Nicolas Cuntz, Ekkart Kindler

On the semantics of EPCs

Efficient calculation and simulation

One of the most debatable features of Event driven Process Chains (EPCs) is their non-local semantics, which results in some difficulties when formalising their semantics. Recently, we have overcome these problems by using techniques from fixed-point theory for defining the semantics of an EPC, which consists of a pair of related transition relations. This fixed-point characterisation of the semantics of EPCs provides a mathematical characterisation 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 iteration for calculating the transition relations results in a practically useless algorithm.

In this paper, we show how to calculate the semantics of an EPC in a more efficient way by employing different techniques and optimisations from symbolic model checking. We also 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, our 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 project called EPC Tools, which could be 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 [KeNS92]. 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 was *non-local*. This non-local semantics, however, results in severe problems when it comes to a formalisation of the semantics of EPCs and, recurrently, resulted in a debate on the semantics of EPCs [LaSW98, Ritt00]. It turned out that these problems are inherent to the informal non-local semantics of EPCs. In [AaDK02], we pin-pointed these arguments and proved that a formal semantics that exactly captures the non-local semantics of EPCs in terms of a single transition relation does not exist. But, we could define a semantics for an EPC that consists of a pair of two correlated transition relations by using fixed-point theory [Kind04b].

Due to their non-local semantics, EPCs cannot be simulated by looking at the current state only; 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 two transition relations by naive fixed-point iteration is very inefficient and intractable in practise. In this paper, we will show that some techniques from *symbolic model checking* [BCM+92, McMi93, ClGP99] and *ordered binary decision diagrams* (OBDDs) [Brya86] can be used for calculating the semantics of EPCs in a more efficient way.

We have implemented an EPC tool based on these techniques, which simulates practically relevant EPCs within a reasonable response time. Since this tool calculates the transition relations of an EPC anyway, it was easy to implement some simple semantical checks; and it should be easy to add all kinds of more sophisticated analysis and verification methods. The tool is open source and is based on the Eclipse platform [Ecli]. Therefore, it could serve as the starting point of an open source project for an

EPC tool with all kinds of analysis, simulation, and verification features, 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 motivated, discussed and formalised in [Kind04b], which was based on the informal ideas as presented in [KeNS92, NüRu02].

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 the 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.

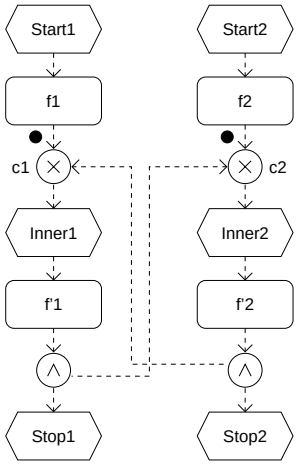

Figure 1: An EPC

Mathematically, the nodes are represented by three pairwise disjoint sets E, F, and C, which represent the events, functions, and connectors, respectively. We denote the set of all nodes by $N = E \cup F \cup C$. The type of each connector is defined by a mapping $t : C \to \{ \mathit{and}, \mathit{or}, \mathit{xor} \}$. The control flow arcs are a subset $A \subseteq N \times N$.

For some node $n \in N$, n_{in} denotes the set of its ingoing arcs, and n_{out} denote the set of its outgoing arcs. With this notation, we can formalise the syntactical restrictions of EPCs: Each connector $c \in C$ is either a *join connector*, i.e. $|c_{in}| > 1$ and $|c_{out}| = 1$, or it is a *split connector*, i.e. $|c_{in}| = 1$ and $|c_{out}| > 1$. Moreover, every function $f \in F$ has exactly one ingoing and one outgoing arc (i.e. $|f_{in}| = |f_{out}| = 1$), and every event $e \in E$ has at most one ingoing arc and at most one outgoing arc (i.e. $|e_{in}| \leq 1$ and $|e_{out}| \leq 1$). Note that there are some more syntactical restrictions on EPCs. But, we omit these restrictions here because they are not important for our semantical considerations (see [NüRu02] for details).

