal{E}_{\rangle e\langle} \widehat{\backslash\backslash}_{M\backslash(l(e),\widehat{M(l(e))}(l,\mathcal{M}(e)))} A & \textit{if } l(e) \notin A \wedge e \in \mathrm{dom}(\mathcal{M}) \\ \mathcal{E}_{\rangle e\langle} \widehat{\backslash\backslash}_M A & \textit{if } l(e) = \tau \end{cases}$$

Let $\mathcal{E}_r = \widetilde{Ref}^{se}_{A,M_A}(\mathcal{E},\theta,\vec{\theta})_{\rangle(e,\hat{e})\langle}$ *then*

$$\mathcal{E}_r \simeq \begin{cases} \widetilde{Ref}^{se}_{A,[l(e),0]\cdot M_A}(\mathcal{E}_{\rangle e\langle},\theta,\vec{\theta}) & \textit{if } l(e) \in \mathsf{Obs} \wedge \hat{e} = e \wedge e \notin \mathrm{dom}(\mathcal{M}) \\ \widetilde{Ref}^{se}_{A,M_A\backslash(l(e),\widehat{M_A(l(e))}(0,\mathcal{M}(e)))}(\mathcal{E}_{\rangle e\langle},\theta,\vec{\theta}) & \textit{if } l(e) \in \mathsf{Obs} \wedge \hat{e} = e \\ \widetilde{Ref}^{se}_{A,M_A}(\mathcal{E}_{\rangle e\langle},\theta,\vec{\theta}) & \textit{if } l(e) = \tau \wedge \hat{e} = e \end{cases}$$

whenever $l(e) \notin A$

$$\mathcal{E}_r \simeq \begin{cases} \widetilde{Ref}^{se}_{A,[l_{\theta(l(e))}(\hat{e}),(l(e),1)]\cdot(M_A \diamond l(e))}(\mathcal{E}_{\rangle e\langle},\theta,\vec{\theta}[l(e) \to \theta(l(e))_{\rangle\hat{e}\langle} \cdot \vec{\theta}(l(e))]) & \\ & \textit{if } l_{\theta(l(e))} \in \mathsf{Obs} \\ \widetilde{Ref}^{se}_{A,M_A \diamond l(e)}(\mathcal{E}_{\rangle e\langle},\theta,\vec{\theta}[l(e) \to \theta(l(e))_{\rangle\hat{e}\langle} \cdot \vec{\theta}(l(e))]) & \textit{if } l_{\theta(l(e))} = \tau \wedge \neg\Upsilon(T_{\theta(l(e))},\hat{e}) \\ \widetilde{Ref}^{se}_{A,M_A}(\mathcal{E}_{\rangle e\langle\rangle e\langle},\theta,\vec{\theta}) & \textit{if } l_{\theta(l(e))} = \tau \wedge \Upsilon(T_{\theta(l(e))},\hat{e}) \end{cases}$$

whenever $l(e) \in A \wedge e \notin \mathrm{dom}(\mathcal{M})$

$$\mathscr{E}_r \simeq \begin{cases} \widetilde{Ref}^{se}_{A,[b,(l(e),\mathcal{M}(e))]\cdot M_A}(\mathscr{E},\theta,\vec{\theta}[l(e)\rightarrow\vec{\theta}(l(e))\pm(\mathcal{M}(e),\vec{\theta}(l(e))_{\rangle\hat{e}\langle})]) \\ \quad \textit{if } b\in\mathsf{Obs}\wedge\hat{e}\notin\mathrm{dom}(\pi_2(\vec{\theta}(l(e))[\mathcal{M}(e)])) \\ \widetilde{Ref}^{se}_{A,M_A\setminus(b,\widehat{M_A(b)}((l(e),\mathcal{M}(e)),\mathcal{M}'(\hat{e})))}(\mathscr{E},\theta,\vec{\theta}[l(e)\rightarrow\vec{\theta}(l(e))\pm(\mathcal{M}(e),\vec{\theta}(l(e))_{\rangle\hat{e}\langle})]) \\ \quad \textit{if } b\in\mathsf{Obs}\wedge\mathcal{M}'=\pi_2(\vec{\theta}(l(e))[\mathcal{M}(e)])\wedge\neg\Upsilon(T_{\vec{\theta}(l(e))[\mathcal{M}(e)]},\hat{e})\wedge\hat{e}\in\mathrm{dom}(\mathcal{M}') \\ \widetilde{Ref}^{se}_{A,M_A}(\mathscr{E},\theta,\vec{\theta}[l(e)\rightarrow\vec{\theta}(l(e))\pm(\mathcal{M}(e),\vec{\theta}(l(e))_{\rangle\hat{e}\langle})]) \\ \quad \textit{if } b=\tau\wedge\neg\Upsilon(T_{\vec{\theta}(l(e))[\mathcal{M}(e)]},\hat{e}) \\ \widetilde{Ref}^{se}_{A,(M_A\dagger(l(e),\mathcal{M}(e)))\setminus(b,\widehat{M_A(b)}((l(e),\mathcal{M}(e)),\mathcal{M}'(\hat{e})))}(\mathscr{E}_{\rangle e\langle},\theta,\vec{\theta}[l(e)\rightarrow\vec{\theta}(l(e))\setminus\mathcal{M}(e)]) \\ \quad \textit{if } b\in\mathsf{Obs}\wedge\mathcal{M}'=\pi_2(\vec{\theta}(l(e))[\mathcal{M}(e)])\wedge\Upsilon(T_{\vec{\theta}(l(e))[\mathcal{M}(e)]},\hat{e})\wedge\hat{e}\in\mathrm{dom}(\mathcal{M}') \\ \widetilde{Ref}^{se}_{A,(M_A\dagger(l(e),\mathcal{M}(e)))}(\mathscr{E}_{\rangle e\langle},\theta,\vec{\theta}[l(e)\rightarrow\vec{\theta}(l(e))\setminus\mathcal{M}(e)]) \\ \quad \textit{if } b=\tau\wedge\Upsilon(T_{\vec{\theta}(l(e))[\mathcal{M}(e)]},\hat{e}) \end{cases}$$

whenever $l(e)\in A\wedge l_{\theta(l(e))}(\hat{e})=b\wedge e\in\mathrm{dom}(\mathcal{M})$

$\widetilde{Shift}_i(\mathscr{E})_{\rangle(\star_i,e)\langle}\simeq\widetilde{Shift}_i(\mathscr{E}_{\rangle e\langle})$ *whenever* $i\in\{1,2\}$

$\widetilde{Shift}_r(\mathscr{E})_{\rangle(\star,e)\langle}\simeq\widetilde{Shift}_r(\mathscr{E}_{\rangle e\langle})$

$\widetilde{Shift}_l(\mathscr{E})_{\rangle(e,\star)\langle}\simeq\widetilde{Shift}_l(\mathscr{E}_{\rangle e\langle})$

Proof: Can be straightforwardly checked by using Lemma 7.46. □

The following lemma states that every transition of $\longrightarrow'$ is matched by $\hookrightarrow^c$.

Lemma 7.48 *Suppose* $G\in\mathsf{EXP}^{\mathsf{O}'}_{\mathsf{se}}$ *and* $\mathrm{decl}:\mathsf{Var}\rightarrow\mathsf{EXP}_{\mathsf{se}}$. *Then for all* $G'\in\mathsf{EXP}^{\mathsf{O}'}_{\mathsf{se}}$ *and* $\gamma\in\mathcal{L}_{\mathsf{se}}$ *we have*

$$G\xrightarrow[e]{\gamma}{}'_{\mathrm{decl}}\,G'\Rightarrow(\llbracket(\mathrm{decl},G)\rrbracket'\overset{\gamma}{\hookrightarrow}{}^{c}\llbracket(\mathrm{decl},G')\rrbracket'\wedge\llbracket(\mathrm{decl},G')\rrbracket'=\llbracket(\mathrm{decl},G)\rrbracket'_{\rangle e\langle})$$

Proof: It follows by induction on the depth of inferences of $G\xrightarrow[e]{\gamma}{}'_{\mathrm{decl}}\,G'$, where Lemma 7.46 and Lemma 7.47 are used. Furthermore, in the case of Rec also Lemma 7.42 is applied. □

Every started action can be immediately finished in $\longrightarrow'_{\mathrm{decl}}$:

Lemma 7.49 *Suppose* $G,G'\in\mathsf{EXP}^{\mathsf{O}'}_{\mathsf{se}}$, $\mathrm{decl}:\mathsf{Var}\rightarrow\mathsf{EXP}_{\mathsf{se}}$, $a\in\mathsf{Obs}$ *and* $e\in\mathcal{U}$. *Then*

$$G\xrightarrow[e]{a^+}{}'_{\mathrm{decl}}\,G'\Rightarrow\exists G'':(G'\xrightarrow[e]{a_1^-}{}'_{\mathrm{decl}}\,G''\vee G'\xrightarrow[e]{a_1^-\surd}{}'_{\mathrm{decl}}\,G'')$$

Proof: It follows straightforwardly by induction on the depths of inferences of $G\xrightarrow[e]{a^+}{}'_{\mathrm{decl}}\,G'$ and is hence omitted. □

The following lemma states that every transition of $\hookrightarrow^c$ is matched by $\longrightarrow'$.

Lemma 7.50 *Suppose* $G\in\mathsf{EXP}^{\mathsf{O}'}_{\mathsf{se}}$, $\mathrm{decl}:\mathsf{Var}\rightarrow\mathsf{EXP}_{\mathsf{se}}$, $(\mathcal{E},\mathcal{M})=\llbracket(\mathrm{decl},G)\rrbracket'$ *and* $e\in\mathrm{init}(\mathcal{E})$. *Then there exists* $G'\in\mathsf{EXP}^{\mathsf{O}'}_{\mathsf{se}}$ *and* $\gamma\in\mathcal{L}_{\mathsf{se}}$ *such that*

$$G\xrightarrow[e]{\gamma}{}'_{\mathrm{decl}}\,G'\wedge\gamma\in\begin{cases}\{l(e)^-_{\mathcal{M}(e)},l(e)^-_{\mathcal{M}(e)}\surd\} & \textit{if } l(e)\in\mathsf{Obs}\wedge e\in\mathrm{dom}(\mathcal{M}) \\ \{l(e)^+\} & \textit{if } l(e)\in\mathsf{Obs}\wedge e\notin\mathrm{dom}(\mathcal{M}) \\ \{\tau,\tau\surd\} & \textit{if } l(e)=\tau\end{cases}.$$

Proof: First we show for any $\mathrm{decl} : \mathsf{Var} \to \mathsf{EXP}_{\mathsf{se}}$:

$$\begin{array}{l}\forall n \in \mathbb{N} : \forall B \in \mathsf{EXP}_{\mathsf{se}} : \forall \mathcal{E} : (\mathcal{E} = [\![B]\!]_{\mathcal{F}^n_{\mathrm{decl}}(\perp)} \wedge e \in \mathrm{init}(\mathcal{E})) \Rightarrow \exists G', \gamma : \\ \quad (B \xrightarrow[e]{\gamma}{}'_{\mathrm{decl}} G' \wedge (l(e) = \tau \Rightarrow \gamma \in \{\tau, \tau\surd\}) \wedge (l(e) \in \mathsf{Obs} \Rightarrow \gamma = l(e)^+))\end{array} \tag{7.4}$$

This is done by induction on n combined with the structure of B where the lexicographical order is used. We only present case $B = x$ in detail: $e \in \mathrm{init}([\![x]\!]_{\mathcal{F}^n_{\mathrm{decl}}(\perp)})$ implies that $n > 0$. Therefore, $[\![x]\!]_{\mathcal{F}^n_{\mathrm{decl}}(\perp)} = \mathcal{F}^n_{\mathrm{decl}}(\perp)(x) = [\![\mathrm{decl}(x)]\!]_{\mathcal{F}^{n-1}_{\mathrm{decl}}(\perp)}$. The rest follows by induction, since n is reduced. In the case of the refinement operator, Lemma 7.49 is applied. Thus (7.4) is established.

