Decidability of Model Checking with the Temporal Logic EF

Richard Mayr

Institut für Informatik
Technische Universität München
Arcisstr. 21, D-80290 München, Germany
e-mail: mayrri@informatik.tu-muenchen.de

Abstract

The branching-time temporal logic EF is a simple, but natural fragment of computation-tree logic (CTL) and the modal μ-calculus. We study the decidability of the model checking problem for EF and infinite-state systems. We use process rewrite systems (PRS) to describe infinite-state systems and define a hierarchy of subclasses of PRS that includes Petri nets, pushdown processes, Basic Parallel Processes (BPP), context-free processes and PA-Processes. Then we establish the exact limits of the decidability of model checking with EF in this hierarchy.

Model checking with EF is undecidable for Petri nets and even for parallel pushdown automata (the pushdown extension of Basic Parallel Processes). On the other hand, model checking with EF is decidable for PAD, a process model that subsumes both PA-processes and pushdown processes.

Key words: infinite-state systems, temporal logic, EF, model checking, process algebra, PA-processes, pushdown processes

1 Introduction

The branching-time temporal logic EF (also called UB^- in [11] and [19]) uses the boolean operators, the one-step next operator EX (for some successor), and the operator EF (for some path eventually in the future). It is a fragment of computation tree logic (CTL), which in turn is weaker than the modal μ-calculus [5]. EF-formulae are interpreted over (possibly infinite) trees describing all possible computations of a process. The processes can also have infinite state spaces.

Preprint submitted to Elsevier Preprint *28 June 1999*

There are many models for systems with infinite state spaces. Some of the most common are Milner's Calculus of Communicating Systems (CCS) [25], Basic Parallel Processes (BPP) [9], context-free processes (BPA), pushdown processes and Petri nets. The process algebra PA is a common generalization of BPP and BPA and has operators for nondeterministic choice, parallel composition, sequential composition and recursion. Unlike BPP, PA is not a syntactical subset of CCS [25], because CCS does not have an explicit operator for sequential composition. However, as CCS can simulate sequential composition by parallel composition and synchronization, PA is still a weaker model than CCS. PA-processes, pushdown processes and Petri nets are mutually incomparable (see Section 2).

Except for CCS, all these models can be represented by special subclasses of a general rewriting formalism. These rewrite systems called "Process Rewrite Systems (PRS)" were introduced in [18,22], together with a hierarchy of its subclasses (the "PRS-hierarchy"). The PRS-hierarchy is a common generalization of two separate hierarchies for rewrite systems with sequential and parallel composition that were defined by Stirling, Moller and Caucal [26] (see also [11,7]) in analogy to the Chomsky-hierarchy. In this hierarchy, there is a natural common generalization of PA-processes and pushdown processes. This model was called PAD (for PA + PD) in [18,22] and it is strictly more general than PA and pushdown processes with respect to bisimulation equivalence.

The model checking problem is the problem if a given process satisfies a property encoded as a formula in a certain temporal logic. We study the model checking problem for the logic EF and the models in the PRS-hierarchy. The main new result in this paper is that model checking with EF is decidable even for PAD. This completes the general picture of the decidability of model checking with EF.

In Section 2 we define Process Rewrite Systems (PRS) and the PRS-hierarchy of its subclasses. In Section 3 we define the logic EF and some generalizations of EF. In Section 4 we show that model checking PAD with EF is decidable. In Section 5 we describe a small example. In Section 6 we show that model checking with EF is undecidable for PPDA, the pushdown extension of BPP, which is a subclass of Petri nets. In the last section we present a general view of the limits of the decidability of model checking with EF and other temporal logics.

2 Process Models

Many classes of concurrent systems can be described by a (possibly infinite) set of process terms, representing the states, and a finite set of rewrite rules describing the dynamics of the system.

Definition 1 *Let $Act = \{a, b, \dots\}$ be a countably infinite set of atomic actions and $Const = \{\epsilon, X, Y, Z, \dots\}$ a countably infinite set of process constants. The process terms that describe the states of the system have the following form:*

$$t ::= \epsilon \mid X \mid t_1.t_2 \mid t_1 \| t_2$$

where ϵ is the empty term, $X \in Const$ is a process constant (used as an atomic process in this context), "$\|$" means parallel composition and "." means sequential composition. Parallel composition is associative and commutative. Sequential composition is associative. Let $\mathcal{T}$ be the set of process terms.

Convention 1: *We always work with equivalence classes of terms modulo commutativity and associativity of parallel composition and modulo associativity of sequential composition. Also we define that $\epsilon.t = t = t.\epsilon$ and $t \| \epsilon = t$.*

Convention 2: *We defined that sequential composition is associative. However, when we look at terms we think of it as left-associative. So when we say that a term t has the form $t_1.t_2$, then we mean that t_2 is either a single constant or a parallel composition of process terms.*

The size *of a process term is defined as the number of occurrences of constants in it plus the number of occurrences of operators in it.*

$$\begin{aligned} size(\epsilon) &:= 0 \\ size(X) &:= 1 \\ size(t_1.t_2) &:= size(t_1) + size(t_2) + 1 \\ size(t_1 \| t_2) &:= size(t_1) + size(t_2) + 1 \end{aligned}$$

For a term t the set $Const(t)$ is the set of constants that occur in t.

$$\begin{aligned} Const(\epsilon) &:= \emptyset \\ Const(X) &:= \{X\} \\ Const(t_1.t_2) &:= Const(t_1) \cup Const(t_2) \\ Const(t_1 \| t_2) &:= Const(t_1) \cup Const(t_2) \end{aligned}$$

The dynamics of the system is described by a finite set of rules Δ of the form $(t_1 \stackrel{a}{\to} t_2)$ where t_1 and t_2 are process terms and $a \in Act$ is an atomic action. The finite set of rules Δ induces a (possibly infinite) labeled transition system with relations $\stackrel{a}{\to}$ with $a \in Act$. For every $a \in Act$, the transition relation $\stackrel{a}{\to}$

is the smallest relation that satisfies the following inference rules.

$$\frac{(t_1 \xrightarrow{a} t_2) \in \Delta}{t_1 \xrightarrow{a} t_2} \quad \frac{t_1 \xrightarrow{a} t_1'}{t_1 \| t_2 \xrightarrow{a} t_1' \| t_2} \quad \frac{t_1 \xrightarrow{a} t_1'}{t_1.t_2 \xrightarrow{a} t_1'.t_2}$$

where t_1, t_2, t_1', t_2' are process terms. Note that parallel composition is commutative and thus the inference rule for parallel composition also holds with t_1 and t_2 exchanged.

Since Δ is finite, the generated LTS is finitely branching. (For some classes of systems (e.g. Petri nets) the branching-degree is bounded by a constant that depends on Δ. For other classes (e.g. PA) the branching-degree is finite at every state, but it can get arbitrarily high.) Also every single Δ uses only a finite subset $Const(\Delta) := \bigcup_{(t_1 \xrightarrow{a} t_2) \in \Delta} (Const(t_1) \cup Const(t_2))$ of constants and only a finite subset $Act(\Delta) := \bigcup_{(t_1 \xrightarrow{a} t_2) \in \Delta} \{a\}$ of atomic actions. Thus for every Δ only finitely many of the generated transition relations $\xrightarrow{a_i}$ for $a_i \in Act$ are nonempty. (Those for which $a_i \in Act(\Delta)$). Still the generated transition system can be infinite. (Consider the analogy: Every labeled Petri net has only finitely many transitions and uses only finitely many different atomic actions, but the state space can be infinite.) The relation $\xrightarrow{a}$ is generalized to sequences of actions in the standard way. Sequences are denoted by σ.

Remark 2 *There is no operator "+" for nondeterministic choice in the process terms, because this is encoded in the set of rules Δ! There can be several rules with the same term on the left hand side. It is also possible that several rules are applicable at different places in a term. The rule that is applied and the position where it is applied are chosen nondeterministically. Also there is no such thing as action prefixes in the process terms. The atomic actions are introduced by the rules.*

Many common models of systems fit into this scheme. In the following we characterize subclasses of rewrite systems. The expressiveness of a class depends on what kind of terms are allowed on the left hand side and right hand side of the rewrite rules in Δ.

Definition 3 (Classes of process terms)
We distinguish four classes of process terms:

1 *Terms consisting of a single process constant like X.*
S *Terms consisting of a single constant or a sequential composition of process constants like $X.Y.Z$.*
P *Terms consisting of a single constant or a parallel composition of process constants like $X \| Y \| Z$.*
G *General process terms with arbitrary sequential and parallel composition like $(X.(Y \| Z)) \| W$.*

Also let $\epsilon \in S, P, G$, but $\epsilon \notin 1$. It is easy to see that the relations between these classes of process terms are: $1 \subset S$, $1 \subset P$, $S \subset G$ and $P \subset G$. S and P are incomparable and $S \cap P = 1 \cup \{\epsilon\}$.

We characterize classes of process rewrite systems (PRS) by the classes of terms allowed on the left hand sides and the right hand sides of rewrite rules.

Definition 4 (PRS)
Let $\alpha, \beta \in \{1, S, P, G\}$. A (α, β)-PRS is a finite set of rules Δ where for every rewrite rule $(l \stackrel{a}{\to} r) \in \Delta$ the term l is in the class α and $l \neq \epsilon$ and the term r is in the class β (and can be ϵ). The initial state is given as a term $t_0 \in \alpha$. A (G, G)-PRS is simply called PRS.

Remark 5 *W.l.o.g. it can be assumed that the initial state t_0 of a PRS is a single constant. There are only finitely many terms $t_1, \ldots, t_n$ s.t. $t_0 \stackrel{a_i}{\to} t_i$. If t_0 is not a single constant then we can achieve this by introducing a new constant X_0 and new rules $X_0 \stackrel{a_i}{\to} t_i$ and declaring X_0 to be the initial state.*

(α, β)-PRS where α is more general than β or incomparable to β (for example $\alpha = G$ and $\beta = S$) do not make any sense. This is because the terms that are introduced by the right side of rules must later be matched by the left sides of other rules. So in a (G, S)-PRS the rules that contain parallel composition on the left hand side will never be used (assuming that the initial state is a single constant). Thus one may as well use a (S, S)-PRS. So we restrict our attention to (α, β)-PRS with $\alpha \subseteq \beta$.

Figure 1 shows a graphical description of the hierarchy of (α, β)-PRS.

Many of these (α, β)-PRS correspond to widely known models like Petri nets, pushdown processes, context-free processes and others.

(1) A $(1, 1)$-PRS is a finite-state system. Every process constant corresponds to a state and the state space is bounded by $|Const(\Delta)|$. Every finite-state system can be encoded as a $(1, 1)$-PRS.
(2) $(1, S)$-PRS are equivalent to context-free processes (also called "Basic Process Algebra (BPA)") [7,11]. They are transition systems associated with Greibach normal form (GNF) context-free grammars in which only left-most derivations are permitted.
(3) It is easy to see that pushdown automata can be encoded as a subclass of (S, S)-PRS (with at most two constants on the left side of rules). Caucal [8] showed that any unrestricted (S, S)-PRS can be presented as a pushdown automaton (PDA), in the sense that the transition systems are isomorphic up to the labeling of states. Thus (S, S)-PRS are equivalent to pushdown processes, the processes described by pushdown automata.
(4) (P, P)-PRS are equivalent to Petri nets. Every constant corresponds to a place in the net and the number of occurrences of a constant in a term

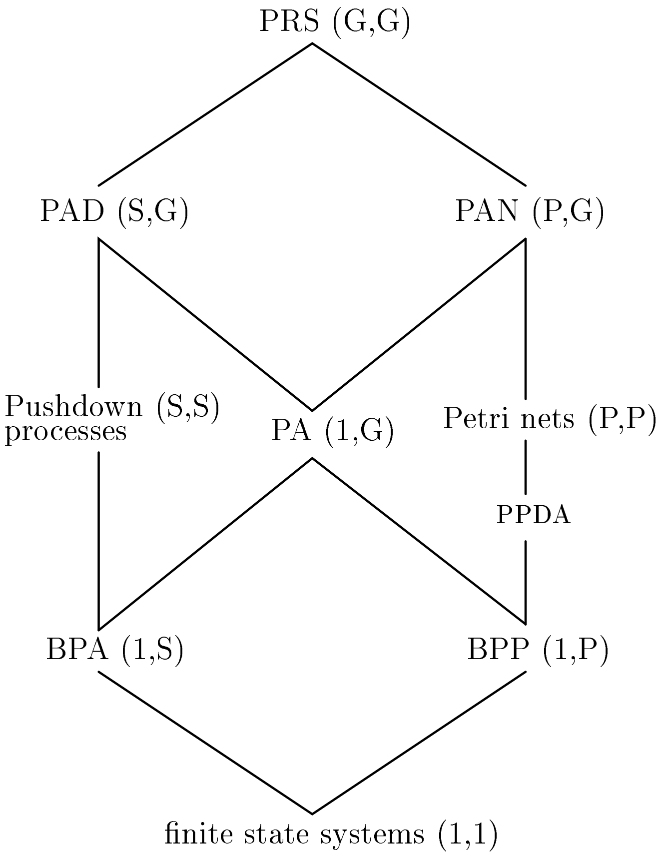

Fig. 1. The PRS-hierarchy

corresponds to the number of tokens in this place. This is because we work with classes of terms modulo commutativity of parallel composition. Every rule in Δ corresponds to a transition in the net.

