In: M. Nüttgens, F. J. Rump (Eds.):
EPK 2004, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten.
Proceedings. 3. Workshop. October 6, 2004, Luxembourgh.
pp 7-26.

On the semantics of EPCs: Efficient calculation and simulation

Nicolas Cuntz, Ekkart Kindler
Computer Science Department, University of Paderborn, Germany
cuntz|kindler@upb.de

Abstract: One of the most debatable features of *Event driven Process Chains* (EPCs) is their non-local semantics, which results in some difficulties when defining a formal semantics for EPCs. Recently, we have overcome these problems by using techniques from fixed-point theory for the definition of the semantics for an EPC, which consists of a pair of related transition relations for each EPC.

The fixed-point characterization of the semantics for EPCs, however, provides a mathematical characterization of the semantics of EPCs only. For simulating an EPC based on this semantics, we need an efficient way for calculating the corresponding pair of transition relations. A naive implementation of the underlying fixed-point approximation for calculating the transition relations, however, results in a practically useless algorithm.

In this paper, we show how to calculate the transition relations of EPCs in a more efficient way by employing different techniques and tricks from symbolic model checking for the calculation of the pair of transition relations. In addition, we analysed all kinds of simplifications of EPCs to make the calculation of the semantics more efficient, but it turned out that most of these techniques are ineffective. Still, the algorithms are fast enough for simulating practical size EPCs.

In order to demonstrate the efficiency of our algorithms and data structures, we have started an open source tool for EPCs, which we call *EPC Tools*. Right now, EPC Tools can simulate EPCs and can check some simple properties. But, EPC Tools is open for adding more sophisticated analysis and verification algorithms, and could provide a good starting point for an open source tool for the EPC community.

1 Introduction

Event driven Process Chains (*EPCs*) have been introduced in the early 90ties for modelling business processes [KNS92]. Initially, EPCs have been used informally only, without a fixed formal semantics. For easing the modelling of business processes with EPCs, the informal semantics proposed for the OR-join and the XOR-join connectors of EPCs is *non-local*. This non-local semantics, however, results in severe problems when it comes to a formalization of the semantics of EPCs and, recurrently, resulted in a debate on the semantics of EPCs [LSW98, Ri00]. It turned out that these problems are inherent to the informal non-local semantics of EPCs. In [vdADK02], we pin-pointed these arguments, which render a formal semantics that exactly captures the non-local semantics of an EPC in terms of a single transition relation impossible. But, we have shown that we can define

a semantics for an EPC that consists of a pair of two correlated transition relations [Ki04b] by using fixed-point theory.

Due to their non-local semantics EPCs cannot be simply simulated by looking at the current state; rather it requires calculating the transition relations beforehand. In principle, the two transition relations defined as the semantics of an EPC can be calculated by fixed-point iteration. The problem, however, is that the calculation of the transition relations by naive fixed-point iteration is very inefficient and intractable in practice. In this paper, we will show that some techniques from *symbolic model checking* [BCM$^+$92, Mc93, CGP99] and *ordered binary decision diagrams* (ROBDDs) [Br86] in particular can be used for calculating the semantics of an EPC in a more efficient way.

We have implemented an EPC tool based on these techniques, which simulates practically relevant EPCs with a reasonable response time. Since this tool needs to calculate the transition relations of an EPC anyway, it was easy to also implement some simple semantical checks, and it should be easy to add all kinds of more sophisticated analysis and verification methods due to the fact that we use model checking techniques already for the computation of the transition relation. The tool is open source and is based on the Eclipse platform [Ecl]. Therefore, it could serve as the starting point of an open source project for a collection of EPC tools, which is the reason for calling it *EPC Tools*.

2 Syntax and Semantics of EPCs

In this section, we introduce the syntax and the semantics of EPCs as defined and motivated in [Ki04b]. The syntax, basically follows the presentation of [NR02] and the semantics is inspired by the ideas of [KNS92, NR02], but resolving the technical problems of the non-locality of EPCs as pointed out in [vdADK02].

2.1 Syntax

Figure 1 shows an example of an EPC. It consists of three kinds of nodes: *events*, which are graphically represented as hexagons, *functions*, which are represented as rounded boxes, and *connectors*, which are represented as circles. The dashed arcs between the different nodes represent the *control flow*. The two black circles do not belong to the EPC itself; they represent a *state* of an EPC. A state, basically, assigns a number of *process folders* to each arc of the EPC. Each black circle represents a process folder at the corresponding arc. In order to express some of the syntactical restrictions, we introduce a simple notation for the ingoing and outgoing arcs of a node first:

Notation 1 (Ingoing and outgoing arcs) Let N be a set of *nodes* and let $A \subseteq N \times N$ be a binary relation over N, the *arcs*. For each node $n \in N$, we define the set of its *ingoing arcs* $n_{in} = \{(x, n) \mid (x, n) \in A\}$, and we define the set of its *outgoing arcs* $n_{out} = \{(n, y) \mid (n, y) \in A\}$.

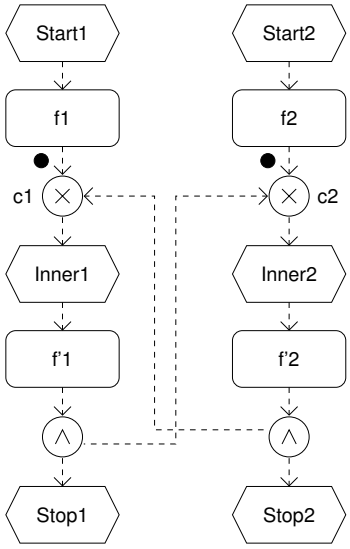

Figure 1: An EPC

A connector can be either an AND-, an OR-, or an XOR-connector, which is indicated by labelling the connector correspondingly. Each function has exactly one ingoing and one outgoing arc, whereas each event has at most one ingoing and at most one outgoing arc. A connector has multiple ingoing arcs and one outgoing arc, or it has one ingoing arc and multiple outgoing arcs:

Definition 2 (EPC) An *EPC* $M = (E, F, C, l, A)$ consists of three pairwise disjoint sets E, F, and C, a mapping $l : C \to \{and, or, xor\}$ and a binary relation $A \subseteq (E \cup F \cup C) \times (E \cup F \cup C)$ such that

- $|e_{in}| \leq 1$ and $|e_{out}| \leq 1$ for each $e \in E$,
- $|f_{in}| = |f_{out}| = 1$ for each $f \in F$, and
- either $|c_{in}| > 1$ and $|c_{out}| = 1$ or $|c_{in}| = 1$ and $|c_{out}| > 1$ for each $c \in C$.

An element of E is called an *event*, an element of F is called a *function*, an element of C is called a *connector*, and an element of A is called a *control flow* arc.

Note, that we do not consider subprocesses (called process signs in EPCs) here and that we have omitted some syntactical restrictions for EPCs. Subprocesses and the syntactical restrictions are important from a practical point of view, but they are not relevant for defining the semantics of EPCs. So, we do not formalize these restrictions here. For a complete exposition of the syntax of EPCs, we refer to [NR02] .

In the definition of the semantics, we will need to distinguish among different types of connectors: AND-, OR-, and XOR-, each of which can be either a *split* or a *join connector*.