An immediate consequence of (7.4) is

$$\begin{array}{l}\forall B \in \mathsf{EXP}_{\mathsf{se}} : \forall \mathcal{E} : (\mathcal{E} = [\![\langle \mathrm{decl}, B\rangle]\!] \wedge e \in \mathrm{init}(\mathcal{E})) \Rightarrow \exists G', \gamma : \\ \quad (B \xrightarrow[e]{\gamma}{}'_{\mathrm{decl}} G' \wedge (l(e) = \tau \Rightarrow \gamma \in \{\tau, \tau\surd\}) \wedge (l(e) \in \mathsf{Obs} \Rightarrow \gamma = l(e)^+))\end{array} \tag{7.5}$$

The main statement follows now by structural induction on G, where (7.5) is used. In the case of the refinement operator, Lemma 7.49 is applied. □

Now we are ready to obtain the second bisimulation result, which establishes Theorem 7.26.

Lemma 7.51 *Let* $\langle \mathrm{decl}, B\rangle \in \mathsf{PA}_{\mathsf{se}}$*, then the transition system* $(\mathsf{EXP}^{\mathsf{O}'}_{\mathsf{se}}, \mathcal{L}_{\mathsf{se}}, \longrightarrow''_{\mathrm{decl}}, B)$ *is bisimilar to* $(\mathbf{SEBES}_{\mathcal{M}}, \mathcal{L}_{\mathsf{se}}, \hookrightarrow^c, [\![(\mathrm{decl}, B)]\!]')$*, where* $\longrightarrow''$ *is defined as in Lemma 7.41.*

Proof: Define $\mathcal{R} = \{(G, [\![(\mathrm{decl}, G)]\!]') \mid G \in \mathsf{EXP}^{\mathsf{O}'}_{\mathsf{se}}\}$. Then $(B, [\![(\mathrm{decl}, B)]\!]') \in \mathcal{R}$ by definition.
Suppose $G_1 \in \mathsf{EXP}^{\mathsf{O}'}_{\mathsf{se}}$ and $G_1 \xrightarrow{\gamma}{}'' G_2$. Then $G_1 \xrightarrow[e]{\gamma}{}' G_2$ for some e. Hence, by Lemma 7.48 we get $[\![(\mathrm{decl}, G_1)]\!]' \overset{\gamma}{\hookrightarrow}{}^c [\![(\mathrm{decl}, G_2)]\!]'$, as required.
Suppose $G_1 \in \mathsf{EXP}^{\mathsf{O}'}_{\mathsf{se}}$ and $[\![(\mathrm{decl}, G_1)]\!]' \overset{\gamma}{\hookrightarrow}{}^c \mathscr{E}_2$. Then there is $e \in \mathrm{init}([\![(\mathrm{decl}, G_1)]\!]')$ such that $\mathscr{E}_2 = [\![(\mathrm{decl}, G_1)]\!]'_{\rangle e\langle}$ and $[\![(\mathrm{decl}, G_1)]\!]' \overset{\gamma}{\hookrightarrow}{}^c [\![(\mathrm{decl}, G_1)]\!]'_{\rangle e\langle}$. From Lemma 7.50 we get the existence of $G_2 \in \mathsf{EXP}^{\mathsf{O}'}_{\mathsf{se}}$ and γ' such that $G_1 \xrightarrow[e]{\gamma'}{}' G_2$. Moreover, $[\![(\mathrm{decl}, G_1)]\!]'_{\rangle e\langle} = [\![(\mathrm{decl}, G_2)]\!]'$ and $\gamma = \gamma'$ by Lemma 7.48, which concludes the proof. □

Final conclusions.

The only proof left is that of Theorem 7.23. Before we do this, we introduce the following lemma, which shows a correspondence between $\hookrightarrow$ and $\hookrightarrow^c$.

Lemma 7.52 *For all* $a \in \mathcal{A}ct$*,* $\mathcal{E}_1 \in \mathbf{SEBES}$ *and* $(\mathcal{E}_2, \mathcal{M}_2) \in \mathbf{SEBES}_{\mathcal{M}}$ *we have*

$$(\mathcal{M}_2 = \perp \wedge \mathcal{E}_1 \overset{a(\surd)}{\hookrightarrow} \mathcal{E}_2) \Leftrightarrow \begin{cases} (\mathcal{E}_1, \perp) \overset{a^+}{\hookrightarrow}{}^c \overset{a^-_1(\surd)}{\hookrightarrow}{}^c (\mathcal{E}_2, \mathcal{M}_2) & \text{if } a \in \mathsf{Obs} \\ (\mathcal{E}_1, \perp) \overset{\tau(\surd)}{\hookrightarrow}{}^c (\mathcal{E}_2, \mathcal{M}_2) & \text{if } a = \tau \end{cases}$$

Proof: It can be easily checked, since $\mathcal{E}_{\langle e\rangle[e]} = \mathcal{E}_{[e]}$. □

Now we are ready to verify Theorem 7.23.

Proof of Theorem 7.23: From Lemma 7.41, Lemma 7.51 and from the transitivity of bisimilarity it follows that $(\mathsf{EXP}^{\mathsf{O}}_{\mathsf{se}}, \mathcal{L}_{\mathsf{se}}, \longrightarrow^c_{\mathrm{decl}}, B)$ and $(\mathbf{SEBES}_{\mathcal{M}}, \mathcal{L}_{\mathsf{se}}, \hookrightarrow^c, [\![(\mathrm{decl}, B)]\!]')$ are bisimilar (Theorem 7.26). Let $\widetilde{\mathcal{R}}$ be such a corresponding bisimulation.
Define $\mathcal{R}$ by $\mathcal{R} = \{(C, \mathcal{E}) \in \mathsf{EXP}^{\mathsf{O}}_{\mathsf{se}} \times \mathbf{SEBES} \mid (C, (\mathcal{E}, \perp)) \in \widetilde{\mathcal{R}}\}$. Then $(B, [\![\langle \mathrm{decl}, B\rangle]\!]) \in \mathcal{R}$ by definition.
Suppose $(C_1, \mathcal{E}_1) \in \mathcal{R}$ and $C_1 \overset{a(\surd)}{\Rightarrow} C_2$. We assume that $a \in \mathsf{Obs}$ (the case when $a = \tau$ follows analogously). By definition $C_1 \overset{a^+}{\longrightarrow}^c \overset{a_1^-(\surd)}{\longrightarrow}^c C_2$. Then there is $(\mathcal{E}_2, \mathcal{M}_2)$ such that $(\mathcal{E}_1, \perp) \overset{a^+}{\hookrightarrow}^c \overset{a_1^-(\surd)}{\hookrightarrow}^c (\mathcal{E}_2, \mathcal{M}_2) \wedge (C_2, (\mathcal{E}_2, \mathcal{M}_2)) \in \widetilde{\mathcal{R}}$. Thus, $\mathcal{E}_1 \overset{a(\surd)}{\hookrightarrow} \mathcal{E}_2 \wedge (C_2, (\mathcal{E}_2, \perp)) \in \widetilde{\mathcal{R}}$ by Lemma 7.52.
Suppose $(C_1, \mathcal{E}_1) \in \mathcal{R}$ and $\mathcal{E}_1 \overset{a(\surd)}{\hookrightarrow} \mathcal{E}_2$. We assume that $a \in \mathsf{Obs}$ (the case when $a = \tau$ follows analogously). From Lemma 7.52 we obtain that $(\mathcal{E}_1, \perp) \overset{a^+}{\hookrightarrow}^c \overset{a_1^-(\surd)}{\hookrightarrow}^c (\mathcal{E}_2, \perp)$. Then there is C_2 such that $C_1 \overset{a^+}{\longrightarrow}^c \overset{a_1^-(\surd)}{\longrightarrow}^c C_2 \wedge (C_2, (\mathcal{E}_2, \perp)) \in \tilde{\mathcal{R}}$. Hence, $C_1 \overset{a(\surd)}{\Rightarrow} C_2$ by definition. □

7.8.2 Proof of the Congruence Results

Proof of Theorem 7.28: Let $\mathcal{R}$ be a bisimulation such that $((\mathcal{E}, \perp), (\mathcal{E}', \perp)) \in \mathcal{R}$ and let $\mathcal{R}_a$ be bisimulations such that $((\theta(a), \perp), (\theta'(a), \perp)) \in \mathcal{R}_a$. Then define

$$\begin{aligned}\mathcal{R}_{Ref} = \{ \; & (\widetilde{Ref}^{se}_{A,M_A}(\mathcal{E}, \theta, \vec{\theta}), \widetilde{Ref}^{se}_{A,M_A}(\mathcal{E}', \theta, \vec{\theta}')) \mid (\mathcal{E}, \mathcal{E}') \in \mathcal{R} \wedge \\ & \forall a \in A : \forall i \in \mathbb{N}^+ : (\vec{\theta}(a)[i] \text{ is defined} \Leftrightarrow \vec{\theta}'(a)[i] \text{ is defined}) \wedge \\ & (\vec{\theta}(a)[i] \text{ is defined} \Rightarrow (\vec{\theta}(a)[i], \vec{\theta}'(a)[i]) \in \mathcal{R}_a)\}\end{aligned}$$

It is clear that $(Ref^{se}_A(\mathcal{E}, \theta), \perp) = \widetilde{Ref}^{se}_{A,M_A}((\mathcal{E}, \perp), \theta, \perp)$, and therefore we obtain the fact that $((Ref^{se}_A(\mathcal{E}, \theta), \perp), (Ref^{se}_A(\mathcal{E}, \theta), \perp)) \in \mathcal{R}_{Ref}$ as required.

The verification that Rel_{Ref} is a bisimulation is a straightforward consequence of Lemma 7.46 and Lemma 7.47 and is omitted here.

The proof of the other operators is straightforward. □

Proof of Theorem 7.29: The idea of this proof is the same as in Lemma 6.11:

Define $A \subseteq \mathcal{A}ct$ to be the set of all action-names occurring in $\mathcal{E}$ or in $\mathcal{E}'$, i.e. $A = \{l(e) | e \in E\} \cup \{l'(e') | e' \in E'\}$. Let $\mu : \{1, 2\} \times A \times \mathbb{N} \to \mathcal{A}ct \backslash A$ be an injective function. Such a function exists. We define for all $a \in A$ a sebes $\mathcal{E}_a$, which corresponds to the process algebra term $X = \mu(1, a, 0); \mu(2, a, 0) \oplus X[f]$ where $f(\mu(i, a, n)) = \mu(i, a, n + 1)$ [2].