(5) $(1, P)$-PRS are equivalent to communication-free nets, the subclass of Petri nets where every transition has exactly one place in its preset [7,11]. This class of Petri nets is equivalent to *Basic Parallel Processes (BPP)* [9].

(6) $(1, G)$-PRS are equivalent to PA-processes, a process algebra with sequential and parallel composition, but no communication (see [1,21,16]).

(7) (P, G)-PRS are called *PAN-processes* in [20]. It is a common generalization of Petri nets and PA-processes and it is strictly more general than both of them (e.g. PAN can describe all Chomsky-2 languages while Petri nets cannot).

(8) (S, G)-PRS are a common generalization of pushdown processes and PA-processes. They are called *PAD* (PA + PD) in [23].

(9) The most general case is (G, G)-PRS (here simply called PRS). PRS have been introduced in [18,22]. They subsume all the previously mentioned classes.

What does it mean that parallel/sequential/arbitrary composition is allowed in terms on the left/right hand sides of rules? The general intuition is as follows:

If parallel composition is allowed on the right hand side of rules, then there can be rules of the form $t \xrightarrow{a} t_1 \| t_2$. This means that it is possible to create processes that run in parallel. The rule can be interpreted that, by action a, the process t becomes the process t_1 and spawns off the process t_2 or vice versa. If sequential composition is allowed on the right hand side of rules, then there are rules of the form $t \xrightarrow{a} t_1.t_2$. The interpretation is that process t calls a subroutine t_1 and becomes process t_2. It resumes its execution when the subroutine t_1 terminates. If arbitrary sequential and parallel composition is allowed on the right hand side of rules then both parallelism and subroutines are possible. If parallel composition is allowed on the left hand side of rules, then there are rules of the form $t_1 \| t_2 \xrightarrow{a} t$. This can be interpreted as synchronization/communication of the parallel processes t_1 and t_2. This is because this action can only occur if both t_1 and t_2 change in a certain defined way. If sequential composition is allowed on the left hand side of rules, then there can be rules of the form $t_1'.t_2 \xrightarrow{a} t'$ and $t_1''.t_2 \xrightarrow{a} t''$. The intuition is that a process t called a subroutine t_1 and became process t_2 by a rule $t \xrightarrow{a} t_1.t_2$. The subroutine may in its computation reach a state t_1' or t_1''. Now one of these rules is applicable. This means that the result of the computation of the subroutine affects the behavior of the caller when it becomes active again, since the caller can become t' or t''. The interpretation is that the subroutine returns a value to the caller when it terminates. If arbitrary sequential and parallel composition is allowed on the left hand sides of rules then both synchronization and returning of values by subroutines are possible. Rules with nested sequential and parallel composition (on the left side or the right side) do not increase the expressiveness [18].

Thus, for example, the processes of class PAD (type (S, G) in the PRS-hierarchy) have parallel composition and subroutines that can return values to their caller, but they lack the ability to synchronize parallel processes.

It has been shown in [18] (with the help of earlier results from [6,26]) that this hierarchy of subclasses of PRS is strict w.r.t. bisimulation equivalence, i.e., more general subclasses are strictly more expressive. (See [25,26] for more on bisimulation.)

Theorem 6 ([18])
The PRS-hierarchy is strict with respect to bisimulation equivalence.

It has also been shown in [18] that PRS are not Turing-powerful.

Theorem 7 ([18])
The reachability problem is decidable for PRS.

3 The Temporal Logic EF

Temporal logics are used to describe properties of systems. The verification process consists in showing that a given system satisfies a property encoded in a given formula. We use the logic $EF_{DC}^{=}$, an extended version of the logic EF [11,7]. In addition to the standard operators of EF, the logic $EF_{DC}^{=}$ uses strong atomic propositions of the form 'The current state is term t' and can thus express the reachability problem. The "=" in the name stands for these strong propositions, because they express that the current state is equal to a given state t. Note that, because of this feature, the logic $EF_{DC}^{=}$ (unlike EF) is not a fragment of CTL or the modal μ-calculus. The modal μ-calculus (and CTL) cannot distinguish bisimilar states, but $EF_{DC}^{=}$ can. The logic $EF_{DC}^{=}$ can also express weak constraints on sequences of actions. These constraints are called *decomposable constraints* (thus the DC in the name).

Definition 8 ($EF_{DC}^{=}$)
The syntax of the formulae is as follows:

$$\Phi ::= t \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \Diamond_C \Phi$$

where $t \in \mathcal{T}$ is a process term and C is a decomposable constraint *(see Def. 10).*

Let $\mathcal{F}$ be the set of all $EF_{DC}^{=}$-formulae. Let $\mathcal{T}$ be the set of all processes terms (as in Def. 1) in the process algebra. The denotation $[\![\Phi]\!]$ of an $EF_{DC}^{=}$-formula Φ is the set of process terms defined inductively by the following rules:

$$\begin{aligned}
[\![t]\!] &:= \{t\} \\
[\![\neg\Phi]\!] &:= \mathcal{T} - [\![\Phi]\!] \\
[\![\Phi_1 \wedge \Phi_2]\!] &:= [\![\Phi_1]\!] \cap [\![\Phi_2]\!] \\
[\![\Diamond_C \Phi]\!] &:= \{t \in \mathcal{T} \mid \exists t', \sigma.\ t \xrightarrow{\sigma} t' \ \wedge\ t' \in [\![\Phi]\!] \ \wedge\ C(\sigma)\}
\end{aligned}$$

Disjunction can be expressed by conjunction and negation.

The property $t \in [\![\Phi]\!]$ is also denoted by $t \models \Phi$.

The model checking problem is the problem if a process satisfies a property encoded as a formula in a temporal logic.

<u>Model Checking</u>

Instance: A description of a process (for example an (α, β)-PRS Δ with $\alpha, \beta \in \{1, S, P, G\}$ (see Def. 3)) and a state t and a formula Φ from a temporal logic (for example EF or $EF_{DC}^{=}$).
Question: Is it true that $t \models \Phi$?

Model checking finite-state systems with EF can be done in polynomial time, since EF is a fragment of the alternation-free modal μ-calculus [11,7]. Model checking Petri nets with EF has been shown to be undecidable [11,7] by reduction of the reachability set containment problem for Petri nets. Model checking with EF is *PSPACE*-complete for Basic Parallel Processes (BPP) [23,19], and context-free processes (BPA) [2,24]. For pushdown processes the complexity of model checking with EF is between *PSPACE* and *EXPTIME* [2,29,30]. It was claimed in [2] that model checking pushdown processes with EF is *PSPACE*-complete. Unfortunately, the given proof is wrong. It assumes that an accepting polynomial space-bounded Turing-machine has an accepting computation of polynomial length, which is not true in general. It was shown in [21] that model checking with EF is decidable for PA-processes. Lugiez and Schnoebelen [17] later proved the same result by using a completely different method (using tree-automata to represent infinite sets of configurations). In the next section we show that model checking with the more general logic $EF_{\mathcal{DC}}^{=}$ is decidable for the more general model PAD (type (S,G) in the PRS-hierarchy). But first we reduce the problem to a simpler form.

Definition 9 *For any $EF_{\mathcal{DC}}^{=}$-formula Φ let terms(Φ) be the set of process terms used in Φ as atomic propositions.*

$$\begin{aligned}
terms(t) &:= \{t\} \\
terms(\neg\Phi) &:= terms(\Phi) \\
terms(\Phi_1 \wedge \Phi_2) &:= terms(\Phi_1) \cup terms(\Phi_2) \\
terms(\Diamond_C \Phi) &:= terms(\Phi)
\end{aligned}$$

The logic $EF_{\mathcal{DC}}^{=}$ uses constraints on sequences of actions. These constraints are called *decomposable*, because they can be decomposed with respect to sequential and parallel composition of sequences of actions.

Definition 10 (Decomposable Constraints)
A set of decomposable constraints $\mathcal{DC}$ is a finite set of predicates on finite sequences of actions that satisfy the following conditions.

(1) $\mathcal{DC}$ contains the predicates true (all sequences satisfy it) and false (no sequence satisfies it).
(2) For every predicate $C \in \mathcal{DC}$ it is decidable if C is satisfiable.
(3) For every $C \in \mathcal{DC}$ there is a finite index set I and a finite set of decomposable constraints $\{C_i^1, C_i^2 \in \mathcal{DC} \mid i \in I\}$ s.t.

$$\forall \sigma, \sigma_1, \sigma_2.\ \sigma_1\sigma_2 = \sigma \ \Rightarrow \ \left(C(\sigma) \iff \bigvee_{i \in I} C_i^1(\sigma_1) \wedge C_i^2(\sigma_2) \right)$$

(4) For every $C \in \mathcal{DC}$ there is a finite index set I and a finite set of decom-

posable constraints $\{C'_i \in \mathcal{DC} \mid i \in I\}$ *s.t.*

$$\forall \sigma, \sigma'.\ a\sigma' = \sigma \ \Rightarrow \ \left(C(\sigma) \iff \bigvee_{i \in I} C'_i(\sigma') \right)$$

(5) *For every* $C \in \mathcal{DC}$ *there is a finite index set* I *and a finite set of decomposable constraints* $\{C^1_i, C^2_i \in \mathcal{DC} \mid i \in I\}$ *s.t.*

$$\forall \sigma, \sigma_1, \sigma_2 . \forall a \in Act.\ \sigma_1 a \sigma_2 = \sigma \ \Rightarrow \ \left(C(\sigma) \iff \bigvee_{i \in I} C^1_i(\sigma_1) \wedge C^2_i(\sigma_2) \right)$$

(6) *For every* $C \in \mathcal{DC}$ *there is a finite index set* I *and a finite set of decomposable constraints* $\{C^1_i, C^2_i \in \mathcal{DC} \mid i \in I\}$ *s.t.*

$$\forall \sigma_1, \sigma_2. \left(\ (\exists \sigma \in interleave(\sigma_1, \sigma_2).\, C(\sigma)) \iff \bigvee_{i \in I} (C^1_i(\sigma_1) \wedge C^2_i(\sigma_2)) \right)$$

$\sigma \in interleave(\sigma_1, \sigma_2)$ *means that* σ *is an arbitrary interleaving of* σ_1 *and* σ_2. *The formal definition of the function interleave is as follows: Let* λ *be the empty sequence.*

$$\begin{aligned} interleave(\lambda, \sigma) &:= \{\sigma\} \\ interleave(\sigma, \lambda) &:= \{\sigma\} \\ interleave(a_1\sigma_1, a_2\sigma_2) &:= \begin{array}{l} \{a_1\sigma \mid \sigma \in interleave(\sigma_1, a_2\sigma_2)\} \cup \\ \{a_2\sigma \mid \sigma \in interleave(a_1\sigma_1, \sigma_2)\} \end{array} \end{aligned}$$

Lemma 11 *If* $\mathcal{DC}$ *is a set of decomposable constraints, then the closure* $\mathcal{DC}'$ *of* $\mathcal{DC}$ *under the boolean operations of conjunction and disjunction is also a set of decomposable constraints.*

PROOF. *The formulae in* $\mathcal{DC}'$ *can be transformed into disjunctive normal form, such that the formulae in* $\mathcal{DC}$ *are the atomic formulae. Since* $\mathcal{DC}$ *is finite,* $\mathcal{DC}'$ *is finite too.*

Remark 12 *A set of decomposable constraints need not be closed under negation.*

Now we give an example for a set of decomposable constraints. Let $A \subset Act$, be a finite set of atomic actions. For any $a \in A$ let $\#_a(\sigma)$ be the number of occurrences of action a in σ. For $u, v \in \mathbb{N}$ let $[u]_v$ denote u modulo v. We define the following constraints:

(1) $length(\sigma) \geq i$ or $length(\sigma) \leq i$ for all $i \leq k$ for some fixed constant k.
(2) $\#_a(\sigma) \geq i$ or $\#_a(\sigma) \leq i$ for all $i \leq n$ for some fixed constant n.