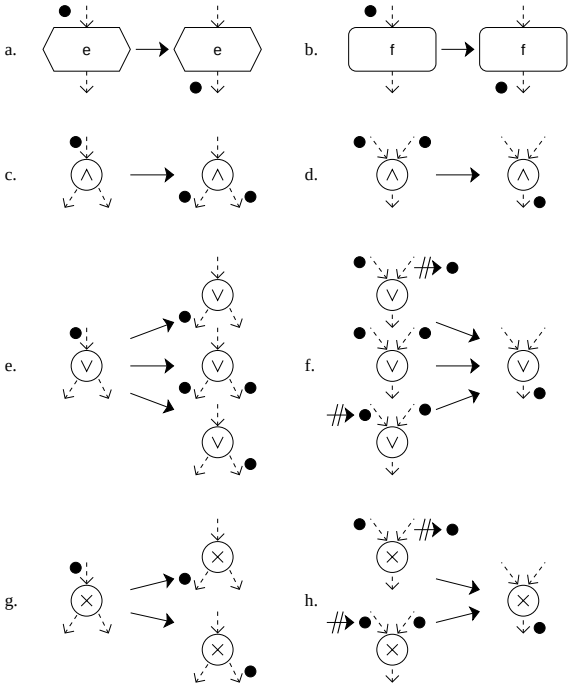

Figure 2: The transition relations for the different nodes

A *state* of an EPC assigns a number of *process folders* to each arc of the EPC; here, we assume that there is at most one process folder at each arc. So a state σ is a mapping $\sigma : A \to \{ 0, 1 \}$. The set of all states will be denoted by Σ.

2.2 Transition relation

The semantics of an EPC defines how process folders are propagated through an EPC. This can be formalised by a *transition relation* $R \subseteq \Sigma \times N \times \Sigma$, where the first component denotes the source state, the third component denotes the target state, and the middle component denotes the involved node. Clearly, the definition of this relation depends on the involved node, which is the reason for defining it for each node n separately $R_n \subseteq \Sigma \times \{ n \} \times \Sigma$ later in this paper.

For events and functions, a process folder is simply propagated from the ingoing arc to the outgoing arc. The transition relation R_n for events and functions is graphically represented in the top row of Figure 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 ingoing arc to all outgoing arcs. Note that an AND-split connector can propagate the folder only when there are no folders on the outgoing arcs, because we do not consider multiple folders here. The AND-join connector (d.) needs one folder on each ingoing arc, all of 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 ingoing 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 ingoing arc.

In Figure 2 (h), this additional condition is represented by a label with a crossed out arrow at the other arc. Note that this condition cannot be checked locally in the current state: 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. In order to resolve this problem, we assume that some transition relation P is given already, and whenever we refer to the non-local condition, we refer to this transition relation P. Thus, Figure 2 defines a mapping $R(P)$ for each node, which defines a transition relation $R(P)$ for some given transition relation P. Actually, we define a mapping $R_n(P)$ for each node n separately, where $R(P)$ is the union of all $R_n(P)$.

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 [Kind04b] for more details).

2.3 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 the mapping R 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 $Q = R(P)$, where P is the least such transition relation and Q is the greatest such transition relation. In [Kind04b], we proved that this pair is uniquely defined by 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 [Kind04b], we did not bother to give an operational characterisation 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 the pair (P, Q), which is called fixed-point approximation or fixed-point iteration:

Let $P_0 = \varnothing$ and $Q_0 = \Sigma \times \Sigma$. For each i, 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, $\Sigma \times \mathrm{N} \times \Sigma$ is finite, which implies that, for some i, 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 pair (P_i, Q_i) is the semantics of the EPC. So, starting with $P_0 = \varnothing$ and $Q_0 = \Sigma \times \Sigma$ 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. So a naive explicit implementation of the fixed-point approximation does not work in practise.

[1] Note that not all of these states will be reachable in the final semantics, but they must be considered during the calculation of the semantics.

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 ordered binary decision diagrams and techniques from symbolic model checking. We use formulas and temporal formulas for representing the transition relation $R_n(P)$ of each node n of the EPC, and we will show how these formulas can be used for efficiently calculating the semantics of the underlying EPC.

3.1 Representing $R_n(P)$ by formulas

Let us start with the formula for 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 introduce the primed version i' and $o_1', \ldots, o_n'$ for each arc, 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 (cf. Fig. 2 (c)):

$$i \wedge \neg o_1 \wedge \ldots \wedge \neg o_n \quad \wedge \quad \neg i' \wedge o_1' \wedge \ldots \wedge o_n'$$

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 Figure 2 (c) (as formalised in [Kind04b]), where we assume that variables not occurring in the formula do not change.

