# Finding Finite Herbrand Models

Stefan Borgwardt and Barbara Morawska*

Theoretical Computer Science, TU Dresden, Germany
{stefborg,morawska}@tcs.inf.tu-dresden.de

**Abstract.** We show that finding finite Herbrand models for a restricted class of first-order clauses is ExpTime-complete. A Herbrand model is called finite if it interprets all predicates by finite subsets of the Herbrand universe. The restricted class of clauses consists of anti-Horn clauses with monadic predicates and terms constructed over unary function symbols and constants. The decision procedure can be used as a new goal-oriented algorithm to solve linear language equations and unification problems in the description logic $\mathcal{FL}_0$. The new algorithm has only worst-case exponential runtime, in contrast to the previous one which was even best-case exponential.

## 1 Introduction

Satisfiability of formulas in First Order Logic (FOL) has always been of interest for computer science and is an active field of research. The main problem is that satisfiability of such formulas is not even semi-decidable. Thus, the focus lies on finding algorithms that decide satisfiability for restricted classes. A possible approach is to use restrictions on the resolution or superposition calculi to obtain decision procedures [8,10].

Related to this is the problem of *model building* that asks for an actual model witnessing the satisfiability of the given clauses. Additionally, one usually asks for a finite representation of such a model. For example, the completeness proofs of resolution-style inference systems sometimes explicitly construct (counter-)models, but there are also other approaches [2,11,16].

Here, we want to study the related problem of finding finite Herbrand models. We call a Herbrand model *finite* if each predicate is interpreted by a finite subset of the Herbrand universe. This problem is semi-decidable since the finite Herbrand interpretations over a fixed signature can be recursively enumerated. It has not been studied before and it is unknown whether it is decidable for arbitrary first-order formulae. The existence of finite Herbrand models implies the existence of finite models in the usual sense, where the domain is required to be finite, but the other implication does not hold in general.

We restrict ourselves to finite sets of *propagation rules*, which are anti-Horn clauses that use only monadic predicates and function symbols, one constant symbol, and one variable. In particular, we do not allow the equality predicate.

* The authors are supported by DFG under grant BA 1122/14-1.

These sets of clauses can be seen as skolemized versions of Ackermann formulas, for which satisfiability is known to be decidable [7,10]. This class of clause sets is also similar to the decidable Bernays-Schönfinkel class [10], but neither is actually included in the other.

In this paper, we show that the problem of deciding the existence of a finite Herbrand model for a finite set of propagation rules is ExpTime-complete. Our decision procedure is aided by a new computational model that we call *propagation nets*. The process of building a model is simulated by the process of saturating the net with terms. This process terminates iff a finite Herbrand model exists. We decide this by analyzing the structure of the net.

The problem of finding finite Herbrand models for a set of propagation rules occurred while designing a new unification procedure for the description logic $\mathcal{FL}_0$. The unification problem in this logic was shown to be ExpTime-complete in [1]. There, solving unification in $\mathcal{FL}_0$ is shown to be equivalent to solving linear language equations. The problem of solving these equations reduces in a natural way to the problem of finding finite Herbrand models for propagation rules. In this reduction, variables become predicates and their finite interpretation in the Herbrand universe defines a solution to the original language equation.

Our decision procedure thus provides a new way to solve linear language equations. It is worst-case exponential, but there are cases in which our algorithm runs in polynomial time. Thus, it has advantages over the previous algorithm [1], which is always exponential.

We think that this method of finding finite Herbrand models can be generalized to larger classes of clauses. As detailed above, it has an immediate application to unification and solving formal language equations.

This paper does not include the formal proofs of our results. These and more detailed explanations can be found in the technical report [4].

## 2 Propagation Rules

We first introduce *propagation rules*, which are clauses over a signature of finitely many unary predicates $\mathcal{P}$, finitely many unary function symbols $\mathcal{F}$, one constant $a$, and one variable $x$. Every ground term over this signature is of the form $f_1(\dots f_n(a)\dots)$, which we will abbreviate as $f_1 \dots f_n(a)$. A *propagation rule* is a clause of the form $\top \to P_1(a) \vee \dots \vee P_n(a)$ *(positive clause)*, $P_0(a) \to P_1(a) \vee \dots \vee P_n(a)$, or $P_0(t_0) \to P_1(t_1) \vee \dots \vee P_n(t_n)$ for $P_0, \dots, P_n \in \mathcal{P}$ and non-ground terms $t_0, \dots, t_n$ over $\mathcal{F}$ and $x$.[1]