(3) $[\#_a(\sigma)]_k = i$ for all $i, k \leq m$ for some fixed constant m.
(4) $\mathit{first}(\sigma) = a$ for any action $a \in A$.

For any choice of A, k, n, m let $\mathcal{C}_{A,k,n,m}$ denote the closure of the set of these constraints under conjunction and disjunction.

Lemma 13 *For any A, k, n, m, the set $\mathcal{C}_{A,k,n,m}$ is a set of decomposable constraints. It is even closed under negation.*

PROOF. *Directly from the definitions.*

Example 14 *The constraint $[\#_a(\sigma)]_2 = 0$ expresses that the number of occurrences of action a in σ is even. Let $\sigma \in interleave(\sigma_1, \sigma_2)$ be an interleaving of two sequences. Then the number of occurrences of the action a in σ is even iff it is either even in both σ_1 and σ_2 or odd in both σ_1 and σ_2. This can be expressed by the following decomposition.*

$$\begin{aligned}[\#_a(\sigma)]_2 = 0 \iff & ([\#_a(\sigma_1)]_2 = 0 \ \wedge \ [\#_a(\sigma_2)]_2 = 0) \ \vee \\ & ([\#_a(\sigma_1)]_2 = 1 \ \wedge \ [\#_a(\sigma_2)]_2 = 1)\end{aligned}$$

Decomposable constraints increase the expressiveness of the logic. They have proved to be useful for constructing characteristic formulae for finite-state systems up to (different kinds of) bisimulation-like equivalences [15]. For more details on decomposable constraints and decomposable languages see [17,28].

We use these constraints to show that the usual definition of EF is a fragment of $EF^{\equiv}_{DC}$. The usual $\Diamond$ is just $\Diamond_{true}$. The normal one-step nexttime operator EX is often denoted by $\langle a \rangle$ and defined by

$$[\![\langle a \rangle \Phi]\!] := \{t \mid \exists t'.\, t \xrightarrow{a} t' \ \wedge \ t' \in [\![\Phi]\!]\}$$

It is clear that $\langle a \rangle = \Diamond_C$ with $C := [\mathit{first}(\sigma) = a \wedge \mathit{length}(\sigma) = 1]$. The normal version of EF also does not have atomic propositions t (meaning that the state is equal to t; see Def. 8), but propositions "a" (meaning that the atomic action a is enabled). This can be expressed by $\langle a \rangle \mathit{true}$, where $\mathit{true} = t \vee \neg t$ for any term t.

It is also possible to express the modal operator $\Box$ (meaning 'always') by defining $\Box_C := \neg \Diamond_C \neg$. $\Box_C \Phi$ then means that Φ holds in all states that are reachable via a sequence of actions σ s.t. $C(\sigma)$.

Definition 15 *The* nesting-depth $nd(\Phi)$ *of an $EF^{\equiv}_{DC}$-formula Φ is defined by*

$$
\begin{aligned}
nd(t) &:= 0 \\
nd(\neg\Phi) &:= nd(\Phi) \\
nd(\Phi_1 \wedge \Phi_2) &:= max\{nd(\Phi_1), nd(\Phi_2)\} \\
nd(\Diamond_C \Phi) &:= nd(\Phi) + 1
\end{aligned}
$$

Definition 16 *$\mathcal{F}_d \subset \mathcal{F}$ is defined as the set of all $EF^{=}_{DC}$-formulae with a nesting-depth of modal operators $\Diamond_C$ of at most d.*

$$\mathcal{F}_d := \{\Phi \in \mathcal{F} \mid nd(\Phi) \leq d\}$$

It follows that formulae in $\mathcal{F}_0$ contain no modal operators.

In order to simplify the notation we use some abbreviations:
Let $T = \{t_1, \ldots, t_n\} \subseteq \mathcal{T}$ be a finite set of process terms, then

$$t \models -T \ :\Longleftrightarrow \ t \models \neg t_1 \wedge \cdots \wedge \neg t_n$$

For reasons of symmetry we also define

$$t \models T :\Longleftrightarrow \ t \models t_1 \wedge \cdots \wedge t_n$$

Of course this cannot be true if $n \geq 2$.

Definition 17 *We define a subset $\mathcal{F}^c_d \subset \mathcal{F}_d$ of formulae that do not contain disjunction. Thus the formulae in $\mathcal{F}^c_d$ are called* conjunctive formulae. *$\mathcal{F}^c_d$ is defined as the minimal set of formulae Φ_d that are defined by the following grammar.*

$$\Phi_0 = T^+ \wedge -T^-$$

for every finite $T^+, T^- \subset \mathcal{T}$ and

$$\Phi_d = T^+ \wedge -T^- \mid \Phi_d \wedge \Diamond_C \Phi_{d-1} \mid \Phi_d \wedge \neg\Diamond_C \Phi_{d-1}$$

for every finite $T^+, T^- \subset \mathcal{T}$ and every decomposable constraint C and every $\Phi_{d-1} \in \mathcal{F}^c_{d-1}$.

It follows that every formula in $\mathcal{F}^c_d$ has the form

$$T^+ \wedge -T^- \wedge \bigwedge_{i \in I} \Diamond_{C_i} \Psi_i \wedge \bigwedge_{j \in J} \neg\Diamond_{D_j} \Upsilon_j$$

where $T^+, T^- \subset \mathcal{T}$, and C_i, D_j are decomposable constraints and $\Psi_i \in \mathcal{F}^c_{d-1}$ and $\Upsilon_j \in \mathcal{F}^c_{d-1}$.

A formula Φ is in normal form *if $\Phi = \bigvee_{i \in I} \Diamond_{C_i} \Psi_i$ s.t. the Ψ_i are conjunctive formulae.*

Lemma 18 *Any $EF^{=}_{DC}$-formula $\Diamond_C \Phi$ is equivalent to a formula in normal form.*

PROOF. *By induction on the nesting-depth d of modal operators in Φ. The important property here is that $\Diamond_C(\Phi_1 \vee \Phi_2) = \Diamond_C\Phi_1 \vee \Diamond_C\Phi_2$. We transform the subformulae into disjunctive normal form, and then push the disjunctions outwards.*

Lemma 19 *Every model checking problem for $EF^{=}_{DC}$ is decidable iff it is decidable for all formulae $\Diamond_C\Phi$ with $\Phi \in \bigcup_{d\in\mathbb{N}} \mathcal{F}^c_d$.*

PROOF. *If it is decidable for formulae of the form $\Diamond_C\Phi$ with $\Phi \in \mathcal{F}^c_d$, then it is decidable for formulae in normal form and thus by Lemma 18 for all formulae of the form $\Diamond_C\Psi$, with $\Psi \in \mathcal{F}$. Simple boolean operations yield the decidability of the whole model checking problem. The other direction is trivial.*

4 Model Checking PAD

We prove the decidability of the model checking problem for $EF^{=}_{DC}$ and PAD by construction of a sound and complete tableau. By Lemma 19 it suffices to consider formulae of the form $\Diamond_C\Phi$ for $\Phi \in \mathcal{F}^c_d$ for any d and C in some set of decomposable constraints.

4.1 Decomposition

The key to the construction of the tableau system in Subsection 4.2 is that properties of the form $t_1.t_2 \models \Diamond_C\Phi$ or $t_1\|t_2 \models \Diamond_C\Phi$ can be decomposed into properties of t_1 and properties of t_2. First we give a small example how this is done and then we do it in general.

Example 20 *We show how to do the decomposition for the following simple formula of nesting-depth two:*

$$\Phi := \Diamond(\neg u \ \wedge \ \Diamond(v) \ \wedge \ \neg\Diamond(w))$$

where u, v, w are process terms. No decomposable constraints are used, except for the constraint true ($\Diamond_{true} = \Diamond$). This formula means that there is a reachable state different from u, s.t. from this state the state v is reachable, but the state w is not reachable.

Let t_1, t_2 be process terms. Then the property

$$t_1 \| t_2 \models \Diamond(\neg u \ \wedge \ \Diamond(v) \ \wedge \ \neg\Diamond(w))$$

is equivalent to

$$\begin{aligned}
&(t_1 \models \Diamond(\epsilon) \ \wedge \ t_2 \models \Phi) \ \vee \\
&(t_1 \models \Phi \ \wedge \ t_2 \models \Diamond(\epsilon)) \ \vee \\
&\exists \sigma_1, \sigma_2, t_1', t_2'.\ t_1 \xrightarrow{\sigma_1} t_1' \wedge t_2 \xrightarrow{\sigma_2} t_2' \ \wedge \\
&\qquad\qquad \bigwedge_{\alpha_1 \| \alpha_2 = u} (t_1' \neq \alpha_1 \vee t_2' \neq \alpha_2) \ \wedge \ t_1' \| t_2' \models \Diamond(v) \ \wedge \ t_1' \| t_2' \models \neg\Diamond(w)
\end{aligned}$$

where α_1, α_2 are process terms. This is equivalent to

$$\begin{aligned}
&(t_1 \models \Diamond(\epsilon) \ \wedge \ t_2 \models \Phi) \ \vee \\
&(t_1 \models \Phi \ \wedge \ t_2 \models \Diamond(\epsilon)) \ \vee \\
&\exists \sigma_1, \sigma_2, t_1', t_2'.\ t_1 \xrightarrow{\sigma_1} t_1' \wedge t_2 \xrightarrow{\sigma_2} t_2' \ \wedge \bigwedge_{\alpha_1 \| \alpha_2 = u} (t_1' \neq \alpha_1 \vee t_2' \neq \alpha_2) \ \wedge \\
&\qquad\qquad \bigvee_{\beta_1 \| \beta_2 = v} (t_1' \models \Diamond(\beta_1) \ \wedge \ t_2' \models \Diamond(\beta_2)) \ \wedge \\
&\qquad\qquad \neg \left(\bigvee_{\gamma_1 \| \gamma_2 = w} t_1' \models \Diamond(\gamma_1) \wedge t_2' \models \Diamond(\gamma_2) \right)
\end{aligned}$$

This is equivalent to

$$\begin{aligned}
&(t_1 \models \Diamond(\epsilon) \ \wedge \ t_2 \models \Phi) \ \vee \\
&(t_1 \models \Phi \ \wedge \ t_2 \models \Diamond(\epsilon)) \ \vee \\
&\exists \sigma_1, \sigma_2, t_1', t_2'.\ t_1 \xrightarrow{\sigma_1} t_1' \wedge t_2 \xrightarrow{\sigma_2} t_2' \wedge \bigwedge_{\alpha_1 \| \alpha_2 = u} (t_1' \neq \alpha_1 \vee t_2' \neq \alpha_2) \ \wedge \\
&\qquad\qquad \bigvee_{\beta_1 \| \beta_2 = v} (t_1' \models \Diamond(\beta_1) \ \wedge \ t_2' \models \Diamond(\beta_2)) \ \wedge \\
&\qquad\qquad \bigwedge_{\gamma_1 \| \gamma_2 = w} (t_1' \models \neg\Diamond(\gamma_1) \vee t_2' \models \neg\Diamond(\gamma_2))
\end{aligned}$$

