REIHE INFORMATIK
07/97

A Categorical View of Action Refinement in Models of Concurrency

Friederike Benjes and Mila Majster-Cederbaum
Universität Mannheim
Seminargebäude A5
D-68131 Mannheim

A Categorical View of Action Refinement in Models of Concurrency

Friederike Benjes, Mila Majster-Cederbaum

November 20, 1997

Abstract

We define a categorical characterization of refinement and show that refinement definitions for various models of concurrency can be captured be our view.

1 Introduction

Various efforts have been made in the past years to investigate suitable notions of refinement for the stepwise development of process systems (see for example [Ba81], [dBV92], [DGR92], [CGG91], [DD93], [DG91], [DG95], [GG89], [Re95], [Vo93]). Particular attention was given to action refinement, also called vertical refinement, where an abstract action is substituted by some more complex construct. Action refinement can be considered as a syntactic operator ([AH93], [NEL89]) as well as a semantic operator in a specific semantic model, see [GG90a],[DD93],[DGR93] for event structures, [BDE93],[Vo89],[JM92] for petrinets, [DG95] for causal trees, and [DD91] for synchronization trees. In [GGR94], [Be96] the connection between syntactic and semantic refinement for flow event structures is studied. For practical considerations more liberal notions of refinement have been investigated recenctly ([Hu96], [We93], [CR92]).

We are here interested in an abstract view of action refinement, i.e. we wish to state properties that are common to the various refinement operators. For this we follow the approach of [JNW94], [WN95] who defined categories for various semantic domains and gave an abstract characterization of bisimulation and some standard operations, like relabelling, restriction, parallel operator and sum.

A categorical characterization of an operation can be viewed as a unification of the various approaches and identifies those features that are essential for this operation. It may serve as a point of reference for defining such an operation in further models.

The paper is organized as follows:
In section 2 we present our categorical definition of refinement. In section 3 we show that the classical definitions of refinement on prime event structures and flow event structures fit into our framework. In section 4 we deal with action refinement in two interleaving models, i.e. synchronization trees and languages and show that the refinement operators in these models also satisfy our categorical characterization.

2 A Categorical Characterization of Refinement

We follow here an approach of [WN95] where categories are defined for various models of concurrency as e.g. event structures, petrinets and synchronization trees. Operations on these models originating from process languages are interpreted in terms of category theoretical concepts. E.g. it is shown in [WN95] that non deterministic choice corresponds to the coproduct in a fiber and that restriction can be viewed as cartesian lifting. In addition the relation between models is expressed by (co)reflections between the corresponding categories. In [JNW94] a categorical treatment of bisimulation is proposed where bisimulation in some model is viewed as a certain object in the corresponding category.

Along these lines we propose a categorical characterization of action refinement. We show that for selected examples of models that the corresponding refinement operators satisfy our definition.

If object A is a refinement of object B then there should be an epimorphism h from A to B, expressing the fact that A is more detailed. The "structure" of A should moreover be similar to that of B. We capture this by demanding that the set $\Gamma(h)$ of sections of h "covers" A, where $\Gamma(h)$ consists of all morphisms $h' : B \to A$ with $h \circ h' = id_B$. For the notions of category theory we refer to [ML71] and [BW90].

Definition 2.1 *Let* $\mathbf{C}$ *be a category,* A, B *be objects in* $\mathbf{C}$ *and* $w \in Hom_{\mathbf{C}}(A, B)$. *Define* $\Gamma(w)$, *the set of* sections *of* w *by*
$\Gamma(w) := \{w' \in Hom_{\mathbf{C}}(B, A) \mid w \circ w' = id_B\}$.

Definition 2.2 *Let* $\mathbf{C}$ *be a category,* A, B *be objects in* $\mathbf{C}$ *and* $w \in Hom_{\mathbf{C}}(A, B)$. (A, B, w) *is called a* categorical refinement *iff the set of sections* $\Gamma(w)$ *covers* A, *i.e.* $\Gamma(w) \neq \emptyset$ *and for all objects* X *in* $\mathbf{C}$: $\forall u_1 \neq u_2 \in Hom_{\mathbf{C}}(A, X) \exists w' \in \Gamma(w) : u_1 \circ w' \neq u_2 \circ w'$.

Remark 2.3 *If* (A, B, w) *is a categorical refinement then* w *is epimorphic.*

The following lemmata are used to transfer categorical refinements from one category to another:

Lemma 2.4 *Let* $\mathbf{D}$ *and* $\mathbf{C}$ *be categories and* $F : \mathbf{D} \to \mathbf{C}$ *a full and faithful functor. Let* A, B *be objects of* $\mathbf{D}$ *and* $w \in Hom_{\mathbf{D}}(A, B)$. *Then*
a) $\Gamma(Fw) = \{Fg \mid g \in \Gamma(w)\}$
b) If (FA, FB, Fw) *is a categorical refinement in* $\mathbf{C}$ *then* (A, B, w) *is a categorical refinement in* $\mathbf{D}$.

Proof:

a) Let $h \in \Gamma(Fw)$, i.e. $Fw \circ h = id_{FB}$. Since F is full there exists $g \in Hom_{\mathbf{D}}(B, A)$: $Fg = h$. Thus $Fw \circ Fg = F(w \circ g) = id_{FB} = F(id_B)$. Since F is faithful it follows that $w \circ g = id_B$ and therefore $g \in \Gamma(w)$.
Let now $g \in \Gamma(w)$, i.e. $w \circ g = id_B$. Then $F(w \circ g) = Fid_B$ and therefore $Fw \circ Fg = id_{FB}$. Thus $Fg \in \Gamma(Fw)$.

b) Let now X be an object of $\mathbf{D}$ and $u_1, u_2 \in Hom_{\mathbf{D}}(A, X)$ such that $u_1 \neq u_2$. Since F is faithful then $Fu_1 \neq Fu_2$. Thus there exists $h \in \Gamma(Fw)$ such that $Fu_1 \circ h \neq Fu_2 \circ h$.

Let $g \in \Gamma(w)$ such that $Fg = h$. Then $Fu_1 \circ Fg = F(u_1 \circ g) \neq Fu_2 \circ Fg = F(u_2 \circ g)$ and therefore $u_1 \circ g \neq u_2 \circ g$. □

Lemma 2.5 *Let* **C** *be a category and* (A, B, w) *a categorical refinement in* **C**. *Let* $h \in Hom_{\mathbf{C}}(A', A)$ *an ismorphism. Then* $(A', B, w \circ h)$ *also is a categorical refinement.*

3 Categorical Refinement on Event Structures

Prime event structures ([Wi87]) and flow event structures ([Bo90]) were introduced as true concurrency and branching time models of parallel processes. In contrast to [Bo90] we admit flow event structures where not all events are labelled, because otherwise the categorical characterization of refinement would not hold if we admit event structures containing self-conflicting events.

Remark 3.1 *Let* $\mathbf{Set}_*$ *be the category that has as objects all sets that do not contain* $*$ *and as morphisms all (total) functions* $h : A \cup \{*\} \to B \cup \{*\}$ *with* $h(*) = *$.
Each partial function h *from a set* A *to a set* B *with* $* \notin A \cup B$ *can be seen as an element in* $Hom_{\mathbf{Set}_*}(A, B)$. *We then write* $h : A \to^* B$. *For a set* $X \subseteq A$ *we put* $h(X) = \{h(x) \mid x \in X \ \& \ h(x) \neq *\}$.

Definition 3.2 $\mathcal{E} = (E, \prec, \#, l)$ *is a* partially-labelled flow event structure *if*

- E *is a set of events*
- $\prec \subseteq E \times E$ *is irreflexive (flow relation)*
- $\# \subseteq E \times E$ *symmetric (conflict relation)*
- $l : E \to^* L$ *labelling function*

A flow event structure is called *prime event structure* if l is total, $\#$ is irreflexive and $\prec$ is an irreflexive partial order such that the principle of finite causes ($\forall e \in E : \{e' \in E \mid e' \leq e\}$ is finite) and the principle of conflict heredity ($\forall e, e', e'' \in E : e \# e' \leq e'' \Rightarrow e \# e''$) are satisfied.

A subset $X \subseteq E$ is called *configuration* of $\mathcal{E}$ iff

(i) X conflict-free, i. e. $\forall d, e \in X : \neg(d \# e)$

(ii) $\leq_X := (\prec \cap (X \times X))^*$ (the reflexive and transitive closure of $\prec$ in X) is a partial order, i.e. $\prec$ is cycle-free on X.