Notation 3 (Nodes and connectors) For the rest of this paper, we fix the EPC $M = (E, F, C, l, A)$. We denote the set of all its nodes by $N = E \cup F \cup C$ and we define the following sets of connectors:

	split connectors	join connectors
$\wedge$	$C_{as} = \{c \in C \mid l(c) = and \wedge \lvert c_{in}\rvert = 1\}$	$C_{aj} = \{c \in C \mid l(c) = and \wedge \lvert c_{out}\rvert = 1\}$
$\vee$	$C_{os} = \{c \in C \mid l(c) = or \wedge \lvert c_{in}\rvert = 1\}$	$C_{oj} = \{c \in C \mid l(c) = or \wedge \lvert c_{out}\rvert = 1\}$
$\times$	$C_{xs} = \{c \in C \mid l(c) = xor \wedge \lvert c_{in}\rvert = 1\}$	$C_{xj} = \{c \in C \mid l(c) = xor \wedge \lvert c_{out}\rvert = 1\}$

2.2 States

For defining the semantics of EPCs, we need to define the *states* of an EPC first. In general, a state is an assignment of a number of process folders to the arcs of the EPC. In order to keep the state space and the transition relations finite, we consider the case of at most one folder at each arc here.

Definition 4 (State of an EPC) For an EPC $M = (E, F, C, l, A)$, we call a mapping $\sigma : A \rightarrow \{0, 1\}$ a *state* of M. The set of all states of M is denoted by Σ.

2.3 Transition relation

The semantics of an EPC defines how process folders are propagated through an EPC. Clearly, this depends on the involved node. For events and functions, a process folder is simply propagated from the incoming arc to the outgoing arc. The *transition relation* for events and functions is graphically represented in the top row of Fig. 2 (a. and b.). For connectors, the propagation of folders depends on the type of the connector (*AND*, *OR*, resp. *XOR*) and whether it is a *join* or a *split connector*. Figure 2 shows the transition relation for the connectors. For example, the AND-split connector (c.) propagates a folder from its incoming arc to all outgoing arcs. The AND-join connector (d.) needs one folder on each incoming arc, which are propagated to a single folder on the outgoing arc.

The more interesting connectors are the OR-join and the XOR-join connectors. Here, we focus on the XOR-join. An XOR-join (h.) waits for a folder on one incoming arc, which is then propagated to the outgoing arc. But, there is one additional condition: The XOR-join must not propagate the folder, if there is or there could arrive a folder on the other incoming arc. In Fig. 2.h, this is represented by a label #•• at the other arc. Note that this condition cannot be checked locally: whether a folder could arrive on the other arc or not depends on the overall behaviour of the EPC. Therefore, we call the semantics of the XOR-join connector *non-local*. Likewise, the OR-join (f.) has a non-local semantics.

Note that, in this informal definition of the *transition relation*, we refer to the transition relation itself when we require that no folders should arrive at some arcs according to the transition relation. Therefore, we cannot immediately translate it to a mathematically sound definition. For now, we resolve this problem by assuming that one transition relation P is given, and we define a function $R(P)$, which defines the transition relation by referring to P instead of referring to $R(P)$.

For defining $R(P)$, we introduce some notation for restricting transition relations and for

its induced reachability relation:

Notation 5 (Restriction and reachability) For some transition relation $R \subseteq \Sigma \times N \times \Sigma$, and some subset $N' \subseteq N$, we define the *restriction of* R to N' as $R|_{N'} = \{(\sigma, n, \sigma') \in R \mid n \in N'\}$. By slight abuse of notation, we define the *reachability relation* R^* of R as the reflexive and transitive closure of the binary relation $\rightarrow = \{(\sigma, \sigma') \in \Sigma \times \Sigma \mid \exists n \in N : (\sigma, n, \sigma') \in R\}$.

Next we can define $R(P)$, where we define a separate transition relation $R_n(P)$ for each node $n \in N$ of the EPC first. Then, the overall transition relation $R(P)$ is just the union of the transition relations $R_n(P)$ of all nodes n.

Definition 6 (Transition relation $R(P)$) Let P be a transition relation for an EPC M. For each node $n \in N$, we define the transition relation $R_n(P) \subseteq \Sigma \times N \times \Sigma$ as follows:

a. For $n = e \in E$ with $e_{in} = \{i\}$ and $e_{out} = \{o\}$, we define $R_e(P) \subseteq \Sigma \times N \times \Sigma$ by $(\sigma, e, \sigma') \in R_e(P)$ iff $\sigma(i) = 1$, $\sigma(o) = 0$, $\sigma'(i) = 0$, $\sigma'(o) = 1$, and $\sigma'(a) = \sigma(a)$ for each $a \in A \setminus \{i, o\}$.

a': For $n = e \in E$ with $e_{in} = \emptyset$ or $e_{out} = \emptyset$, we define $R_e(P) = \emptyset$.

b. For $n = f \in F$ with $f_{in} = \{i\}$ and $f_{out} = \{o\}$, we define $R_f(P) \subseteq \Sigma \times N \times \Sigma$ by $(\sigma, f, \sigma') \in R_f(P)$ iff $\sigma(i) = 1$, $\sigma(o) = 0$, $\sigma'(i) = 0$, $\sigma'(o) = 1$, and $\sigma'(a) = \sigma(a)$ for each $a \in A \setminus \{i, o\}$.

c. For $n = c \in C_{as}$ with $c_{in} = \{i\}$, we define $R_c(P) \subseteq \Sigma \times N \times \Sigma$ by $(\sigma, c, \sigma') \in R_c(P)$ iff $\sigma(i) = 1$, $\sigma(o) = 0$ for each $o \in c_{out}$, $\sigma'(i) = 0$, $\sigma'(o) = 1$ for each $o \in c_{out}$, and $\sigma'(a) = \sigma(a)$ for each $a \in A \setminus (\{i\} \cup c_{out})$.

d. For $n = c \in C_{aj}$ with $c_{out} = \{o\}$, we define $R_c(P) \subseteq \Sigma \times N \times \Sigma$ by $(\sigma, c, \sigma') \in R_c(P)$ iff $\sigma(i) = 1$ for each $i \in c_{in}$, $\sigma(o) = 0$, $\sigma'(i) = 0$ for each $i \in c_{in}$, $\sigma'(o) = 1$, and $\sigma'(a) = \sigma(a)$ for each $a \in A \setminus (c_{in} \cup \{o\})$.

e. For $n = c \in C_{os}$ with $c_{in} = \{i\}$, we define $R_c(P) \subseteq \Sigma \times N \times \Sigma$ by $(\sigma, c, \sigma') \in R_c(P)$ iff, for some $S \subseteq c_{out}$ with $|S| \geq 1$, we have $\sigma(i) = 1$, $\sigma(o) = 0$ for each $o \in S$, $\sigma'(i) = 0$, $\sigma'(o) = 1$ for each $o \in S$, and $\sigma'(a) = \sigma(a)$ for each $a \in A \setminus (\{i\} \cup S)$.

f. For $n = c \in C_{oj}$ with $c_{out} = \{o\}$, we define $R_c(P) \subseteq \Sigma \times N \times \Sigma$ by $(\sigma, c, \sigma') \in R_c(P)$ iff, for some $S \subseteq c_{in}$ with $|S| \geq 1$, we have $\sigma(i) = 1$ for each $i \in S$, $\widehat{\sigma}(a) = 0$ for each $\widehat{\sigma}$ with $\sigma(P|_{N \setminus \{c\}})^* \widehat{\sigma}$ and for each $a \in c_{in} \setminus S$, $\sigma(o) = 0$, $\sigma'(i) = 0$ for each $i \in S$, $\sigma'(o) = 1$, and $\sigma'(a) = \sigma(a)$ for each $a \in A \setminus (S \cup \{o\})$.

g. For $n = c \in C_{xs}$ with $c_{in} = \{i\}$, we define $R_c(P) \subseteq \Sigma \times N \times \Sigma$ by $(\sigma, c, \sigma') \in R_c(P)$ iff, for some $o \in c_{out}$, we have $\sigma(i) = 1$, $\sigma(o) = 0$, $\sigma'(i) = 0$, $\sigma'(o) = 1$, and $\sigma'(a) = \sigma(a)$ for each $a \in A \setminus \{i, o\}$.

h. For $n = c \in C_{xj}$ with $c_{out} = \{o\}$, we define $R_c(P) \subseteq \Sigma \times N \times \Sigma$ by $(\sigma, c, \sigma') \in R_c(P)$ iff, for some $i \in c_{in}$, we have $\sigma(i) = 1$, $\widehat{\sigma}(a) = 0$ for each $\widehat{\sigma}$ with $\sigma(P|_{N \setminus \{c\}})^* \widehat{\sigma}$ and for each $a \in c_{in} \setminus \{i\}$, $\sigma(o) = 0$, $\sigma'(i) = 0$, $\sigma'(o) = 1$, and $\sigma'(a) = \sigma(a)$ for each $a \in A \setminus \{i, o\}$.