Altogether, we can apply this standard technique [ClGP99, HuRy00] 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 (cf. Fig. 2):

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

$$i \wedge \neg o \quad \wedge \quad \neg i' \wedge o'.$$

c.: For $n = c \in C$ with $l(c) = and$, $c_{in} = \{i\}$, and $c_{out} = \{o_1, o_2\}$,

$$i \wedge \neg o_1 \wedge \neg o_2 \quad \wedge \quad \neg i' \wedge o_1' \wedge o_2'.$$

d.: For $n = c \in C$ with $l(c) = and$, $c_{in} = \{i_1, i_2\}$, and $c_{out} = \{o\}$, the formula for $R_n(P)$ is

$$i_1 \wedge i_2 \wedge \neg o \quad \wedge \quad \neg i_1' \wedge \neg i_2' \wedge o'.$$

e.: For $n = c \in C$ with $l(c) = or$, $c_{in} = \{i\}$, 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 $n = c \in C$ with $l(c) = xor$, $c_{in} = \{i\}$, 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-split and the XOR-split connectors are a bit more involved. For the OR-split connector (cf. Fig. 2 (e)), it is required that no outgoing arcs has less folders than before and at least one has more. Since we do not consider multiple folders, this constraint can be formulated in terms of an 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. Fig. 2 (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 can the formulas for the non-local operators be formalised? 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[2] P. Since we use very simple temporal formulas only, we do not introduce temporal logic in full detail, here. The only temporal operator needed for now is the CTL operator EF (see [ClGP99, HuRy00] for details): For some formula φ, the temporal formula EF φ is *true* in exactly those states from which a state can be reached (with respect to P) in which φ is valid[3]. This way, we can express that no folder can arrive on some arc i by the formula $\neg \mathrm{EF}\, i$.

With this temporal formula, it is easy to express the behaviour of the XOR-join connector: For an XOR-

[2] Technically, P is considered as a Kripke structure on which the temporal formula is interpreted.

[3] The temporal operator EF can be read „there Exists a Future".

join connector with two ingoing arcs i_1 and i_2 and one outgoing arc o, the formula

$$((i_1 \wedge \neg \mathrm{EF}\, i_2) \vee (\neg \mathrm{EF}\, i_1 \wedge i_2)) \wedge \neg o \wedge \neg i_1' \wedge \neg i_2' \wedge o'$$

precisely captures its behaviour. The formulas $\neg \mathrm{EF}\, i_1$ and $\neg \mathrm{EF}\, i_2$ guarantee that a transition does occur only when no folder can arrive from the respective other arc.

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

f.: For $n = c \in C$ with $l(c) = or$, $c_{in} = \{i_1, i_2\}$, and $c_{out} = \{o\}$, the formula for $R_n(P)$ is

$$((i_1 \wedge i_2) \vee (i_1 \wedge \neg \mathrm{EF}\, i_2) \vee (\neg \mathrm{EF}\, i_1 \wedge i_2)) \wedge \neg o \wedge \neg i_1' \wedge \neg i_2' \wedge o'.$$

h.: For $n = c \in C$ with $l(c) = xor$, $c_{in} = \{i_1, i_2\}$, and $c_{out} = \{o\}$, the formula for $R_n(P)$ is

$$((i_1 \wedge \neg \mathrm{EF}\, i_2) \vee (\neg \mathrm{EF}\, i_1 \wedge i_2)) \wedge \neg o \wedge \neg i_1' \wedge \neg i_2' \wedge o'.$$

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, there is no harm in mixing primed variables and temporal operators in a single formula. Such a formula defines a new transition relation based on a given transition relation, which is exactly what we need for calculating $R_n(P)$.

3.2 Computing 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 for the local nodes, and $h_1, \ldots, h_k$ are the formulas representing the transition relations 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*[4] (ROBDD), which has 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 [ClGP99, HuRy00]. We denote this procedure by `f.toROBDD()`, which is close to the corresponding methods of our object oriented model checker MCiE [Kind04a].

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)` calculates the resulting ROBDD.

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

```
Pnext:={ g1.toROBDD(),..., gj.toROBDD()};
for i:= 1 to k do
  Pnext:=  Pnext.add(hi.toROBDD(Pcurr));
```

In the first line, we insert all the transitions of the local nodes to P_{next}; in the loop, we add the transition relation for each non-local node to P_{next}. To be precise, the calculation is a bit more involved: In order to exactly capture the semantics formalised in [Kind04b], 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 execute the fixed-point iteration discussed in Section 2: We start with $P_0 = \varnothing$ and $Q_0 = \Sigma \times \Sigma$ and the iteratively calculate $P_{i+1} = R(Q_i)$ 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, ... in a zig-zag way. As stated before,

[4] 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.

