Formalization and Verification of EPCs with OR-Joins Based on State and Context

Jan Mendling[1] and Wil van der Aalst[2]

[1] Vienna University of Economics and Business Administration
Augasse 2-6, 1090 Vienna, Austria
jan.mendling@wu-wien.ac.at
[2] Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
w.m.p.v.d.aalst@tue.nl

Abstract. The semantics of the OR-join in business process modeling languages like EPCs or YAWL have been discussed for a while. Still, the existing solutions suffer from at least one of two major problems. First, several formalizations depend upon restrictions of the EPC to a subset. Second, several approaches contradict the modeling intuition since the structuredness of the process does not guarantee soundness. In this paper, we present a novel semantical definition of EPCs that addresses these aspects yielding a formalization that is applicable for all EPCs and for which structuredness is a sufficient condition for soundness. Furthermore, we introduce a set of reduction rules for the verification of an EPC-specific soundness criterion and present a respective implementation.

1 Introduction

The Event-driven Process Chain (EPC) is a business process modeling language for the represention of temporal and logical dependencies of activities in a business process (see [1]). EPCs offer *function type* elements to capture the activities of a process and *event type* elements describing pre- and post-conditions of functions. Furthermore, there are three kinds of *connector types* (i.e. AND, OR, and XOR) for the definition of complex routing rules. Connectors have either multiple incoming and one outgoing arc (join connectors) or one incoming and multiple outgoing arcs (split connectors). As a syntax rule, functions and events have to alternate, either directly or indirectly when they are linked via one or more connectors. *Control flow arcs* are used to link elements.

The informal (or intended) semantics of an EPC can be described as follows. The AND-split activates all subsequent branches in a concurrent fashion. The XOR-split represents a choice between exclusive alternative branches. The OR-split triggers one, two or up to all of multiple branches based on conditions. In both cases of the XOR- and OR-split, the activation conditions are given in events subsequent to the connector. Accordingly, splits from events to functions are forbidden with XOR and OR since the activation conditions do not become clear in the model. The AND-join waits for all incoming branches to complete,

J. Krogstie, A.L. Opdahl, and G. Sindre (Eds.): CAiSE 2007, LNCS 4495, pp. 439–453, 2007.

then it propagates control to the subsequent EPC element. The XOR-join merges alternative branches. The *OR-join* synchronizes all active incoming branches, i.e., it needs to know whether the incoming branches may receive tokens in the future. This feature is called *non-locality* since the state of all (transitive) predecessor nodes has to be considered.

Since the informal description cannot be directly translated into proper semantics (see [2]), EPCs arguably belong to those process modeling languages for which state based correctness criteria such as soundness are not directly applicable. Instead, several authors have proposed to consider structuredness of the process graph as an alternative criterion for correctness (see e.g. [3,4,5]). Essentially, in a structured process model each split connector matches a join connector of the same type and loops have one XOR-join as entry and one XOR-split as exit point. These building blocks can be nested and extended with sequences of functions and events. The structuredness of a process model can be tested by repeatedly applying reduction rules that collapse several nodes of the respective building blocks. If the reduction yields a single node for the whole process, the model is structured. While structuredness represents a sufficient condition for soundness of Petri nets (see [6,7]), the application of reduction rules to EPCs such as proposed in [5] rather represents a heuristic. Figure 1 gives an example of a structured EPC that can be reduced to a single node by first collapsing the two OR-blocks, the AND-block, and then the loop. The EPC of Figure 2 extends this model with two additional start events $e4$ and $e5$. Due to the introduction of the OR-joins $c9$ and $c10$, there are only two structured blocks left between $c3$ and $c4$ and between $c5$ and $c6$. Still, if we assume that the start event $e1$ is always triggered, there is no problem to execute this unstructured EPC. If the start event $e4$ is triggered, it will synchronize with the first loop entry at $c1$.

Against this background, we present a novel EPC semantics definition that has the following qualities. First, it is applicable for all EPCs that are syntactically

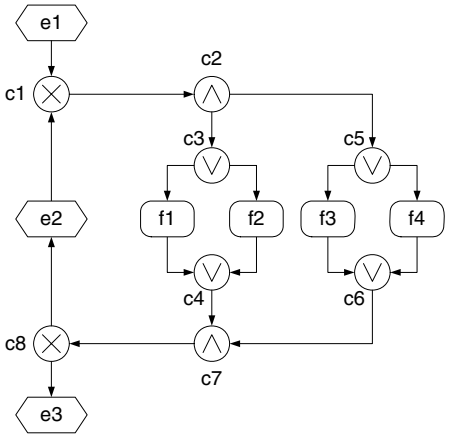

Fig. 1. A structured EPC with two OR-blocks $c2 - c5$ and $c3 - c4$ on a loop

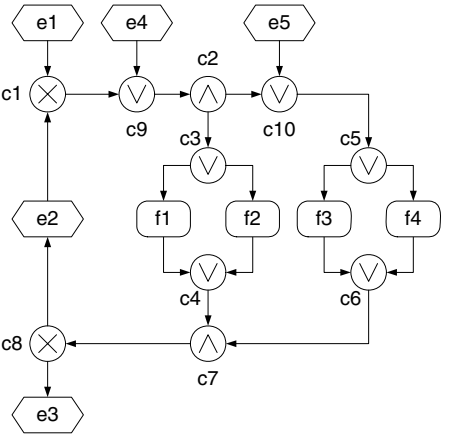

Fig. 2. An unstructured EPC with one OR-block $c3 - c4$ and an OR loop entry

correct while several existing proposals restrict themselves only to a subset. Second, for this new semantics structuredness is a sufficient condition for soundness. This aspect is of central importance both for the intuition of the semantics and for the efficient verification based on reduction rules. The remainder of this paper is structured as follows. In Section 2 we use the EPCs of Figures 1 and 2 as a running example to discuss related work on process modeling languages with OR-joins, i.e. EPCs and YAWL, in particular. This discussion reveals that none of the existing formalizations captures both the complete set of syntactically correct EPCs and at the same time supports the intuition that structured models are sound. In Section 3 we give an EPC syntax definition and present our novel EPC semantics definition based on state and context. In Section 4 we elaborate on the relationship of structuredness and soundness showing that a structured EPC is indeed sound according to the new semantics. This is an important result that makes the semantics also a candidate for the formalization of YAWL nets without cancellation areas. Section 5 concludes the paper and gives an outlook on future research.