[2] Here we use a relabeling operator as defined in Section 5.2. An isomorphic event structure can also be derived by the refinement operator.

In the following definition, sequences $\star_2^n\star_1\star_i\bullet$ are considered to be right bracketed and therefore elements of $\mathcal{U}$.

$$\begin{aligned}\mathcal{E}_a \quad = (\quad & \{\star_2^n\star_1\star_i\bullet \mid n\in\mathbb{N}\wedge i\in\{1,2\}\},\\ & \emptyset,\\ & \{(\{\star_2^n\star_1\star_1\bullet \mid n\in\mathbb{N}\backslash\{j\}\}\cup\{\star_2^j\star_1\star_i\bullet\},\star_2^j\star_1\star_i\bullet) \mid j\in\mathbb{N}\wedge i\in\{1,2\}\},\\ & \{(\{\star_2^n\star_1\star_1\bullet\},\star_2^n\star_1\star_2\bullet) \mid n\in\mathbb{N}\},\\ & \{\{\star_2^n\star_1\star_2\bullet \mid n\in\mathbb{N}\}\},\\ & \{(\star_2^n\star_1\star_i\bullet,\mu(i,a,n)) \mid n\in\mathbb{N}\wedge i\in\{1,2\}\})\end{aligned}$$

Define $\theta' : A\to\mathbf{SEBES}$ by $\theta'(a)=\mathcal{E}_a$. Furthermore, define $\mathcal{E}^{(a,n)}$ by

$$\mathcal{E}^{(a,n)}=(\{\star_2^n\star_1\star_2\bullet\},\emptyset,\{(\{\star_2^n\star_1\star_2\bullet\},\star_2^n\star_1\star_2\bullet)\},\emptyset,\{\{\star_2^n\star_1\star_2\bullet\}\},\{(\star_2^n\star_1\star_2\bullet,\mu(2,a,n))\})$$

Let $\mathcal{R}_b$ be a strong bisimulation such that $(Ref_A^{se}(\mathcal{E},\theta'),Ref_A^{se}(\mathcal{E}',\theta'))\in\mathcal{R}_b$. Without loss of generality, R_b contains only elements which can be derived from $Ref_A^{se}(\mathcal{E},\theta'),Ref_A^{se}(\mathcal{E}',\theta')$. Furthermore, let $\kappa:\mathcal{U}\to\mathbb{N}$ be an isomorphism. We define the relation $\mathcal{R}_{ST}$ by

$$\begin{aligned}\mathcal{R}_{ST}=\{\quad & ((\tilde{\mathcal{E}},\tilde{\mathcal{M}}),(\tilde{\mathcal{E}}',\tilde{\mathcal{M}}') \mid \exists\tilde{\vartheta},\tilde{\vartheta}':\exists f:\mathrm{dom}(\tilde{\mathcal{M}})\to\mathrm{dom}(\tilde{\mathcal{M}}'):\\ & f \text{ is a labeling preserving isomorphisms}\wedge\\ & (\forall e\in\mathrm{dom}(\tilde{\mathcal{M}}):\tilde{\mathcal{M}}(e)=\tilde{\mathcal{M}}'(f(e)))\wedge\\ & \left(\forall e\in\tilde{E}:\tilde{\vartheta}(e)=\begin{cases}\mathcal{E}_{\tilde{l}(e)} & \text{if } e\notin\mathrm{dom}(\tilde{\mathcal{M}})\\ \mathcal{E}^{(\tilde{l}(e),\kappa(f(e)))} & \text{if } e\in\mathrm{dom}(\tilde{\mathcal{M}})\end{cases}\right)\wedge\\ & \left(\forall e'\in\tilde{E}':\tilde{\vartheta}'(e')=\begin{cases}\mathcal{E}_{\tilde{l}'(e')} & \text{if } e'\notin\mathrm{dom}(\tilde{\mathcal{M}}')\\ \mathcal{E}^{(\tilde{l}'(e'),\kappa(e'))} & \text{if } e'\in\mathrm{dom}(\tilde{\mathcal{M}}')\end{cases}\right)\wedge\\ & (\underline{Ref}_A^{se}(\tilde{\mathcal{E}},\tilde{\vartheta}),\underline{Ref}_A^{se}(\tilde{\mathcal{E}}',\tilde{\vartheta}'))\in\mathcal{R}_b\ \}\end{aligned}$$

Obviously, $((\mathcal{E},\bot),(\mathcal{E}',\bot))\in\mathcal{R}_{ST}$.

The verification that Rel_{Ref} is a bisimulation is straightforward, where Lemma 7.46 and Lemma 7.47 are used. □

7.8.3 Proof of Theorem 7.34

It is only necessary to check the correctness of (7.1): From Lemma 7.33 we get that the transition systems obtained from $\langle\mathrm{decl},H_i\rangle$ and $\langle\mathrm{decl},\mathrm{decl}'(x_i)\{(H_j/x_j)^{j\in\{0,..,n\}}\}\rangle$ are bisimilar for any $i\le n$. Therefore, let R_i be corresponding bisimulations. Without loss of generality, let every R_i be reflexive. In the following, we write $(H,H')\in R_i$ instead of $(\langle\mathrm{decl},H\rangle,\langle\mathrm{decl},H'\rangle)\in R_i$. Define

$$R=\{(\langle\mathrm{decl},H\rangle,\langle\mathrm{decl}',x_q\rangle) \mid q\le n\wedge\exists i,f:\mathbb{N}\to\{1,...,n\}:(H,H_q)\in R_{f(1)}\circ\cdots\circ R_{f(i)}\}$$

Now we show that R is a bisimulation. Therefore, let $(\langle\mathrm{decl},H\rangle,\langle\mathrm{decl}',x_q\rangle)\in R$ such that $(H,H_q)\in R'$ where $R'=R_{f(1)}\circ\cdots\circ R_{f(i)}$. Then

$$(H,\mathrm{decl}'(x_q)\{(H_j/x_j)^{j\in\{0,..,n\}}\})\in R'\circ R_q, \tag{7.6}$$

since $(H_q,\mathrm{decl}'(x_q)\{(H_j/x_j)^{j\in\{0,..,n\}}\})\in R_q$.

$x_q \xrightarrow{\gamma}{}^{z}_{\mathrm{decl}'} \tilde{H}'$: Then there is $m \leq n$ such that $\tilde{H}' = x_m$ since $(\mathrm{decl}', \{x_0, ..., x_n\}) \in \mathsf{SeqG}$. Thus $\mathrm{decl}'(x_q)\{(H_j/x_j)^{j\in\{0,..,n\}}\} \xrightarrow{\gamma}{}^{z}_{\mathrm{decl}} H_m$. Since $R' \circ R_q$ is a bisimulation, we obtain from (7.6) the existence of $\tilde{H}$ such that $H \xrightarrow{\gamma}{}^{z}_{\mathrm{decl}} \tilde{H}$ and $(\tilde{H}, H_m) \in R' \circ R_q$. Thus $(\langle \mathrm{decl}, \tilde{H}\rangle, \langle \mathrm{decl}', \tilde{H}'\rangle) \in R$.

$H \xrightarrow{\gamma}{}^{z}_{\mathrm{decl}} \tilde{H}$: Since $R' \circ R_q$ is a bisimulation we obtain from (7.6) the existence of $\tilde{H}'$ such that $\mathrm{decl}'(x_q)\{(H_j/x_j)^{j\in\{0,..,n\}}\} \xrightarrow{\gamma}{}^{z}_{\mathrm{decl}} \tilde{H}'$ and $(\tilde{H}, \tilde{H}') \in R' \circ R_q$. Thus $x_q \xrightarrow{\gamma}{}^{z}_{\mathrm{decl}'} x_m \wedge \tilde{H}' = H_m$ for some $m \leq n$, since decl' is sequential guarded with respect to $\{x_0, ..., x_n\}$. Hence, $(\langle \mathrm{decl}, \tilde{H}\rangle, \langle \mathrm{decl}', x_m\rangle) \in R$.

Furthermore, $(\langle \mathrm{decl}, H_0\rangle, \langle \mathrm{decl}', x_0\rangle) \in R$ follows from the reflexivity of R_0, which establish Theorem 7.34.

7.8.4 Proof of Theorem 7.39

The verification of the completeness is similar to [44], which uses the technique from [137, 139]. The verification of our completeness result differs a little bit from [44], since we do not split the internal action, and termination is determined by the final action. More precisely, the above conditions lead to rules R_6 and R_6', where we have a transition $(C' \xrightarrow{a_1^-(\surd)} C'')$ in which the left process (C') is no subterm of the original process ($C[\phi, \varphi]_{M_A}$). Consequently, structural induction can not be used to verify completeness.

For this reason, we introduce a weight function on $\mathsf{PA}^{\mathsf{Gu}}_{\mathsf{se}}$, which is used for induction. Weight function Λ counts an upper bound of the possible numbers of actions that can start when no action finishes. This is done respectively for every action to guarantee the well-definedness of the weight function for the refinement operators. Λ also counts the numbers of the reachable variables ($\Lambda(\mho)$). Function Λ^c counts the reachable variables together with the upper bound of the possible number of actions that can start when no action ends, i.e. Λ^c is the sum over Λ.

Definition 7.53 *Define* $\Lambda : \mathsf{PA}^{\mathsf{Gu}}_{\mathsf{se}} \to ((\mathsf{Obs} \cup \{\mho\}) \to^{fin} \mathbb{N})$ *as follows, where we do not mention* decl *explicitly*

$$
\begin{array}{lcl}
\Lambda(H)(c) & = & 0 \quad \textit{if } H \in \{\mathbf{0}, \tau, b^-, b_q^-, b_q^-\surd; H', \tau\surd; H'\} \\
\Lambda(b)(c) & = & \begin{cases} 1 & \textit{if } c = b \\ 0 & \textit{otherwise} \end{cases} \\
\Lambda(b^+; H')(c) & = & \begin{cases} 1 + \Lambda(H')(c) & \textit{if } c = b \\ \Lambda(H')(c) & \textit{otherwise} \end{cases} \\
\Lambda(H)(c) & = & \Lambda(H')(c) \quad \textit{if } H \in \{H'; H'', H'\backslash\backslash_M A\} \\
\Lambda(H)(c) & = & \Lambda(H_1)(c) + \Lambda(H_1)(c) \quad \textit{if } H \in \{H_1 + H_2, H_1 \dot{+} H_2, H_1 \dot{+}^- H_2, \\
 & & \qquad H_1 \oplus_M H_2, H_1 \oplus_M^- H_2, H_1 \| _{A,M} H_2, \\
 & & \qquad H_1 \|_{A,M}^- H_2, H_1 |_{A,M} H_2\}
\end{array}
$$

$$\begin{array}{l}\Lambda(H[(a \to H_a)^{a\in A}, (a \to \vec{H}_a)^{a\in\tilde{A}}]_{M_A}\rangle)(c) = \\ \qquad \Lambda(H)(c) + (\sum_{a\in A} \Lambda(H)(a)\cdot\Lambda(H_a)(c)) + \sum_{a\in\tilde{A}} \sum_{i\le|\vec{H}_a|} \Lambda(\vec{H}_a[i])(c) \\ \Lambda(H[(a \to H_a)^{a\in A}, (a \to H'_a)^{a\in A}, (a \to \vec{H}_a)^{a\in\tilde{A}}]_{M_A} H'\rangle)(c) = \\ \qquad \Lambda(H)(c) + (\sum_{a\in A} \Lambda(H)(a)\cdot\Lambda(H_a)(c) + \Lambda(H')(a)\cdot\Lambda(H'_a)(c)) + \\ \qquad \Lambda(H')(c) + \sum_{a\in\tilde{A}} \sum_{i\le|\vec{H}_a|} \Lambda(\vec{H}_a[i])(c) \\ \Lambda(x)(c) \;=\; \begin{cases} 1 + \Lambda(decl(x))(c) & \textit{if } c = \mho \\ \Lambda(decl(x))(c) & \textit{otherwise} \end{cases}\end{array}$$

Furthermore, define $\Lambda^c : \mathsf{PA}^{\mathsf{Gu}}_{\mathsf{se}} \to \mathbb{N}$ *by* $\Lambda^c(H) = \sum_{c\in\mathsf{Obs}\cup\{\mho\}} \Lambda(H)(c)$.