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.

The pseudo code for the resulting algorithm is shown below:

```
P_next:={ false }; // P_0
P_curr:={ 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 P_{curr} and P_{next} contain the two transition relations for the EPC. The question, however, is which of them is the pessimistic transition relation and which of them is the optimistic transition relation. In order to decide this, we use the `step` counter. If it is odd, P_{curr} is a P_i relation and thus represents the pessimistic transition relation and P_{next} is the optimistic transition relation; if the `step` counter is even, P_{curr} is a Q_i relation and, thus, represents the optimistic transition relation and P_{next} is the pessimistic transition relation.

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_n for the transition relation of node n separately, checking the enabledness is simple. Let `enabled` be the CTL formula $\mathrm{EX}\ true$, which is valid in all states for which the underlying transition relation has a successor. Then `enabled.toROBDD(`P_n`)` calculates 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 can be calculated locally.

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 immediately 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 optimisations, it worked reasonably well on small EPCs. For calculating the semantics for larger EPCs, however, we had to come up with some optimisations, which will be discussed below.

4 Optimisations

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

The first kind exploits 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 optimisations have been investigated in [Cunt04]. 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 optimisations impossible—except for very trivial ones.

The second kind is a smart application and combination of optimisation 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 optimisations, 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 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.

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, i.e. in the number of 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 Figure 3 (labelled *Event* and *Function*) can be omitted when computing the enabledness of the XOR-join connectors. We call this optimisation *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 could influence 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 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 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 case, 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 simulating 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 could be applied. If the condition is not met, we must switch back to calculating the semantics of the original EPC, which of course is less efficient.

If both conditions 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 eliminated only event and function nodes, those connectors are still contained in the reduced EPC.

The main disadvantage of the chain elimination approach is that it cannot be applied for arbitrary EPCs and that its applicability depends on the current state. 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

Another idea for simplifying the simulation problem was to identify some restricted classes of EPCs for which no fixed-point iteration would be necessary. For example, we considered EPCs without cycles on non-local nodes such as the ones shown in Figure 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 spoilt this approach. A detailed discussion of these negative results can be found in [Cunt04].