2 Related Research

The transformation to Petri nets plays an important role in early EPC formalizations. A problem of these approaches is their restriction to a subset of EPCs. The first concept is presented by *Chen and Scheer* [8] who define a mapping of structured EPCs with OR-blocks to colored Petri nets. A similar proposal is repeated by *Rittgen* [9]. Yet, while these first Petri net semantics provide a formalization for structured EPCs such as in Figure 1, it does not provide semantics for OR-joins in unstructured EPCs.

The transformation approach by *Langner, Schneider, and Wehler* [10] maps EPCs to Boolean nets, a variant of colored Petri nets whose token colors are 0 (negative token) and 1 (positive token). A connector propagates both negative and positive tokens according to its logical type. This mechanism is able to capture the non-local synchronization semantics of the OR-join similar to dead-path elimination (see [11]). A drawback is that the EPC syntax has to be restricted: arbitrary structures are not allowed. If there is a loop it must have an XOR-join as entry point and an XOR-split as exit point which are both mapped to one place in the resulting Boolean net. As a consequence, this approach does not provide semantics for the unstructured EPC in Figure 2.

Van der Aalst [12] presents an approach to derive Petri nets from EPCs. While this mapping provides clear semantics for XOR- and AND-connectors as well as for the OR-split, it does not cover the OR-join. Dehnert presents an extension of this approach by mapping the OR-join to a Petri net block [13]. Since the resulting Petri net block may not necessarily synchronize multiple tokens at runtime (i.e., a non-deterministic choice), its state space is larger than the actual state space with synchronization. Based on the so-called relaxed soundness criterion it is possible to check whether a join should synchronize (cf. [13]).

Nüttgens and Rump [14] define a transition relation for EPCs that addresses also the non-local semantics of the OR-join, yet with a problem: the transition

relation for the OR-join refers to itself under negation. *Van der Aalst, Desel, and Kindler* show, that a fixed point for this transition relation does not always exist [15]. They present an example to prove the opposite: an EPC with two OR-joins on a circle waiting for each other. This vicious circle is the starting point for the work of *Kindler* towards a sound mathematical framework for the definition of non-local semantics for EPCs [2]. The technical problem is that for the OR-join transition relation R depends upon R itself in negation. Instead of defining one transition relation, he considers a pair of transition relations (P, Q) on the state space Σ of an EPC and a monotonously decreasing function R. Then, a function $\varphi((P,Q)) = (R(Q), R(P))$ has a least and a greatest fixed point. P is called pessimistic transition relation and Q optimistic transition relation. An EPC is called *clean*, if $P = Q$. For most EPCs, this is the case. Some EPCs such as the vicious circle EPC are *unclean* since the pessimistic and the optimistic semantics do not coincide. The EPC of Figure 2 belongs to the class of unclean EPCs.

Van der Aalst and Ter Hofstede define a workflow language called YAWL [16] which also offers an OR-join with non-local semantics. The authors propose a definition of the transition relation $R(P)$ with a reference to a second transition relation P that ignores all OR-joins. A similar semantics that is calculated on history-logs of the process is proposed by *Van Hee et al.* in [17]. *Mendling, Moser, and Neumann* relate EPCs to YAWL by the help of a transformation [18]. Even though this definition provides semantics for the full set of models, it yields a deadlock if the OR-joins $c4$ and $c6$ are activated. In cases of chained OR-joins, there might be a lack of synchronization (see [19]). Motivated by these problems *Wynn et al.*, present a novel approach based on a mapping to Reset nets. Whether an OR-join can fire (i.e. $R(P)$) is decided depending on (a) a corresponding Reset net (i.e. P) that treats all OR-joins as XOR-joins and (b) a predicate called *superM* that hinders firing if an OR-join is on a directed path from another enabled OR-join. In particular, the Reset net is evaluated using backward search techniques that grant coverability to be decidable (see [21,22]). A respective verification approach for YAWL nets is presented in [23]. The approach based on Reset nets provides interesting semantics but in some cases also leads to deadlocks, e.g. if the OR-joins $c4$ and $c6$ are activated.

Table 1 summarizes existing work on the formalization of the OR-join. Several early approaches define syntactical restrictions such as OR-splits to match corresponding OR-joins or models to be acyclic (see [8,10]). Newer approaches impose little or even no restrictions (see [2,16,23]), but exhibit unexpected behavior for OR-block refinements on loops with further OR-joins on it.