Now we transform this expression into disjunctive normal form. We define the set F of all functions f that assign to every pair (α_1, α_2) s.t. $\alpha_1 \| \alpha_2 = u$, a value in $\{1, 2\}$. For every $f \in F$ let $A_f^1 := \{\alpha_1 \mid f((\alpha_1, \alpha_2)) = 1\}$ and

$A^2_f := \{\alpha_2 \mid f((\alpha_1,\alpha_2)) = 2\}$. *Then the expression is equivalent to*

$$
\begin{aligned}
&(t_1 \models \Diamond(\epsilon) \;\wedge\; t_2 \models \Phi) \vee \\
&(t_1 \models \Phi \;\wedge\; t_2 \models \Diamond(\epsilon)) \vee \\
&\exists \sigma_1, \sigma_2, t_1', t_2'.\ t_1 \xrightarrow{\sigma_1} t_1' \wedge t_2 \xrightarrow{\sigma_2} t_2' \wedge \bigvee_{f \in F} (t_1' \notin A^1_f \;\wedge\; t_2' \notin A^2_f) \wedge \\
&\qquad \bigvee_{\beta_1 \| \beta_2 = v} (t_1' \models \Diamond(\beta_1) \;\wedge\; t_2' \models \Diamond(\beta_2)) \wedge \\
&\qquad \bigwedge_{\gamma_1 \| \gamma_2 = w} (t_1' \models \neg\Diamond(\gamma_1) \vee t_2' \models \neg\Diamond(\gamma_2))
\end{aligned}
$$

In the same way we define the set G of all functions g that assign to every pair (γ_1,γ_2) s.t. $\gamma_1\|\gamma_2 = w$, a value in $\{1,2\}$. For every $g \in G$ let $B^1_g := \{\gamma_1 \mid g((\gamma_1,\gamma_2)) = 1\}$ and $B^2_g := \{\gamma_2 \mid g((\gamma_1,\gamma_2)) = 2\}$.

Then the expression is equivalent to

$$
\begin{aligned}
&(t_1 \models \Diamond(\epsilon) \;\wedge\; t_2 \models \Phi) \vee \\
&(t_1 \models \Phi \;\wedge\; t_2 \models \Diamond(\epsilon)) \vee \\
&\exists \sigma_1, \sigma_2, t_1', t_2'.\ t_1 \xrightarrow{\sigma_1} t_1' \wedge t_2 \xrightarrow{\sigma_2} t_2' \;\wedge \bigvee_{f \in F} (t_1' \models -A^1_f \;\wedge\; t_2' \models -A^2_f) \wedge \\
&\qquad \bigvee_{\beta_1 \| \beta_2 = v} (t_1' \models \Diamond(\beta_1) \;\wedge\; t_2' \models \Diamond(\beta_2)) \wedge \\
&\qquad \bigvee_{g \in G} \left(\bigwedge_{\gamma_1 \in B^1_g} t_1' \models \neg\Diamond(\gamma_1) \;\wedge\; \bigwedge_{\gamma_2 \in B^2_g} t_2' \models \neg\Diamond(\gamma_2) \right)
\end{aligned}
$$

This is equivalent to

$$
\begin{aligned}
&(t_1 \models \Diamond(\epsilon) \;\wedge\; t_2 \models \Phi) \vee \\
&(t_1 \models \Phi \;\wedge\; t_2 \models \Diamond(\epsilon)) \vee \\
&\exists \sigma_1, \sigma_2, t_1', t_2'.\ t_1 \xrightarrow{\sigma_1} t_1' \wedge t_2 \xrightarrow{\sigma_2} t_2' \;\wedge \\
&\bigvee_{f \in F, \beta_1\|\beta_2 = v, g \in G} \quad t_1' \models -A^1_f \wedge t_1' \models \Diamond(\beta_1) \wedge \bigwedge_{\gamma_1 \in B^1_g} t_1' \models \neg\Diamond(\gamma_1) \wedge \\
&\qquad\qquad\qquad t_2' \models -A^2_f \wedge t_2' \models \Diamond(\beta_2) \wedge \bigwedge_{\gamma_2 \in B^2_g} t_2' \models \neg\Diamond(\gamma_2)
\end{aligned}
$$

Finally, this is equivalent to

$$
\begin{aligned}
&(t_1 \models \Diamond(\epsilon) \ \wedge \ t_2 \models \Phi) \vee \\
&(t_1 \models \Phi \ \wedge \ t_2 \models \Diamond(\epsilon)) \vee \\
&\bigvee_{f \in F, \beta_1 \| \beta_2 = v, g \in G} \quad t_1 \models \Diamond(-A_f^1 \wedge \Diamond(\beta_1) \wedge \bigwedge_{\gamma_1 \in B_g^1} \neg\Diamond(\gamma_1)) \wedge \\
&\qquad\qquad\qquad\qquad t_2 \models \Diamond(-A_f^2 \wedge \Diamond(\beta_2) \wedge \bigwedge_{\gamma_2 \in B_g^2} \neg\Diamond(\gamma_2))
\end{aligned}
$$

This is a boolean combination of properties of t_1 and properties of t_2.

Now we show how the decomposition is done in the general case. In order to simplify the presentation, we define the following sets of expressions. Let $\mathcal{DC}$ be a set of decomposable constraints, $T \subset \mathcal{T}$ a finite set of process terms and $d \in \mathbb{N}$.

$$
\begin{aligned}
\mathit{Cform}(d, T, \mathcal{DC}) &:= \left\{ \begin{array}{l} \left(\bigwedge_{i \in I} t_i \models \Diamond_{C_i} \Phi_i \ \wedge \ \bigwedge_{j \in J} t'_j \models \neg\Diamond_{D_j} \Psi_j \right) \ \mid \\ \forall i, j.\ t_i, t'_j \in T, C_i, D_j \in \mathcal{DC}, \Phi_i \in \mathcal{F}_d^c, \Psi_j \in \mathcal{F}_{d-1}^c \end{array} \right\} \\
\mathit{Cform}'(d, T, \mathcal{DC}) &:= \text{like } \mathit{Cform}(d, T, \mathcal{DC}), \text{ except that } \Psi_j \in \mathcal{F}_d^c \\
\mathit{Dform}(d, T, \mathcal{DC}) &:= \left\{ \bigvee_{i \in I} F_i \ \mid \ F_i \in \mathit{Cform}(d, T, \mathcal{DC}) \right\}
\end{aligned}
$$

The next two lemmas show the decomposition of properties for sequential composition. The general idea is that properties of the form $t_1.t_2 \models \Diamond_C \Phi$ are decomposed into properties of t_1 and properties of t_2. However, the details are more complex. It does not always suffice to use properties of t_1 and properties of t_2, but sometimes also properties of other terms are needed. These other terms are the terms that occur in Φ as atomic propositions and the terms that occur in the rules of the PAD-process. Fortunately, these are only finitely many.

We defined that sequential composition is left-associative, so if we write $t_1.t_2$, then the term t_2 is either a single constant or a parallel composition. The following lemma describes the decomposition for the case that t_2 is a single constant.

Lemma 21 *Let t be a process term, X a process constant, Δ a PAD, Φ a formula in $\mathcal{F}_d^c$ that contains only constraints from a set $\mathcal{DC}$ of decomposable constraints and $C \in \mathcal{DC}$. Let $T := \{\epsilon, t, X\} \cup \mathit{terms}(\Phi) \cup \{r \mid (l \xrightarrow{a} r) \in \Delta\}$*

Then an expression $F \in Dform(d, T, \mathcal{DC})$ can be effectively constructed s.t.

$$t.X \models \Diamond_C \Phi \iff F$$

PROOF. *by induction on d.*

$\Phi = (T^+ \wedge -T^- \wedge \bigwedge_{i\in I} \Diamond_{C_i}\Phi_i \wedge \bigwedge_{j\in J} \neg\Diamond_{D_j}\Psi_j)$ for some $T^+, T^- \subset \mathcal{T}$, $\Phi_i, \Psi_j \in \mathcal{F}^c_{d-1}$ and $C_i, D_j \in \mathcal{DC}$ decomposable constraints. In the base case $d = 0$ the sets I and J are empty and we solve the problem without referring to the induction hypothesis.

If $|T^+| \geq 2$ then $t.X \models \Diamond_C \Phi$ is equivalent to false.

If $|T^+| = 1$ s.t. the term in T^+ is not $t'.X$ for some t', then $t.X \models \Diamond_C \Phi$ is equivalent to

$$\bigvee_{i\in I} (t \models \Diamond_{C_1^i}(\epsilon) \wedge X \models \Diamond_{C_2^i}\Phi)$$
$$\vee$$
$$\bigvee_{j\in J,\ \left(l.X \xrightarrow{a} r\right)\in\Delta} (t \models \Diamond_{D_1^j}(l) \wedge r \models \Diamond_{D_2^j}\Phi)$$

where the C_k^i, D_k^j are the decompositions of C as defined in Def. 10 (cases 3 and 5). This expression is the F that we are looking for. It is in $Dform(d, T, \mathcal{DC})$.

Now we consider the case that $T^+ = \{u.X\}$ for some term u. If $u.X \in T^-$ then $t.X \models \Diamond_C \Phi$ is equivalent to false. Otherwise $t.X \models \Diamond_C \Phi$ is equivalent to

$$\bigvee_{i\in I} (t \models \Diamond_{C_1^i}(\epsilon) \wedge X \models \Diamond_{C_2^i}\Phi)$$
$$\vee$$
$$\bigvee_{j\in J,\ \left(l.X \xrightarrow{a} r\right)\in\Delta} (t \models \Diamond_{D_1^j}(l) \wedge r \models \Diamond_{D_2^j}\Phi)$$
$$\vee$$
$$t \models \Diamond_C(u) \wedge \bigwedge_{i\in I} u.X \models \Diamond_{C_i}\Phi_i \wedge \bigwedge_{j\in J} u.X \models \neg\Diamond_{D_j}\Psi_j$$

where the C_k^i, D_k^j are the decompositions of C as defined in Def. 10 (cases 3 and 5). This is the expression F that we are looking for. It is in $Dform(d, T, \mathcal{DC})$.

Now we consider the case that $T^+ = \{\}$. Then $t.X \models \Diamond_C \Phi$ is equivalent to

$$\bigvee_{i \in I} (t \models \Diamond_{C_1^i}(\epsilon) \wedge X \models \Diamond_{C_2^i}\Phi)$$

$$\vee$$

$$\bigvee_{j \in J,\ \left(l.X \xrightarrow{a} r\right) \in \Delta} (t \models \Diamond_{D_1^j}(l) \wedge r \models \Diamond_{D_2^j}\Phi)$$

$$\vee$$

$$\exists \sigma, t'. \begin{pmatrix} t \xrightarrow{\sigma} t' \wedge C(\sigma) \wedge (\forall (\alpha.X) \in T^-.\ t' \neq \alpha) \wedge \\ \bigwedge_{i \in I} t'.X \models \Diamond_{C_i}\Phi_i \wedge \bigwedge_{j \in J} t'.X \models \neg\Diamond_{D_j}\Psi_j \end{pmatrix}$$

where the C_k^i, D_k^j are the decompositions of C as defined in Def. 10 (cases 4 and 5).