(iii) $\forall e \in X$: $\{e' \in X \mid e' \leq_X e\}$ is finite (principle of finite causes)

(iv) $\forall e \in X \ \forall e' \in E \setminus X : e' \prec e \Rightarrow \exists e'' \in X : \ e' \# e'' \prec e$ (left-closed up to conflicts)

$Conf(\mathcal{E})$ is the set of all configurations of $\mathcal{E}$ (the finite ones and the infinite ones).
$\mathcal{R}(\mathcal{E}) := \bigcup_{X \in Conf(\mathcal{E})} X$ is the set of *reachable* events of $\mathcal{E}$.
A configuration X is called *complete* iff $\forall e \in E \setminus X : \exists e' \in X : e' \# e$.

We write $Conf_f(\mathcal{E})$ for the set of all finite configurations of $\mathcal{E}$ and $MConf_f(\mathcal{E})$ for the set of all maximal (w.r.t. $\subseteq$) and finite configurations of $\mathcal{E}$.
For $X \in Conf(\mathcal{E})$ and $e \in X$ write $\downarrow_X(e) := \{e' \in X \mid e' \leq_X e\}$.

For a prime event structure $\mathcal{E} = (E, \leq, \#, l)$ a subset $X \subseteq E$ is a configuration of $\mathcal{E}$ iff X is left-closed, i.e. $\forall e \in X : \downarrow e := \{e' \in E \mid e' \leq e\} \subseteq X$ and conflict-free, i.e. $\forall e, e'' \in X : \neg(e \# e')$.

Note that for prime event structures each maximal configuration is also complete, whereas for flow event structures there may exist maximal configurations that are not complete – such configurations can be seen as deadlock.

We call flow event structures that do not contain maximal and incomplete configurations *deadlock-free.*

A flow event structure $\mathcal{E}$ is called *empty* iff $\mathcal{R}(\mathcal{E}) = \emptyset$. Note that in contrast to prime event structures there exist empty flow event structures containing events (for example an event that is self-conflicting).

A flow event structure is called *well-labelled* iff exactly the reachable events are labelled, i.e. $l(e) = * \Leftrightarrow e \notin \mathcal{R}(\mathcal{E})$.

Each partially-labelled flow event structure $\mathcal{E} = (E, \prec, \#, l : E \to^* L)$ in which all reachable events are labelled can easily be transformed into a well-labelled event structure $\mathcal{W}(\mathcal{E}) := (E, \prec, \#, l' : E \to^* L)$ with $l'(e) := \begin{cases} l(e) & \text{if } e \in \mathcal{R}(\mathcal{E}) \\ * & \text{otherwise} \end{cases}$

Note that all prime event structures are well-labelled flow event structures because in prime event structures all events are reachable.

Definition 3.3 *Let $\mathcal{E}_1 = (E_1, \prec_1, \#_1, l_1), \mathcal{E}_2 = (E_2, \prec_2, \#_2, l_2)$ be flow event structures. $\mathcal{E}_1$ and $\mathcal{E}_2$ are* event-structure-isomorphic *($\mathcal{E}_1 \cong_e \mathcal{E}_2$) iff $L_1 = L_2$ and there exists a bijection $f : E_1 \to E_2$ such that for all $e, e' \in E_1$:*
$e \prec_1 e' \Leftrightarrow f(e) \prec_2 f(e')$, $e \#_1 e' \Leftrightarrow f(e) \#_2 f(e')$, and $l_1(e) = l_2(f(e))$.
$\mathcal{E}_1$ and $\mathcal{E}_2$ are domain isomorphic *($\mathcal{E}_1 \cong_d \mathcal{E}_2$) iff $L_1 = L_2$ and there exists a bijection $f : Conf(\mathcal{E}_1) \to Conf(\mathcal{E}_2)$ such that for all $X, X' \in Conf(\mathcal{E}_1)$:*
$X \subseteq X' \Leftrightarrow f(X) \subseteq f(X')$ and $l_1(X) = l_2(f(X))$.
(compare [GGR94] and [Bo90]).

[Bo90] showed that for each flow event structure $\mathcal{E}$ there exists a domain isomorphic prime event structure $\mathcal{P}(\mathcal{E}) := (E', <', \#', l')$ with
$E' := \{\downarrow_X(e) \mid e \in X \in Conf(\mathcal{E})\}$, $\downarrow_X(e) <' \downarrow_Y(f)$ iff $\downarrow_X(e) \subset \downarrow_Y(f)$,
$\downarrow_X(e) \#' \downarrow_Y(f)$ iff $\downarrow_X(e) \cup \downarrow_Y(f) \notin Conf(\mathcal{E})$ and $l'(\downarrow_X(e)) := l(e)$.
This definition can directly be transferred to partially-labelled flow event structures.

Note that $\cong_d$ and $\cong_e$ coincide on prime event structures. Thus for all prime event structures $\mathcal{E}$ holds: $\mathcal{P}(\mathcal{E}) \cong_e \mathcal{E}$.

Adapting the definition of [GGR94] we define action refinement for well-labelled flow event structures:

Definition 3.4 *Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be well-labelled flow event structures with $\mathcal{E}_2$ being non-empty. Define $\mathcal{E}_1[a \rightsquigarrow \mathcal{E}_2]_{\mathbf{F}} := \mathcal{W}(E, \prec, \#, l : E \to^* L)$ with*
$E = \{(e_1, e_2) \in E_1 \times (E_2 \cup \{*\}) \mid e_2 = * \Leftrightarrow l_1(e_1) \neq a\}$,
$(e_1, e_2) \prec (e_1', e_2') \Leftrightarrow e_1 \prec_1 e_1' \vee (e_1 = e_1' \;\&\; e_2 \prec_2 e_2')$,

$(e_1, e_2)\#(e_1', e_2') \Leftrightarrow e_1\#_1 e_1' \vee e_1 = e_1' \ \& \ e_2\#_2 e_2'$,

$L := (L_1 \setminus \{a\}) \cup L_2$ *and* $l(e_1, e_2) := \begin{cases} l_1(e_1) & \text{if } e_2 = * \\ l_2(e_2) & \text{otherwise} \end{cases}$

$(\mathcal{E}_i = (E_i, \prec_i, \#_i, l_i : E_i \to^* L_i))$.

Remark 3.5 *If $\mathcal{E}$ and $\mathcal{F}$ are well-labelled flow event structures so is $\mathcal{E}[a \rightsquigarrow \mathcal{F}]_{\mathbf{F}}$.*

If a prime event structures is refined in the same way as a flow event structure with an event structure containing conflicts the result may be no longer a prime event structure. Hence we define

$$\mathcal{E}_1[a \rightsquigarrow \mathcal{E}_2]_{\mathbf{E}} := \mathcal{P}(\mathcal{E}_1[a \rightsquigarrow \mathcal{E}_2]_{\mathbf{F}})$$

for prime event structures $\mathcal{E}_1$ and $\mathcal{E}_2$ with $\mathcal{E}_2$ being non-empty.

This definition builds the flow event structure refinement of two prime event structures and then transforms the result into a prime event structure. It yields isomorphic results to the one in [DGR93] (see [Be96]) and if no conflicting event structures are inserted also to the one in [GG90b].

[GG90a] and [Co95] gave a simple characteristic of refined event structures that we transferred to well-labelled flow event structures. This lemma will be used for the proof that refinement on flow event structures yields a categorical refinement.

Lemma 3.6 *Let $\mathcal{E}_1, \mathcal{E}_2$ be well-labelled flow event structures such that $\mathcal{E}_2$ is non-empty, $\mathcal{E}' = (E', \prec', \#', l') = \mathcal{E}_1[a \rightsquigarrow \mathcal{E}_2]_{\mathbf{F}}$ and $X \subseteq E'$.*
Then $X \in Conf(\mathcal{E}_1[a \rightsquigarrow \mathcal{E}_2]_{\mathbf{F}})$ if and only if

- $\pi_1(X) \in Conf(\mathcal{E}_1)$ *(with π_1 being the projection from $E_1 \times (E_2 \cup \{*\})$ to E_1.*
- $\forall e_1 \in E_1 : X(e_1) := \{e_2 \in E_2 \mid (e_1, e_2) \in X\} \in Conf(\mathcal{E}_2)$
- $\forall e_1 \in \pi_1(X) \setminus Max(\pi_1(X)) : X(e_1)$ *is a complete and finite configuration of $\mathcal{E}_2$ (with $Max(\pi_1(X))$ denoting the set of maximal (w.r.t. $\leq_{\pi_1(X)}$) elements in $\pi_1(X)$).*

Remark 3.7 *If $X \in Conf(\mathcal{E}_1[a \rightsquigarrow \mathcal{E}_2]_{\mathbf{F}})$ is complete so is $\pi_1(X)$.*

We now define a category for flow event structures and then show that refining a flow event structure yields a categorical refinement.