In the following section, we propose a novel semantics definition that provides soundness for structured EPCs without restricting the set of models based on the concepts reported in [19]. For verification we follow a reduction rules approach similar to the one proposed in *Sadiq & Orlowska* [3]. Unfortunately, the verification algorithm presented in [3] turned out to be incorrect since the set of reduction rules provided was shown to be incomplete [24,7]. In [24] there was an attempt to repair this by adding additional reduction rules. In [7] it was shown that the considered class of process models coincides with the well-known class

Table 1. Overview of OR-join semantics and their limitations

OR-join semantics	Restricted to	Correctness of structured models
Chen et al. [8]	structured EPCs	correct
Langner et al. [10]	structured EPCs	correct
Kindler [2]	clean EPCs	correct (no proof available)
van der Aalst et al. [16]	no restriction	potential deadlock, lack of synchronization
Wynn et al. [23]	no restriction	potential deadlock

of free-choice nets for which a compact and complete set of reduction rules exist [6]. Moreover, using the well-known Rank Theorem for free-choice nets it is possible to find any errors in polynomial time for the class of workflow considered in [3] extended with loops. A set of reduction rules for EPCs was first mentioned in *van Dongen, van der Aalst, and Verbeek* [5]. Still, their reduction rules are not related to a semantics definition of EPCs, but rather given as heuristics. In this paper, we extend this work by relating reduction rules to EPC soundness and provide specific rules to deal with multiple start and end events.

3 EPC Syntax and Semantics

3.1 EPC Syntax

There is not only one, but there are several approaches towards the formalization of EPC syntax because the original paper introduces them only in an informal way (see [1]). The subsequent syntax definition of EPCs is an abbreviation of a more elaborate definition given in [19] that consolidates prior work.

Definition 1 (EPC Syntax). A flat $EPC = (E, F, C, l, A)$ consists of four pairwise disjoint and finite sets E, F, 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

- An element of E is called *event*. $E \neq \emptyset$.
- An element of F is called *function*. $F \neq \emptyset$.
- An element of C is called *connector*.
- The mapping l specifies the type of a connector $c \in C$ as *and*, *or*, or *xor*.
- A defines the control flow as a coherent, directed graph. An element of A is called an *arc*. An element of the union $N = E \cup F \cup C$ is called a *node*.

In order to allow for a more concise characterization of EPCs, notations are introduced for incoming and outgoing arcs, paths, and several subsets.

Definition 2 (Incoming and Outgoing Arcs, Path). Let N be a set of *nodes* and $A \subseteq N \times N$ a binary relation over N defining the arcs. For each *node* $n \in N$, we define the set of incoming arcs $n_{in} = \{(x, n) | x \in N \wedge (x, n) \in A\}$, and the set of outgoing arcs $n_{out} = \{(n, y) | y \in N \wedge (n, y) \in A\}$. A *path* $a \hookrightarrow b$ refers to a sequence of nodes $n_1, \ldots, n_k \in N$ with $a = n_1$ and $b = n_k$ such that for all $i \in 1, \ldots, k$ holds: $(n_1, n_2), \ldots, (n_i, n_{i+1}), \ldots, (n_{k-1}, n_k) \in A$. This includes the empty path of length zero, i.e., for any node $a : a \hookrightarrow a$.

Definition 3 (Subsets). For an EPC, we define the following subsets of its nodes and arcs:

- $E_s = \{e \in E \mid |e_{in}| = 0\}$ being the set of start-events,
  $E_{int} = \{e \in E \mid |e_{in}| = 1 \wedge |e_{out}| = 1\}$ being the set of intermediate-events,
  $E_e = \{e \in E \mid |e_{out}| = 0\}$ being the set of end-events.
- $A_s \subseteq \{(x, y) \in A \mid x \in E_s\}$ as the set of start-arcs,
  $A_{int} \subseteq \{(x, y) \in A \mid x \notin E_s \wedge y \notin E_e\}$ as the set of intermediate-arcs, and
  $A_e \subseteq \{(x, y) \in A \mid y \in E_e\}$ as the set of end-arcs.

In contrast to other approaches, we assume only a very limited set of constraints for a EPC to be correct. For an extensive set of constraints see e.g. [19].

Definition 4 (Syntactically Correct EPC). An $EPC = (E, F, C, l, A)$ is called syntactically correct, if it fulfills the requirements:

1. EPC is a directed and coherent graph such that $\forall n \in N : \exists e_1 \in E_s, e_2 \in E_e$ such that $e_1 \hookrightarrow n \hookrightarrow e_e$
2. $|E| \geq 2$. There are at least two events in an EPC.
3. Events have at most one incoming and one outgoing arc.
   $\forall e \in E : |e_{in}| \leq 1 \wedge |e_{out}| \leq 1$.
4. Functions have exactly one incoming and one outgoing arcs.
   $\forall f \in F : |f_{in}| = 1 \wedge |f_{out}| = 1$.
5. Connectors have one incoming and multiple outgoing arcs or multiple incoming and one outgoing arc. $\forall c \in C : (|c_{in}| = 1 \wedge |c_{out}| > 1) \vee (|c_{in}| > 1 \wedge |c_{out}| = 1)$. If a connector does not have multiple incoming or multiple outgoing arcs, it is treated as if it was an event.

3.2 EPC Semantics Based on State and Context

In this subsection, we introduce a novel formalization of the EPC semantics. The principal idea of these semantics lends some concepts from *Langner, Schneider, and Wehler* [10] and adapts the idea of Boolean nets with true and false tokens in an appropriate manner. The reachability graph that we will formalize afterwards depends on the state and the context of an EPC. The *state* of an EPC is basically an assignment of positive and negative tokens to the arcs. Positive tokens signal which functions have to be carried out in the process, negative tokens indicate which functions are to be ignored. In order to signal OR-joins that it is not possible to have a positive token on an incoming branch, we define the *context* of an EPC. The context assigns a status of *wait* or *dead* to each arc of an EPC. A wait context indicates that it is still possible that a positive token might arrive; a dead context status means that no positive token can arrive anymore. For example, XOR-splits produce a dead context on those output branches that are not taken and a wait context on the output branch that receives a positive token. A dead context at an input arc is then used by an OR-join to determine whether it has to synchronize with further positive tokens or not.