If $d = 0$ then $I = J = \{\}$ and the induction hypothesis is not needed. If $d > 0$ then by induction hypothesis there are expressions $F_i, G_j \in Dform(d-1, (T-\{t\}) \cup \{t'\}, \mathcal{DC})$ s.t. the above expression is equivalent to

$$\bigvee_{i \in I} (t \models \Diamond_{C_1^i}(\epsilon) \wedge X \models \Diamond_{C_2^i}\Phi)$$

$$\vee$$

$$\bigvee_{j \in J,\ \left(l.X \xrightarrow{a} r\right) \in \Delta} (t \models \Diamond_{D_1^j}(l) \wedge r \models \Diamond_{D_2^j}\Phi)$$

$$\vee$$

$$\exists \sigma, t'.\ t \xrightarrow{\sigma} t' \wedge C(\sigma) \wedge (\forall (\alpha.X) \in T^-.\ t' \neq \alpha)\ \wedge \bigwedge_{i \in I} F_i \wedge \bigwedge_{j \in J} \neg G_j$$

By transformation to disjunctive normal form there are finite index sets K, N_k, N'_k, M_k and formulae $\Phi'_i, \Psi'_i \in \mathcal{F}^c_{d-1}$ and decomposable constraints $E_i, E'_i \in \mathcal{DC}$ and $H_j \in Cform'(d-1, T-\{t\}, \mathcal{DC})$ s.t. the above expression is equivalent *to*

$$\bigvee_{i\in I}(t \models \Diamond_{C_1^i}(\epsilon) \wedge X \models \Diamond_{C_2^i}\Phi)$$

$$\vee$$

$$\bigvee_{j\in J,\ \left(l.X \xrightarrow{a} r\right)\in\Delta} (t \models \Diamond_{D_1^j}(l) \wedge r \models \Diamond_{D_2^j}\Phi)$$

$$\vee$$

$$\exists \sigma, t'. \left(\begin{array}{l} t \xrightarrow{\sigma} t' \wedge C(\sigma) \wedge (\forall (\alpha.X) \in T^{-}.\ t' \neq \alpha) \wedge \\ \bigvee_{k\in K} \left[\bigwedge_{i\in N_k} t' \models \Diamond_{E_i}\Phi'_i \bigwedge_{i\in N'_k} t' \models \neg\Diamond_{E'_i}\Psi'_i \bigwedge_{j\in M_k} H_j \right] \end{array} \right)$$

Note that the expressions H_j do not contain the terms t or t'. This is equivalent to

$$\bigvee_{i\in I}(t \models \Diamond_{C_1^i}(\epsilon) \wedge X \models \Diamond_{C_2^i}\Phi)$$

$$\vee$$

$$\bigvee_{j\in J,\ \left(l.X \xrightarrow{a} r\right)\in\Delta} (t \models \Diamond_{D_1^j}(l) \wedge r \models \Diamond_{D_2^j}\Phi)$$

$$\vee$$

$$\bigvee_{k\in K} \left[t \models \Diamond_C \left(-\{\alpha \mid \alpha.X \in T^{-}\} \wedge \bigwedge_{i\in N_k} \Diamond_{E_i}\Phi'_i \bigwedge_{i\in N'_k} \neg\Diamond_{E'_i}\Psi'_i \right) \wedge \bigwedge_{j\in M_k} H_j \right]$$

This is the expression F that we are looking for. It is in $Dform(d, T, \mathcal{DC})$.

The following lemma does the same decomposition for the case that the second component in the sequential composition is itself a parallel composition.

Lemma 22 *Let t_1, t_2, t_3 be process terms, Δ a PAD, Φ a formula in $\mathcal{F}_d^c$ that contains only constraints from a set $\mathcal{DC}$ of decomposable constraints and $C \in \mathcal{DC}$. Let $T := \{\epsilon, t_1, t_2 \| t_3\} \cup terms(\Phi) \cup \{r \mid (l \xrightarrow{a} r) \in \Delta\}$*

Then an expression $F \in Dform(d, T, \mathcal{DC})$ can be effectively constructed s.t.

$$t_1.(t_2 \| t_3) \models \Diamond_C \Phi \iff F$$

PROOF. *The proof is similar to Lemma 21 with only the following differences:*

(1) Leave out the part

$$\bigvee_{j\in J,\ \left(l.X \xrightarrow{a} r\right)\in\Delta} t \models \Diamond_{D_1^j}(l) \wedge r \models \Diamond_{D_2^j}\Phi$$

of the disjunction, and

(2) Substitute $(t_2 \| t_3)$ for X everywhere. $\square$

Now we show an analogous property for parallel composition.

Lemma 23 *Let t_1, t_2 be process terms, Δ a PAD, Φ a formula in $\mathcal{F}_d^c$ that contains only constraints from a set $\mathcal{DC}$ of decomposable constraints and $C \in \mathcal{DC}$. Let $T := \{\epsilon, t_1, t_2\} \cup terms(\Phi)$.*

Then an expression $F \in Dform(d, T, \mathcal{DC})$ can be effectively constructed s.t.

$$t_1 \| t_2 \models \Diamond_C \Phi \iff F$$

PROOF. *by induction on d.*
Φ has the form $(T^+ \wedge -T^- \wedge \bigwedge_{i\in I} \Diamond_{C_i}\Phi_i \wedge \bigwedge_{j\in J} \neg\Diamond_{D_j}\Psi_j)$ for some $T^+, T^- \subset \mathcal{T}$ and $\Phi_i, \Psi_j \in \mathcal{F}_{d-1}^c$.

If $|T^+| \geq 2$ then $\Diamond_C\Phi$ is equivalent to false.

Now we consider the case that $T^+ = \{t\}$ for some term t. If $t \in T^-$ then $t_1 \| t_2 \models \Diamond_C\Phi$ is equivalent to false. Otherwise it is equivalent to

$$\bigvee_{k\in K} \begin{pmatrix} (t_1 \models \Diamond_{C'_k}(\epsilon) \wedge t_2 \models \Diamond_{C''_k}\Phi) \vee \\ (t_2 \models \Diamond_{C'_k}(\epsilon) \wedge t_1 \models \Diamond_{C''_k}\Phi) \end{pmatrix}$$

$$\vee$$

$$\bigvee_{l\in L} \bigvee_{\alpha_1\|\alpha_2 = t} \begin{pmatrix} t_1 \models \Diamond_{D'_l}(\alpha_1) \wedge t_2 \models \Diamond_{D''_l}(\alpha_2) \wedge \\ \bigwedge_{i\in I} t \models \Diamond_{C_i}\Phi_i \wedge \bigwedge_{j\in J} t \models \neg\Diamond_{D_j}\Psi_j \end{pmatrix}$$

where the C'_k, C''_k, D'_l, D''_l are the decompositions of C as defined in Def. 10 (case 6). This is the F that we are looking for. It is in $Dform(d, T, \mathcal{DC})$.

Now we consider the case that $T^+ = \{\}$. Then $t_1 \| t_2 \models \Diamond_C \Phi$ is equivalent to

$$\bigvee_{k \in K} \begin{pmatrix} (t_1 \models \Diamond_{C'_k}(\epsilon) \wedge t_2 \models \Diamond_{C''_k}\Phi) \vee \\ (t_2 \models \Diamond_{C'_k}(\epsilon) \wedge t_1 \models \Diamond_{C''_k}\Phi) \end{pmatrix}$$

$$\vee$$

$$\bigvee_{l \in L} \exists \sigma_1, \sigma_2, t'_1, t'_2. \begin{pmatrix} t_1 \stackrel{\sigma_1}{\to} t'_1 \wedge t_2 \stackrel{\sigma_2}{\to} t'_2 \wedge D'_l(\sigma_1) \wedge D''_l(\sigma_2) \wedge \\ \bigwedge_{\alpha_1 \| \alpha_2 \in T^-} (t'_1 \neq \alpha_1 \vee t'_2 \neq \alpha_2) \wedge \\ \bigwedge_{i \in I} t'_1 \| t'_2 \models \Diamond_{C_i} \Phi_i \wedge \bigwedge_{j \in J} t'_1 \| t'_2 \models \neg \Diamond_{D_j} \Psi_j \end{pmatrix}$$

where the C'_k, C''_k, D'_l, D''_l are the decompositions of C as defined in Def. 10 (case 6). In the base case $d = 0$ we have $I = J = \{\}$ and don't need the induction hypothesis. For $d > 0$, by induction hypothesis, there are formulae $F_i, G_j \in Dform(d-1, \{t'_1, t'_2\} \cup terms(\Phi), \mathcal{DC})$ such that $t'_1 \| t'_2 \models \Diamond_{C_i} \Phi_i \iff F_i$ and $t'_1 \| t'_2 \models \Diamond_{D_j} \Psi_j \iff G_j$. Now we transform the expression into disjunctive normal form. We define the set Func of all functions

$$f : \{(\alpha_1, \alpha_2) \mid \alpha_1 \| \alpha_2 \in T^-\} \mapsto \{1, 2\}$$

that assign to every pair (α_1, α_2) s.t. $\alpha_1 \| \alpha_2 \in T^-$, a value in $\{1, 2\}$. For every $f \in$ Func let $A^1_f := \{\alpha_1 \mid f((\alpha_1, \alpha_2)) = 1\}$ and $A^2_f := \{\alpha_2 \mid f((\alpha_1, \alpha_2)) = 2\}$. Then the expression is equivalent to

$$\bigvee_{k \in K} \begin{pmatrix} (t_1 \models \Diamond_{C'_k}(\epsilon) \wedge t_2 \models \Diamond_{C''_k}\Phi) \vee \\ (t_2 \models \Diamond_{C'_k}(\epsilon) \wedge t_1 \models \Diamond_{C''_k}\Phi) \end{pmatrix}$$

$$\vee$$

$$\bigvee_{l \in L} \exists \sigma_1, \sigma_2, t'_1, t'_2. \begin{pmatrix} t_1 \stackrel{\sigma_1}{\to} t'_1 \wedge t_2 \stackrel{\sigma_2}{\to} t'_2 \wedge D'_l(\sigma_1) \wedge D''_l(\sigma_2) \wedge \\ \bigvee_{f \in Func} (t'_1 \notin A^1_f \wedge t'_2 \notin A^2_f) \wedge \\ \bigwedge_{i \in I} F_i \wedge \bigwedge_{j \in J} \neg G_j \end{pmatrix}$$

By transformation to disjunctive normal form there must be finite index sets O and $M(o), M'(o), N(o), N'(o)$ for every $o \in O$ and formulae $\Phi'_n, \Psi'_{n'}, \Phi''_m, \Psi''_{m'} \in \mathcal{F}^c_{d-1}$ and decomposable constraints $E_n, E'_{n'}, F_m, F'_{m'} \in \mathcal{DC}$ s.t. the condition is *equivalent to*

$$\bigvee_{k\in K}\begin{pmatrix}(t_1 \models \Diamond_{C'_k}(\epsilon) \wedge t_2 \models \Diamond_{C''_k}\Phi) \vee \\ (t_2 \models \Diamond_{C'_k}(\epsilon) \wedge t_1 \models \Diamond_{C''_k}\Phi)\end{pmatrix}$$

$$\vee$$

$$\bigvee_{l\in L} \exists \sigma_1, \sigma_2, t'_1, t'_2. \begin{bmatrix} t_1 \xrightarrow{\sigma_1} t'_1 \wedge t_2 \xrightarrow{\sigma_2} t'_2 \wedge D'_l(\sigma_1) \wedge D''_l(\sigma_2) \wedge \\ \bigvee_{f\in Func} (t'_1 \notin A^1_f \wedge t'_2 \notin A^2_f) \wedge \\ \bigvee_{o\in O}\begin{pmatrix} \bigwedge_{n\in N(o)} t'_1 \models \Diamond_{E_n}\Phi'_n \bigwedge_{n'\in N'(o)} t'_1 \models \neg\Diamond_{E'_{n'}}\Psi'_{n'} \\ \bigwedge_{m\in M(o)} t'_2 \models \Diamond_{F_m}\Phi''_m \bigwedge_{m'\in M'(o)} t'_2 \models \neg\Diamond_{F'_{m'}}\Psi''_{m'}\end{pmatrix}\end{bmatrix}$$

This is equivalent to

$$\bigvee_{k\in K}\begin{pmatrix}(t_1 \models \Diamond_{C'_k}(\epsilon) \wedge t_2 \models \Diamond_{C''_k}\Phi) \vee \\ (t_2 \models \Diamond_{C'_k}(\epsilon) \wedge t_1 \models \Diamond_{C''_k}\Phi)\end{pmatrix}$$

$$\vee$$

$$\bigvee_{l\in L, f\in Func, o\in O}\begin{bmatrix} t_1 \models \Diamond_{D'_l}\left(-A^1_f \wedge \bigwedge_{n\in N(o)} \Diamond_{E_n}\Phi'_n \bigwedge_{n'\in N'(o)} \neg\Diamond_{E'_{n'}}\Psi'_{n'}\right) \wedge \\ t_2 \models \Diamond_{D''_l}\left(-A^2_f \wedge \bigwedge_{m\in M(o)} \Diamond_{F_m}\Phi''_m \bigwedge_{m'\in M'(o)} \neg\Diamond_{F'_{m'}}\Psi''_{m'}\right)\end{bmatrix}$$

This is the expression F that we are looking for. It is in $Dform(d, T, \mathcal{DC})$.