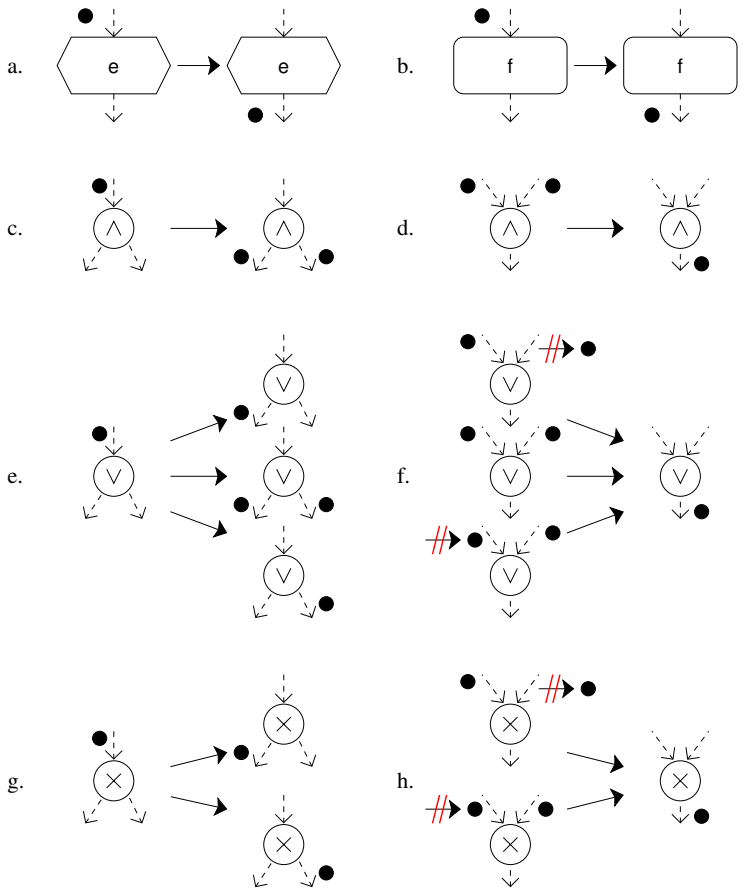

Figure 2: The transition relation $P_n = R(P)$ for the different nodes n

We define the *transition relation* $R(P) = \bigcup_{n \in N} R_n(P)$

Below, we briefly discuss the important cases f. and h. of the above definition, which concerns the non-local semantics of the OR-join and the XOR-join connectors: When there is a folder on at least one of its incoming arcs $S \subseteq c_{in}$ of an OR-join connector and no folder can arrive (according to P) on the other arcs without the occurrence of c, the folder is propagated to the outgoing arc. In order to formalize that no folder can arrive on the other incoming arcs $a \in c_{in} \setminus S$, the definition refers to the states $\widehat{\sigma}$ that can be reached from σ (with respect to P) without the occurrence of c. This is formalized by $\sigma \ (P|_{N \setminus \{c\}})^* \ \widehat{\sigma}$. The XOR-join is similar to the definition of the OR-join. Instead of selecting some set S of incoming arcs on which a folder must be present, we select exactly one incoming arc i. We require that no folder can arrive on the other incoming arcs (with respect to P) before the occurrence of c. Again, this can be formalized by using the relation $(P|_{N \setminus \{c\}})^*$.

The most important property of $R(P)$ is that it is monotonously decreasing in P, i. e. for each two transition relations P and P' with $P \subseteq P'$ we have $R(P) \supseteq R(P')$. The reason is that P occurs under a negation in the definition of $R(P)$ (see [Ki04b] for more details).

2.4 Semantics

Based on $R(P)$, we can now define the semantics of the EPC. Ideally, we would like to define it to be a fixed-point $P = R(P)$. Unfortunately, there are EPCs for which $R(P)$ does not have a fixed-point. So, we define it as a pair of transition relations P and Q such that $P = R(Q)$ and $Q = R(P)$, were P is the least such transition relation and Q is the greatest such transition relation. In [Ki04b], we have proved that this pair is uniquely defined by applying standard fixed-point theory, exploiting the fact that R is monotonously decreasing. We called P the *pessimistic transition* relation of the EPC, and we called Q the *optimistic transition* relation of the EPC. Unfortunately, P and Q can be different for some (nasty) EPCs, and we have argued that these are exactly the EPCs for which a single transition relation cannot fully capture the informal semantics of EPCs. For EPCs for which P and Q coincide the semantics exactly captures the informal semantics. Therefore, we call EPCs with $P = Q$ *clean*.

In [Ki04b], we did not bother to give an operational characterization of this semantics, since we were interested only in defining a precise semantics. But, the fixed-point theorem of Kleene immediately gives us a simple algorithm for calculating this pair (which is called fixed-point approximation):
Let $P_0 = \emptyset$ and $Q_0 = \Sigma \times N \times \Sigma$. For each $i \in \mathbb{N}$, we define $P_{i+1} = R(Q_i)$ and $Q_{i+1} = R(P_i)$. Since $R(P)$ is monotonously decreasing, we have that $P_i \subseteq P_{i+1}$ and $Q_i \supseteq Q_{i+1}$ for each i. Moreover, $\Sigma \times N \times \Sigma$ is finite, which implies that for some $i \in \mathbb{N}$ we will have $P_{i+1} = P_i$ and $Q_{i+1} = Q_i$. For this i, we have $R(P_i) = Q_{i+1} = Q_i$ and $R(Q_i) = P_{i+1} = P_i$. And this (P_i, Q_i) is the semantics of the EPC. So, starting with $P_0 = \emptyset$ and $Q_0 = \Sigma \times N \times \Sigma$ and iteratively computing the next P_{i+1} and Q_{i+1} will eventually terminate with the semantics of the EPC.

Unfortunately, an explicit representation of the transition relations P_i and Q_i and an explicit calculation of $P_{i+1} = R(Q_i)$ and $Q_{i+1} = R(P_i)$ is extremely inefficient. For realistic EPCs, there are millions of potential states Σ and billions of potential arcs in the transition relation[1]. Moreover, an explicit calculation of $R(P)$ involves a reachability analysis on P (at least for the non-local connectors). So a naive explicit implementation of the fixed-point approximation does not work in practice.

3 Calculating the transition relations

In the previous sections, we have rephrased the semantics of EPCs in an operational way. Next, we will show how the two transition relations can be calculated in a more efficient way. To this end, we will use techniques from symbolic model checking and ordered binary decision diagrams. We use formulas and temporal formulas for representing the transition relations $R_n(P)$, and we will show how these formulas can be used for efficiently calculating the semantics of the underlying EPC.

[1] Note that not all of these states will be reachable in the final semantics; but they will be necessary in the calculation of the semantics.