Definition 5 (State and Context). For an $EPC = (E, F, C, l, A)$ the mapping $\sigma : A \rightarrow \{-1, 0, +1\}$ is called a state of an EPC. The positive token captures the state as it is observed from outside the process. It is represented by a black circle. The negative token depicted by a white circle with a minus on it has a similar semantics as the negative token in the Boolean nets formalization. Arcs with no state tokens on them have no circle depicted. Furthermore, the mapping $\kappa : A \rightarrow \{wait, dead\}$ is called a context of an EPC. A wait context is represented by a w and a dead context by a d next to the arc.

In contrast to Petri nets we distinguish the terms marking and state: the term marking refers to state σ and context κ collectively.

Definition 6 (Marking of an EPC). For a syntactically correct EPC the mapping $m : A \rightarrow \{-1, 0, +1\} \times \{wait, dead\}$ is called a marking. The set of all markings M of an EPC is called marking space with $M = A \times \{-1, 0, +1\} \times \{wait, dead\}$. The projection of a given marking m to a subset of arcs $S \subseteq A$ is referred to as m_S. If we refer to the κ- or the σ-part of m, we write κ_m and σ_m, respectively, i.e. $m(a) = (\sigma_m(a), \kappa_m(a))$.

The propagation of context status and state tokens is arranged in a four phase cycle: (1) dead context, (2) wait context, (3) negative token, and (4) positive token propagation. Whether a node is enabled and how it fires is illustrated in Figure 3. A formalization of the transitions for each phase is presented in [25].

1. In the first phase, all *dead context* information is propagated in the EPC until no new dead context can be derived.
2. Then, all *wait context* information is propagated until no new wait context can be derived. It is necessary to have two phases (i.e., first the dead context propagation and then the wait context propagation) in order to avoid infinite cycles of context changes (see [25]).
3. After that, all *negative tokens* are propagated until no negative token can be propagated anymore. This phase cannot run into an endless loop (see [25]).
4. Finally, one of the enabled nodes is selected and propagates *positive tokens* leading to a new iteration of the four phase cycle.

In order to set the start and the end point of the four phases, we define the initial and the final marking of an EPC similar to the definition in *Rump* [26].

Definition 7 (Initial Marking of an EPC). For an EPC $I \subseteq M$ is defined as the set of all possible initial markings, i.e. $m \in I$ if and only if [1]:

- $\exists a_s \in A_s : \sigma_m(a_s) = +1$,
- $\forall a_s \in A_s$: $\sigma_m(a_s) \in \{-1, +1\}$,
- $\forall a_s \in A_s$: $\kappa_m(a_s) = wait$ if $\sigma_m(a_s) = +1$ and $\kappa_m(a_s) = dead$ if $\sigma_m(a_s) = -1$, and
- $\forall a \in A_{int} \cup A_e : \kappa_m(a) = wait$ and $\sigma_m(a) = 0$.

[1] Note that the marking is given in terms of arcs.