Lemma 7.54 *Function* Λ *is well defined and consequently function* Λ^c *is also well defined.*

Proof: We show that

$$\forall(\langle \mathrm{decl}, H\rangle, V) \in \mathsf{VarSp} : \mathcal{G}((\langle \mathrm{decl}, H\rangle, V), \tilde{V}) \Rightarrow \Lambda(H) \text{ is well defined.}$$

This is done by induction on $|V\backslash\tilde{V}|$ combined with the structure of H where the lexicographic order is used.

The rest is an immediate consequence of Lemma 7.37. □

The nice property that the start of an action reduces the weight holds:

Lemma 7.55 *Suppose* $\langle \mathrm{decl}, H\rangle \in \mathsf{PA}^{\mathsf{Gu}}_{\mathsf{se}}$, $a \in \mathsf{Obs}$ *and* $H \stackrel{a^+}{\longrightarrow}{}^z{}_{\mathrm{decl}}\ H'$. *Then*

$$\Lambda(H)(a) > \Lambda(H')(a) \wedge \forall c \in (\mathsf{Obs}\cup\{\mho\}) : \Lambda(H)(c) \ge \Lambda(H')(c)$$

and consequently $\Lambda^c(H) > \Lambda^c(H')$.

Proof: This can be verified by induction on the depth of inference of $H \stackrel{a^+}{\longrightarrow}{}^z\ H'$. □

Guardedness implies finitely branching:

Lemma 7.56 *Suppose* $\langle \mathrm{decl}, H\rangle \in \mathsf{PA}^{\mathsf{Gu}}_{\mathsf{se}}$ *then* H *is finitely branching with respect to* $\longrightarrow^z$.

Proof: It follows by induction on $\Lambda^c(H)$ combined with the structure of H where the lexicographic order is used. In the case of rules R_6 and R'_6, we make use of Lemma 7.55. □

It can be derived from our axioms that a process is equivalent to the choice of its branches. This is illustrated by the following lemma, where $\sum_{H \stackrel{\gamma}{\longrightarrow}{}^z{}_{\mathrm{decl}} H'} \gamma; H'$ is really an expression, i.e. it has a finite choice, by Lemma 7.56.

Lemma 7.57 *Suppose* $\langle \mathrm{decl}, H\rangle \in \mathsf{PA}^{\mathsf{Gu}}_{\mathsf{se}}$, *then*

$$\vdash_{\mathrm{decl}} H = \sum_{H \stackrel{\gamma}{\longrightarrow}{}^z{}_{\mathrm{decl}} H'} \gamma; H'.$$

Proof: It follows by induction on $\Lambda^c(H)$ combined with the structure of H where the lexicographic order is used. In the case of the refinement operator, we make use of Lemma 7.55. □

As a consequence of the previous lemma, we can derive from our axioms that every guarded and finite state process is equivalent to a sequentially guarded process.

Corollary 7.58 *Suppose* $\langle \mathrm{decl}, H\rangle \in \mathsf{PA}^{\mathsf{Gu}}_{\mathsf{se}}$ *is finite state, then there is* $(\mathrm{decl}', \{x_0, ..., x_n\}) \in \mathsf{SeqG}$ *such that* $\vdash \langle \mathrm{decl}, H_0\rangle = \langle \mathrm{decl}', x_0\rangle$.

Proof: There is only a finite number of different expressions reachable from H by the transition rules, since $\langle \mathrm{decl}, H\rangle$ is finite state. Let $\{H_0, ..., H_n\}$ be the set of expressions reachable from H with $H_0 = H$.

Define $\mathrm{decl}'(x_i) = \sum_{H_i \xrightarrow{\gamma}_{\mathrm{decl}} H_j} \gamma; x_j$, which is well defined since $\langle \mathrm{decl}, H\rangle$ is finitely branching by Lemma 7.56. It is easily seen that $(\mathrm{decl}', \{x_0, ..., x_n\}) \in \mathsf{SeqG}$. Furthermore, $\vdash_{\mathrm{decl}} H_i = \mathrm{decl}'(x_i)\{(H_j/x_j)^{j\in\{0,..,n\}}\}$ by Lemma 7.57. Thus $\vdash \langle \mathrm{decl}, H_0\rangle = \langle \mathrm{decl}', x_0\rangle$ follows by rule (7.1). □

Our derivation system derives that two processes are equivalent whenever they are sequentially guarded and ST-equivalent.

Lemma 7.59 *Suppose* $(\mathrm{decl}, \{y_0, ..., y_m\}), (\mathrm{decl}', \{x_0, ..., x_n\}) \in \mathsf{SeqG}$ *such that* $\langle \mathrm{decl}, y_0\rangle$ *is ST-equivalent to* $\langle \mathrm{decl}', x_0\rangle$. *Then there is* $(\mathrm{decl}'', V_z) \in \mathsf{SeqG}$ *such that* $|V_z| < |\mathbb{N}|$ *and* $\vdash \langle \mathrm{decl}, y_0\rangle = \langle \mathrm{decl}'', z_{00}\rangle$ *and* $\vdash \langle \mathrm{decl}', x_0\rangle = \langle \mathrm{decl}'', z_{00}\rangle$ *for some* $z_{00} \in V_z$.

Proof: Let $R \subseteq \{y_0, ..., y_m\} \times \{x_0, ..., x_n\}$ be a bisimulation such that $(y_0, x_0) \in R$. Define $V_z = \{z_{ij} \mid i \leq n \wedge j \leq m \wedge (x_i, x_j) \in R\}$. Let

$$\mathrm{decl}''(z_{ij}) = \sum_{y_i \xrightarrow{\gamma}_{\mathrm{decl}} y_k,\ x_j \xrightarrow{\gamma}_{\mathrm{decl}'} x_l,\ (y_k,x_l)\in R} \gamma; z_{kl}.$$

Then $(\mathrm{decl}'', V_z) \in \mathsf{SeqG}$. Furthermore, $\vdash_{\mathrm{decl}} y_i = \mathrm{decl}''(z_{ij})\{(\mathrm{decl}(y_k)/z_{kl})^{k\in\{0,..,m\},l\in\{0,..,n\}}\}$ for any $z_{ij} \in V_z$, since R is a bisimulation. Thus $\vdash \langle \mathrm{decl}, y_0\rangle = \langle \mathrm{decl}'', z_{00}\rangle$ by rule (7.1). With symmetrical arguments we obtain $\vdash \langle \mathrm{decl}', x_0\rangle = \langle \mathrm{decl}'', z_{00}\rangle$. □

Now, we are ready to show the completeness result for guarded and finite state processes.

Proof of Theorem 7.39: It is an immediate consequence of Corollary 7.58, Lemma 7.59 and Theorem 7.34. □

Chapter 8

Conclusion

In this thesis, we motivated the approach that considers a choice operator as end-based triggered, in particular in the context of action refinement. We established the end-based choice operator by investigating a process algebra, which contains an end-based choice and an action refinement operator, and by developing a denotational, an operational and an axiomatical semantics for this process algebra. We showed that these semantics are consistent. More precisely, the operational and the denotational semantics are bisimilar, and the axiomatical semantics is sound and complete (for guarded and finite state processes) with respect to the bisimulation equivalence obtained from the operational (denotational) semantics.

We had to investigate a new technique (approximation closedness) to restrict event structures that are based on the bundle technique in order to obtain a complete partial order. Furthermore, we used a new technique to show the bisimilarity between the operational and the denotational semantics. This new technique can handle unguarded recursion.

We also investigated new equivalences in the extended bundle event structures setting. These new equivalences are congruences for the action refinement operator that considers the conflict relation in extended bundle event structures to be end-based triggered. The valid relations between the trace equivalence, the bisimulation equivalence and these new equivalences are summarized in Figure 8.1 (if two equivalences are connected via a line, then the lower one identifies more elements than the upper one).

We pointed out that extended bundle event structures are not appropriate to model the end-based view, since the intuitive congruence equivalences fail to be the coarsest for the end-based action

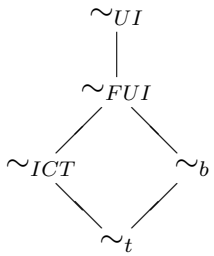


Figure 8.1: Relations Between the Equivalences

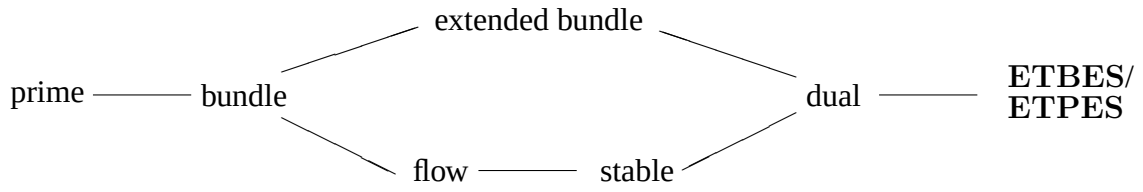


Figure 8.2: Hierarchy of Event Structures

refinement operator with respect to trace (respectively bisimulation) equivalence. Therefore, we investigated new event structures, namely extended termination bundle and extended termination precursor event structures, which have more general disabling relations. These event structures were examined in the context of a process algebra that contains a disrupt operator. The expressive power with respect to the set of event traces, describable by the classes of event structures, was examined. The hierarchy of this expressive power is depicted in Figure 8.2, where prime event structures can describe less set of event traces than the other event structures. The ICT-equivalence is the coarsest equivalence with respect to trace equivalence, and the FUI-equivalence is the coarsest equivalence with respect to bisimulation equivalence for the end-based action refinement operator in the extended termination event structures setting.

We introduced a choice operator where one side triggers the choice by ending actions and the other side triggers the choice by starting actions. Furthermore, we argued that it is useful to have this kind of choice as well as a start-based and an end-based choice in a single setting.

In the context of a process algebra with a start-based choice, we investigated a new technique of defining an operational semantics. This technique can handle action refinement and disruption in a reasonable way. Moreover, it is not necessary to introduce new syntactic terms in order to give the operational semantics.

Bibliography

[1] Samson Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors. *Handbook of Logic in Computer Science*, volume 2. Oxford University Press, 1992.