4.2 The Tableau System

We show the decidability of the model checking problem for PAD and $EF^{=}_{DC}$ by induction on the nesting-depth d of the formula. We describe a tableau system that solves the model checking problem for formulae $\Diamond_C\Phi$ with $\Phi \in \mathcal{F}^c_d$ under the condition that we can already solve the problem for formulae $\Diamond_C\Psi$ with $\Psi \in \mathcal{F}^c_{d-1}$. This is because we use properties of the form $t' \models \Diamond_C\Psi$ for $\Psi \in \mathcal{F}^c_{d-1}$ as side conditions in the construction of the tableau. By induction hypothesis we can assume this. In the base case of $d = 0$ the condition is trivially satisfied, as $\mathcal{F}^c_{-1} = \{\}$.

Every node in the tableau is a set of expressions of the form $t \vdash \Phi$, where t is a process term and Φ an $EF^{=}_{DC}$-formula. We use the symbol $\vdash$ in the tableau instead of $\models$. The expression $t \vdash \Phi$ means that one attempts to prove the property $t \models \Phi$. The meaning of $t \models \Phi$ is defined semantically (Def. 8). The sets of expressions that form the tableau nodes are denoted by Γ and interpreted as sets of subgoals that should be proved. These subgoals are interpreted conjunctively. The branches in the tableau are interpreted disjunctively, so the tableau is successful iff there is at least one successful branch. Every branch in the tableau can be seen as an attempt to construct a proof.

The following tableau rules are meant to be applied to a problem of the form $t \models \Diamond_C \Phi$ with $\Phi \in \mathcal{F}_d^c$. In the rules Induct1–Induct4 we apply the induction hypothesis that we can already solve the problem for formulae of a smaller nesting-depth.

SEQ1 $\quad \dfrac{\{t.X \vdash \Diamond_C \Phi\} \cup \Gamma}{\{F\} \cup \Gamma}$ where F is from Lemma 21

SEQ2 $\quad \dfrac{\{t_1.(t_2 \| t_3) \vdash \Diamond_C \Phi\} \cup \Gamma}{\{F\} \cup \Gamma}$ where F is from Lemma 22

PAR $\quad \dfrac{\{t_1 \| t_2 \vdash \Diamond_C \Phi\} \cup \Gamma}{\{F\} \cup \Gamma}$ where F is from Lemma 23

STEP1 $\quad \dfrac{\{X \vdash \Diamond_C \Phi\} \cup \Gamma}{\{X \vdash \Phi\} \cup \Gamma \quad \{\bigvee_{i \in I_1} t_1 \vdash \Diamond_{D_1^i} \Phi\} \cup \Gamma \ \ldots \ \{\bigvee_{i \in I_n} t_n \vdash \Diamond_{D_n^i} \Phi\} \cup \Gamma}$

if $C(\lambda)$, where λ is the empty sequence, $(X \stackrel{a_k}{\to} t_k) \in \Delta$, $k = 1, \ldots, n$

and the D_k^i are the decompositions of C (Def. 10 (case 4)).

STEP2 $\quad \dfrac{\{X \vdash \Diamond_C \Phi\} \cup \Gamma}{\{\bigvee_{i \in I_1} t_1 \vdash \Diamond_{D_1^i} \Phi\} \cup \Gamma \quad \ldots \quad \{\bigvee_{i \in I_n} t_n \vdash \Diamond_{D_n^i} \Phi\} \cup \Gamma}$

if not $C(\lambda)$, $(X \stackrel{a_k}{\to} t_k) \in \Delta$, $k = 1, \ldots, n$

and the D_k^i are decompositions of C (Def. 10 (case 4)).

Unsat $\quad \dfrac{\{t \vdash \Diamond_C \Phi\} \cup \Gamma}{\{\mathit{false}\}}$ if C is unsatisfiable

conj1 $\quad \dfrac{\{t \vdash \Phi \wedge \Psi\} \cup \Gamma}{\{t \vdash \Phi, t \vdash \Psi\} \cup \Gamma}$

conj2 $\quad \dfrac{\{F \wedge G\} \cup \Gamma}{\{F, G\} \cup \Gamma}$

disj1 $\quad \dfrac{\{t \vdash \Phi \vee \Psi\} \cup \Gamma}{\{t \vdash \Phi\} \cup \Gamma \quad \{t \vdash \Psi\} \cup \Gamma}$

disj2 $\dfrac{\{F \vee G\} \cup \Gamma}{\{F\} \cup \Gamma \quad \{G\} \cup \Gamma}$

Induct1 $\dfrac{\{t \vdash \Diamond_C \Psi\} \cup \Gamma}{\Gamma}$ if $\Psi \in \mathcal{F}^c_{d-1}$ and $t \models \Diamond_C \Psi$

Induct2 $\dfrac{\{t \vdash \Diamond_C \Psi\} \cup \Gamma}{\{\mathit{false}\}}$ if $\Psi \in \mathcal{F}^c_{d-1}$ and not $t \models \Diamond_C \Psi$

Induct3 $\dfrac{\{t \vdash \neg\Diamond_C \Psi\} \cup \Gamma}{\Gamma}$ if $\Psi \in \mathcal{F}^c_{d-1}$ and not $t \models \Diamond_C \Psi$

Induct4 $\dfrac{\{t \vdash \neg\Diamond_C \Psi\} \cup \Gamma}{\{\mathit{false}\}}$ if $\Psi \in \mathcal{F}^c_{d-1}$ and $t \models \Diamond_C \Psi$

Term1 $\dfrac{\{t \vdash T^+\} \cup \Gamma}{\Gamma}$ if $T^+ = \{t\}$ or $T^+ = \{\}$

Term2 $\dfrac{\{t \vdash T^+\} \cup \Gamma}{\{\mathit{false}\}}$ if $T^+ \neq \{\} \wedge T^+ \neq \{t\}$

Term3 $\dfrac{\{t \vdash -T^-\} \cup \Gamma}{\Gamma}$ if $t \notin T^-$

Term4 $\dfrac{\{t \vdash -T^-\} \cup \Gamma}{\{\mathit{false}\}}$ if $t \in T^-$

In order to avoid any unnecessary growth of the proof tree, we define that the rules with names in capital letters (PAR, SEQ1, SEQ2, STEP1 and STEP2) have a **lower** precedence than the other rules. So in the construction of a branch of the proof tree we only use such a rule if none of the others is applicable.

Lemma 24 *For any instance of a tableau-rule, the antecedent is true iff at least one of the succedents is true.*

PROOF. *This follows immediately from the definition of the tableau-rules and Lemma 21, Lemma 22 and Lemma 23.*

Definition 25 (Termination conditions)
A node in the tableau consisting of a set of formulae Γ *is a* terminal node *if one of the following conditions is satisfied:*

(1) $\Gamma = \{\}$
(2) *false* $\in \Gamma$.
(3) *There is a previous node in the same branch that is marked with the same set* Γ.

Terminal nodes of type 1 are successful, while terminal nodes of types 2,3 are unsuccessful.

The construction of a branch of the tableau stops when a terminal node is reached. The branch is successful if this terminal node is successful. The tableau is successful if there is at least one successful branch.

The intuition is that every branch in the tableau is an attempt to construct a proof. A terminal node of type 1 means that all subgoals have been solved. A terminal node of type 2 means that this attempt to construct a proof failed. A terminal node of type 3 means that the proof is 'running in circles'. If there is a proof, then it can be found elsewhere in the tableau by a shorter branch.

The construction of the tableau starts with a root-node of the form $\{t \vdash \Diamond_C \Phi\}$ where t is a process term and $\Phi \in \mathcal{F}_d^c$. The tableau for a given root is not unique, because the sequents are sets of expressions and the element to which a rule is applied is chosen nondeterministically. However, all tableaux are equivalent semantically, because the order in which subgoals are solved does not matter.

4.3 Decidability

In this section we show that the tableau system of the previous section is sound and complete and produces only finite tableaux for any given root. Thus it yields a decision procedure for the model checking problem for PAD and $EF_{DC}^{=}$.

Lemma 26 *If the root node has the form* $\{t \vdash \Diamond_C \Phi\}$*, for* $\Phi \in \mathcal{F}_d^c$*, then for every node in a tableau with this root at least one of the following conditions is satisfied:*

(1) A tableau rule is applicable
(2) The node is a terminal node.

PROOF. *The only problematic cases are the expressions of the form* $t \vdash \neg\Diamond_C \Phi$*. If such an expression occurs, then it must be due to the rules SEQ1, SEQ2 or STEP1. By definition of these rules and Lemma 21 and Lemma 22 we know that* $\Phi \in \mathcal{F}_{d-1}^c$*. Then the rules Induct3 or Induct4 are applicable,* *because we assumed (by induction hypothesis) that we can already solve the problem for formulae of a smaller nesting depth.*

Lemma 27 *The tableau is finite for every instance of the model checking problem.*

PROOF. *If only process terms of a bounded size are used as atomic propositions, then there are only finitely many formulae in $\mathcal{F}_d^c$ for any fixed d. The tableau rules and the proofs of Lemmas 21, 22 and 23 show that this precondition is satisfied. Any set $\mathcal{DC}$ of decomposable constraints is finite. There are only finitely many rules $(t_1 \xrightarrow{a} t_2) \in \Delta$ with only finitely many subterms of the terms t_2. So there are only finitely many different sets of expressions of the form $t \vdash \Phi$ in the tableau. Therefore the branches of the tableau can only have finite length, because of termination condition 3. Since the tableau is finitely branching, the result follows.*

Now we prove the soundness and completeness of the tableau. The following lemma shows the soundness.

Lemma 28 *Let $\Phi \in \mathcal{F}_d^c$ and $C \in \mathcal{DC}$. If there is a successful tableau with root $\{t \vdash \Diamond_C \Phi\}$, then $t \models \Diamond_C \Phi$.*

PROOF. *A successful tableau has a successful branch ending with a node marked by the empty set of expressions. Since these sets are interpreted conjunctively this node is true. By repeated application of Lemma 24 all its ancestor-nodes must be true and thus the root-node must be true.*

We need some new definitions to show the completeness of the tableau system.

Definition 29 *A* valid sequent Γ *in a tableau is a set of expressions which evaluate to true.*

For example if $(t \vdash \Diamond_C \Phi) \in \Gamma$ then $t \models \Diamond_C \Phi$. If $(F \wedge G) \in \Gamma$ then F and G evaluate to true.

It follows from the construction of the tableau system that every expression in a valid sequent is a disjunction of conjunctions of expressions of the form $t \vdash \Diamond_C \Phi$ or $t \vdash \neg \Diamond_C \Phi$.

Now we define a total order on valid sequents.

Definition 30 *For an expression $t \vdash \Diamond_C \Phi$ with $t \models \Diamond_C \Phi$ we define*

$$xnorm(t \vdash \Diamond_C \Phi) := min\{length(\sigma) \mid t \xrightarrow{\sigma} t' \in [\![\Phi]\!] \wedge C(\sigma)\}$$

and

$$ynorm(t \vdash \Diamond_C \Phi) := size(t)$$

For an expression F in a valid sequent we define

$$xnorm(F) := max\{xnorm(t \vdash \Diamond_C \Phi) \mid t \vdash \Diamond_C \Phi \text{ is subterm of } F,\ nd(\Phi) = d\}$$

and

$$ynorm(F) := max\{ynorm(t \vdash \Diamond_C \Phi) \mid t \vdash \Diamond_C \Phi \text{ is subterm of } F,\ nd(\Phi) = d\}$$

and

$$znorm(F) := size(F)$$

where $size(F)$ is just the number of letters/symbols needed to write F. The norm of F is a triple, which is defined by

$$norm(F) := (xnorm(F), ynorm(F), znorm(F))$$

These norms are ordered lexicographically. The order is well-founded.