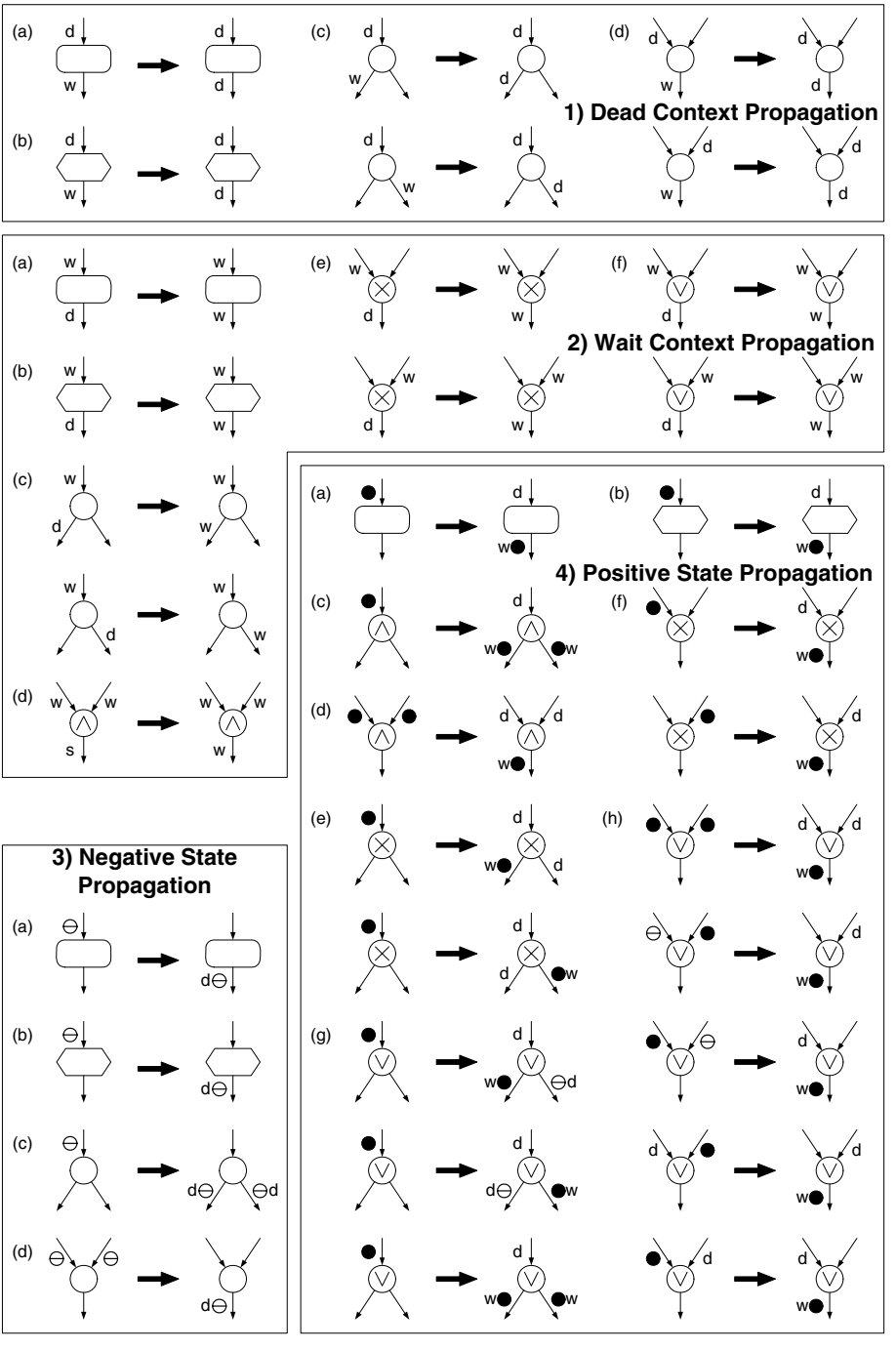

Fig. 3. Propagation of context and state in four phases

Definition 8 (Final Marking of an EPC). For an *EPC* $O \subseteq M$ is defined as the set of all possible final markings, i.e. $m \in O$ if and only if:

- $\exists a_e \in A_e$: $\sigma_m(a_e) = +1$ and
- $\forall a \in A_{int} \cup A_s : \sigma_m(a) \leq 0$.

Initial and final markings are the start and end points for calculating the reachability graph of an EPC. In this context a marking m' is called reachable[2] from another marking m if and only if after applying the phases of dead and wait context and negative token propagation on m, there exists a node n whose firing in the positive token propagation phase produces m'. Then, we write $m \xrightarrow{n} m'$, or only $m \rightarrow m'$ if there exists some node n such that $m \xrightarrow{n} m'$. Furthermore, we write $m_1 \xrightarrow{\tau} m_q$ if there is a firing sequence $\tau = n_1 n_2 ... n_{q-1}$ that produces from marking m_1 the new marking m_q with $m_1 \xrightarrow{n_1} m_2, m_2 \xrightarrow{n_2} ... \xrightarrow{n_{q-1}} m_q$. If there exists a sequence τ exists such that $m_1 \xrightarrow{\tau} m_q$, we write $m_1 \xrightarrow{*} m_q$. Accordingly, we define the reachability graph *RG* as follows.

Definition 9 (Reachability Graph of an EPC). $RG \subseteq M_{RG} \rightarrow N \times M_{RG}$ is called the reachability graph of an EPC if and only if:

(i) $\forall i \in I : i \in M_{RG}$.
(ii) $\forall m, m' \in RG : (m, n, m') \Leftrightarrow m \xrightarrow{n} m'$.

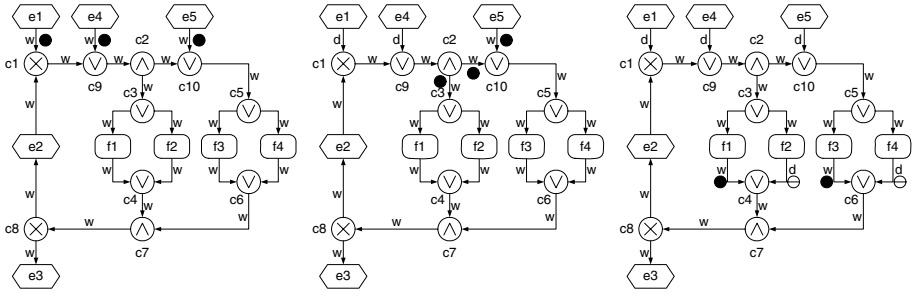

Fig. 4. Applying the Transition Relations

Based on the previous definitions we can discuss the behavior of the unstructured example EPC of Figure 2. This EPC and three markings are depicted in Figure 4. The *first marking* shows the example EPC in an initial marking with all start arcs carrying a positive token represented by a black circle. In this marking only the XOR-join $c1$ is allowed to fire – the other OR-joins have a *wait* context on one of their incoming arcs; therefore, they are not allowed to fire. In the *second marking* a token is propagated from $c1$, via synchronizing with the second token at $c9$, to the AND-split $c2$. The context of the start arcs has changed to *dead*, but

[2] A formalization of *reachability* is given in [25].

the arcs between the connectors $c1 - c9$ and $c9 - c2$ are still in *wait* since a token may arrive here via $e2$ on the loop. In order to arrive at the *third marking*, first the connector $c10$ has to fire. After that both OR-splits $c3$ and $c5$ are activated and fire a positive token to the left branch and a negative token to the right branch. After passing functions $f1$ to $f4$ we achieve the current marking with the OR-joins $c4$ and $c6$ being activated since both input arcs carry a token (a positive and a negative). *After this marking*, the two positive tokens generated by $c4$ and $c6$ synchronize at the AND-join $c7$. Then the loop can be run again, or the end arc can be reached. The loop can be executed without a problem since both OR-joins $c9$ and $c10$ have a *dead* context on the arcs coming from the start events. Therefore, the OR-join can fire using the last transition rule of (h) in positive state propagation.