[2] Samson Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors. *Handbook of Logic in Computer Science*, volume 4. Oxford University Press, 1995.

[3] Samson Abramsky and Achim Jung. Domain theory. In Samson Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Clarendon Press, 1994.

[4] L. Aceto and M. Hennessy. Termination, deadlock, and divergence. *Journal of the ACM*, 39:147–187, 1992.

[5] L. Aceto and M. Hennessy. Towards action-refinement in process algebras. *Information and Computation*, 103:204–269, 1993.

[6] L. Aceto and M. Hennessy. Adding action refinement to a finite process algebra. *Information and Computation*, 115:179–247, 1994.

[7] Luca Aceto. *Action refinement in process algebras*. Cambridge University Press, 1992.

[8] Luca Aceto. On "Axiomatising Finite Concurrent Processes". *SIAM Journal on Computing*, 23:852–863, 1994.

[9] Luca Aceto, Wan Fokkink, and Chris Verhoef. Structural operational semantics. In Bergstra et al. [27], pages 197–292.

[10] Luca Aceto and David Murphy. Timing and causality in process algebra. *Acta Informatica*, 33:317–350, 1996.

[11] Marco Ajmone Marsan, Andrea Bianco, Luigi Ciminiera, Riccardo Sisto, and Adriano Valenzano. A LOTOS extension for the performance analysis of distributed systems. *IEEE/ACM Transactions on Networking*, 2:151–165, 1994.

[12] Marco Ajmone Marsan, Gianni Conte, and Gianfranco Balbo. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, 2:93–122, 1984.

[13] Egidio Astesiano and Gianna Reggio. Formalism and method. In M. Bidoit and M. Dauchet, editors, *TAPSOFT '97: Theory and Practice of Software Development*, volume 1214 of *LNCS*, pages 93–114. Springer-Verlag, 1997.

[14] J. C. M. Baeten and J. A. Bergstra. Mode transfer in process algebra. Report CSR 00-01, Vakgroep Informatica, Technische Universiteit Eindhoven, 2000.

[15] J. C. M. Baeten, J. A. Bergstra, and J. W. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae*, 9:127–168, 1986.

[16] J. C. M. Baeten and J. W. Klop, editors. *CONCUR '90*, volume 458 of *LNCS*. Springer-Verlag, 1990.

[17] J. C. M. Baeten and C. A. Middelburg. Process algebra with timing: Real time and discrete time. In Bergstra et al. [27], pages 627–684.

[18] J. C. M. Baeten and C. Verhoef. Concrete process algebra. In Abramsky et al. [2], pages 149–268.

[19] Christel Baier and Mila Majster-Cederbaum. How to interpret and establish consistency results for semantics of concurrent programming languages. *Fundamenta Informaticae*, 29:225–256, 1997.

[20] Christel Baier and Mila E. Majster-Cederbaum. The connection between an event structure semantics and an operational semantics for TCSP. *Acta Informatica*, 31:81–104, 1994.

[21] Christel Baier and Mila E. Majster-Cederbaum. Denotational semantics in the cpo and metric approach. *Theoretical Computer Science*, 135:171–220, 1994.

[22] H. P. Barendregt. Lambda calculi with types. In Abramsky et al. [1], pages 117–309.

[23] Marek A. Bednarczyk. Hereditary history preserving bisimulation or what is the power of the future perfect in program logics. Technical report, Institute of Computer Science, Polish Academy of Science, 1991.

[24] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, L. Petrucci, Ch. Schnoebelen, and P. McKenzie. *System and Software Verification*. Springer, 2001.

[25] J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60:109–137, 1984.

[26] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:77–121, 1985.

[27] J. A. Bergstra, A. Ponse, and S. A. Smolka, editors. *Handbook of Process Algebra*. North-Holland, 2001.

[28] Jan A. Bergstra, Wan Fokkink, and Alban Ponse. Process algebra with recursive operations. In Bergstra et al. [27], pages 333–389.

[29] Marco Bernardo and Roberto Gorrieri. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202:1–54, 1998.

[30] E. Best, editor. *CONCUR '93*, volume 715 of *LNCS*. Springer-Verlag, 1993.

[31] Eike Best, Raymond Devillers, and Javier Esparza. General refinement and recursion operators for the perti box calculus. In P. Enjalbert, A. Finkel, and K. W. Wagner, editors, *STACS 93*, volume 665 of *LNCS*, pages 130–140. Springer-Verlag, 1993.

[32] Tommaso Bolognesi and Ed Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14:25–59, 1987.

[33] G. Boudol and I. Castellani. On the semantics of concurrency: Partial orders and transition systems. In H. Ehrig, R. Kowalski, G. Levi, and U. Montanari, editors, *TAPSOFT '87 (Volume 1)*, volume 249 of *LNCS*, pages 123–137. Springer-Verlag, 1987.

[34] Gérard Boudol. Atomic actions. *Bulletin of the European Association for Theoretical Computer Science*, 38:136–144, 1989.

[35] Gérard Boudol. Flow event structures and flow nets. In I. Guessarian, editor, *Semantics of Systems of Concurrent Processes*, volume 469 of *LNCS*, pages 62–95. Springer-Verlag, 1990.

[36] Gérard Boudol and Ilaria Castellani. Permutation of transitions: an event structure semantics for CCS and SCCS. In de Bakker et al. [66], pages 411–427.

[37] Gérard Boudol and Ilaria Castellani. Flow models of distributed computations: event structures and nets. Report 1482, INRIA, 1991.

[38] Gérard Boudol and Ilaria Castellani. Flow models of distributed computations: Three equivalent semantics for CCS. *Information and Computation*, 114:247–314, 1994.

[39] Howard Bowman and John Derrick. Extending LOTOS with time: A true concurrency perspective. In M. Bertran and T. Rus, editors, *Transformation - Based Reactive Systems Development*, volume 1231 of *LNCS*, pages 383–399. Springer-Verlag, 1997.

[40] Howard Bowman and Joost-Pieter Katoen. A true concurrency semantics for ET-LOTOS. In *Applications of Concurrency to System Design*, pages 228–239. IEEE Computer Society Press, 1998.

[41] W. Brauer, W. Reisig, and G. Rozenberg, editors. *Petri Nets: Applications and Relationship to Other Models of Concurrency, Advances in Petri Nets 1986, Part II*, volume 255 of *LNCS*. Springer-Verlag, 1987.

[42] Wilfried Brauer, Robert Gold, and Walter Vogler. A survey of behaviour and equivalence preserving refinements of Petri nets. In G. Rozenberg, editor, *Advances in Petri Nets*, volume 483 of *LNCS*, pages 1–46. Springer-Verlag, 1991.

[43] Mario Bravetti and Marco Bernardo. Compositional asymmetric cooperations for process algebras with probabilities, priorities, and time. In Flavio Corradini and Paola Inverardi, editors, *MTCS 2000*, volume 39 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2000.

[44] Mario Bravetti and Robert Gorrieri. Deciding and axiomatizing weak ST bisimulation for a process algebra with recursion and action refinement. *ACMTCL: ACM Transactions on Computational Logic*, 3, 2002.

[45] Ed Brinksma, Joost-Peter Katoen, Rom Langerak, and Diego Latella. A stochastic causality-based process algebra. *The Computer Journal*, 38(7):552–565, 1995.

[46] Ed Brinksma, Joost-Peter Katoen, Rom Langerak, and Diego Latella. Partial order models for quantitative extensions of LOTOS. *Computer Networks and ISDN Systems*, 30:925–950, 1998.

[47] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.

[48] Manfred Broy and Ernst-Rüdiger Olderog. Trace-oriented models of concurrency. In Bergstra et al. [27], pages 101–196.

[49] Nadia Busi, Rob van Glabbeek, and Roberto Gorrieri. Axiomatising ST bisimulation equivalence. In E.-R. Olderog, editor, *Proceedings IFIP Working Conference on Programming Concepts, Methods and Calculi*, pages 169–188. Elsevier Science, 1994.

[50] Juanito Camilleri and Glynn Winskel. CCS with priority choice. *Information and Computation*, 116:26–37, 1995.

[51] Edmund M. Clarke, Jeanette M. Wing, et al. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28:626–643, 1996.

[52] Edmund M., Jr. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[53] Rance Cleaveland, Gerald Lüttgen, and V. Natarajan. A process algebra with distributed priorities. *Theoretical Computer Science*, 195:227–258, 1998.

[54] Rance Cleaveland, Gerald Lüttgen, and V. Natarajan. Priority in process algebra. In Bergstra et al. [27], pages 711–765.

[55] Rance Cleaveland, Scott A. Smolka, et al. Strategic directions in concurrency research. *ACM Computing Surveys*, 28:607–625, 1996.

[56] W. R. Cleaveland, editor. *CONCUR '92*, volume 630 of *LNCS*. Springer-Verlag, 1992.

[57] Flavio Corradini. Absolute versus relative time in process algebras. *Information and Computation*, 156:122–172, 2000.

[58] Flavio Corradini, Rocco De Nicola, and Anna Labella. Graded modalities and resource bisimulation. In C. Pandu Rangan, V. Raman, and R. Ramanujam, editors, *FSTTCS'99*, volume 1738 of *LNCS*, pages 381–393. Springer-Verlag, 1999.

[59] Flavio Corradini, Rocco De Nicola, and Anna Labella. Models of nondeterministic regular expressions. *Journal of Computer and System Sciences*, 59:412–449, 1999.

[60] Ingo Czaja, Rob van Glabbeek, and Ursula Goltz. Interleaving semantics and action refinement with atomic choice. In G. Rozenberg, editor, *Advances in Petri Nets*, volume 609 of *LNCS*, pages 89–107. Springer-Verlag, 1992.

[61] Ph. Darondeau and P. Degano. Causal trees. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Roccha, editors, *Automata, Languages and Programming*, volume 372 of *LNCS*, pages 234–248. Springer-Verlag, 1989.

[62] Philippe Darondeau and Pierpaolo Degano. About semantic action refinement. *Fundamenta Informaticae*, 14:221–234, 1991.

[63] Philippe Darondeau and Pierpaolo Degano. Refinement of actions in event structures and causal trees. *Theoretical Computer Science*, 118:21–48, 1993.

[64] J. W. de Bakker and E. P. de Vink. Bisimulation semantics for concurrency with atomicity and action refinement. *Fundamenta Informaticae*, 20:3–34, 1994.

[65] J. W. de Bakker and J. I. Zucker. Processes and the denotational semantics of concurrency. *Information and Control*, 54:70–120, 1982.

[66] J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors. *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *LNCS*. Springer-Verlag, 1989.

[67] P. Degano, R. De Nicola, and U. Montanari. Observational equivalences for concurrency models. In M. Wirsing, editor, *Formal Description of Programming Concepts – III, Proceedings of the* 3^{th} *IFIP WG 2.2 working conference*, Ebberup 1986, pages 105–129. North-Holland, 1987.

[68] Pierpaolo Degano, Rocco De Nicola, and Ugo Montanari. On the consistency of "truly concurrent" operational and denotational semantics (extended abstract). In *Proceedings of the 3rd Annual IEEE Symposium on Logic in Computer Science*, pages 133–141. IEEE Computer Society Press, 1988.