3.1 Representing $R_n(P)$

Let us start with the semantics of an AND-split connector, with ingoing arc i and outgoing arcs $o_1, \ldots, o_n$. In order to define the corresponding behaviour, we assume that i and $o_1, \ldots, o_n$ are boolean variables. The values of these variables represent the state before the transition, where value true means that there is a process folder on the corresponding arc, and value false means that there is no process folder. Moreover, we assume that, for each variable, there is a primed version i' and $o'_1, \ldots, o'_n$, which represent the state after the transition. With this notation and understanding, the behaviour of the AND-split can be expressed by the following formula:

$$i \wedge \neg o_1 \wedge \ldots \wedge \neg o_n \ \wedge \ \neg i' \wedge o'_1 \wedge \ldots \wedge o'_n$$

This formula exactly captures the fact that there must be a folder on the ingoing arc i of the AND-split and there must be no folders on the outgoing arcs $o_1, \ldots, o_n$ before firing the AND-split; and, after firing the AND-split, the ingoing arc has no folder anymore, but the outgoing arcs have a folder each. Altogether the formula is an immediate translation of Def. 6 (c), where we assume that variables not occurring in the formula do not change.

Altogether, we can apply this standard technique [CGP99, HR00] for defining the behaviour of all EPC nodes with a local semantics. The complete list of formulas for all connectors is shown below, where, for simplicity, we assume that connectors have at most two input and output arcs:

a. / b. For $n \in E \cup F$ with $n_{in} = \{i\}$ and $n_{out} = \{o\}$, the formula for $R_n(P)$ is $i \wedge \neg o \wedge \neg i' \wedge o'$.

c. For $n = c \in C_{as}$ with $c_{in} = \{i\}$ and $c_{out} = \{o_1, o_2\}$, the formula for $R_n(P)$ is $i \wedge \neg o_1 \wedge \neg o_2 \wedge \neg i' \wedge o'_1 \wedge o'_2$.

d. For $n = c \in C_{aj}$ with $c_{in} = \{i_1, i_2\}$ $c_{out} = \{o\}$, the formula for $R_n(P)$ is $i_1 \wedge i_2 \wedge \neg o \wedge \neg i'_1 \wedge \neg i'_2 \wedge o'$.

e. For $n = c \in C_{os}$ with $c_{in} = \{i\}$ and $c_{out} = \{o_1, o_2\}$, the formula for $R_n(P)$ is $i \wedge \neg(o_1 \wedge o_2) \wedge \neg i' \wedge (o_1 \Rightarrow o'_1) \wedge (o_2 \Rightarrow o'_2) \wedge (o'_1 \neq o_1 \vee o'_2 \neq o_2)$.

g. For $n = c \in C_{xs}$ with $c_{in} = \{i\}$ and $c_{out} = \{o_1, o_2\}$, the formula for $R_n(P)$ is $i \wedge \neg(o_1 \wedge o_2) \wedge \neg i' \wedge (o_1 \Rightarrow o'_1) \wedge (o_2 \Rightarrow o'_2) \wedge (o'_1 \neq o_1 \ xor \ o'_2 \neq o_2)$.

The formulas for the OR- and the XOR-split connectors are a bit more involved. For the OR-split connector (cf. e.), it is required that no outgoing arcs has less folders than before and at least one has more, which can be formulated in terms of a implication $o \Rightarrow o'$ (i. e. if there is a folder on o in the source state of the transition then there is a folder on o in the target state of the transition).

For the XOR-split (cf. g.) connector, we also require that no outgoing arc has less folders than before and exactly one arc has one more. Some formulas are a bit involved, but, in principle, there is no problem with these formulas for the local connectors, because the transition relation $R_n(P)$ does not refer to P.

But, how about the formulas for the non-local operators? For these connectors the definition of $R_n(P)$ refers to P. So, we need to refer to P in the formula for $R_n(P)$ somehow. To this end, we use a temporal logic formula that is interpreted on the transition relation P. Since we use very simple formulas only, we do not bother to introduce temporal logic in full detail. The only temporal operator needed is the CTL operator EF: For some formula φ the temporal formula $EF\varphi$ is true in exactly those states from which a state can be reached (with respect to P) in which φ is valid. This way, we can express that no folder can arrive on some arc i by the formula $\neg EFi$.

With this temporal formula, it is easy to express the behaviour of the XOR-join connector: For an XOR-join connector with the two incoming arcs i_1 and i_2 and one outgoing arc o,

$$((i_1 \wedge \neg EFi_2) \vee (\neg EFi_1 \wedge i_2)) \wedge \neg o \wedge \neg i_1' \wedge \neg i_2' \wedge o'$$

precisely captures its behaviour. The formulas $\neg EFi_1$ resp. $\neg EFi_2$ guarantee that a transition does occur only when no folder can arrive from the other arc, respectively.

For the OR-join connector, the transition relation is similar. It requires that there is one folder on one incoming arc and, if there is no folder on the other incoming arc no folder can arrive at this arc anymore. Altogether, we define:

f. For $n = c \in C_{oj}$ with $c_{in} = \{i_1, i_2\}$ $c_{out} = \{o\}$, the formula for $R_n(P)$ is $((i_1 \wedge i_2) \vee (i_1 \wedge \neg EFi_2) \vee (\neg EFi_1 \wedge i_2)) \wedge \neg o \wedge \neg i_1' \wedge \neg i_2' \wedge o'$

h. For $n = c \in C_{xj}$ with $c_{in} = \{i_1, i_2\}$ and $c_{out} = \{o\}$, the formula for $R_n(P)$ is $((i_1 \wedge \neg EFi_2) \vee (\neg EFi_1 \wedge i_2)) \wedge \neg o \wedge \neg i_1' \wedge \neg i_2' \wedge o'$

Experts in model checking may be a bit concerned about mixing primed variables and temporal operators in a single formula. Usually, there are transition formulas that may contain primed variables, but no temporal operators, and there are temporal formulas that must not contain primed variables. A transition formula or a set of transition formulas represents the underlying system; the temporal formulas represent properties to be verified for that system. Though uncommon, in principle, there is no harm in mixing primed variables and temporal operators in a single formula. It defines a new transition relation based on a given transition relation, which is exactly what we need for calculating $R_n(P)$.

3.2 Calculating the transition relations

Next, we will discuss how to calculate the two transition relations that actually represent the semantics of an EPC, where we assume that the EPC has the local nodes $l_1, \ldots, l_j$ and the non-local nodes $n_1, \ldots, n_k$, and $g_1, \ldots, g_j$ are the formulas representing the transition relations $R_{l_i}(P)$ for the local nodes, and $h_1, \ldots, h_k$ are the formulas representing the transition relations $R_{n_i}(P)$ for the non-local nodes.

Let us first discuss the operations from model checking that we need for this calculation. In symbolic model checking, a transition relation given as a formula (with primed variables) is transformed into a data structure that is called a *reduced ordered binary decision*

diagram[2] (ROBDD), which have the nice feature that equivalent formulas will have exactly the same ROBDD representation. For a formula f with primed variables without temporal operators, there is a standard procedure for this transformation [CGP99, HR00]. We denote this procedure by f`.toROBDD()`, which is close the corresponding methods of our object oriented model checker MCiE [Ki04a].

Formulas with primed variables and temporal variables are very uncommon. So there is no standard procedure for converting it to an ROBDD. But, there is a standard procedure for calculating an ROBDD representing the set of states of a transition system in which a given temporal formula is true. We assume that the transition system is given as a set P of ROBDDs representing the transitions of the system. This procedure can be easily extended to formulas that contain primed variables. For such a formula f and an ROBDD-representation P of the transition relation, f`.toROBDD(`P`)` denotes the resulting ROBDD.