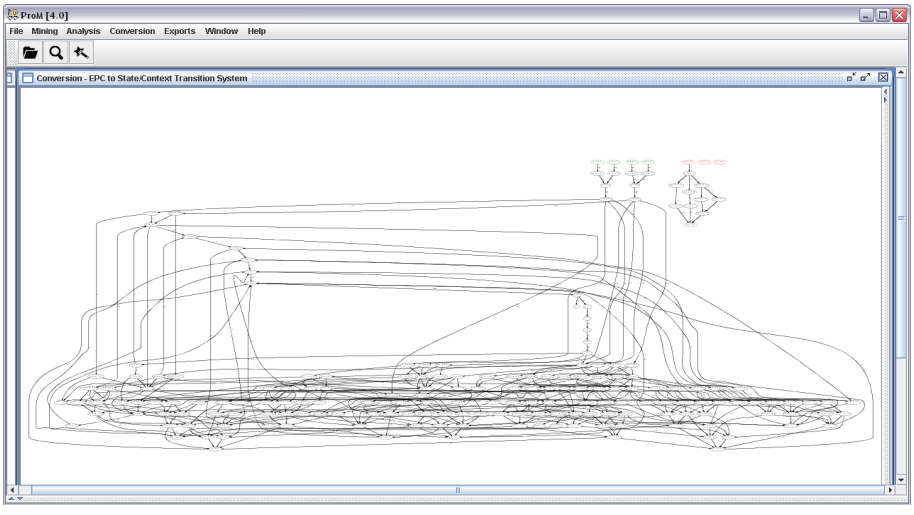

Fig. 5. Reachability Graph for the unstructured example EPC

We have implemented the reachability graph calculation as a conversion plug-in for the ProM framework [27]. Figure 5 displays the reachability graph of the unstructured example EPC that we used to illustrate the behavioral semantics. It can be seen that this graph is already quite complex for a small EPC. The complexity of this example basically stems from three facts. First, there are seven different initial markings. Second, parts of the loop can be executed in concurrency. Third, there are two OR-splits that both can activate either one or the other or both output arcs. Similar to the state explosion in Petri nets, the calculation of the reachability (or coverability) graph can turn out to be very inefficient for verification. Therefore, we discuss an EPC-specific variant of soundness and it verification using reduction rules in the following section.

4 EPC Verification Based on Reduction Rules

Soundness is an important correctness criterion for business process models introduced in [28]. The original soundness property is defined for a Workflow net, a Petri net with one source and one sink, and requires that (i) for every state reachable from the source, there exists a firing sequence to the sink (option to complete); (ii) the state with a token in the sink is the only state reachable from the initial state with at least one token in it (proper completion); and (iii) there are no dead transitions [28]. For EPCs, this definition cannot be used directly since EPCs may have multiple start and end events. Based on the definitions of the initial and final marking of an EPC, we define soundness of an EPC analogously to soundness of Workflow nets.

Definition 10 (Soundness of an EPC). An EPC is sound if there is a set of initial markings I such that:

(i) For each start-arc a_s there exists an initial marking $i \in I$ where the arc (and hence the corresponding start event) holds a positive token. Formally: $\forall a_s \in A_s : \exists i \in I : \sigma_i(a_s) = +1$
(ii) For every marking m reachable from an initial state $i \in I$, there exists a firing sequence leading from marking m to a final marking $o \in O$. Formally: $\forall i \in I : \forall m \in M \ (i \overset{*}{\rightarrow} m) \Rightarrow \exists o \in O \ (m \overset{*}{\rightarrow} o)$
(iii) The final markings $o \in O$ are the only markings reachable from a marking $i \in I$ such that there is no node that can fire. Formally: $\forall m \in M : \nexists m'(m \rightarrow m') \Rightarrow m \in O$

Given this definition, the EPCs of Figures 1 and Figure 2 are sound, and any initial marking of the second must include the state $\sigma_i(e_1, c_1) = +1$ for all $i \in I$.

Related to this soundness definition, we identify a set of reduction rules that is soundness preserving. A *reduction rule* T is a binary relation that transforms a source EPC_1 to a simpler target EPC_2 that has less nodes and/or arcs (cf. e.g. [6]). A reduction rule is bound to a *condition* that defines for which arcs and nodes it is applicable. The reduction rules for sound EPCs include (1) Sequence Elimination, (2) Block Reduction, (3) Simple Loop Reduction, (4) Join Reduction, (5) Split Reduction, (6) Start-Join Reduction, and (7) Split-End Reduction (see Figure 6). Some of these rules (i.e.,1-5) were defined in previous work by [5]. In the following we sketch why these rules are soundness preserving for the given EPC semantics definition.

(1) Sequence Elimination: An element n with one input and one output arc can be eliminated. This rule is applicable for functions and intermediate events, but also connectors with such cardinality can be produced by the other rules. As mentioned before in Def. 4, these connectors are treated as if they were events. The idea for proving that the rule preserves soundness can be sketched as follows. Based on the soundness of the unreduced EPC_1 we have to show that the reduced EPC_2 is also sound. In order to meet (i) we consider $I_2 = I_1$ of EPC_1. For (ii) we consider the node x that enables n, i.e. $m_1 \overset{x}{\rightarrow} m_2$, and the firing of n, i.e.