[69] Pierpaolo Degano and Roberto Gorrieri. Atomic refinement in process description languages. In Tarlecki [168], pages 121–130.

[70] Pierpaolo Degano and Roberto Gorrieri. A causal operational semantics of action refinement. *Information and Computation*, 122:97–119, 1995.

[71] John Derrick, Eerke Boiten, Jim Woodcock, and Joakim von Wright, editors. *REFINE 2002*, volume 70 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2002.

[72] Raymond Devillers. Maximality preserving bisimulation. *Theoretical Computer Science*, 102:165–183, 1992.

[73] B. Diertens. New features in PSF I – interrupts, disrupts, and priorities. Report P9417, Programming Research Group - University of Amsterdam, 1994.

[74] J. Dugundji. *Topology*. Allyn and Bacon, Boston, Mass., 1966.

[75] A. Engels and Th. Cobben. Interrupt and disrupt in MSC: Possibilities and problems. In Y. Lahav, A. Wolisz, J. Fischer, and E. Holz, editors, *Proceedings fo the 1st Workshop of the SDL Forum Society on SDL and MSC*, number 104 in Informatikberichte. Humboldt-Universitt zu Berlin, 1998.

[76] Harald Fecher. Denotational semantics of untyped object-based programming languages. Master's thesis, Technische Universität Darmstadt, 1999.

[77] Harald Fecher. A real-time process algebra with open intervals and maximal progress. *Nordic Journal of Computing*, 8:346–365, 2001.

[78] Harald Fecher and Mila Majster-Cederbaum. Taking decisions late: End-based choice combined with action refinement. In Derrick et al. [71].

[79] Harald Fecher, Mila Majster-Cederbaum, and Jinzhao Wu. Action refinement for probabilistic processes with true concurrency models. In H. Hermanns and R. Segala, editors, *PAPM-PROBMIV 2002. Performance Modeling and Verification*, volume 2399 of *LNCS*, pages 77–94. Springer-Verlag, 2001.

[80] Harald Fecher, Mila Majster-Cederbaum, and Jinzhao Wu. Bundle event structures: A revised cpo approach. *Information Processing Letters*, 83:7–12, 2002.

[81] Harald Fecher, Mila Majster-Cederbaum, and Jinzhao Wu. Refinement of actions in a real-time process algebra with a true concurrency model. In Derrick et al. [71].

[82] Miguel Felder, Angelo Gargantini, and Angelo Morzenti. A theory of implementation and refinement in timed Petri nets. *Theoretical Computer Science*, 202:127–161, 1998.

[83] Robert W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Mathematical Aspects of Computer Science*, volume 19 of *Proceedings of Symposia in Applied Mathematics*, pages 19–32. American Mathematical Society, 1967.

[84] Wan Fokkink. *Introduction to Process Algebra*. Springer-Verlag, 2000.

[85] R. J. van Glabbeek. The refinement theorem for ST-bisimulation semantics. In M. Broy and C.B. Jones, editors, Proceedings IFIP TC2 Working Conference on *Programming Concepts and Methods*, Sea of Gallilee, Israel, April 1990, pages 27–52. North Holland, 1990.

[86] R. J. van Glabbeek. The linear time–branching time spectrum I. the semantics of concrete, sequential processes. In Bergstra et al. [27], pages 3–99.

[87] R. J. van Glabbeek and G. D. Plotkin. Configuration structures. In *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, pages 199–209. IEEE Computer Society Press, 1995.

[88] Rob J. van Glabbeek, Scott A. Smolka, and Bernhard Steffen. Reactive, generative and stratified models of probabilistic processes. *Information and Computation*, 121:59–80, 1995.

[89] Rob van Glabbeek. The linear time–branching time spectrum II: The semantics of sequential systems with silent moves (extended abstract). In Best [30], pages 66–81.

[90] Rob van Glabbeek and Ursula Goltz. Refinement of actions in causality based models. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems. Models, Formalisms, Correctness*, volume 430 of *LNCS*, pages 267–300. Springer-Verlag, 1990.

[91] Rob van Glabbeek and Ursula Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Informatica*, 37:229–327, 2001.

[92] Rob van Glabbeek and Frits Vaandrager. Petri net models for algebraic theories of concurrency. In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, editors, *PARLE, Parallel Architectures and Languages Europe (Volume II)*, volume 259 of *LNCS*, pages 224–242. Springer-Verlag, 1987.

[93] Ursula Goltz, Roberto Gorrieri, and Arend Rensink. Comparing syntactic and semantic action refinement. *Information and Computation*, 125:118–143, 1996.

[94] Roberto Gorrieri. A hierarchy of system descriptions via atomic linear refinement. *Fundamenta Informaticae*, 16:289–336, 1992.

[95] Roberto Gorrieri and Cosimo Laneve. The limit of split$_n$-bisimulations for CCS agents. In Tarlecki [168], pages 170–180.

[96] Roberto Gorrieri and Cosimo Laneve. Split and ST bisimulation semantics. *Information and Computation*, 118:272–288, 1995.

[97] Roberto Gorrieri, Sergio Marchetti, and Ugo Montanari. A^{2}CCS: Atomic actions for CCS. *Theoretical Computer Science*, 72:203–223, 1990.

[98] Roberto Gorrieri and Arend Rensink. Action refinement. In Bergstra et al. [27], pages 1047–1147.

[99] Roberto Gorrieri, Marco Roccetti, and Enrico Stancampiano. A theory of processes with durational actions. *Theoretical Computer Science*, 140:73–94, 1995.

[100] Michael R. Hansen and Chaochen Zhou. Duration calculus: Logical foundations. *Formal Aspects of Computing*, 9:283 – 330, 1997.

[101] Constance Heitmeyer, James Kirby, Jr., Bruce Labaw, Myla Archer, and Ramesh Bharadwaj. Using abstraction and model checking to detect safety violations in requirements specifications. *IEEE Transactions on Software Engineering*, 24(11):927–948, 1998.

[102] Matthew Hennessy. Concurrent testing of processes. *Acta Informatica*, 32:509–543, 1995.

[103] Matthew Hennessy and Tim Regan. A process algebra for timed systems. *Information and Computation*, 117:221–239, 1995.

[104] Holger Hermanns and Michael Rettelbach. Syntax, semantics, equivalences, and axioms for MTIPP. In U. Herzog and M. Rettelbach, editors, *Proceedings of the 2nd Int. Workshop on Process Algebra and Performance Modelling (PAPM'94)*, 1994.

[105] C. A. R. Hoare. An axiomatic basis of computer programming. *Communications of the ACM*, 12:576–580, 1969.

[106] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[107] C. A. R. Hoare. A model for communicating sequential processes. In R. M. McKeag and A. M. Macnaghten, editors, *On the Construction of Programs*. Cambridge University Press, 1980.

[108] C. A. R. Hoare. *Communications Sequential Processes*. International Series in Computer Science. Prentice Hall, 1985.

[109] C. A. R. Hoare. Theories of programming: Top-down and bottom-up and meeting in the middle. In J.M. Wing, J. Woodcock, and J. Davies, editors, *FM'99 – Formal Methods (Volume I)*, volume 1708 of *LNCS*, pages 1–27. Springer-Verlag, 1999.

[110] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[111] Michaela Huhn. Action refinement and property inheritance in systems of sequential agents. In U. Montanari and V. Sassone, editors, *CONCUR '96: Concurrency Theory*, volume 1119 of *LNCS*, pages 639–654. Springer-Verlag, 1996.

[112] Pankaj Jalote and Robert H. Campbell. Atomic actions in concurrent systems. In *Proceedings of the 5th International Conference on Distributed Computing Systems*, pages 184–191, Denver, Colorado, May 1985. IEEE Computer Society.

[113] Wil Janssen, Mannes Poel, and Job Zwiers. Action systems and action refinement in the development of parallel systems. In J. C. M. Baeten and J. F. Groote, editors, *CONCUR '91*, volume 527 of *LNCS*, pages 298–316. Springer-Verlag, 1991.

[114] Lalita Jategaonkar and Albert Meyer. Testing equivalence for Petri nets with action refinement. In Cleaveland [56], pages 17–31.

[115] Bengt Jonsson, Wang Yi, and Kim G. Larsen. Probabilistic extensions pf process algebras. In Bergstra et al. [27], pages 685–710.

[116] Joost-Pieter Katoen. *Quantitative and Qualitative Extension of Event Structures*. PhD thesis, Enschede: Centre for Telematics and Information Technology, P.O. Box 217 - 7500 AE Enschede - The Netherlands, 1996.

[117] Joost-Pieter Katoen, Christel Baier, and Diego Latella. Metric semantics for true concurrent real time. *Theoretical Computer Science*, 254:501–541, 2001.

[118] Joost-Pieter Katoen, Ed Brinksma, Diego Latella, and Rom Langerak. Stochastic simulation of event structures. In Ribaudo [160], pages 21–49.

[119] Joost-Pieter Katoen, Rom Langerak, and Diego Latella. Modelling systems by probabilistic process algebra: an event structures approach. In R. L. Tenney et al., editors, *Formal Description Techniques, VI*, pages 253–268. Elsevier, 1994.

[120] Joost-Pieter Katoen, Rom Langerak, Diego Latella, and Ed Brinksma. On specifying real-time systems in a causality-based setting. In B. Jonsson and J. Parrow, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 1135 of *LNCS*, pages 385–404. Springer-Verlag, 1996.

[121] Robert M. Keller. Formal verification of parallel programs. *Communications of the ACM*, 19:371–384, 1976.

[122] Maciej Koutny and Eike Best. Operational and denotational semantics for the box algebra. *Theoretical Computer Science*, 211:1–83, 1999.

[123] Dexter Kozen. Results on the propositional mu -calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[124] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16:872–923, 1994.

[125] Rom Langerak. *Transformations and Semantics for LOTOS*. PhD thesis, Department of Computer Science, University of Twente, 1992.

[126] Rom Langerak. Bundle event structures: A non-interleaving semantics for LOTOS. In M. Diaz and R. Groz, editors, *Formal Description Techniques, V*, pages 331–346. Elsevier, 1993.

[127] Rom Langerak, Ed Brinksma, and Joost-Pieter Katoen. Causal ambiguity and partial orders in event structures. In A. Mazurkiewicz and J. Winkowski, editors, *CONCUR '97: Concurrency Theory*, volume 1243 of *LNCS*, pages 317–331. Springer-Verlag, 1997.

[128] Luc Léonard and Guy Leduc. An introduction to ET-LOTOS for the description of time-sensitive systems. *Computer Networks and ISDN Systems*, 29:271–292, 1997.

[129] Gavin Lowe. Probabilistic and prioritized models of timed CSP. *Theoretical Computer Science*, 138:315–352, 1995.

[130] M. Majster-Cederbaum, F. Salger, and M. Sorea. A priori verification of reactive systems. In Tommaso Bolognesi and Diego Latella, editors, *Formal Methods for Distributed System Development (Proc. FORTE/PSTV 2000)*, pages 35–50. Kluwer Academic Publishers, 2000.

[131] Mila Majster-Cederbaum and Markus Roggenbach. Transition systems from event structures revisited. *Information Processing Letters*, 67:119–124, 1998.