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: optimisation of the variable order and partitioning of the transition relation.

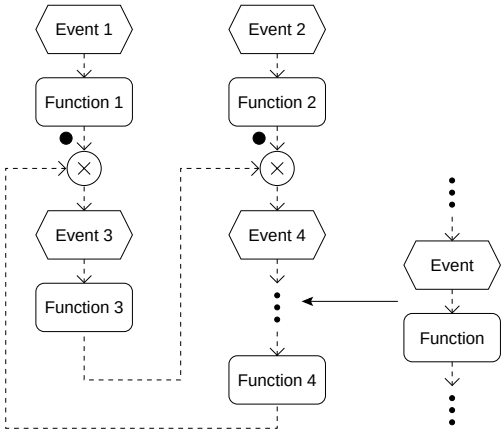

Figure 3: The example used for the measurements in Figure 4

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 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, which needs some further investigation.

Partitioning 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 optimisation* resulted in significantly better computation times.

4.3 Measurements

In order to illustrate the benefits of the above optimisations, Figure 4 shows the computation times for calculating the semantics of the example of Figure 3 for the different optimisation 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 EPC, the y-axis shows the computation times for the different optimisations.

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 optimisation for unchanged variables in the transition relations. The fourth graph shows the time with an optimised 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 dramatically 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).

Once the transition relation was computed, the simulation itself could be done in virtually no time.

The above figures come from a technical example. In order to get some experience with real world

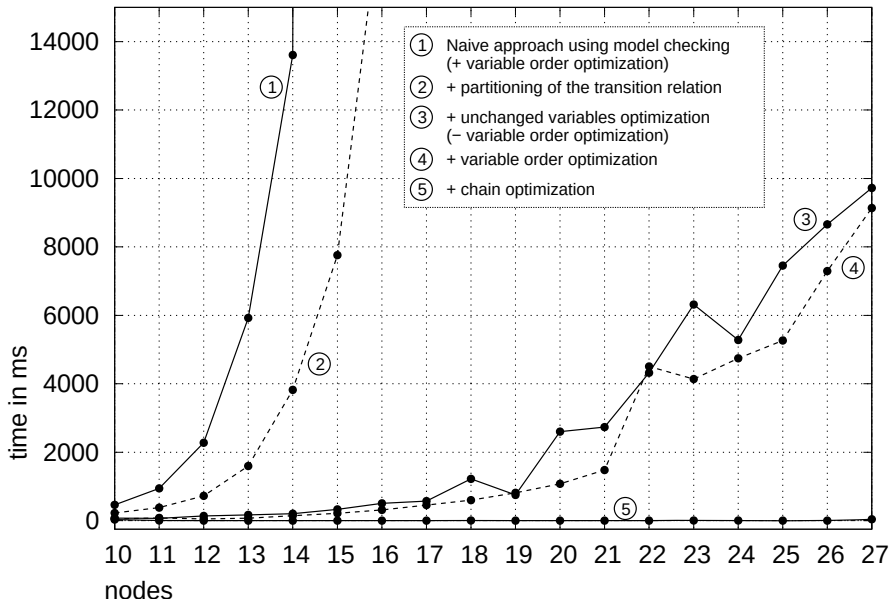

Figure 4: Benefit of the implemented optimisation techniques (compare to Figure 3)

examples, we have done some experiments with some EPC models from the SAP reference processes of the ARIS Toolset[5]:

For EPCs with 20 to 30 nodes the semantics was calculated within milliseconds to about 4 seconds, depending on the structure of the EPC. The chosen EPCs examples and the precise figures can be found in [CuKi04b].

5 EPC Tools

The algorithms for calculating the semantics of an EPC and for simulating an EPC based on this semantics is integrated into an Eclipse [Ecli] based tool, which we call *EPC Tools*. EPC Tools is open source and can be down-loaded from the EPC Tools web site [CuKi04a].

Figure 5 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* [MeNü04a], and there are converters between the AML format of the ARIS Toolset and EPML [MeNü04b].

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. 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 [AaHe02] or we could check properties by applying model checking. This should take less time than calculating the semantics.

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

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

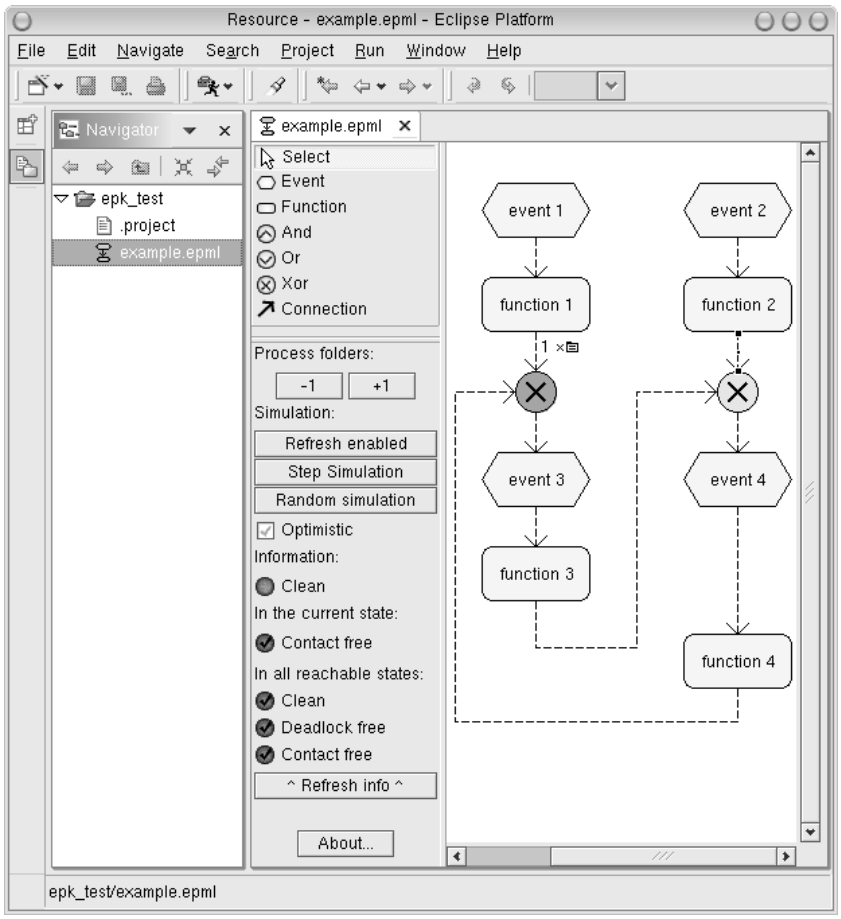

Figure 5: EPC Tools in the Eclipse environment

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 (cf. Fig. 5). In addition, EPC Tools provides a print function and allows a user to zoom into and out of EPC diagrams by using the standard Eclipse toolbar and the main menu.

The simulator functions are located in the lower part of the panel in the middle. It is possible to highlight all currently enabled nodes (button "refresh enabled"), and then to simulate one step by specifying a node or by randomly choosing a node (button "step simulation"). The randomised simulation (button "random 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. In the same panel, there are some LEDs corresponding to the properties of the EPC. This information is updated by pushing the "refresh info" 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 optimisations, the simulation of medium size EPCs works quite well and is practically feasible—even when taking into account the non-