Definition 3.8 *The category* **F** *of flow event structures:*

Objects: *well-labelled flow event structures*

Morphisms: $(g, \lambda) \in Hom_{\mathbf{F}}(\mathcal{E}_1, \mathcal{E}_2) \Leftrightarrow$

a) $g : E_1 \to^* E_2$, $\lambda : L_1 \to^* L_2$ *with* $e_1 \notin \mathcal{R}(\mathcal{E}_1) \Rightarrow g(e_1) = *$

b) $\forall X \in Conf(\mathcal{E}_1)\ \forall e, e' \in X$ *holds:* $g(e) \prec_2 g(e') \Rightarrow e \prec_1 e'$

c) $\forall X \in Conf(\mathcal{E}_1) \exists Y \in Conf(\mathcal{E}_2) : g(X) \subseteq Y$, *and*
$\forall e, e' \in X : g(e) <_Y g(e') \Rightarrow e <_X e'$, *and*
if X is complete so is Y.

d) $l_2 \circ g = \lambda \circ l_1$.

Note that c) implies $g(e_1) \neq * \Rightarrow g(e_1) \in \mathcal{R}(\mathcal{E}_2)$. (One thus also could define $g : \mathcal{R}(\mathcal{E}_1) \rightarrow^* \mathcal{R}(\mathcal{E}_2)$ because all other values are necessarily $*$). The use of well-labelled instead of totally labelled flow event structure thus leads to a simple morphism definition only depending on the reachable events. The identity morphism of an event structure $\mathcal{E} = (E, \prec, \#, l : E \rightarrow^* L)$ will be denoted by $(\gamma_\mathcal{E}, id_L)$ with $\gamma_\mathcal{E} : E \rightarrow^* E$,

$$\gamma_\mathcal{E} : e \mapsto \begin{cases} e & \text{if } e \in \mathcal{R}(\mathcal{E}) \\ * & \text{otherwise} \end{cases} .$$

It coincides with (id_E, id_L) if this is a morphism, which is not necessarily the case.

Note that $\cong_e$ implies $\cong_\mathbf{F}$ (the categorical isomorphism on $\mathbf{F}$ – but not vice versa. But if labelling is not considered $\cong_\mathbf{F}$ is stronger than domain isomorphism.

Theorem 1 *For $i = 1, 2$ let $\mathcal{E}_i = (E_i, \prec_i, \#_i, l_i : E_i \rightarrow^* L_i)$ be well-labelled flow event structures and $\mathcal{E}_2$ finite, non-empty and deadlock-free, $l_2(\mathcal{R}(\mathcal{E}_2)) = L_2$, $a \in l_1(\mathcal{R}(\mathcal{E}_1))$ and $L_1 \cap L_2 = \emptyset$. Let $\mathcal{E} = (E, \prec, \#, l : E \rightarrow^* L) = \mathcal{E}_1[a \rightsquigarrow \mathcal{E}_2]_\mathbf{F}$. Let $g : E \rightarrow E_1$, $\lambda : L \rightarrow L_1$ be given by:*

$$g : (e_1, e_2) \mapsto \begin{cases} e_1 & \textit{if } (e_1, e_2) \in \mathcal{R}(\mathcal{E}_1[a \rightsquigarrow \mathcal{E}_2]_\mathbf{F}) \\ * & \textit{otherwise} \end{cases} , \lambda : b \mapsto \begin{cases} a & \textit{if } b \in L_2 \\ b & \textit{otherwise} \end{cases}$$

Then $(\mathcal{E}_1[a \rightsquigarrow \mathcal{E}_2]_\mathbf{F}, \mathcal{E}_1, (g, \lambda))$ is a categorical refinement in $\mathbf{F}$.

Proof: (using lemma 3.6)
Obviously $(g, \lambda) \in Hom_\mathbf{F}(\mathcal{E}', \mathcal{E})$ because $\forall X \in Conf(\mathcal{E}')$ holds $g(X) = \pi_1(X) \in Conf(\mathcal{E})$ (see lemma 3.6) and completeness of X implies completeness of $g(X)$.
We have to show that for an arbitrary $\mathcal{E}'$ and $\forall (g_1, \lambda_1), (g_2, \lambda_2) \in Hom_\mathbf{F}(\mathcal{E}, \mathcal{E}')$:
$(g_1, \lambda_1) \neq (g_2, \lambda_2) \Rightarrow \exists (g', \lambda') \in \Gamma(g, \lambda) : (g_1, \lambda_1) \circ (g', \lambda') \neq (g_2, \lambda_2) \circ (g', \lambda')$.
For each event $f \in \mathcal{R}(\mathcal{E}_2)$ define $(g_f, \lambda_f) \in \Gamma(g, \lambda)$:

$$g_f : e_1 \mapsto \begin{cases} (e_1, f) & \text{if } l_1(e_1) = a \\ (e_1, *) & \text{if } l_1(e_1) \neq a \ \&\ e_1 \in \mathcal{R}(\mathcal{E}_1) \\ * & \text{otherwise} \end{cases} , \lambda_f : b \mapsto \begin{cases} l_2(f) & \text{if } b = a \\ b & \text{otherwise} \end{cases}$$

$(g_f, \lambda_f) \in Hom_\mathbf{F}(\mathcal{E}_1, \mathcal{E})$ because for each configuration X_1 of $\mathcal{E}_1$ there exists a configuration X of $\mathcal{E}$ with $\pi_1(X) = X_1$ and if X_1 is complete X also can be chosen complete (this is due to the fact that $\mathcal{E}_2$ is finite and deadlock-free). Obviously $(g, \lambda) \circ (g_f, \lambda_f) = id_{\mathcal{E}_1}$ and therefore $(g_f, \lambda_f) \in \Gamma(g, \lambda)$.

If $\lambda_1 \neq \lambda_2$ then there exists $b \in L_\mathcal{E} : \lambda_1(b) \neq \lambda_2(b)$.
If $b \in L_2$ then there exists $(e_1, f) \in \mathcal{R}(\mathcal{E})$ with $l_2(f) = b$ and then $\lambda_1(\lambda_f(a)) = \lambda_1(b) \neq \lambda_2(b) = \lambda_2(\lambda_f(a))$. If $b \notin L_2$ then $b \in L_1$. Choose an arbitrary $f \in \mathcal{R}(\mathcal{E}_2)$, then $\lambda_1(\lambda_f(b)) = \lambda_1(b) \neq \lambda_2(b) = \lambda_2(\lambda_f(b))$. Thus in both situations $(g_1, \lambda_1) \circ (g_f, \lambda_f) \neq (g_2, \lambda_2) \circ (g_f, \lambda_f)$.

If $\lambda_1 = \lambda_2$ then $g_1 \neq g_2$. Then there exists $e \in \mathcal{R}(\mathcal{E})$ with $g_1(e) \neq g_2(e)$.
If exists $e_1 \in \mathcal{R}(\mathcal{E}_1) : e = (e_1, *)$ then for an arbitrary $f \in \mathcal{R}(\mathcal{E}_2)$ holds $g_1(g_f(e_1)) = g_1(e_1, *) \neq g_2(e_1, *) = g_2(g_f(e_1))$. If exists $e_1 \in \mathcal{R}(\mathcal{E}_1), f \in \mathcal{R}(\mathcal{E}_2)$ such that $e = (e_1, f)$ then $g_1(g_f(e_1)) = g_1(e_1, f) \neq g_2(e_1, f) = g_2(g_f(e_1))$. Thus in both situations $(g_1, \lambda_1) \circ (g_f, \lambda_f) \neq (g_2, \lambda_2) \circ (g_f, \lambda_f)$. □

If we used morphisms also depending on unreachable events this result only would hold for a small subclass of flow event structures, because then also the unreachable events of the refined event structure must be in the range of one of the sections.

For the category of prime event structures we have to relax the conditions for morphisms: Consider for example $\mathcal{E}_1$ and $\mathcal{E}_2$ in Figure 1. If morphisms have to satisfy the condition $l_2 \circ g = \lambda \circ l_1$ the event $\downarrow_X(e_2, f_1)$ with $X = \{(e_0, f_0), (e_1, *), (e_2, f_1)\}$ cannot be reached by a morphism in $\Gamma(g, \lambda)$ with (g, λ) chosen appropriately because $\lambda(a)$ must be x and $\lambda(a)$ must be y.