[132] Mila Majster-Cederbaum and Frank Salger. Correctness by construction: Towards verification in hierarchical system development. In K. Havelund, J. Penix, and W. Visser, editors, *SPIN Model Checking and Software Verification*, volume 1885 of *LNCS*, pages 163–180. Springer-Verlag, 2000.

[133] Mila Majster-Cederbaum and Jinzhao Wu. Action refinement for true concurrent real-time. In *Proc. 7th IEEE int. Conf. on Engineering of Complex Computer Systems*, pages 58–68. IEEE Computer Sciety Press, 2001.

[134] Mila Majster-Cederbaum, Naijun Zhan, and Harald Fecher. Action refinement from a logical point of view. In L. Zuck, P. Attie, A. Cortesi, and S. Mukhopadhyay, editors, *VMCAI 2003*, volume 2575 of *LNCS*, pages 253–267. Springer-Verlag, 2003.

[135] F. Erich Marschner. Practical challenges for industrial formal verification tools. In O. Grumberg, editor, *Computer Aided Verification*, volume 1254 of *LNCS*, pages 1–2. Springer-Verlag, 1997.

[136] Robin Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[137] Robin Milner. A complete inference system for a class of regular behaviors. *Journal of Computer and System Sciences*, 28:439–466, 1984.

[138] Robin Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[139] Robin Milner. A complete axiomatisation for observational congruence of finite-state behaviors. *Information and Computation*, 81:227–247, 1989.

[140] Ugo Montanari and Marco Pistore. Minimal transition systems for history-preserving bisimulation. In R. Reischuk and M. Morvan, editors, *STACS 97*, volume 1200 of *LNCS*, pages 413–425. Springer-Verlag, 1997.

[141] David Murphy. Time and duration in noninterleaving concurrency. *Fundamenta Informaticae*, 19:403–416, 1993.

[142] Xavier Nicollin and Joseph Sifakis. An overview and synthesis on timed process algebras. In J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, volume 600 of *LNCS*, pages 526–548. Springer-Verlag, 1992.

[143] Xavier Nicollin and Joseph Sifakis. The algebra of timed processes, ATP: Theory and application. *Information and Computation*, 114:131–178, 1994.

[144] Mogens Nielsen, Uffe Engberg, and Kim S. Larsen. Fully abstract models for a process language with refinement. In de Bakker et al. [66], pages 523–548.

[145] Mogens Nielsen, Gordon Plotkin, and Glynn Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13:85–108, 1981.

[146] E.-R. Olderog, editor. *Programming Concepts, Methods and Calculi, Proceedings of the IFIP TC2/WG2.1/WG2.2/WG2.3 Working Conference on Programming Concepts, Methods and Calculi (PROCOMET '94)*, volume A–56 of *IFIP Transactions*, 1994.

[147] C.-H. L. Ong. Correspondence between operational and denotational semantics: the full abstraction problem for PCF. In Abramsky et al. [2], pages 269–356.

[148] Doron A. Peled. *Software Reliability Methods*. Springer, 2001.

[149] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[150] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, Aarhus, Denmark, 1981.

[151] A. Pnueli. System specification and refinement in temporal logic. In R. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 652 of *LNCS*, pages 1–38. Springer-Verlag, 1992.

[152] Lucia Pomello. Some Equivalence Notions for Concurrent Systems. An Overview. In G. Rozenberg, editor, *Advances in Petri Nets 1985*, volume 222 of *LNCS*, pages 381–400. Springer-Verlag, 1986.

[153] Vaughan Pratt. Modeling concurrency with partial orders. *International Journal of Parallel Programming*, 15:33–71, 1986.

[154] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114–125, 1959.

[155] W. Reisig. *Petri Nets: An Introduction*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1985.

[156] Arend Rensink. Methodological aspects of action refinement. In Olderog [146], pages 227–246.

[157] Arend Rensink. An event-based SOS for a language with refinement. In *Structures in Concurrency Theory*, Workshops in Computing, pages 294–309, 1995.

[158] Arend Rensink and Roberto Gorrieri. Vertical implementation. *Information and Computation*, 170:95–133, 2001.

[159] Arend Rensink and Heike Wehrheim. Process algebra with action dependencies. *Acta Informatica*, 38:155–234, 2001.

[160] M. Ribaudo, editor. *Proceedings of the Fourth Process Algebra and Performance Modelling Workshop (PAPM'96)*, 1996.

[161] John Rushby. Formal methods and their role in the certification of critical systems. Technical Report SRI-CSL-95-1, Computer Science Laboratory, SRI International, 1995. Also available as NASA Contractor Report 4673, August 1995, and to be issued as part of the *FAA Digital Systems Validation Handbook* (the guide for aircraft certification).

[162] Frank Salger. *Verification in the Hierarchical Development of Reactive Systems*. PhD thesis, Universität Mannheim, 2001.

[163] D. Sangiorgi and R. de Simone, editors. *CONCUR '98: Concurrency Theory*, volume 1466 of *LNCS*. Springer-Verlag, 1998.

[164] Steve Schneider. *Concurrent and Real-time Systems: The CSP Approach*. Wiley, 2000.

[165] Karen Seidel. Probabilistic communicating processes. *Theoretical Computer Science*, 152:219–249, 1995.

[166] Susan Stepney, David Cooper, and Jim Woodcock. More powerful data refinement in Z: pushing the state of the art in industrial refinement. In J.P. Bowen, A. Fett, and M.G. Hinchey, editors, *ZUM'98: The Z Formal Specification Notation*, volume 1493 of *LNCS*, pages 284–307. Springer-Verlag, 1998.

[167] Colin Stirling. Modal and temporal logics. In Abramsky et al. [1], pages 477–563.

[168] A. Tarlecki, editor. *Mathematical Foundations of Computer Science*, volume 520 of *LNCS*. Springer-Verlag, 1991.

[169] Wolfgang Thomas. Logic for computer science: The engineering challenge. In R. Wilhelm, editor, *Informatics. 10 Years Back. 10 Years Ahead*, volume 2000 of *LNCS*, pages 257–267. Springer-Verlag, 2001.

[170] Simone Veglioni and Rocco De Nicola. Possible worlds for process algebras. In Sangiorgi and de Simone [163], pages 179–193.

[171] C. Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. In B. Jonsson and J. Parrow, editors, *CONCUR '94: Concurrency Theory*, volume 836 of *LNCS*, pages 433–448. Springer-Verlag, 1994.

[172] Walter Vogler. Failures semantics based on interval semiwords is a congruence for refinement. *Distributed Computing*, 4:139–162, 1991.

[173] Walter Vogler. Bisimulation and action refinement. *Theoretical Computer Science*, 114:173–200, 1993.

[174] Walter Vogler. Timed testing of concurrent systems. *Information and Computation*, 121:149–171, 1995.

[175] Y. Wang. Real-time behaviour of asynchronous agents. In Baeten and Klop [16], pages 502–520.

[176] Heike Wehrheim. Parametric action refinement. In Olderog [146], pages 247–266.

[177] Glynn Winskel. Event structures. In Brauer et al. [41], pages 325–392.

[178] Glynn Winskel. An introduction to event structures. In de Bakker et al. [66], pages 364–397.

[179] Glynn Winskel and Mogens Nielsen. Models for concurrency. In Abramsky et al. [2], pages 1–148.

[180] Niklaus Wirth. Program development by stepwise refinement. *Communications of the ACM*, 14:221–227, 1971.

Index

Symbols,
$[\![\ ]\!]$, 39, 93, 127
$\vdash$, 142
$\sim_b$, 21, 60, 111
$\sim_c$, 67
$\sim_{FUI}$, 65, 113
$\sim_{ICT}$, 63, 111
$\sim_{ST}$, 138
$\sim_t$, 21, 60, 111
$\sim_{UI}$, 64
$\rightarrow$, 18
$\rightharpoonup$, 19
$\rightharpoondown$, 19
$\rightarrow^{fin}$, 19
$\cong$, 20
$\trianglelefteq$, 33, 90, 124
$\hat{\trianglelefteq}$, 96
0, 30, 81, 122, 139
1, 30
., 30
$+$, 30, 81, 122, 139
$\dot{+}$, 122, 129, 139
$\dot{+}^-$, 139
$\oplus$, 122, 129, 139
$\oplus^-$, 139
;, 30, 81, 122, 129, 139
$\lceil\!>$, 30, 81
$\|_A$, 30, 81, 122, 129, 139
$\|_A^-$, 139
$\lfloor_A$, 139
$\backslash A$, 30
$\backslash\backslash A$, 81, 122, 129, 139
$\widehat{\cdot}$, 37
$\widehat{+}$, 37, 90, 125
$\widehat{\dot{+}}$, 125
$\widehat{\oplus}$, 125
$\widehat{;}$, 37, 91, 125
$\widehat{\lceil\!>}$, 37, 91
$\widehat{\|_A}$, 37, 91, 126
$\widehat{\backslash A}$, 38
$\widehat{\backslash\backslash A}$, 91, 126
*, 17
$\longrightarrow_{\mathcal{O}^s}$, 43
$\longrightarrow^c$, 131
$\longrightarrow^s$, 42
$\longrightarrow^t$, 82
$\longrightarrow^z$, 139
$\twoheadrightarrow$, 136
$\hookrightarrow$, 36, 88, 136
$\hookrightarrow^c$, 137

ACP, 11
$\mathcal{A}ct$, 80
$\mathcal{A}ct_{\surd}$, 30
$\mathcal{A}ct^T$, 81
action, 10
  atomic, 10
  instantaneous, 10
  internal, 10, 30
  observable, 10, 30, 44
  set of, 30, 80
  termination, 30, 81
action refinement, 12, 29–30
  atomic, 29
  non-atomic, 29
  syntactical, 14
  vertical, 30
α-equivalence, 43
approximation
  closed, 23
  finite, monotone, 23
automaton
  finite, 9
axioms, 142

bijective, 19
bisimilar, 21
  FUI, 65, 113

ST, 138, 139
strong, 60, 111
UI, 64
bisimulation, 21
finite unique initial, *see* bisimulation, FUI
FUI, 65, 112
UI, 64
unique initial, *see* bisimulation, UI
bundle stability constraint, 32, 85, 93

cardinality, 17
causal ambiguity, 32
causal trees, 11
CBES, 35
cbes, *see* event structures, closed bundle
CCS, 11
chain, 22
choice
end-based, 57, 109, 121
end-start, 121
start-based, 121
coarsest, *see* congruence, coarsest
completeness, 146
configurations, 83
congruence, 20
coarsest, 20, 67, 112, 114, 138
FUI, 66, 113
ICT, 64, 112
ST, 138
UI, 65
consistency, 15, 43, 93, 136
continuous, 22
componentwise, 23
on events, 48
countable, 17
cpo, *see* partial order, complete
CSP, 11

decl, 30, 81, 122, 139
declaration, *see* decl
design errors, 9
dom, 19
domain, 19
dual language approach, 9
dynamic names, 127