For a valid sequent Γ let

$$\gamma_{x,y,z} := |\{F \in \Gamma \mid norm(\Gamma) = (x, y, z)\}|$$

Since Γ is valid and finite, there is a largest x s.t. $\gamma_{x,y,z} \neq 0$ for some y, z. This largest x will be called x_{max}. It depends on Γ. Also for every $x \leq x_{max}$ there is a largest y (called $y(x)$) s.t. $\gamma_{x,y,z} \neq 0$ for some z. Finally, for every x, y there is a largest $z(x, y)$ s.t. $\gamma_{x,y,z} \neq 0$.

We define a well-founded ordering on valid sequents. Let Γ and Γ' be two valid sequents and $\gamma_{x,y,z}$ and $\gamma'_{x,y,z}$ be defined as above. Then

$$\Gamma < \Gamma' \ :\Leftrightarrow\ \exists(x, y, z).\gamma_{x,y,z} < \gamma'_{x,y,z} \ \wedge\ \forall(x', y', z') \geq_{lex} (x, y, z).\ \gamma_{x',y',z'} = \gamma'_{x',y',z'}$$

The intuition is that if a tableau-rule is applied to a valid sequent Γ, then there is at least one valid succedent sequent that is smaller. This is because an expression $F \in \Gamma$ is replaced with several others with a lower norm. Since the ordering is well-founded, the process must eventually terminate.

Note that these definitions do not apply to non-valid sequents.

Lemma 31 *Let Γ be a valid sequent. Then every tableau with root Γ has at least one successful branch that ends with the empty sequent.*

PROOF. *By Lemma 24 every tableau with root Γ has at least one branch that only contains valid sequents. Choose one such branch of minimal length. We show that the order of the sequents on this branch must strictly decrease. We do this by showing that every application of a tableau rule to a valid sequent yields a smaller sequent.*

SEQ1,SEQ2 *It follows from the construction of the expressions in Lemma 21 and Lemma 22 that in these expressions one of two cases holds:*
(1) The remaining sequence is shorter (lower xnorm) or
(2) The remaining sequence has the same length and the terms are smaller (lower ynorm).
Thus the succedent sequent is smaller.

PAR *It follows from the construction of the expression in Lemma 23 that the terms are always smaller (since t_1, t_2 are smaller than $t_1 \| t_2$). The xnorm is the same or smaller and the ynorm is smaller. Thus the succedent is smaller.*

STEP1,STEP2 *Here we have two sub-cases:*

- *In the first branch of the rule STEP1 the sequence has length 0. In the succedent the xnorm and ynorm are the same, but the znorm is smaller.*
- *In the other branches of STEP1 and all branches of STEP2 we choose the valid succedent that corresponds to the shortest sequence that leads to a state that satisfies Φ. In this succedent the sequence is shorter and thus the xnorm is smaller.*

In both cases the succedent is smaller.

Unsat *This rule is never applied in this branch, because all sequents are valid.*

conj1,conj2 *For these rules the succedent is smaller, because the znorm decreases.*

disj1,disj2 *For these rules the succedent is smaller, because the znorm decreases.*

Induct,Term *For the rules Induct1,Induct3 and Term1,Term3 the succedent must be smaller, because expressions are removed from the sequent. The rules Induct2,Induct4,Term2,Term4 are never applied in this branch, because all sequents are valid.*

The construction of this branch cannot be stopped by termination condition 3, because the order strictly decreases. Since the order of the sequents strictly decreases on this branch, it must eventually end with the empty sequent and thus it is successful.

Corollary 32 *If $t \models \Diamond_C \Phi$ for $\Phi \in \mathcal{F}_d^c$ and $C \in \mathcal{DC}$ then every tableau with root $\{t \vdash \Diamond_C \Phi\}$ is successful.*

PROOF. *The root-sequent is valid. By Lemma 31 every tableau must have a branch that ends with the empty sequent. This branch is successful and thus the tableau is successful.*

Lemma 33 *Let t be a process term, Δ a PAD, $\Phi \in \mathcal{F}_d^c$, $\mathcal{DC}$ a set of decomposable constraints and $C \in \mathcal{DC}$. Then the following conditions are equivalent:*

- $t \models \Diamond_C \Phi$
- *A tableau with root $\{t \vdash \Diamond_C \Phi\}$ is successful.*
- *Every tableau with root $\{t \vdash \Diamond_C \Phi\}$ is successful.*

PROOF. *Directly from Lemma 28 and Corollary 32.*

Theorem 34 *The model checking problem for $EF^{=}_{DC}$ and PAD is decidable.*

PROOF. *By Lemma 19 it suffices to prove decidability for formulae of the form $\Diamond_C \Phi$ with Φ in $\mathcal{F}_d^c$ for any d. We prove this by induction on d. By Lemma 33 and Lemma 27 it suffices to construct a finite tableau. During the construction we must decide problems of the form $t' \models \Diamond_C \Psi$ for $\Psi \in \mathcal{F}_{d-1}^c$. In the base case $d = 0$ this is trivial, since $\mathcal{F}_{-1}^c = \emptyset$. For $d > 0$ this is possible by induction hypothesis.*

Since EF is weaker than $EF^{=}_{DC}$, we get the following corollary.

Corollary 35 *Model checking PAD with EF is decidable.*

5 Example

In this section we describe a small example of the model checking problem for EF and PAD. The PAD-process is described by the following set of rules Δ:

$$\begin{aligned} X &\xrightarrow{a} (Y \| X).Z \\ Y &\xrightarrow{b} \epsilon \\ X.Z &\xrightarrow{c} X \end{aligned}$$

The initial state is X. By using the algorithm derived from the tableau system described in Section 4 we can show a property of the process X.

$$X \models \Box\Diamond\langle a\rangle true \;\wedge\; \Box\Diamond\langle c\rangle true \;\wedge\; \neg\Diamond\langle a\rangle\langle c\rangle true$$

This means that process X can always get back into states where it can do action "a" or action "c", but never a "c" directly after an "a".

6 Parallel Pushdown Automata

Parallel Pushdown Automata (PPDA) are defined as the pushdown extension of BPP. They are the class of systems that can be described by a synchronization of a BPP with a finite-state system. In the framework of PRS they can be described as follows: Let $R := \{X_1, \ldots, X_k\} \subset \mathit{Const}$ be the process constants that represent the states in the finite state system. Then a PPDA is a PRS where all rules in Δ have the form

$$X_i \| Y \xrightarrow{a} X_j \| t$$

where $X_i, X_j \in R$, $Y \in \mathit{Const}$ and $t \in P$ is a parallel composition of constants that does not contain any constants from R. This is a subclass of Petri net (type (P, P)) rules. In the case of sequential composition the same construction yields pushdown automata, which are equivalent to (S, S)-PRS [8]. However, PPDA are slightly weaker than Petri nets w.r.t. bisimulation.

We prove that model checking with EF is undecidable for PPDA by showing that the proof of undecidability for Petri nets carries over to PPDA. Undecidability of model checking Petri nets with EF was first proved by Esparza in [10]. The proof there contains a slight error, which was corrected in [11]. The idea is to prove undecidability by reduction from the *reachability set containment problem.*

<u>REACHABILITY SET CONTAINMENT</u>

Instance: Two Petri nets N_1 and N_2 having the same number of places and a bijection f between the sets of places of N_1 and N_2. f can be extended to a bijection on markings in the obvious way.

Question: Is it true that for every reachable marking M of N_1, $f(M)$ is a reachable marking of N_2 ?

Rabin showed that this problem is undecidable by reduction of Hilbert's 10th problem. Later Jančar [13,14] gave a more direct proof by a reduction from the halting problem for counter machines.

We sketch the reduction of the reachability set containment problem to the model checking problem. It is similar to the one in [11], but slightly simpler. We assume that the transitions in the Petri nets N_1, N_2 are not labeled with atomic actions.

Figure 2 illustrates the following construction.

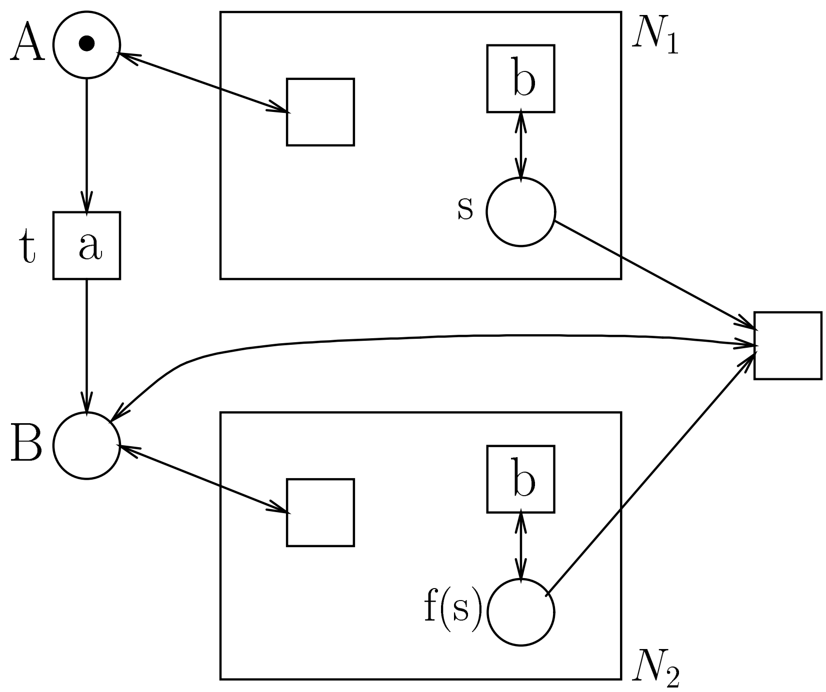

Fig. 2. Reducing reachability set containment to model checking with EF

(1) Put N_1 and N_2 side by side.
(2) Add a place A and arcs from A to every transition in N_1 and back. Put one token on A.
(3) Add a new transition t and a place B and arcs from A to t and from t to B. The transition t is labeled with the atomic action a. Place B is initially unmarked.
(4) Add arcs from B to every transition in N_2 and back.
(5) For every pair of places $(s, f(s))$ add a transition t_s and arcs from s to t_s, $f(s)$ to t_s, B to t_s and t_s to B.
(6) For every place s in N_1 add a transition t'_s labeled with action b and arcs from s to t'_s and back. Do the same for N_2.

Proposition 36 *An instance of the reachability set containment problem has answer 'yes' iff the newly constructed Petri net satisfies the EF-formula*

$$\Box(\neg a \ \vee \ \Diamond(\neg a \ \wedge \ \neg b))$$

Now we construct a PPDA that weakly simulates the Petri net of Figure 2. Assign a unique process constant to every place in this net. Every transition t in the net then corresponds to a rule $Y_1 \| \ldots \| Y_n \xrightarrow{a} Y'_1 \| \ldots Y'_m$. (These constants need not be pairwise different.) Now replace every such rule by the following rules:

$$
\begin{aligned}
X\|Y_1 &\xrightarrow{\tau} X_1 \\
X_i\|Y_{i+1} &\xrightarrow{\tau} X_{i+1} \quad \text{for i=1, \ldots ,n-2} \\
X_{n-1}\|Y_n &\xrightarrow{a} X\|Y'_1\| \ldots Y'_m
\end{aligned}
$$

where τ is a new action. The X_j are states of the finite control and are specific for every transition t. X is also a state of the finite control, but it is global and only exists once. The intuition is that these rules simulate the original transition in n steps. The initial state of the PPDA is $X\|I$, where I is the initial state of the Petri net.