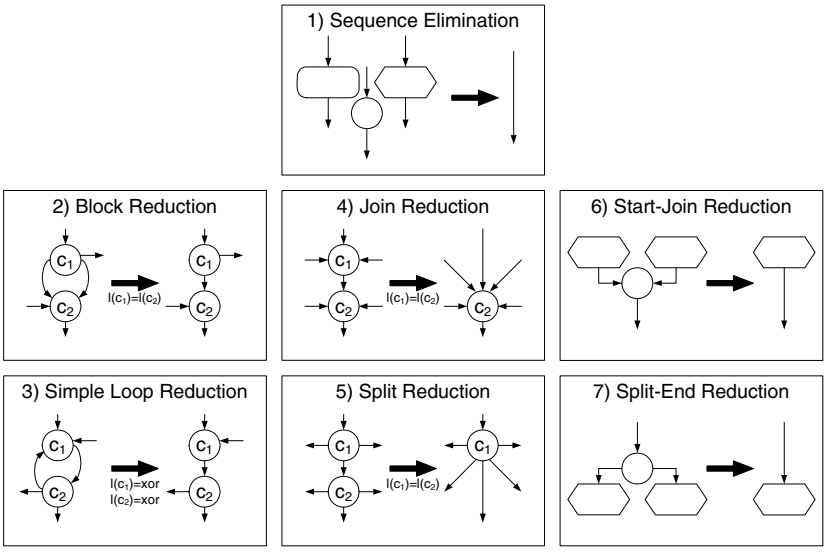

Fig. 6. Soundness preserving reduction rules for EPCs

$m_2 \xrightarrow{n} m_3$ of EPC_1. Obviously, in EPC_2 every marking that corresponds to m_3 is reachable from m_1 by firing x. Therefore, still for all markings that can be reached from some initial marking, some final marking is reachable. Since no new transitions are introduced, the final markings are still the only markings that meet (iii). Therefore, EPC_2 is also sound.

(2) Block Reduction: Multiple arcs from split- to join-connectors of the same type can be fused to a single arc. This might result in connectors with one input and one output arc. The above argument also holds for this reduction, but it must be adapted to cover all states that might be produced by firing c_1.

(3) Simple Loop Reduction: The arc from an XOR-split to an XOR-join can be deleted if there is also an arc from the join to the split. This rule might produce connectors with one input and one output arc. The above argument also holds for this rule.

(4) Join Reduction: Multiple join connectors having the same label are merged to one join. The above argument on soundness can be adapted here.

(5) Split Reduction: Multiple split connectors are reduced to one split. The above argument can be adapted for this rule.

(6) Start-Join Reduction: Multiple start events that are merged to one start event. We replace the two joined start events of EPC_1 in each initial marking by the merged start event such that (i) is met for EPC_2. Since any marking that is reachable by firing the join in EPC_1 is also reachable directly from the start event in EPC_2, but no additional marking is reached, (ii) and (iii) hold respectively for EPC_2. Therefore, EPC_2 is sound.

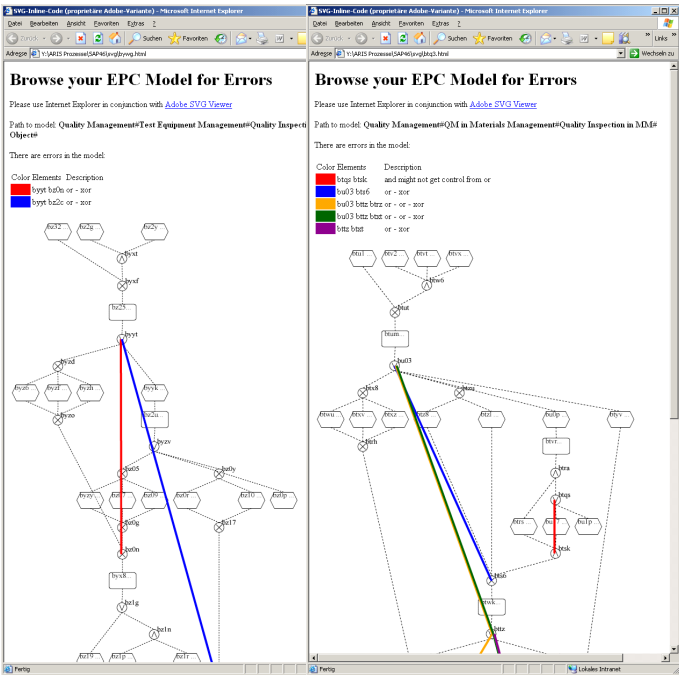

Fig. 7. Examples from the SAP reference model

(7) Split-End Reduction: Splits to multiple end events can be reduced to one end event. The above argument can be adapted for this rule.

Based on these reduction rules, it can be shown that the structured EPC of Figure 1 is indeed sound. Beyond that, we have implemented the reduction rules for EPCs that are available as ARIS XML files. Figure 7 shows two EPCs from the SAP reference model [29]. Both these models were analyzed with an existing verification approach based on the relaxed soundness criterion [13]. Even though they are relaxed sound, they still have structural problems. Using reduction rules we found that the EPCs are not sound according to the definition reported in this paper. In both models there are OR-splits that are joined with an XOR. The website http://wi.wu-wien.ac.at/epc offers an interface to the implementation of the reduction rules. Uploading an ARIS XML file generates an error report such as shown in Figure 7.

5 Contribution and Limitations

In this paper we presented a novel semantics definition for EPCs covering also the behavior of the OR-join. In contrast to existing semantical proposals for business process modeling languages with OR-joins, our definition provides semantics that are (1) applicable for any EPC without imposing a restriction on the

syntax, and (2) intuitive since structuredness of the process model yields sound behavior. This is an important finding because there is up to now no solution reported that covers both aspects (1) and (2) in a formalization of the OR-join. Furthermore, the reduction rules that we presented and their implementation as a web interface are a useful tool for the verification of EPCs. In particular, the start-join and the split-end reduction rule directly address the definition of a soundness notion for EPCs. Moreover, they provide a novel solution for the problem of multiple start and end events in an EPC which is not appropriately covered by existing approaches so far. Still, our approach has a limitation with respect to the completeness of the reduction rules. While for free-choice Petri nets there is a complete set of reduction rules, this completeness is not achieved by the seven rules for EPCs. In future work, we aim to enhance our set by adding further rules in order to provide for an efficient verification of EPC soundness.