$\mathcal{E}$, 32, 34, 86, 123
$\mathscr{E}$, 137
$\mathcal{E}$, 94
EBES, 32
ebes, *see* event structures, extended bundle
environment, 10
equivalence, 60–61
bisimulation, *see* bisimilar
FUI, *see* bisimilar, FUI
history preserving, 61
ICT, 63, 111
initial corresponding trace, *see* equivalence, ICT
pomset, 61
resource bisimulation, 61
ST, 61, 138, 139
step, 61
trace, 21, 60, 111
UI, *see* bisimilar, UI
ETBES, 86
eTbes, *see* event structures, extended termination bundle
ETPES, 94
eTpes, *see* event structures, extended termination precursor
event, 18
initial, 35, 87, 95, 123
set of, 32, 85, 123
termination, 35, 87, 123
trace, 89
initial, 63
set of, 89, 97
universe of, 18
event structures, 11
bundle, 31, 89
closed bundle, 34, 83
dual, 33, 83, 89
extended bundle, 32, 89
extended termination bundle, 86
extended termination precursor, 94
flow, 83, 89
prime, 83, 89
restriction of, 33, 90, 124
stable, 83, 89
start-end, 123
time bundle, 57
exit, 35

$\mathsf{EXP}_{\mathsf{se}}$, 122
$\mathsf{EXP}^{\mathsf{Ax}}_{\mathsf{se}}$, 139
$\mathsf{EXP}_{\mathsf{sr}}$, 30
$\mathsf{EXP}_{\mathsf{st}}$, 81

$\mathcal{F}$, 97
$\mathcal{F}_{\mathrm{decl}}$, 127
$\mathcal{F}_{\mathrm{decl}}$, 39, 93
F_E, 97
$\mathcal{F}_L$, 80
fa-approach, 79, 80
finite state, 146
finitely determined, 94
formal methods, 9
function, 18–19
  action-labeling, 32, 85, 123
  inverse, 19
  labeling preserving, 63
  partial, *see* partial function
  relabeling, 80

$\mathcal{G}$, 145
guarded, 145
  sequentially, 142

Id, 18
init, 35, 62, 87, 123, 136
$\widehat{\mathrm{init}}$, 95
injective, 19
interleaving, 10

$\mathcal{L}$, 41
$\mathcal{L}_{\mathsf{se}}$, 130
$\widehat{\mathrm{Lab}}$, 91
label, 20
Λ^c, 162
least
  element, 22
  fixpoint, 23
  upper bound, 22
literals
  negative, 138
logic, 9
LOTOS, 11

metric space, 22
model checking, 9, 12
monotonic, 20

$\mathbb{N}^+$, 17
natural numbers
  positive, 17

$\mathcal{O}^s$, 43
Obs, 30
Obs_a, 44
Obs_P, 44
ω-chain, *see* chain
ω-continuous, *see* continuous
order, *see* partial order
  componentwise, 23
  pointwise, 23

$\mathcal{P}$, 17
$\mathcal{P}_{count}$, 17
$\mathcal{P}_{fin}$, 17
$\mathsf{PA}_{\mathsf{se}}$, 122
$\mathsf{PA}^{\mathsf{Ax}}_{\mathsf{se}}$, 139
$\mathsf{PA}^{\mathsf{Gu}}_{\mathsf{se}}$, 145
$\mathsf{PA}_{\mathsf{sr}}$, 30
$\mathsf{PA}_{\mathsf{st}}$, 81
panth, 138
partial function, 19
  bijective, 19
  domain, 19
  injective, 19
  inverse, 19
  surjective, 19
partial order
  complete, 22, 35, 90, 97, 124
  ω-complete, *see* partial order, complete
performance analysis, 11
petri net, 11
pointers, 127
pomsets, 11
preserved
  relation, 20, 61
process, 30
  action prefix, 31
  choice, 31, 81
    end-based, 122
    end-start, 122
    start-based, 121
  disrupt, 31, 81
  hiding, 31
  inactive, 31, 81

parallel, 31, 81
refinement, 31, 122
relabeling, 81
restriction, 81
sequential, 31, 81
specified by, 145
terminate, 31
variable, 30, 81
process algebra, 9, 11
priority, 12
probability, 12
stochastic, 12
time, 12
projection, 19

race policy, 13
reactive system, 10
concurrent, 10
Ref^{e}_{A}, 59
$\overline{Ref^{e}_{A}}$, 69
Ref^{eT}_{A}, 109
$\overline{Ref^{eT}_{A}}$, 115
$\overline{Ref}^{s}_{A}$, 38
Ref^{se}_{A}, 126
relation
bundle, 32
causality, 85, 123
conflict, 85
asymmetric, 33
precursor, 94
start, 123
symmetric, 32
identity, 18, 34
transition, *see* transition, relation
witness, 86
end, 123
relative active number, 128
remainder, 36, 88, 95, 135

SEBES, 123
SEBES$_{\mathcal{M}}$, 136
sebes, *see* event structures, start-end
semantics
axiomatic, 11
denotational, 11, 40, 93, 127
operational, 11, 42, 81, 130
SeqG, 142
soundness, 142
specification, 9
descriptive/property-based, 9
imperative/operational-based, 9
stack technique, 127
state, 20
initial, 20
state explosion problem, 12
static names, 127
string, 17
empty, 17
substitution, 142
syntactic, 30
surjective, 19
system design
hierarchical, 12
top down, 29

T, 21
T^{ic}, 63, 111
τ, 30, *see* action, internal
TBES, 85
Tbes, *see* event structures, termination bundle
TCSP, 11
termination
action, *see* action, termination
event, *see* event, termination
predicate, 87, 95, 123
set, 85, 123
Tr^{e}, 89
trace, 21
initial event, 111
semantics, 10
transition
relation, 20, 36, 88, 137
rules, 11, 41, 81, 131, 139
complete, 138
system, 9–11, *20*
true concurrency, 11
TS, 20

$\mathcal{U}$, 18
upper closed, 94
Υ, 87, 123
$\hat{\Upsilon}$, 95

Var, 30

variable assignment, 38, 93, 127
VarSp, 145
verification, 9

$\mathcal{X}$, 24

Z, 14

Zusammenfassung

Der Auswahloperator ist ein wichtiges Element in der Beschreibung von aktionsbasierten, reaktiven Systemen. Wenn man den Ansatz der Atomarität von Aktion aufgibt, zum Beispiel durch Aktionsverfeinerung, muss man festlegen, wann die Auswahl getroffen wird. In der Regel wird die Auswahl beim Starten von Aktionen getroffen.

Diese Doktorarbeit beschäftigte sich mit dem alternativen Ansatz, der darin besteht, dass die Auswahl beim Beenden von Aktionen getroffen wird (end-basierte Auswahl). Ich habe diesen Ansatz motiviert, insbesondere im Kontext des hierarchischen Entwurfs von Systemen (realisiert durch Aktionsverfeinerung). Das Ziel dieser Arbeit war eine Prozess-Algebra zu entwickeln, die einen end-basierten Auswahloperator und einen Aktionsverfeinerungsoperator besitzt. Vor allem sollten konsistente Semantiken (denotationale, operationale und axiomatische) für diese Prozess-Algebra angegeben werden. Weiterhin sollte der Unterschied zwischen dem end-basierten und dem start-basierten Auswahloperator herausgearbeitet werden, insbesondere bezüglich der Äquivalenzbegriffe.

Beim Definieren einer denotationalen Semantik hatte man seither das Problem, dass die Ereignisstrukturen (engl: *event structures*), welche auf der Bundle-Technik basieren, keine cpo_s mit der üblichen Ordnung liefern. Deshalb wurde eine Technik zur Definition von Einschränkungen dieser Ereignisstrukturen entwickelt, so dass diese Einschränkungen cpo_s mit der üblichen Ordnung liefern. Weiterhin wurde auch eine neue Technik vorgestellt, um operational Semantiken, die den denotationalen Semantiken entsprechen, für Prozess-Algebren mit einem Aktionsverfeinerungsoperator zu entwickeln. Mit dieser Technik ist es nicht nötig, die Prozess-Algebren um weitere Ausdrücke zu erweitern, um die operationale Semantik zu definieren.

Ein neuer Aktionsverfeinerungsoperator wurde auf den *extended bundle event structures* definiert. Dieser Operator betrachtet die Konflikt-Relation der *extended bundle event structures* als end-basiert. Neue Äquivalenzen wurden eingeführt, da keine der Standard-Äquivalenzen von diesem Operator erhalten bleiben. Es wurde aufgezeigt, dass die *extended bundle event structures* kein passendes Modell für diese Art von Aktionsverfeinerungsoperator ist, da die intuitiven Kongruenzäquivalenzen nicht die kleinsten bezüglich der Bisimulations- und der Trace-Äquivalenz sind.

Aus diesem Grund wurden zwei neue Klassen von Ereignisstrukturen eingeführt. In diesen Strukturen können Mengen von *events* andere *events* deaktivieren. Es wurde gezeigt, dass diese Klassen von Ereignisstrukturen äquivalente Ansätze liefern. Eine dieser Ereignisstrukturen wurde als denotationale Semantik einer Prozess-Algebra, die einen Abbruchsoperator aber keinen Aktionsverfeinerungsoperator besitzt, benutzt. In dieser Prozess-Algebra findet Termination durch die zuletzt ausgeführte Aktion und nicht durch ein zusätszliches Terminationsevent statt. Weiterhin wurde gezeigt, dass auf dieser Ereignisstruktur die intuitiven

Kongruenzäquivalenzen für den Aktionsverfeinerungsoperator, der die Konfliktrelation als end-basiert betrachtet, tatsächlich die kleinsten Kongruenzen bezüglich der Bisimulations- und der Trace-Äquivalenz liefern.

Schließlich habe ich aufgezeigt, dass es sinnvoll ist, einen start-basierten Auswahloperator zu haben, wann immer man einen end-basierten Auswahloperator und einen Parallelloperator mit Aktionssynchronisierung hat. Zusätzlich wurde auch ein Auswahloperator motiviert und betrachtet, bei dem sich eine Seite end-basiert und die andere Seite start-basiert verhählt. Eine Prozess-Algebra mit diesen drei verschieden Auswahloperatoren und einem Aktionsverfeinerungsoperator wurde eingeführt. Es wurden konsistente denotationale, operationale und axiomatische Semantiken angegeben. Genauer gesagt sind die denotationale und die operationale Semantik bisimulationsäquivalent und die axiomatische Semantik ist korrekt und vollständig bezüglich der von der operationalen Semantik erhaltenen Bisimulations-Äquivalenz. Für die denotationale Semantik wurde eine Ereignisstruktur mit zwei Relationen für Konflikte betrachtet: eine für den start-basierten Konflikt die andere für den end-basierten Konflikt.

Danksagungen

Vor allem möchte ich Frau Prof. Dr. Mila Majster-Cederbaum für Ihre Betreuung danken. Sie ließ mir den nötigen Freiraum, stand allzeit für meine Fragen bereit und half mir beim Formulieren mathematischer Texte in englischer Sprache. Ich möchte mich auch herzlich bei Herrn Prof. Dr. Franz Stetter, meinem Zweitkorrektor, bedanken. Mein Dank gilt außerdem meinen Kollegen, unter anderem Dr. Jinzhao Wu, Jürgen Jaap, Dr. Naijun Zhan, Dr. Sven Helmer und insbesondere meinem Zimmerkollegen Dr. Frank Salger. Bedanken möchte ich mich auch bei Frau Jackowski für ihre hilfreiche Unterstützung bei den anfallenden Formalitäten. Zuletzt möchte ich meiner Frau Annette danken, ohne die ich das alles nicht geschafft hätte.