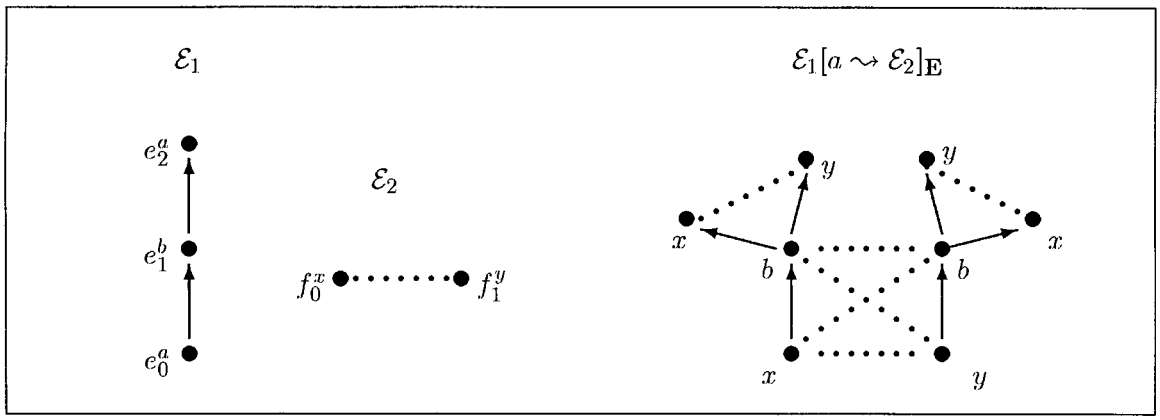

Figure 1: The refinement of an event structure – e_0^a denotes the event e_0 labelled with a while b denotes the label b – note that not all conflict-relations are drawn

To solve this problem in the category **E** also morphisms not respecting labelling are allowed:

Definition 3.9 *The category* **E** *of prime event structures:*

Objects: *prime event structures*

Morphisms: $(g, \lambda) \in Hom_{\mathbf{E}}(\mathcal{E}_1, \mathcal{E}_2) \Leftrightarrow$

a) $g : E_1 \to^* E_2$ *and* $\lambda : L_1 \to^* L_2$

b) $\forall e_1, e_1' \in E_1 : g(e_1) <_2 g(e_1') \Rightarrow e_1 <_1 e_1' \vee e_1 \#_1 e_1'$.

c) $\forall X \in Conf(\mathcal{E}_1) \exists Y \in Conf(\mathcal{E}_2) : g(X) \subseteq Y$

d) $\forall e \in E_1 : g(e) = * \Leftrightarrow \lambda(l_1(e)) = *$

Condition b) ensures that concurrent events cannot be mapped to causally depending events.

Theorem 2 *For* $i = 1, 2$ *let* $\mathcal{E}_i = (E_i, <_i, \#_i, l_i : E_i \to L_i)$ *be prime event structures and* $\mathcal{E}_2$ *finite and non-empty,* $l_2(E_2) = L_2$, $a \in l_1(E_1)$ *and* $L_1 \cap L_2 = \emptyset$. *Let* $\mathcal{E} = (E, <, \#, l : E \to L) = \mathcal{E}_1[a \rightsquigarrow \mathcal{E}_2]_{\mathbf{E}}$. *Let* $g : E \to E_1$, $\lambda : L \to L_1$ *be given by:*

$$g : \downarrow_X(e_1, e_2) \mapsto e_1,\ \lambda : b \mapsto \begin{cases} a & \text{if } b \in L_2 \\ b & \text{otherwise} \end{cases}$$

Then $(\mathcal{E}_1[a \rightsquigarrow \mathcal{E}_2]_{\mathbf{E}}, \mathcal{E}_1, (g, \lambda))$ *is a categorical refinement in* **E**.

Proof:
Obviously $(g, \lambda) \in Hom_{\mathbf{E}}(\mathcal{E}_1[a \rightsquigarrow \mathcal{E}_2], \mathcal{E}_1)$. We have to show that for an arbitrary prime

event structure $\mathcal{E}'$ and for all morphisms $(g_1,\lambda_1),(g_2,\lambda_2) \in Hom_{\mathbf{E}}(\mathcal{E},\mathcal{E}')$:
$(g_1,\lambda_1) \neq (g_2,\lambda_2) \Rightarrow \exists (g',\lambda') \in \Gamma(g,\lambda)$: $(g_1,\lambda_1) \circ (g',\lambda') \neq (g_2,\lambda_2) \circ (g',\lambda')$.
For a complete configuration $Y \in Conf(\mathcal{E}_2)$ and $f \in Y$ define $g_{Y_f} : E_1 \to E$ with

$$g_{Y_f} : e \mapsto \begin{cases} \downarrow_{X_Y}(e,f) & \text{if } l_1(e) = a \\ \downarrow_{X_Y}(e,*) & \text{otherwise} \end{cases} \quad \text{and}$$

and $\lambda_{Y_f} : L_1 \to L$ with $\lambda_{Y_f} : b \mapsto \begin{cases} l_2(f) & \text{if } b = a \\ b & \text{if } b \neq a \end{cases}$
with $X_Y := \{(e',f') \in E \mid e' \leq_1 e \ \&\ f' = * \vee f' \in Y\}$.
Then $(g_{Y_f},\lambda_{Y_f}) \in Hom_{\mathbf{E}}(\mathcal{E}_1,\mathcal{E}_1[a \rightsquigarrow \mathcal{E}_2]_{\mathbf{F}})$ and since $(g,\lambda) \circ (g_{Y_f},\lambda_{Y_f}) = id_{\mathcal{E}_1}$
then $(g_{Y_f},\lambda_{Y_f}) \in \Gamma(g,\lambda)$.

If $\lambda_1 \neq \lambda_2$ then there exists $b \in L : \lambda_1(b) \neq \lambda_2(b)$.
If $b \in L_2$ then there exists a maximal configuration $Y \in Conf(\mathcal{E}_2)$ and $f \in Y$ such that $l_2(f) = b$ and $\downarrow_{X_Y}(e,f) \in E$ and then $\lambda_1(\lambda_{Y_f}(a)) = \lambda_1(b) \neq \lambda_2(b) = \lambda_2(\lambda_{Y_f}(a))$. If $b \in L_1$ then choose an arbitrary $f \in E_2$, $Y \in Conf(\mathcal{E}_2)$ with Y complete and $f \in Y$ and then $\lambda_1(\lambda_{Y_f}(b)) = \lambda_1(b) \neq \lambda_2(b) = \lambda_2(\lambda_{Y_f}(b))$. Thus in both cases $(g_1,\lambda_1) \circ (g_{Y_f},\lambda_{Y_f}) \neq (g_2,\lambda_2) \circ (g_{Y_f},\lambda_{Y_f})$.

If $\lambda_1 = \lambda_2$ then $g_1 \neq g_2$ and then there exists $\downarrow_X(e_1,e_2) \in E$ with
$g_1(\downarrow_X(e_1,e_2)) \neq g_2(\downarrow_X(e_1,e_2))$ Let $X' \in Conf(\mathcal{E}')$ such that X' is complete and $X \subseteq X'$. Let f be an arbitrary event of $\mathcal{E}_2$. Let Y be a complete configuration of $\mathcal{E}_2$ with $f \in Y$.

$$\text{Define } g' : E_1 \to E,\ e_1' \mapsto \begin{cases} \downarrow_{X'}(e_1,e_2) & \text{if } e_1' = e_1 \\ \downarrow_{X'}(e_1',e_2') & \text{if } e_1' \neq e_1 \ \&\ e_1' \in \pi_1(X') \text{ for an} \\ & \text{arbitrary } e_2' \in E_2 \cup \{*\} \text{ with } (e_1',e_2') \in X' \\ \downarrow_{Z_{e_1'}}(e_1',f) & \text{if } e_1' \notin \pi_1(X') \end{cases} \quad \text{and}$$

$$\lambda' : b \mapsto \begin{cases} b & \text{if } b \neq a \\ l_2(f) & \text{if } b = a \end{cases} \quad \text{with } Z_{e_1'} := \{(e_1'',e_2'') \in X' \mid e_1'' \leq e_1'\}$$

$\cup \{(e_1'',e_2'') \in E \mid e_1'' \leq e_1' \ \&\ e_1'' \notin \pi_1(X') \ \&\ e_2'' \in Y\}$. Then $(g',\lambda') \in Hom_{\mathbf{E}}(\mathcal{E}_1,\mathcal{E})$ and $(g',\lambda') \in \Gamma(g,\lambda)$. Furthermore $g_1(g'(e_1)) = g_1(\downarrow_{X'}(e_1,e_2)) = g_1(\downarrow_X(e_1,e_2))$
$\neq g_2(\downarrow_X(e_1,e_2)) = g_2(\downarrow_{X'}(e_1,e_2)) = g_2(g'(e_1))$. Thus $(g_1,\lambda_1) \circ (g',\lambda') \neq (g_2,\lambda_2) \circ (g',\lambda')$. □

4 Categorical Refinement on Interleaving Models

We adopt a refinement operator on synchronization trees from [DD91] and define a suitable refinement operator on languages. Both constructions are shown to be categorical refinements.