Given some transition system (represented as a set of ROBDDs) $\mathtt{P}_{\mathtt{curr}}$, we can calculate $\mathtt{P}_{\mathtt{next}} = R(\mathtt{P}_{\mathtt{curr}})$ as follows:

```
P_next := { g_1.toROBDD(),..., g_j.toROBDD() };
for i := 1 to k do
    P_next := P_next.add(h_i.toROBDD(P_curr));
```

In the first line, we insert all the transitions of the local nodes to $\mathtt{P}_{\mathtt{next}}$; in the loop, we add the transition relation for each non-local node to $\mathtt{P}_{\mathtt{next}}$. To be precise, the calculation is a bit more involved: In order to exactly capture the semantics formalized in Sect. 2.4, we must switch off the transition relation corresponding to node n_i for calculating the next transition relation for node n_i. Since this is a minor technical detail, we do not include this into the presented pseudo code.

Based on this code, we can easily start the calculation of the transition relations P_i and Q_i as stated in the formal definition of the semantics in Sect. 2: $P_0 = \emptyset$ and $Q_0 = \Sigma \times N \times \Sigma$ and $P_{i+1} = R(Q_i)$ and $Q_{i+1} = R(P_i)$. In order to save computation time, we do not calculate every P_i and every Q_i, rather we calculate $Q_0, P_1, Q_2, P_3, \ldots$ in a zig-zag way. As stated before, we will eventually end up with $P_{i+2} = P_i$ and $Q_{i+2} = Q_i$; in order to detect this point, we need to store the last two versions of the calculated transition relations and compare them to the next one. When they are equal, we have calculated the two transition relations that represent the semantics of the EPC.

Altogether, the algorithm for calculating the semantics of an EPC looks as follows.

[2] Often reduced ordered binary decision diagrams are called ordered binary decision diagrams (OBDDs) or even binary decision diagrams (BDDs) only. We stick to the term ROBDD throughout this paper, however.

```
P_curr := { false }; // P_0
P_next := { true }; // Q_0
step := 1;

repeat
    P_prev := P_curr;
    P_curr := P_next;
    step := step + 1;

    // P_next = R(P_curr)
    P_next := { g_1.toROBDD(),..., g_j.toROBDD() };
    for i := 1 to k do
        P_next := P_next.add(h_i.toROBDD(P_curr));

until P_next == P_prev;
```

Upon termination $\texttt{P}_{\texttt{curr}}$ and $\texttt{P}_{\texttt{next}}$ contain the two transition relations for the EPC. The question, however, is which of them is the pessimistic and which is the optimistic transition relation. In order to decide this, we use the `step` counter. If it is odd, $\texttt{P}_{\texttt{curr}}$ represents the pessimistic transition relation and $\texttt{P}_{\texttt{next}}$ is the optimistic transition relation; otherwise, it is the other way round.

3.3 Simulation

Once we have calculated the two transition relations for an EPC, it is easy to simulate it. For some given state, we must calculate all nodes that can propagate a process folder (according to the pessimistic or according to the optimistic transition relation). In that case, we call the corresponding node *enabled* in this state. Since we store the calculated ROBDDs P_x for each transition relation for node x separately, checking the enabledness is simple. Let `enabled` be the CTL formula $EXtrue$, which is valid in all states for which the underlying transition relation has a successor. Then `enabled.toROBDD(`P_x`)` represents all those states in which the node is enabled.

When the user wants to fire an enabled transition, the simulator explicitly removes and adds the folders in the current state according to semantics of the corresponding node. It is not necessary to use ROBDDs here because only the enabledness of a node is non-local. The propagation of the folders itself is local.

3.4 Implementation

It is easy to implement the above algorithms based on some standard ROBDD package. The only tricky part might be the mixed occurrence of primed variables and temporal operators in formulas. Since our own *Model Checking in Education* (MCiE) project im-

mediately supports this kind of formulas, we implemented the algorithm based on MCiE. Though MCiE is implemented in Java and efficiency is not MCiE's highest priority, the first experiments with this algorithm were surprisingly good. Without further optimizations, it worked reasonably well on small EPCs. For calculating the semantics for larger EPCs, however, we had to come up with some optimizations, which will be discussed below.

In principle, we could use any other ROBDD package for implementing the calculation of the semantics of EPCs. Since we were heading for an Eclipse based tool (see Section 5) and MCiE was available in Java, however, we still use MCiE.

4 Optimizations

As mentioned above, we had to apply several tricks and optimizations in order to compute the semantics of larger EPCs. In our discussion, we distinguish between two different kinds of optimizations.

The first kind tries to exploit properties of the semantics of EPCs in order to reduce and to simplify them. The idea is to calculate the semantics of a simpler and smaller EPC and, based on this information, simulate the original EPC. These optimizations have been investigated in [Cu04]. Unfortunately, there are many negative results, which basically can be considered as a backfiring of the non-local semantics of EPCs. The non-local semantics of EPCs seems to have many nasty side effects and renders many ideas for optimizations impossible – except for very trivial ones.

The second kind is a smart application and combination of optimization techniques generally known from model checking. It turned out that these techniques were much more effective than the ones for EPCs and could be used in combination with the ones for EPCs.

Note that, in spite of all our optimizations, the worst case complexity of our algorithms is still very bad: it is exponential. It is an interesting open question whether this is inherent to the semantics of EPCs or not. But, we feel that, again, this worst case complexity cannot be avoided because of the non-local semantics of EPCs. But, our experimental results have shown that, for many practical examples, we can calculate the semantics of many practically relevant EPCs in a reasonable time (see Sect. 4.3).

4.1 EPC techniques

We start with a brief discussion of techniques that exploit the properties of EPCs.

Eliminating chains It is clear that reducing the size of an EPC also reduces the complexity of the simulation problem. For our model checking algorithm, the number of arcs of the simulated EPC is essential, because the computation time is exponential in the number of variables respectively arcs.

One possible approach is to simplify an EPC by eliminating chains of nodes that do not influence the semantical behaviour of other nodes. Obviously, a sequence of consecutive event and function nodes such as the ones shown in Fig. 3 (labelled Event and Function) can be omitted when computing the enabledness of the XOR-join connectors. We call this optimization *chain elimination*,

We can apply chain elimination, when the following two conditions are satisfied:

1. In the considered state, there are no process folders on the arcs eliminated by this simplification.

   It is obvious that, otherwise, a process folder in the predecessor set of an XOR-join connector which would have potentially influenced the behaviour of the XOR-join in the original EPC would be missing in the simplified EPC.

   Note that this condition implies that we can omit only those arcs from a chain that do not have a folder on them. Therefore, chain elimination depends on the considered state of the EPC. For simulation, this is no serious problem because we can compute another simplified EPC each time the state has changed. Since the simplified EPC is much smaller than the original one, we can hope that the fixed-point computation is significantly faster for the reduced EPC. For the analysis and, in particular, for checking whether the semantics of an EPC is clean, however, we cannot apply this chain elimination technique directly.

2. In order to correctly apply chain elimination, it is necessary that in no reachable state of the reduced EPC, a node is blocked because of a process folder on one of its outgoing edges. We call such states *contact situations*. The problem with *contact situations* is, that the simplified EPCs tend to have more contact situations as compared to the original EPCs. In this situation, the behaviour of the original and the simplified EPC are different. The simplified EPC is blocked, whereas the original version could still fire. Therefore, we cannot use the simplified version for simulation the original one. Fortunately, it is easy to calculate whether the simplified EPC has reachable contact situations, which provides us an a posteriori condition, whether chain elimination can be applied. In that case, we can switch back to calculating the semantics of the original EPC, which of course is less efficient.