local semantics of the XOR-join and the OR-join connector. Moreover, it is quite easy to adapt this algorithm to slightly different semantics of EPCs (see [Kind04b] for some alternatives): We need to change only the temporal formulas for defining the transition relations of the different types of nodes of EPCs.

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 downloaded from [CuKi04a].

References

[AaDK02] van der Aalst, W.; Desel, J.; Kindler, E.: On the semantics of EPCs: A vicious circle. In: Nüttgens, M.; Rump, F. J. (eds.): EPK 2002, Geschäftsprozess-management mit Ereignisgesteuerten Prozessketten, November 2002, pp. 71–79.

[AaHe02] van der Aalst, W.; 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.; Hwang, L.: Symbolic model checking: 10^{20} states and beyond. Information and Computation 98 (1992), pp.142–170.

[Brya86] Bryant, R. E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers 35 (1986), pp. 677–691.

[ClGP99] Clarke, E.; Grumberg, O.; Peled, D.: Model checking. MIT Press, 1999.

[Cunt04] Cuntz, N.: Über die effiziente Simulation von Ereignisgesteuerten Prozessketten. Master's thesis. University of Paderborn, Department of Computer Science, June 2004.

[CuKi04a] Cuntz, N.; Kindler, E.: The EPC Tools Project. http://www.upb.de/cs/kindler/research/EPCTools, 2004.

[CuKi04b] Cuntz, N.; Kindler, E.: On the semantics of EPCs: Efficient Calculation and Simulation. In: Nüttgens, M.; Rump, F.~J. (eds.): EPK 2004, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, October 2004, pp. 7–26.

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

[HuRy00] Huth, M.; Ryan, M.: Logic in Computer Science: Modelling and reasoning about systems. Cambridge University Press, 2000.

[Kind04a] Kindler, E.: The Model Checking in Education (MCiE) Project. http://www.upb.de/cs/kindler/teaching/MCiE, 2004.

[Kind04b] Kindler, E.: On the semantics of EPCs: Resolving the vicious circle. In: Desel, J.; Pernici, B.; Weske, M. (eds.): Business Process Management, Second International Conference, BPM 2004, June 2004. Springer, LNCS 3080, pp. 82–97.

[KeNS92] Keller, G.; Nüttgens, M.; 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.

[LaSW98] Langner, P.; Schneider, C.; Wehler, J.: Petri Net Based Certification of Event driven Process Chains. In: Desel, J.; Silva, M. (eds.): Application and Theory of Petri Nets 1998, June 1998. Springer. LNCS 1420, pp. 286–305.

[McMi93] McMillan, K. L.: Symbolic Model Checking. Kluwer Academic Publishers, 1993.

[MeNü04a] Mendling, J.; Nüttgens, M.: Exchanging EPC Business Process Models with EPML. In: Mendling, J.; Nüttgens, M. (eds.): XML interchange formats for business process management, Proceedings of the 1st Workshop XML4BPM, March 2004, pp. 61–80.

[MeNü04b] Mendling, J. and Nüttgens, M.: Transformation of ARIS Markup Language to EPML. In: Nüttgens, M.; Rump, F. J. (eds.): EPK~2004, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, October 2004, pp. 27–38.

[NüRu02] Nüttgens, M. and Rump, F. J.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In: PROMISE 2002, Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen. GI Lecture Notes in Informatics, 2002, P-21, pp. 64–77.

[Ritt00] Rittgen, P.: Quo vadis EPK in ARIS? Wirtschaftsinformatik 42 (2002):pp. 27–35.

Nicolas Cuntz

Computer Graphics and Multimedia Systems Group
University of Siegen
Siegen, Germany
nicolas.cuntz@uni-siegen.de

Ekkart Kindler

Software Engineering Group
University of Paderborn
Paderborn, Germany
kindler@upb.de