4.1 Synchronization Trees

Synchronization trees were introduced by [Mi80] (see also [Mi89]).

Definition 4.1 *$\mathcal{T} = (S,i,L,Tran)$ is a* simple transition system *if S is a set of states, L set of labels, $Tran \subseteq S \times L \times S$ is a cycle-free transition relation such that*
$\forall s,s' \in S : (s,a,s') \in Tran \Rightarrow \nexists b \neq a : (s,b,s') \in Tran$,
$i \in S$ initial state with no predecessor.

A simple transition system $\mathcal{T} = (S, i, L, \mathrm{Tran})$ *is a* synchronization tree *iff the graph of* Tran *is a directed tree with root* i.

We write $s \rightarrow^a s'$ for $(s, a, s') \in \mathrm{Tran}$ and $s \rightarrow s'$ if $\exists a \in L : s \rightarrow^a s'$.
We write $s \rightarrow^* s'$ iff exist $s_1, ...s_n \in S : s = s_1 \rightarrow ... \rightarrow s_n = s'$ and $s \rightarrow^+ s'$ iff $s \rightarrow^* s' \ \& \ s \neq s'$. A transition system is called *empty* if $S = \{i\}$ and $\mathrm{Tran} = \emptyset$. It is called *finite* if S is finite.
$\surd_{\mathcal{T}} := \{s \in S \mid \nexists s' \in S, a \in S : (s, a, s') \in \mathrm{Tran}\}$. For a synchronization tree define $l_{\mathcal{S}} : S \rightarrow^* L$ with $l_{\mathcal{S}}(s) := \begin{cases} a & \text{if } \exists s' \in S : s' \rightarrow^a s \\ * & \text{if } s = i \end{cases}$

Definition 4.2 *The category* **S** *of synchronization trees:*

Objects: *synchronization trees*

Morphisms: $(\sigma, \lambda) \in Hom_{\mathbf{S}}(\mathcal{S}_1, \mathcal{S}_2)$ *iff*

a) $\sigma : S_1 \rightarrow^* S_2, \lambda : L_1 \rightarrow^* L_2$ *such that* $\sigma(i_1) = i_2$
b) $\forall s_1, s_1' \in S_1 : \sigma(s_1) \rightarrow^+ \sigma(s_1') \Rightarrow s_1' \not\rightarrow^+ s_1$
c) $\forall s_1, s_1' \in S_1 : (s_1 \rightarrow^+ s_1' \ \& \ \sigma(s_1) \neq * \ \& \ \sigma(s_1') \neq *) \Rightarrow \sigma(s_1) \rightarrow^* \sigma(s_1')$
d) $\forall s_1 \in S_1 \setminus \{i_1\} : (\sigma(s_1) = * \Leftrightarrow \lambda(l_{\mathcal{S}_1}(s_1)) = *)$.

Simple transition systems can be unfolded to a synchronization trees: For a simple transition system $\mathcal{T} = (S, i, L, \mathrm{Tran})$ define the synchronization tree $Unfold(\mathcal{T}) := (S_1, i_1, L, \mathrm{Tran}_1)$ with $S_1 = \{(s_1, ..., s_n) \mid i = s_1 \rightarrow s_2 ... \rightarrow s_n\}$, $i_1 = (i)$ and
$\mathrm{Tran}_1 := \{((s_1, ..., s_n), b, (s_1, ..., s_{n+1})) \in S_1 \times L \times S_1 \mid s_n \rightarrow^b s_{n+1}\}$.

Lemma 4.3 *For* $\mathcal{S} = (S, i, L, \mathrm{Tran})$ *let* $se(\mathcal{S}) := (E, <, \#, l)$ *with* $E := S \setminus \{i\}$, $e < e' \Leftrightarrow e \rightarrow^+ e'$, $e \# e' \Leftrightarrow \neg(e \rightarrow^* e') \ \& \ \neg(e' \rightarrow^* e)$ *and* $l : E \rightarrow L$ *with* $l(e) := l_{\mathcal{S}}(e)$. *For* $(\sigma, \lambda) \in Hom_{\mathbf{S}}(\mathcal{S}_1, \mathcal{S}_2)$ *let* $se(\sigma, \lambda) := (g_\sigma, \lambda)$ *with*
$g_\sigma(e) := \begin{cases} \sigma(e) & \text{if } \lambda(l_{\mathcal{S}_1}(e)) \neq * \\ * & \text{otherwise} \end{cases}$
Then $se : \mathbf{S} \rightarrow \mathbf{E}$ *is a full and faithful functor.*

Note that this lemma does not follow from [WN95] where different categories have been used.

For each prime event structure $\mathcal{E} = (E, \leq, \#, l : E \rightarrow L)$ define a synchronization tree $es(\mathcal{E}) := Unfold(Conf_f(\mathcal{E}), \emptyset, L, \mathrm{Tran})$ with $(X, a, Y) \in \mathrm{Tran} \Leftrightarrow \exists e \in \mathcal{R}(\mathcal{E}) : Y = X \uplus \{e\} \ \& \ l(e) = a$.

es does not extend to a functor. Consider for example the event structures $\mathcal{E}_1$ and $\mathcal{E}_2$ in Figure 2 and the morphism $(g, \lambda) \in Hom_{\mathbf{E}}(\mathcal{E}_1, \mathcal{E}_2)$ with $\lambda(c) = c$. Then $es(\mathcal{E}_2)$ contains two maximal states $(\emptyset, \{e_0\}, \{e_0, e_1\}, \{e_0, e_1, e_2\})$ and $(\emptyset, \{e_1\}, \{e_0, e_1\}, \{e_0, e_1, e_2\})$ to which the state $(\emptyset, \{e_0\})$ of $es(\mathcal{E}_1)$ should be mapped.

Lemma 4.4 *For all synchronization trees* $\mathcal{S}$ *holds* $es(se(\mathcal{S})) \cong_{\mathbf{S}} \mathcal{S}$.

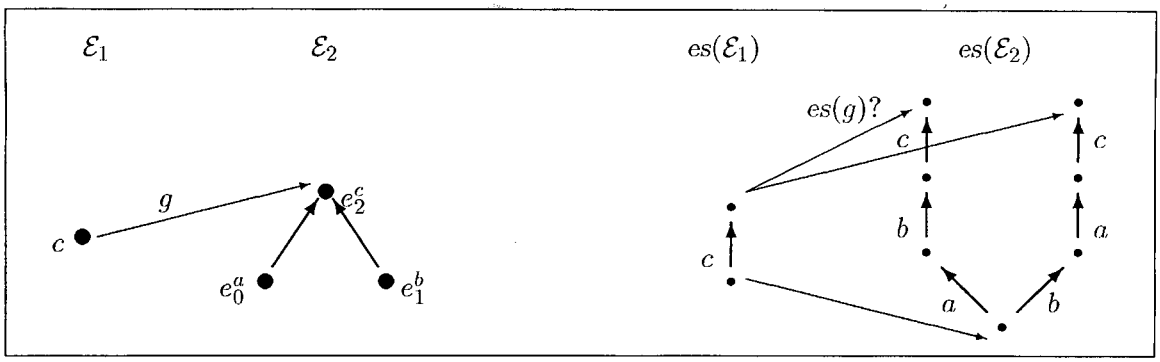

Figure 2: *es* does not extend to a functor

Definition 4.5 *Let* $\mathcal{S}_1 = (S_1, i_1, L_1, \mathrm{Tran}_1)$, $\mathcal{S}_2 = (S_2, i_2, L_2, \mathrm{Tran}_2)$ *be two synchronization trees, such that* $\mathcal{S}_2$ *is non-empty.*
Define $\mathcal{S}_1[a \rightsquigarrow \mathcal{S}_2]_S := \mathit{Unfold}(\mathcal{S}_1[a \rightsquigarrow \mathcal{S}_2]_S)$ *with* $\mathcal{S}_1[a \rightsquigarrow \mathcal{S}_2]_S := (S, i, L, \mathrm{Tran})$ *with*
$S := \{(s_1, *) \mid l(s_1) \neq a\} \cup \{(s_1, s_2) \in S_1 \times S_2 \mid l(s_1) = a \ \&\ s_2 \neq i_2\}$,
$i = (i_1, *)$, $L := (L_1 \setminus \{a\}) \cup L_2$, *and*

$$\begin{aligned}(s_1, s_2) \xrightarrow{b} (s_1', s_2') \Leftrightarrow \quad & (s_1 \xrightarrow{b}_1 s_1' \ \&\ (s_2 = * \vee s_2 \in \surd_{\mathcal{S}_2}) \ \&\ s_2' = *) \\ \vee \quad & (s_1 \xrightarrow{a}_1 s_1' \ \&\ (s_2 = * \vee s_2 \in \surd_{\mathcal{S}_2}) \ \&\ i_2 \xrightarrow{b}_2 s_2') \\ \vee \quad & (s_1 = s_1' \ \&\ s_2 \xrightarrow{b}_2 s_2')\end{aligned}$$

One can easily verify that this definition coincides with the one in [DD91] apart from the fact that our definition also works for infinitely branching trees.