If both requirements have been checked, the simulator can use the transition relation computed for the simplified EPC to determine whether an XOR-join resp. an OR-join connector is enabled in the original EPC or not (other nodes can be checked locally anyway). Because we only eliminate event and function nodes, those connectors are contained in the reduced EPC.

The main disadvantage of the chain elimination approach is that it cannot be applied for arbitrary EPCs because of the above requirements. Also, chain elimination does not allow us to calculate the complete semantics of an EPC. Therefore, it can be used for simulation only; it cannot be used for our analysis and verification algorithms.

Syntactical restrictions An other idea for simplifying the simulation problem was to identify some restricted classes of EPCs for which no fixed-point iteration would be nec-

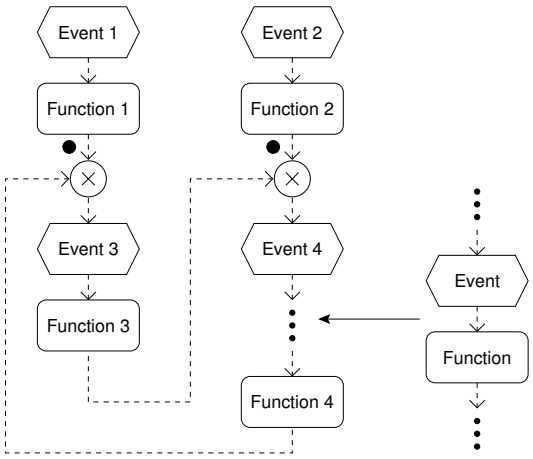

Figure 3: The example used for the measurements in Fig. 4

essary. For example, we considered EPCs without cycles on non-local nodes such as the ones shown in Fig. 1, or EPCs that are constructed from clean EPC constructs only. We hoped that we could calculate the semantics of EPCs from these sub-classes in a much more efficient way. Unfortunately, it turned out that this hope was in vain, and we found some nasty counter-examples, which spoiled this approach. A detailed discussion of these negative results can be found in [Cu04].

4.2 Model checking techniques

There are many techniques that make model checking more efficient. Using ROBDDs as a representation for sets of states and for the transition relations is one of them. It is only this choice, that made our algorithms work for small examples. In addition to using ROBDDs, we used two other techniques: optimization of the variable order and partitioning of the transition relation.

Variable order It is well-known that the size (number of nodes) of the ROBDDs representing some boolean function or formula strongly depends on the chosen variable order. In turn, the computation time of the operations on ROBDDs depends on the size of the ROBDDs. So, it is important to find a good variable order for efficiently calculating the semantics of EPCs. One heuristic for a good variable order is that related variables should be close to each other in the variable order. For EPCs, it is quite easy to identify those variables (arcs) that are related: Two variables resp. arcs are related, when they are attached to the same node. The problem, however, is that each arc belongs to two nodes; so it is impossible to have all related variables close to each other in the variable order, in particular, when the EPC has cycles in its control flow arcs. In order to calculate a good variable order, we thought of some sophisticated schemes. But, in the end, it turned out that a simple breadth first traversal of all nodes starting from the start events of the EPC

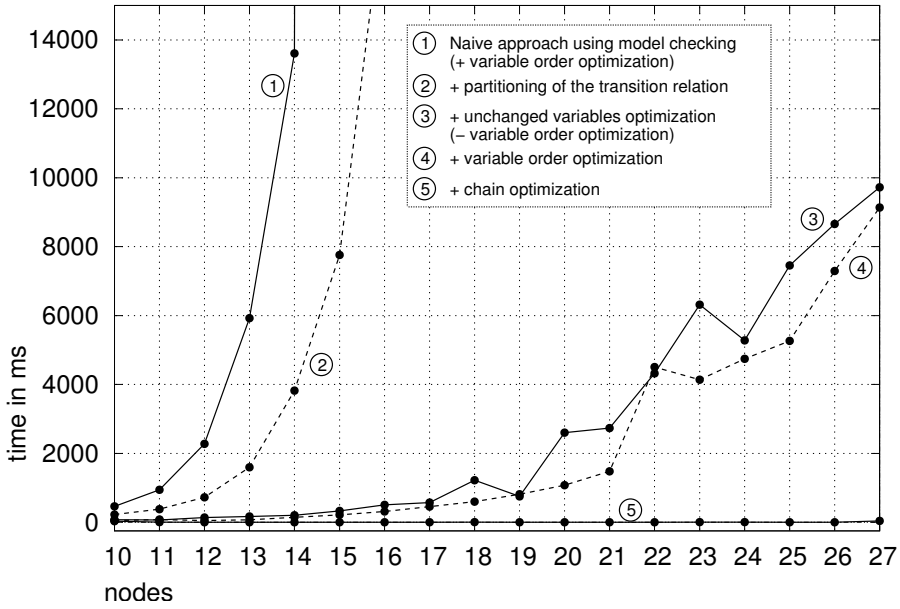

Figure 4: Benefit of the implemented optimization techniques (compare to Fig. 3)

provided a variable order with the best results.

Though this variable order provided satisfactory results, we feel that there is some room for further improvement. But, this needs further investigations.

Partitioning of the transition relation In the algorithm for calculating the transition relations of an EPC, we distinguish the ROBDDs for the transition relations for each node of the ROBDD. It is well known that this results in much less nodes for representing the transition relation than for representing all transitions within a single ROBDD.

In order to make these ROBDDs even smaller, we imposed one additional assumption on the formulas representing the transition relation: we assume that all variables not occurring in the formula do not change. Expressed in a naive way, this means adding the formula $a_1 = a_1' \wedge a_2 = a_2' \wedge \ldots \wedge a_n = a_n'$ for all variables that are not touched by this node. Adding this formula explicitly to the transition relation, however, would result in much bigger ROBDDs, which in turn would result in much longer computation times. Therefore, we did not add this formula to the representation of the transition relation, but we implemented the procedure for calculating EX within the ROBDD library in such a way that these variables were implicitly assumed to be unchanged. This *unchanged variables optimization* resulted in significantly better computation times.

4.3 Measurements

In order to illustrate the benefits of the optimizations, Fig. 4 shows the computation times for calculating the semantics of the example of Fig. 3 for the different optimization techniques. In order to see the influence of the size of the EPC, we measured the computation time for different numbers of nodes on the chain between Event 4 and Function 4. The x-axis represents the number of nodes of the resp. EPC, the y-axis shows the computation times for the different optimizations. The first graph shows the computation time without partitioning the transition relations. The second graph shows the computation time with partitioning, but without the improvement for unchanged variables. The third graph shows the time when incorporating also the optimization for unchanged variables in the transition relations. The fourth graph shows the time with an optimized variable order. Note that partitioning the transition relation along with an explicit algorithm for unchanged variables makes a significant difference in the computation times.

The fifth graph shows that chain elimination can drastically improve the simulation of EPCs. Note, however, that this example is a bit misleading because it was chosen to show the positive effect of chain elimination. In other examples, the figures are not as impressing and, in many situations, chain elimination is not applicable at all (see discussion above).

The above figures come from a technical example. In order to give an impression of computation times for real-world examples, we have considered two examples from the SAP reference processes of the ARIS Toolset[3], which are shown in Fig. 5. The calculation of the semantics of the first EPC took 13 ms; for the second, it took 4.015 s. With the transition relation computed, the other operations (simulation and analysis) can be done in virtually no time.