Finally, we add one rule $X \xrightarrow{\rho} X$, where ρ is a new action. So action ρ is enabled iff the finite control is in state X. This simulation can get stuck, in case there were not enough tokens there to fire the transition in the first place. We call a state in the simulation 'faithful' if it is not forced to get stuck, i.e., it can get back to a state where ρ is enabled again. This can be expressed by the formula $\Diamond(\rho)$.

Theorem 37 *Model checking with EF is undecidable for PPDA.*

PROOF. *An instance of the containment problem has answer 'yes' iff the PPDA simulation of the net in Figure 2 satisfies*

$$\Box(\neg\Diamond(\rho) \vee \neg a \ \vee \ \Diamond(\rho \ \wedge \ \neg a \ \wedge \ \neg b))$$

7 Conclusion

We have shown decidability of the model checking problem for the branching-time temporal logic EF and the process model PAD. The exact complexity of the problem is an open question. The problem is known to be *PSPACE*-complete for the special cases of BPP [23,19] and BPA [2,24]. Model checking pushdown processes with EF is decidable in *EXPTIME* and *PSPACE*-hard [2,29]. It is even *PSPACE*-hard in the size of the system for a small fixed EF-formula. The complexity for PA and PAD is an open question. The two completely different algorithms for PA by Mayr [21] and by Lugiez and Schnoebelen [17] both have the same extremely high complexity of $\mathcal{O}(tower(n))$. The algorithm for PAD described in this paper is a generalization of the one in [21], but not a generalization of the algorithm for BPP in [19]. The *PSPACE*-algorithm for BPP in [23,19] uses a bounded search, while the algorithm for PAD works by decomposition. For a formula of nesting-depth d the complexity of the algorithm derived from the tableau system is d-times exponential. This is because the tableau has a branching degree that is d-times exponential for EF-formulae of nesting depth d. Also there are d-times exponentially many different EF-formulae of nesting depth d. So the overall complexity of the algorithm is $O(tower(n))$, where $tower(0) := 0$ and $tower(i+1) := 2^{tower(i)}$.

The best known lower bound for both PAD and PA is *PSPACE*-hardness, but there is a slight difference. For PAD the problem is *PSPACE*-hard in the size of the system for a fixed formula, because this holds for pushdown processes [2] and PAD subsumes pushdown processes. PA does not subsume pushdown processes and the best known lower bound is the same as for BPP: The problem is Σ_d^p-hard for formulae of nesting depth $\leq d$ [19].

Finally, model checking PPDA with EF is undecidable, as shown in Section 6. This implies undecidability for all models in the PRS-hierarchy that are more general than PPDA, i.e., Petri nets, PAN and PRS.

Model	**Complexity of model checking with EF**
finite-state systems	polynomial
BPA	*PSPACE*-complete
pushdown processes	$\in$ *EXPTIME*, *PSPACE*-hard
BPP	*PSPACE*-complete
PA	decidable, *PSPACE*-hard
PAD	decidable, *PSPACE*-hard
PPDA (and higher)	undecidable

As EF is a fragment of CTL and the modal μ-calculus, it is interesting to compare the limits of decidability for these logics. There is another fragment of CTL (and modal μ-calculus) called EG. EG is like EF, except that the diamond operator EF (for some path eventually in the future) is replaced by the operator EG (for some path always in the future). EG is also a fragment of CTL. Model checking with EG is undecidable even for BPP [12]. On the other hand model checking with the modal μ-calculus is decidable (and *EXPTIME*-complete) for pushdown processes [29] and BPA [24]. Thus in the PRS-hierarchy decidability of the weak logic EG coincides with decidability of the much more expressive modal μ-calculus. In Figure 3 we draw the border of decidability of several branching-time logics in the PRS-hierarchy. Model checking is decidable for all models below the border and undecidable for all those above it. Note that almost all branching-time logics have the same decidability border. EF is the only exception. So EF is 'much more decidable' than all other branching-time logics.

It is interesting to compare the decidability results for branching-time logics with the results for linear-time logics like LTL [27] and the linear-time μ-

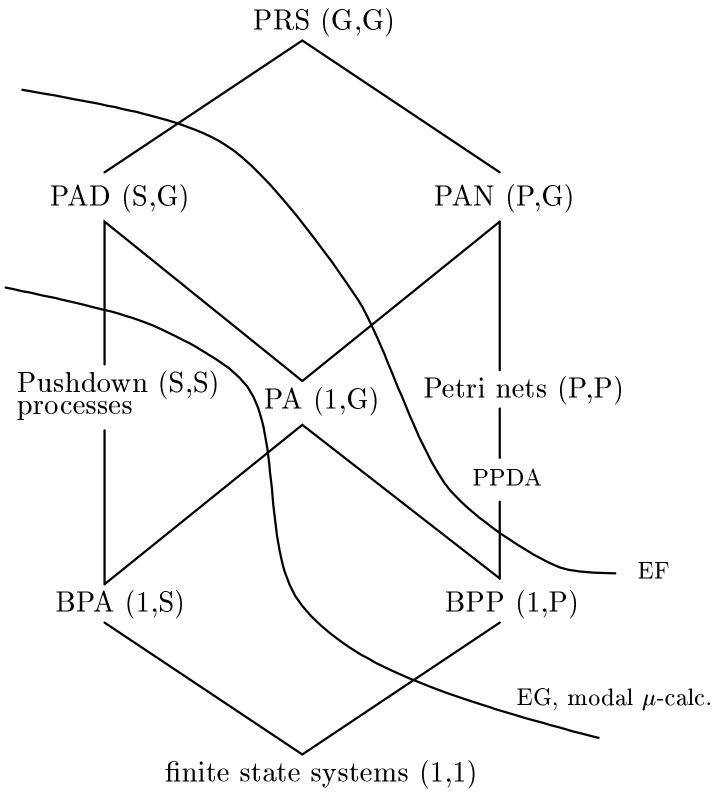

Fig. 3. Limits of the decidability of branching-time logics.

calculus. While model checking PA-processes with EF is decidable, it is undecidable for LTL and the linear-time μ-calculus [3]. For Petri nets the situation is just the opposite. While model checking Petri nets with EF is undecidable, it is decidable for LTL and the linear-time μ-calculus [4,11,7]. In Figure 4 we draw the border of decidability of LTL in the diagram of the PRS-hierarchy.

Acknowledgment: Thanks to Javier Esparza for helpful discussions.

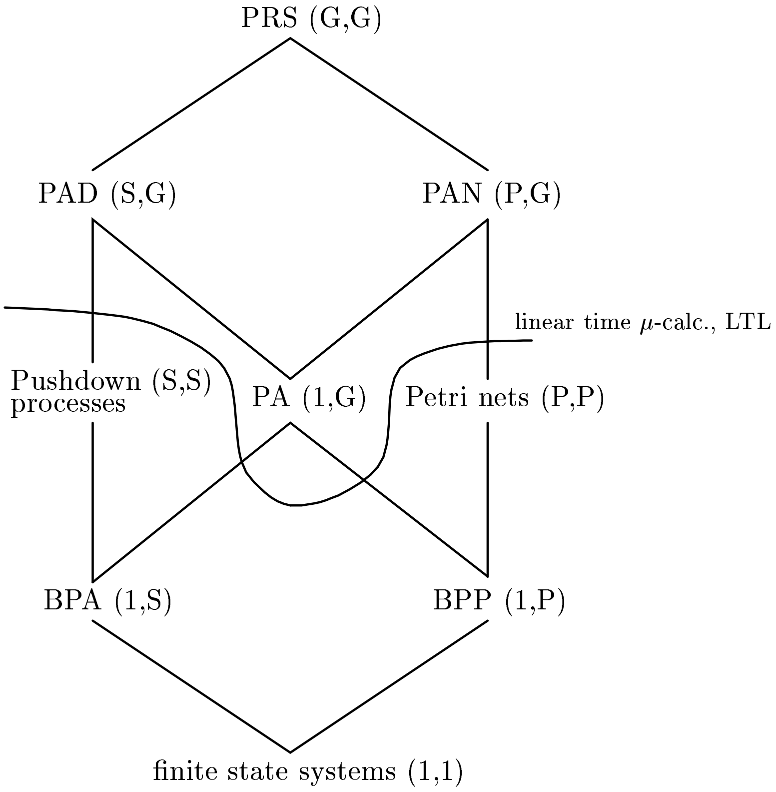

Fig. 4. Limits of the decidability of linear-time logics.

References

[1] J. A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science (TCS)*, 37:77–121, 1985.

[2] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: application to model checking. In *International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*. Springer Verlag, 1997.

[3] A. Bouajjani and P. Habermehl. Constrained properties, semilinear systems, and Petri nets. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[4] A. Bouajjani and R. Mayr. Model checking lossy vector addition systems. In *Proc. of STACS'99*, volume 1563 of *LNCS*. Springer Verlag, 1999.

[5] J. Bradfield. *Verifying Temporal Properties of Systems*. Birkhäuser, 1992.

[6] O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[7] O. Burkart and J. Esparza. More infinite results. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 5, 1997.

[8] D. Caucal. On the regular structure of prefix rewriting. *Journal of Theoretical Computer Science*, 106:61–86, 1992.

[9] S. Christensen. *Decidability and Decomposition in Process Algebras.* PhD thesis, Edinburgh University, 1993.

[10] J. Esparza. On the decidability of model checking for several μ-calculi and Petri nets. In *Trees in Algebra and Programming – CAAP'94*, volume 787 of *LNCS*. Springer Verlag, 1994.

[11] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.

[12] J. Esparza and A. Kiehn. On the model checking problem for branching time logics and Basic Parallel Processes. In *CAV'95*, volume 939 of *LNCS*, pages 353–366. Springer Verlag, 1995.

[13] P. Jančar. Decidability questions for bisimilarity of Petri nets and some related problems. In *Proceedings of STACS'94*, volume 775 of *LNCS*. Springer Verlag, 1994.

[14] P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science*, 148:281–301, 1995.

[15] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. In *Proc. of ICALP'98*, volume 1443 of *LNCS*. Springer Verlag, 1998.

[16] A. Kucera. Regularity is decidable for normed PA processes in polynomial time. In *Foundations of Software Technology and Theoretical Computer Science (FST&TCS'96)*, volume 1180 of *LNCS*. Springer Verlag, 1996.

[17] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. In *Proc. of CONCUR'98*, volume 1466 of *LNCS*. Springer Verlag, 1998.

[18] R. Mayr. Process rewrite systems. *Information and Computation.* To appear.

[19] R. Mayr. Weak bisimulation and model checking for Basic Parallel Processes. In *Foundations of Software Technology and Theoretical Computer Science (FST&TCS'96)*, volume 1180 of *LNCS*. Springer Verlag, 1996.

[20] R. Mayr. Combining Petri nets and PA-processes. In Martin Abadi and Takayasu Ito, editors, *International Symposium on Theoretical Aspects of Computer Software (TACS'97)*, volume 1281 of *LNCS*. Springer Verlag, 1997.

[21] R. Mayr. Model checking PA-processes. In *International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*. Springer Verlag, 1997.

[22] R. Mayr. Process rewrite systems. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 7, 1997. Proceedings of Expressiveness in Concurrency (EXPRESS'97).

[23] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems.* PhD thesis, TU-München, 1998.

[24] R. Mayr. Strict lower bounds for model checking BPA. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 18, 1998.

[25] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[26] F. Moller. Infinite results. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[27] A. Pnueli. The temporal logic of programs. In *FOCS'77*. IEEE, 1977.

[28] Ph. Schnoebelen. Decomposable and regular languages and the shuffle op erator. In *EATCS Bulletin*, volume 67. European Association of Theoretical Computer Science , February 1999.

[29] I. Walukiewicz. Pushdown processes: games and model checking. In *International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *LNCS*. Springer Verlag, 1996.

[30] I. Walukiewicz. Pushdown processes: games and model checking. Technical Report RS-96-54, BRICS, Aarhus, Denmark, 1996. Longer version of a CAV'96 paper.