For an example consider Figure 3.

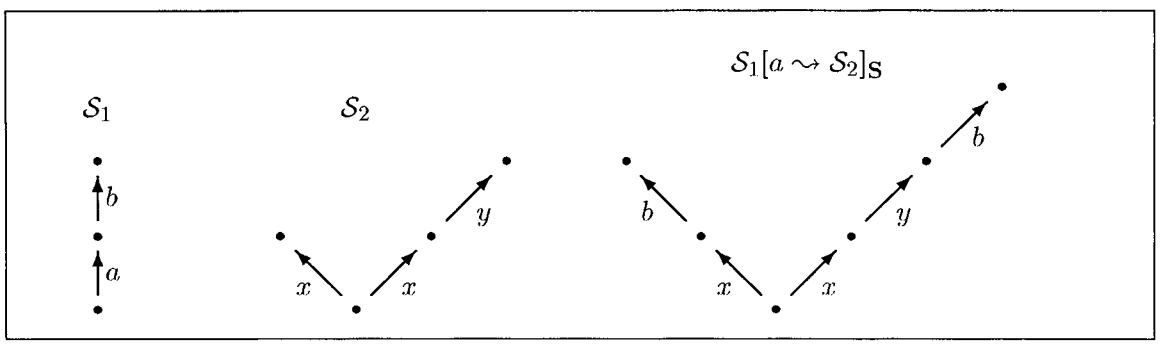

Figure 3: Refinement in Synchronization Trees

4.2 Languages

Languages are left-closed sets of traces ([Ho81]) over an alphabet L (compare with the corresponding category in [WN95]).

Definition 4.6
$\mathcal{L} = (T, L)$ *is called a* language *if* T *is a non-empty left-closed subset of* L^*.

A language $\mathcal{L} = (T, L)$ is called *empty* iff $T = \{\epsilon\}$. The language corresponding to the term $a \parallel b$ for example is $(\{\epsilon, a, ab\}, \{a, b\})$ and the one corresponding to $a.(b + c)$ is $(\{\epsilon, a, ab, ac\}, \{a, b, c\})$.

For a trace $t \in L^*$ define the last label $last(t) := \begin{cases} a & \text{if } \exists t' \in L^* : t = t'.a \\ * & \text{otherwise} \end{cases}$

For traces $t, t' \in L^*$ define $t \leq t'$ if t is a prefix of t', $t < t'$ if $t \leq t' \ \& \ t \neq t'$.

Definition 4.7 *The category* $\mathbf{L}$ *of languages:*

Objects: *languages*

Morphisms: $(\sigma, \lambda) \in Hom_{\mathbf{L}}(\mathcal{L}_1, \mathcal{L}_2)$ *iff*

a) $\sigma : T_1 \to^* T_2, \lambda : L_1 \to^* L_2$ *such that* $\sigma(\epsilon) = \epsilon$

b) $\forall t_1, t_1' \in T_1 : \sigma(t_1) < \sigma(t_1') \Rightarrow t_1' \not< t_1$

c) $\forall t_1, t_1' \in T_1 : (t_1 < t_1' \ \& \ \sigma(t_1) \neq * \ \& \ \sigma(t_1') \neq *) \Rightarrow \sigma(t_1) \leq \sigma(t_1')$

d) $\forall t_1 \in T_1 \setminus \{\epsilon\} : \sigma(t_1) = * \Leftrightarrow \lambda(last(t_1)) = *$.

Lemma 4.8 *For* $\mathcal{L} = (T, L)$ *let* $ls(\mathcal{L}) := (S, i, L, Tran)$ *with* $S := T$, $i := \epsilon$ *and* $Tran := \{(t, a, t') \in T \times L \times T \mid t' = t.a\}$. *For* $(\sigma, \lambda) \in Hom_{\mathbf{L}}(\mathcal{L}_1, \mathcal{L}_2)$ *let* $ls(\sigma, \lambda) := (\sigma, \lambda)$. *Then* $ls : \mathbf{L} \to \mathbf{S}$ *is a full and faithful functor.*

For each synchronization tree $\mathcal{S} = (S, i, L, Tran)$ define a language $sl(\mathcal{S}) := (T, L)$ with $T = Traces(\mathcal{S})$, i.e. $T = \{(a_1 ... a_n) \in L^* \mid \exists i = s_0 \to^{a_1} s_1 ... \to^{a_n} s_n \in \mathcal{S}\}$. sl does not extend to a functor because different states of a synchronization tree may correspond to one trace. But if they are mapped to different states this morphism cannot be transfered to a morphism in $\mathbf{L}$.

Lemma 4.9 *For all languages* $\mathcal{L}$ *holds* $sl(ls(\mathcal{L})) = \mathcal{L}$.

Definition 4.10 *For* $t \in L_1^*$ *and* $T_2 \subseteq L_2^*$ *define* $t[a \rightsquigarrow T_2]_{\mathbf{t}}$ *inductively:* $\epsilon[a \rightsquigarrow T_2]_{\mathbf{t}} := \{\epsilon\}$,
$(t_1.b)[a \rightsquigarrow T_2]_{\mathbf{t}} := \begin{cases} \{t'.b \mid t' \in Max(t_1[a \rightsquigarrow T_2]_{\mathbf{t}})\} & \text{if } b \neq a \\ \{t'.t_2 \mid t' \in Max(t_1[a \rightsquigarrow T_2]_{\mathbf{t}}) \ \& \ t_2 \in T_2\} & \text{otherwise} \end{cases}$
(with $Max(X)$ *denoting the set of maximal elements of* X*)*

For languages $\mathcal{L}_1 = (T_1, L_1)$ *and* $\mathcal{L}_2 = (T_2, L_2)$ *with* $\mathcal{L}_2$ *being non-empty let* $\mathcal{L}_1[a \rightsquigarrow \mathcal{L}_2]_{\mathbf{L}} := (T, L)$ *with* $L := (L_1 \setminus \{a\}) \cup L_2$ *and* $T := \bigcup_{t_1 \in T_1} t_1[a \rightsquigarrow T_2]_{\mathbf{t}}$.

One easily verifies that refining a language by a language again yields a language.

Consider for example the languages $\mathcal{L}_1 = (T_1, L_1)$ corresponding to the term $ab.(c + d)$ and $\mathcal{L}_2 = (T_2, L_2)$ corresponding to $x.(y + z)$. Then $T_1 = \{\epsilon, a, ab, abc, abd\}$, $L_1 = \{a, b, c, d\}$ and $\mathcal{L}_2 = (T_2, L_2)$ with $T_2 = \{\epsilon, x, xy, xz\}$, $L_2 = \{x, y, z\}$. Then $a[a \rightsquigarrow T_2]_{\mathbf{t}} = T_2$, $ab[a \rightsquigarrow T_2]_{\mathbf{t}} = \{xyb, xzb\}$, $abc[a \rightsquigarrow T_2]_{\mathbf{t}} = \{xybc, xzbc\}$ and $abd[a \rightsquigarrow T_2]_{\mathbf{t}} = \{xybd, xzbd\}$. Thus $\mathcal{L}_1[b \rightsquigarrow \mathcal{L}_2]_{\mathbf{L}} = (T, L)$ with $T = \{\epsilon, a, ax, axy, axz, axyc, axzc, axyd, axzd\}$ and $L = \{c, d, x, y, z\}$.

4.3 Categorical Refinement on the Interleaving Models

First we show that the refinement definitions for synchronization trees and languages are compatible with the one for prime event structures in the following sense: Refining in object in an abstract category and then embedding the result in the more concrete one yields an isomorphic result to first embedding an object and then refining it.