5 EPC Tools

The algorithm for calculating the semantics of an EPC and for simulating an EPC based on this semantics is integrated into an Eclipse [Ecl] based tool, which we call *EPC Tools*. EPC Tools can be obtained from [CuKi04] free of charge. Figure 6 shows a screen-shot of Eclipse with the EPC Tools plugin running. EPC Tools comes with a graphical editor and an interactive simulator for EPCs. Moreover, it is easy to import EPCs from other tools because EPC Tools supports the EPC exchange format *EPML* [MN04a], and there are converters between the AML format of the ARIS Toolset and EPML [MN04b].

Moreover, EPC Tools checks simple semantical properties of the EPC. For example, it indicates whether the EPC is clean, i. e. whether both transition relations coincide. This is important, because unclean EPCs can easily lead to different interpretations and should be considered harmful. EPC Tools identifies unclean EPCs right away. In addition, EPC Tools checks whether an EPC might deadlock and whether there are contact situation, i. e.

[3] ARIS Toolset is a registered trademark of IDS Scheer. For more information see `http://www.ids-scheer.com/`.

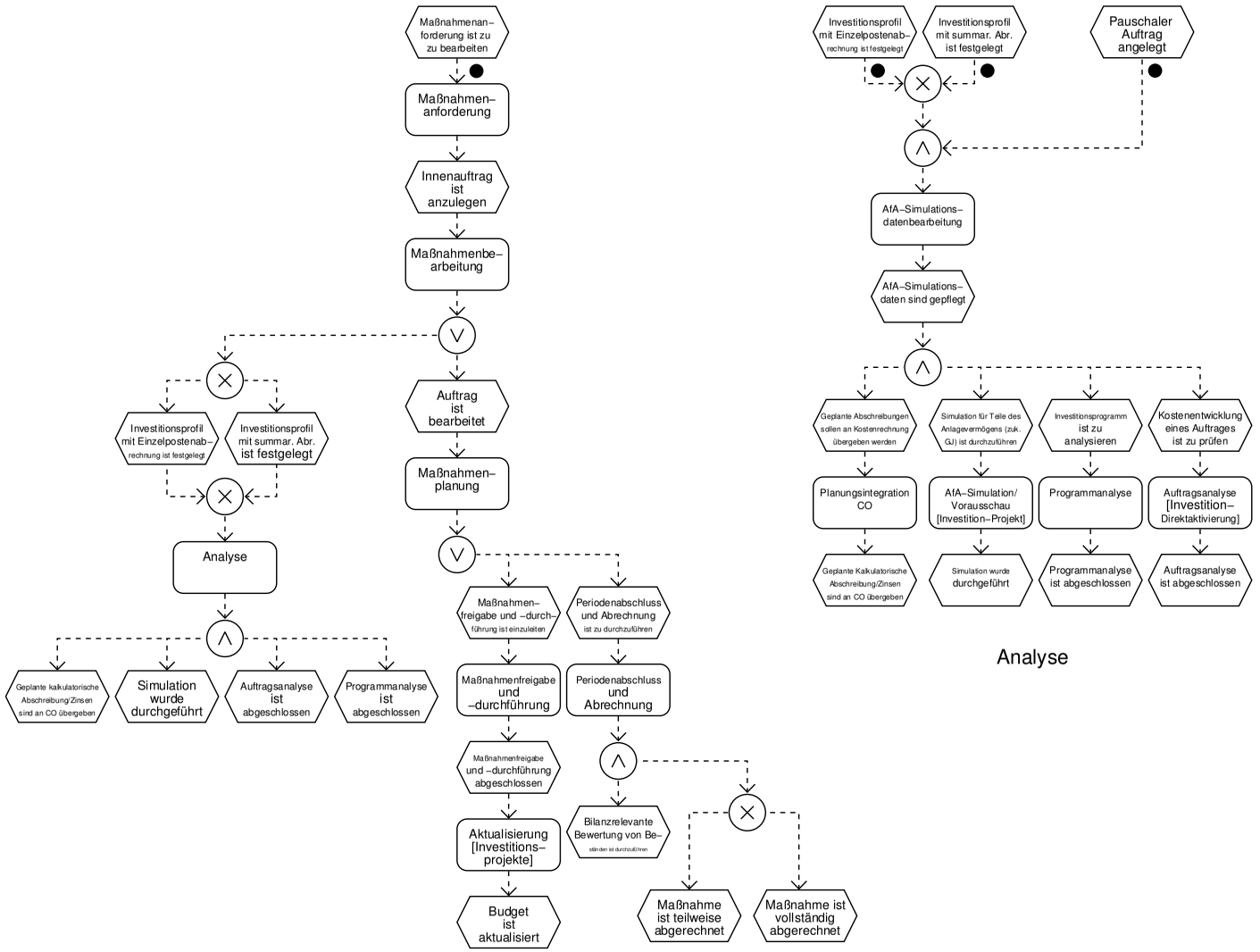

Figure 5: Two examples from the SAP reference processes of the ARIS Toolset

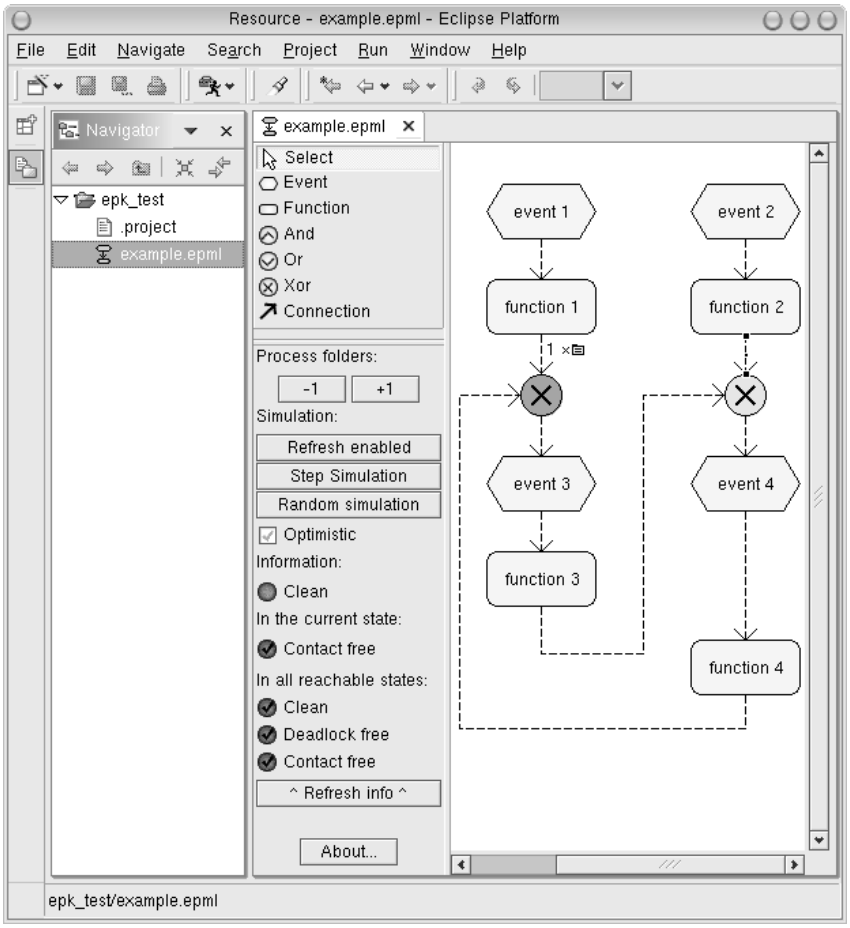

Figure 6: EPC Tools in the Eclipse environment

whether there are situations in which nodes are only blocked because of process folders on their outgoing arcs. Often, such contact situations indicate bad design.