References

1. Keller, G., Nüttgens, M., Scheer, A.W.: Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK). Heft 89, Institut für Wirtschaftsinformatik, Saarbrücken, Germany (1992)
2. Kindler, E.: On the semantics of EPCs: Resolving the vicious circle. Data Knowl. Eng. 56, 23–40 (2006)
3. Sadiq, W., Orlowska, M.E.: Applying graph reduction techniques for identifying structural conflicts in process models. In: Jarke, M., Oberweis, A. (eds.) CAiSE 1999. LNCS, vol. 1626, pp. 195–209. Springer, Heidelberg (1999)
4. Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.: On structured workflow modelling. In: Wangler, B., Bergman, L. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 431–445. Springer, Heidelberg (2000)
5. van Dongen, B., van der Aalst, W., Verbeek, H.M.W.: Verification of EPCs: Using reduction rules and petri nets. In: Pastor, Ó., Falcão e Cunha, J.F. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 372–386. Springer, Heidelberg (2005)
6. Esparza, J.: Reduction and synthesis of live and bounded free choice petri nets. Information and Computation 114, 50–87 (1994)
7. van der Aalst, W., Hirnschall, A., Verbeek, H.: An Alternative Way to Analyze Workflow Graphs. In: Banks-Pidduck, A., Mylopoulos, J., Woo, C., Ozsu, M. (eds.) CAiSE 2002. LNCS, vol. 2348, pp. 535–552. Springer, Heidelberg (2002)
8. Chen, R., Scheer, A.W.: Modellierung von Prozessketten mittels Petri-Netz-Theorie. Heft 107, Institut für Wirtschaftsinformatik, Saarbrücken (1994)
9. Rittgen, P.: Paving the Road to Business Process Automation. In: Proc. of ECIS 2000. pp. 313–319 (2000)
10. Langner, P., Schneider, C., Wehler, J.: Petri Net Based Certification of Event driven Process Chains. In: Desel, J., Silva, M. (eds.) ICATPN 1998. LNCS, vol. 1420, Springer, Heidelberg (1998)
11. Leymann, F., Altenhuber, W.: Managing business processes as an information resource. IBM Systems Journal 33, 326–348 (1994)
12. van der Aalst, W.: Formalization and Verification of Event-driven Process Chains. Information and Software Technology 41, 639–650 (1999)
13. Dehnert, J., Rittgen, P.: Relaxed Soundness of Business Processes. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 151–170. Springer, Heidelberg (2001)

14. Nüttgens, M., Rump, F.J.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In: Desel, J., Weske, M. (eds.): Promise'02. Vol. 21 of LNI. pp. 64–77 (2002)
15. van der Aalst, W., Desel, J., Kindler, E.: On the semantics of EPCs: A vicious circle. In: Nüttgens, M., Rump, F. J. (eds.): Proc. of EPK'02. pp. 71–79 (2002)
16. van der Aalst, W., ter Hofstede, A.: YAWL: Yet Another Workflow Language. Information Systems 30, 245–275 (2005)
17. Hee, K., Oanea, O., Serebrenik, A., Sidorova, N., Voorhoeve, M.: Workflow model compositions perserving relaxed soundness. In: Dustdar, S., Fiadeiro, J.L., Sheth, A. (eds.) BPM 2006. LNCS, vol. 4102, pp. 225–240. Springer, Heidelberg (2006)
18. Mendling, J., Moser, M., Neumann, G.: Transformation of yEPC Business Process Models to YAWL. In: Proc. of ACM SAC, 2, 1262–1267 (2006)
19. Mendling, J., van der Aalst, W.: Towards EPC Semantics based on State and Context. In: Nüttgens, M., Rump, F. J., Mendling, J. (eds.): Proc. of EPK'06 pp. 25–48 (2006)
20. Wynn, M., Edmond, D., van der Aalst, W., ter Hofstede, A.: Achieving a General, Formal and Decidable Approach to the OR-join in Workflow using Reset nets. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 423–443. Springer, Heidelberg (2005)
21. Leuschel, M., Lehmann, H.: Coverability of reset petri nets and other well-structured transition systems by partial deduction. In: Palamidessi, C., Moniz Pereira, L., Lloyd, J.W., Dahl, V., Furbach, U., Kerber, M., Lau, K.-K., Sagiv, Y., Stuckey, P.J. (eds.) CL 2000. LNCS (LNAI), vol. 1861, pp. 101–115. Springer, Heidelberg (2000)
22. Finkel, A., Schnoebelen, P.: Well-structured Transition Systems everywhere! Theoretical Computer Science 256, 63–92 (2001)
23. Wynn, M., van der Aalst, W., ter Hofstede, A., Edmond, D.: Verifying Workflows with Cancellation Regions and OR-joins: An Approach Based on Reset Nets and Reachability Analysis. In: Dustdar, S., Fiadeiro, J.L., Sheth, A. (eds.) BPM 2006. LNCS, vol. 4102, pp. 389–394. Springer, Heidelberg (2006)
24. Lin, H., Zhao, Z., Li, H., Chen, Z.: A novel graph reduction algorithm to identify structural conflicts. In: Proc. of HICSS. 289 (2002)
25. Mendling, J.: Detection and Prediction of Errors in EPC Business Process Models. Ph.D. Thesis, Vienna University of Economics and Business Administration (2007)
26. Rump, F.J.: Geschäftsprozessmanagement auf der Basis ereignisgesteuerter Prozessketten - Formalisierung, Analyse und Ausführung von EPKs. Teubner (1999)
27. van Dongen, B., Medeiros, A., Verbeek, H., Weijters, A., van der Aalst, W.: The ProM framework: A New Era in Process Mining Tool Support. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 444–454. Springer, Heidelberg (2005)
28. van der Aalst, W.: Verification of Workflow Nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997)
29. Keller, G., Teufel, T.: SAP(R) R/3 Process Oriented Implementation: Iterative Process Prototyping. Addison-Wesley, London (1998)