Lemma 4.11

a) For all synchronization trees $\mathcal{S}_1 = (S_1, i_1, L_1, \mathrm{Tran}_1), \mathcal{S}_2 = (S_2, i_2, L_2, \mathrm{Tran}_2)$ *with* $\mathcal{S}_2$ *being non-empty,* $a \in L_1$ *and* $L_1 \cap L_2 = \emptyset$ *holds:*
$se(\mathcal{S}_1[a \rightsquigarrow \mathcal{S}_2]_{\mathbf{S}}) \cong_e se(\mathcal{S}_1)[a \rightsquigarrow se(\mathcal{S}_2)]_{\mathbf{E}}$.

b) For all languages $\mathcal{L}_1 = (T_1, L_1), \mathcal{L}_2 = (T_2, L_2)$ *with* $\mathcal{L}_2$ *being non-empty,* $a \in L_1$ *and* $L_1 \cap L_2 = \emptyset$ *holds:* $ls(\mathcal{L}_1[a \rightsquigarrow \mathcal{L}_2]_{\mathbf{L}}) \cong_{\mathbf{S}} ls(\mathcal{L}_1)[a \rightsquigarrow ls(\mathcal{L}_2)]_{\mathbf{S}}$.

Proof:

a) Let $\mathcal{T} = (S, i, L, \mathrm{Tran}) = \mathcal{S}_1[a \rightsquigarrow \mathcal{S}_2]_S$.
Then $se(\mathcal{S}_1[a \rightsquigarrow \mathcal{S}_2]_{\mathbf{S}}) = se(Unfold(\mathcal{T})) = (E, <, \#, l : E \to L)$ with
$E = \{(s^1, ..., s^n) \mid n > 1 \ \&\ s^1 = (i_1, *) \ \&\ \forall 1 \leq j < n : s^j \to s^{j+1}\}$.
Let $\mathcal{E}' = (E', <', \#', l' : E' \to L') = se(\mathcal{S}_1)[a \rightsquigarrow se(\mathcal{S}_2)]_{\mathbf{F}}$.
Then $L' = (L_1 \setminus \{a\}) \cup L_2 = L$ and $E' = S \setminus \{(i_1, *)\}$. Obviously $X \subseteq E'$ is a configuration of $\mathcal{E}'$ iff it is sequentially ordered. We thus define $(h, 1_L) \in Hom_{\mathbf{E}}(\mathcal{E}, \mathcal{P}(\mathcal{E}'))$ with $h(s^1, ..., s^n) = \downarrow_X (s^n)$ with $X = \{s^1, ..., s^n\}$. One can easily verify that $(h, 1_L)$ is an isomorphism.

b) Let $\mathcal{L} = (T, L) := \mathcal{L}_1[a \rightsquigarrow \mathcal{L}_2]_{\mathbf{L}}$ and $ls(\mathcal{L}) = \mathcal{S} = (S, i, L, \mathrm{Tran})$. Then $S = T$. Let $ls(\mathcal{L}_j) = (S_j, i_j, L_j, \mathrm{Tran}_j)$ (i.e. $S_1 = T_1$ and $S_2 = T_2$) and $\mathcal{S}' = (S', i', L, \mathrm{Tran}') = ls(\mathcal{L}_1)[a \rightsquigarrow ls(\mathcal{L}_2)]_{\mathbf{S}}$.
Define $h : S \to S'$ inductively: $h(\epsilon) = ((\epsilon, *))$. If $h(t) = ((t_1^1, t_2^1), ..., (t_1^n, t_2^n))$ let

$$h(t.b) := \begin{cases} h(t).(t_1^n.b, *) & \text{if } t_1^n.b \in T_1 \ \&\ (t_2^n = * \vee t_2^n \in Max(T_2)) \\ h(t).(t_1^n, t_2^n.b) & \text{if } last(t_1^n) = a \ \&\ t_2^n.b \in T_2 \\ h(t).(t_1^n.a, b) & \text{if } t_1^n.a \in T_1 \& (t_2^n = * \vee t_2^n \in Max(T_2)) \ \&\ b \in T_2 \end{cases}$$

Obviously h is well-defined and total. By induction on trace length one can verify that h is injective. For $s' \in S'$ define $h'(s') \in T$ inductively:
$h'((\epsilon, *)) = \epsilon$, $h'((t_1^1, t_2^1), ..., (t_1^n, t_2^n), (t_1^{n+1}, t_2^{n+1})) = h'((t_1^1, t_2^1), ..., (t_1^n, t_2^n)).b$
with $b := \begin{cases} last(t_1^{n+1}) & \text{if } t_1^{n+1} \neq t_1^n \ \&\ last(t_1^{n+1}) \neq a \\ last(t_2^{n+1}) & \text{if } last(t_1^{n+1}) = a \end{cases}$
Then $h \circ h' = id_{S'}$ and therefore h is surjective. One can easily verify that $\mathrm{Tran}' = \{(h(s), b, h(s')) \mid (s, b, s') \in \mathrm{Tran}\}$ and thus $\mathcal{S} \cong_{\mathbf{S}} \mathcal{S}'$. □

Theorem 3 *Let* $\mathcal{S}_2 = (S_2, i_2, L_2, \mathrm{Tran}_2)$ *be a finite and non-empty synchronization tree with* $l_2(S_2) = L_2$. *Let* $\mathcal{S}_1 = (S_1, i_1, L_1, \mathrm{Tran}_1)$ *be a synchronization tree such that* $a \in l_1(S_1)$ *and* $L_1 \cap L_2 = \emptyset$. *Let* $(\sigma, \lambda) \in Hom_{\mathbf{S}}(\mathcal{S}_1[a \rightsquigarrow \mathcal{S}_2]_{\mathbf{S}}, \mathcal{S}_2)$ *with* $\sigma : ((s_1^1, s_2^1), ...(s_1^n, s_2^n)) \mapsto s_1^n$,

$$\lambda : b \mapsto \begin{cases} a & \text{if } b \in L_2 \\ b & \text{otherwise} \end{cases}$$

Then $(\mathcal{S}_1[a \rightsquigarrow \mathcal{S}_2]_{\mathbf{S}}, \mathcal{S}_1, (\sigma, \lambda))$ *is a categorical refinement in* $\mathbf{S}$.

Proof:
With lemma 2 we know that $(se(\mathcal{S}_1)[a \rightsquigarrow se(\mathcal{S}_2)]_{\mathbf{E}}, se(\mathcal{S}_1), (g, \lambda))$ with
$g : {\downarrow_X}(s_1, s_2) \mapsto s_1$ and λ like above is a categorical refinement in $\mathbf{E}$.

With lemma 4.11 we know that $se(\mathcal{S}_1[a \rightsquigarrow \mathcal{S}_2]_{\mathbf{S}}) \cong_{\mathbf{E}} se(\mathcal{S}_1)[a \rightsquigarrow se(\mathcal{S}_2)]_{\mathbf{E}}$.
Let $(h, 1_L) \in Hom_{\mathbf{E}}(se(\mathcal{S}_1[a \rightsquigarrow \mathcal{S}_2]_{\mathbf{S}}), se(\mathcal{S}_1)[a \rightsquigarrow se(\mathcal{S}_2)]_{\mathbf{E}})$ with
$h : (s^1, ..., s^n) \mapsto {\downarrow_X}(s^n)$ with $X = \{s^1, ..., s^n\}$ (compare the proof of lemma 4.11). Then $(h, 1_L)$ is an isomorphism.

Since $se(\sigma, \lambda) = (g, \lambda) \circ (h, id_L)$ we conclude that $(\mathcal{S}_1[a \rightsquigarrow \mathcal{S}_2]_{\mathbf{S}}, \mathcal{S}_1, (\sigma, \lambda))$ is a categorical refinement in $\mathbf{S}$. □

Theorem 4 *Let $\mathcal{L}_2 = (T_2, L_2)$ be a finite non-empty language satisfying $L_2 = \{b \mid \exists t_2' \in T_2 : t_2'.b \in T_2\}$. Let $\mathcal{L}_1 = (T_1, L_1)$ be an arbitrary language such that $\exists t_1 \in T_1 : t_1.a \in T_1$ and $L_1 \cap L_2 = \emptyset$.*

Let $(\sigma, \lambda) \in Hom_{\mathbf{L}}(\mathcal{L}_1[a \rightsquigarrow \mathcal{L}_2]_{\mathbf{L}}, \mathcal{L}_1)$ with $\lambda : b \mapsto \begin{cases} a & \text{if } b \in L_2 \\ b & \text{otherwise} \end{cases}$ and with σ: $\sigma(\epsilon) = \epsilon$,

$$\sigma(t.b) = \begin{cases} \sigma(t).b & \text{if } b \in L_1 \\ \sigma(t) & \text{if } b \in L_2 \ \&\ \exists t_1, t_2, t_2 \neq \epsilon : t = t_1.t_2 \ \&\ t_2.b \in T_2 \\ \sigma(t).a & \text{otherwise} \end{cases}.$$

Then $(\mathcal{L}_1[a \rightsquigarrow \mathcal{L}_2]_{\mathbf{L}}, \mathcal{L}_1, (\sigma, \lambda))$ form a categorical refinement in $\mathbf{L}$.