The properties checked right now in EPC Tools, however, are quite preliminary. Once the semantics of the EPC is calculated, we could easily do much more. For example, we could check some soundness properties similar to the soundness criteria for workflow nets as proposed by van der Aalst [vdAvH02] or we could check properties by applying model checking. This should take less time than calculating the semantics. The main problem is to identify the properties that are relevant for EPCs.

Overview on the functionality The EPC Tools plugin can be used to edit, to simulate, and to analyse EPCs with the help of graphical control elements integrated into the Eclipse environment. The editor functions are provided by a tool palette containing buttons for adding nodes and arcs to the EPC. Pushing the “select” button allows a user to move, rename, and scale nodes directly by clicking on them. Some other functions like undo commands are accessible through a context menu. Figure 7 shows all these editor functions. In addition, EPC Tools provides a print function and allows a user to zoom into

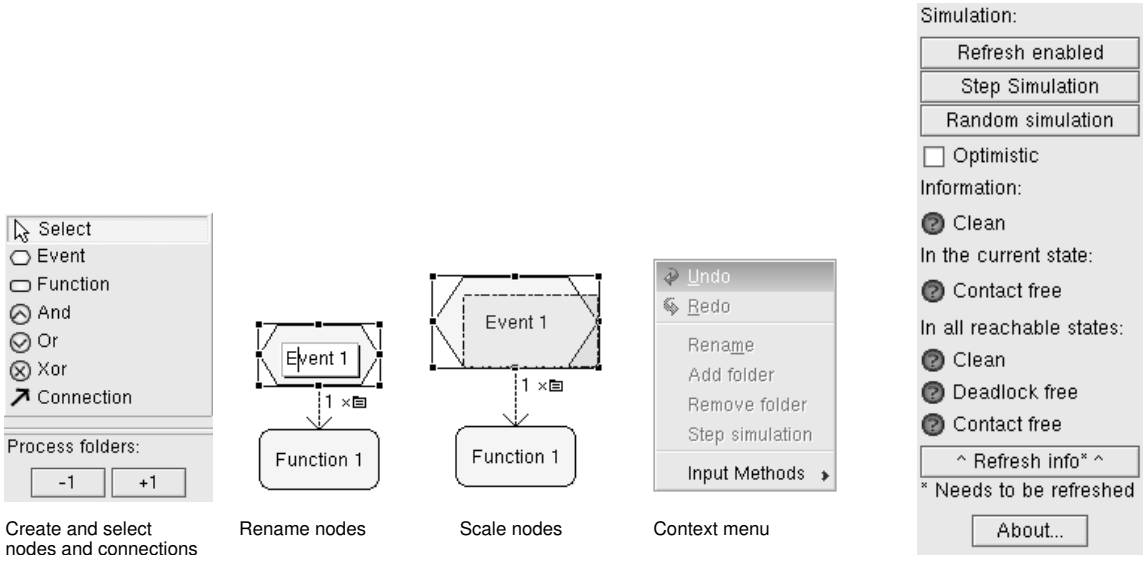

Figure 7: Editor and simulator functions

and out of EPC diagrams by using the standard Eclipse toolbar and the main menu.

The simulator functions are mainly located in the panel shown in Fig. 7 on the right side. It is possible to highlight all currently enabled nodes, and then to simulate one step by specifying a node or by randomly choosing a node. The randomized simulation can be very useful when simulating several steps consecutively by simply clicking one button. A checkbox defines whether the simulation should be done according to the optimistic or according to the pessimistic transition relation computed by the fixed-point iteration algorithm. This option is important when the simulated EPC is unclean, otherwise it does not make any difference. In the same panel, there are some LEDs corresponding to the properties of the EPC. This information can be updated by pushing the “refresh” button. Then, the LEDs light up green or red in order to indicate the valid and invalid properties.

6 Conclusion

In this paper, we have shown that the semantics of an EPC can be efficiently calculated by using ROBDDs and techniques from model checking. With the presented optimizations, the simulation of medium size EPCs works quite well and is practically feasible. Moreover, it is quite easy to adapt this algorithms to slightly different semantics by using different formulas for defining the semantics of the nodes.

The presented algorithms have been implemented in a new Tool for EPCs, which is Eclipse based and is called *EPC Tools*. This tool comes with a graphical editor and it is easy to extend it by new features. EPC Tools is open source published under the GNU Public License, which might make it a good starting point for an open source tool for EPCs. It can be obtained from [CuKi04].

Acknowledgment We would like to thank Jan Mendling for the EPML files of the examples from Fig. 5, which we used as a practical benchmark for the calculation of the semantics.

References

[vdADK02] van der Aalst, W., Desel, J., and Kindler, E.: On the semantics of EPCs: A vicious circle. In: Nüttgens, M. und Rump, F. J. (Eds.), *EPK 2002, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*. pp. 71–79. November 2002.

[vdAvH02] van der Aalst, W. and van Hee, K.: *Workflow Management: Models, Methods, and Systems*. Cooperative Information Systems. The MIT Press, 2002.

[BCM+92] Burch, J., Clarke, E., McMillan, K., Dill, D., and Hwang, L.: Symbolic model checking: 10^{20} states and beyond. *Information and Computation* 98:142–170, 1992.

[Br86] Bryant, R. E.: Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers* 35(8):677–691, 1986.

[CGP99] Clarke, E., Grumberg, O., and Peled, D.: *Model checking*. MIT Press, 1999.

[Cu04] Cuntz, N.: Über die effiziente Simulation von Ereignisgesteuerten Prozessketten. Master's thesis. University of Paderborn, Department of Computer Science, June 2004.

[CuKi04] Cuntz, N. and Kindler, E.: The EPC Tools Project. http://www.upb.de/cs/kindler/research/EPCTools, 2004.

[Ecl] The Eclipse Foundation: The Eclipse platform. http://www.eclipse.org.

[HR00] Huth, M. and Ryan, M.: *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge University Press, 2000.

[Ki04a] Kindler, E.: The Model Checking in Education (MCiE) Project. http://www.upb.de/cs/kindler/teaching/MCiE, 2004.

[Ki04b] Kindler, E.: On the semantics of EPCs: Resolving the vicious circle. In: Desel, J., Pernici, B., and Weske, M. (Eds.), *Business Process Management, Second International Conference, BPM 2004*. *LNCS* 3080, pp. 82–97. Springer, June 2004. (An earlier version of this paper was presented at EPK 03.)

[KNS92] Keller, G., Nüttgens, M., and Scheer, A.-W.: Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK). Technical Report Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi), Heft 89. Universität des Saarlandes, January 1992.

[LSW98] Langner, P., Schneider, C., and Wehler, J.: Petri Net Based Certification of Event driven Process Chains. In: Desel, J. and Silva, M. (Eds.), *Application and Theory of Petri Nets 1998*. *LNCS*1420, pp. 286–305. Springer, 1998.

[Mc93] McMillan, K. L.: *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[MN04a] Mendling, J. and Nüttgens, M.: Exchanging EPC Business Process Models with EPML. In: Mendling, J. and Nüttgens, M. (Eds.), *XML interchange formats for business process management, proceedings of the* 1^{st} *Workshop XML4BPM 2004*, pp. 61–80. March 2004.

[MN04b] Mendling, J. and Nüttgens, M.: Transformation of ARIS Toolset's AML to EPML. To appear.

[NR02] Nüttgens, M. and Rump, F. J.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In: *PROMISE 2002, Prozessorientierte Methoden und Werkzeuge fürr die Entwicklung von Informationssystemen*. *GI Lecture Notes in Informatics* P-21, pp. 64–77. Gesellschaft für Informatik, 2002.

[Ri00] Rittgen, P.: Quo vadis EPK in ARIS? *Wirtschaftsinformatik*. 42:27–35, 2000.