Proof: With lemma 3 we know that $(ls(\mathcal{L}_1)[a \rightsquigarrow ls(\mathcal{L}_2)]_{\mathbf{S}}, ls(\mathcal{L}_1), (\sigma', \lambda))$ with
$\sigma' : ((s_1^1, s_2^1), ...(s_1^n, s_2^n)) \mapsto s_1^n$ and λ like above form a categorical refinement in $\mathbf{S}$. The rest follows analogously to the proof of 3. □

5 Concluding Remark

We introduced a categorical definition of refinement and showed that refinement on event structures, synchronisation trees and languages can be viewed as categorical refinement. For this we adapted categories introduced by [WN95] for our purposes. The the characterization of operations like relabelling, restriction, product and sum that was given in [WN95] – under some restrictions – carries over to our models (see [Be97]).

It should be noted that our notion of categorical refinement allows for constructions that would not be considered as refinement in the various models. Let us consider e.g. flow event structures. One can define an operation satisfying the conditions of categorical refinement that transforms a flow event structure in such a way that two events with the same label are substituted by different event structures. In our setting it is also possible to refine a synchronizing action for each partner in a different way. This kind of refinement was suggested by [Hu96]. But there causality and conflict are not necessarily inherited while in for categorical refinement this inheritance is demanded.

There exists yet another approach to characterize refinement by categorical means: [DGR92] propose to view each event structure as a category and model refinement as a functor between such categories.

Work is in progress that considers refinement in various other models of concurrency.

References

[**AH93**] L. Aceto, M. Hennessy, "Towards action-refinement in process algebras", Information and Computation, vol. 103, pp. 204 -269, 1993.

[**Ba81**] R.J.R. Back, "On Correct Refinement of Programs", Journal of Computer And System Sciences 23, 1981.

[**Be96**] F. Benjes, "Some Properties of Refinement", TR 14/96, Fakultaet fuer Mathematik und Informatik, Universitaet Mannheim, 1996.

[**Be97**] F. Benjes "Verfeinerung in verschiedenen Modellen fuer Paralleles Rechnen", Dissertation, Universitaet Mannheim, 1997.

[**BDE93**] E. Best, R. Devillers, J. Esparza, "General Refinement and Recursion Operators for the Petri Box Calculus" in P. Enjalbert, A. Finkel, K. W. Wagner (eds.), STACS 93, LNCS 665, pp. 130-140, Springer-Verlag, 1993.

[**Bo90**] G. Boudol, "Flow Event Structures and Flow Nets", in I. Guessarian (ed.): Semantics of Systems of Concurrent Processes, LNCS 469, pp. 62-95, Springer-Verlag, 1990.

[**dBV92**] J.W. de Bakker, E.P. de Vink, "Bisimulation Semantics for Concurrency with Atomicity and Action Refinement", CWI technical report, 1992.

[**BW90**] M. Barr, C. Wells, "Category Theory for Computing Science", Prentice Hall, 1990.

[**Co95**] R. Costantini, "Abstraktion in ereignisbasierten Modellen verteilter Systeme", Ph.D Thesis, University of Hildesheim, 1995.

[**CGG91**] I. Czaja, R. J. van Glabbeek, U. Goltz, "Interleaving Semantics and Action Refinement with Atomic Choice", Arbeitspapiere der GMD 594, 1991.

[**CR92**] R. Costantini, A. Rensink, "Abstraction and Refinement in Configuration Structures", Hildesheimer Informatik-Berichte 18/92, 1992.

[**DD91**] P. Darondeau, P. Degano, "About Semantic Action Refinement", Fundamenta Informaticae XIV, pp. 221-234, 1991.

[**DG95**] P. Degano, R. Gorrieri, "A Causal Operational Semantics of Refinement", Information and Computation, vol. 122, pp. 97-119, 1995.

[**DGR92**] P. Degano, R. Gorrieri, G. Rosolini, "A Categorical View of Process Refinement", in J. de Bakker, G. Rozenberg, J. Rutten (eds.): Proc. REX Workshop on Semantics: Theory and Applications, LNCS 666, pp. 138-153, Springer-Verlag, 1992.

[**DGR93**] P. Degano, R. Gorrieri, G. Rosolini, "Graphs and Event Refinement", Proc. Workshop on Semantics: Theory and Applications, 1993

[GG89] R. J. van Glabbeek, U. Goltz: "Partial Order Semantics for Refinement of Actions – neither necessary nor always sufficient but appropriate when used with care –", in BEATCS: Bulletin of the European Association for Theoretical Computer Science, vol. 38, 1989.

[GG90a] R. J. van Glabbeek, U. Goltz, "Refinement of Actions in Causality Based Models", in J.W. de Bakker, W.-P. de Roever & G. Rozenberg (eds.): Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness, LNCS 430, pp. 267-300, Springer-Verlag, 1990.

[GG90b] R. J. van Glabbeek, U. Goltz: "Equivalences and Refinement", in I. Guessarian (ed.): Semantics of Systems of Concurrent Processes, LNCS 469, pp. 309-333, Springer-Verlag, 1990.

[GGR94] U. Goltz, R. Gorrieri, A. Rensink, "On Syntactic and Semantic Action Refinement", in: M. Hagiya, J.C. Mitchell (eds.): Theoretical Aspects of Computer Software, LNCS 789, pp. 385-404, Springer-Verlag, 1994.

[Ho81] C.A.R. Hoare, "A Model for Communicating Sequential Processes", Technical Report PRG-22, Programming Research Group, Oxford University Computing Laboratory, 1981.

[Hu96] M. Huhn, "Action Refinement and Property Inheritance in Systems of Sequential Agents", CONCUR '96, LNCS 1119, Springer-Verlag, 1996.

[JM92] L. Jategaonkar, A. Meyer, "Testing Equivalence for Petri Nets with Action Refinement", in W.R. Cleaveland (ed.): Proc. CONCUR '92, LNCS 630, pp. 17-31, Springer-Verlag, 1992.

[JNW94] A. Joyal, M. Nielsen, G. Winskel, "Bisimulation from open maps", BRICS Report RS-94-7, Aarhus University, 1994.

[Mi80] R. Milner, "Calculus of communicating systems", LNCS 92, Springer-Verlag, 1980.

[Mi89] R. Milner, "Communication and concurrency", Prentice Hall, 1989.

[ML71] S. Mac Lane, "Categories for the Working Mathematician", Graduate Texts in Mathematics, Springer-Verlag, 1971.

[NEL89] M. Nielsen, U. Engberg, K. S. Larsen, "Fully Abstract Models for a Process Language with Refinement", in J. W. de Bakker, W. P. de Roever, G. Rozenberg (eds.): Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, LNCS 354, pp. 523-548, Springer-Verlag, 1989.

[NSW94] M. Nielsen, V. Sassone, G. Winskel, "Relationships Between Models of Concurrency", LNCS 803, pp. 425-476, Springer-Verlag, 1994.

[Re95] A. Rensink, "An Event-Based SOS for a Language with Refinement", presented at STRICT (Structures In Concurrency Theory), 1995.

[**Vo89**] W. Vogler, "Failures Semantics Based on Interval Semiwords is a Congruence for Refinement", 1989.

[**Vo93**] W. Vogler, "Bisimulation and Action Refinement", Theoretical Computer Science, vol. 114, pp. 173-200, 1993.

[**We93**] H. Wehrheim, "Parametric Action Refinement", Hildesheimer Informatik-Berichte, 1993.

[**Wi87**] G. Winskel, "Event Structures", in Brauer, Reissig, Rozenberg (eds.): Petri Nets: Applications and relationships to the models of concurrency, LNCS 255, pp. 325-392, Springer-Verlag, 1987.

[**WN95**] G. Winskel, M. Nielsen "Models for Concurrency", vol. 4 of the Handbook of Logic in Computer Science, Oxford University Press, 1995.