We assume that the reader is familiar with Herbrand interpretations (see, e.g., [10]). We call a Herbrand interpretation $\mathcal{H}$ over the above signature *finite* if it interprets every predicate $P \in \mathcal{P}$ by a finite set $P^{\mathcal{H}}$. The task we are interested in is to decide the existence of finite Herbrand models for finite sets of propagation rules. As a first step, we will flatten the propagation rules to get

[1] Note that $n$ might be 0, in which case the right-hand side of the clause is $\bot$. Positive clauses must be ground since otherwise no finite Herbrand model could exist.

rid of most terms of depth larger than 0. A finite set $\mathcal{C}$ of propagation rules is called *normalized* if there is a set $\mathcal{D}(\mathcal{C}) \subseteq \mathcal{P} \times \mathcal{F}$ such that

- For every $(P, f) \in \mathcal{D}(\mathcal{C})$, we have $P^f \in \mathcal{P}$ and the clauses $P^f(x) \to P(f(x))$ (*increasing clause*) and $P(f(x)) \to P^f(x)$ (*decreasing clause*) in $\mathcal{C}$.
- All other clauses in $\mathcal{C}$ must be *flat*, i.e., of the form $\top \to P_1(a) \vee \cdots \vee P_n(a)$, $P_0(a) \to P_1(a) \vee \cdots \vee P_n(a)$, or $P_0(x) \to P_1(x) \vee \cdots \vee P_n(x)$.

For $f \in \mathcal{F}$, we denote by $\mathcal{D}^f(\mathcal{C})$ the set $\{P \in \mathcal{P} \mid (P, f) \in \mathcal{D}(\mathcal{C})\}$.

The interesting property of such sets is that in order to check whether a flat clause $P_0(x) \to P_1(x) \vee \cdots \vee P_n(x)$ is satisfied by a ground term, one only needs to consider this term. Different terms can only occur in the same instance of a clause if it is an increasing or a decreasing clause, which only allows a very limited connection between the terms, i.e., adding and removing the leading function symbol. The set $\mathcal{D}(\mathcal{C})$ acts as an "interface" between terms of different lengths: A clause can only contain different terms if a predicate $P^f$ with $(P, f) \in \mathcal{D}(\mathcal{C})$ is involved. The special predicate $P^f$ represents those terms in $P$ that have the prefix $f$: For any Herbrand model $\mathcal{H}$ and any word $w \in \mathcal{F}^*$, the term $f(w(a))$ is in $P^{\mathcal{H}}$ iff $w(a)$ is in ${P^f}^{\mathcal{H}}$.

To transform a finite set $\mathcal{C}$ of propagation rules into a normalized set $\mathcal{C}'$, we introduce auxiliary predicates that allow us to replace arbitrary atoms by flat ones. For example, the atom $P(fg(x))$ can be replaced by the equivalent atom $P^{fg}(x)$ if $(P, f)$ and $(P^f, g)$ are added to $\mathcal{D}(\mathcal{C})$. In contrast to common flattening procedures for first-order clauses, we do not use new variables or equality [2].

**Lemma 1.** *For every finite set $\mathcal{C}$ of propagation rules, we can construct in polynomial time a normalized set $\mathcal{C}'$ of propagation rules such that $\mathcal{C}$ has a finite Herbrand model iff $\mathcal{C}'$ does.*

*Example 2.* Consider the propagation rules

$$\begin{aligned}\mathcal{C}_1 := \{&\top \to P_0(a),\ P_0(f(x)) \to \bot,\ P_0(g(x)) \to \bot,\ P_3(a) \to \bot,\ P_3(f(x)) \to \bot,\\ &P_3(g(x)) \to P_0(x),\ P_0(x) \to P_3(g(x)),\ P_0(a) \to P_1(a),\ P_1(a) \to P_0(a),\\ &P_2(x) \to P_3(x) \vee P_1(f(x)),\ P_3(x) \to P_2(x),\ P_1(f(x)) \to P_2(x)\\ &P_1(x) \to P_2(x) \vee P_1(g(x)),\ P_2(x) \to P_1(x),\ P_1(g(x)) \to P_1(x)\}.\end{aligned}$$

To construct the normalized set $\mathcal{C}_1'$, we first rename $P_0$ to $P_3^g$ and add the pair $(P_3, g)$ to $\mathcal{D}(\mathcal{C}_1')$. Afterwards, the pairs $(P_3, f)$, $(P_1, f)$, $(P_1, g)$, $(P_3^f, g)$, and $(P_3^g, g)$ are added, together with the corresponding increasing and decreasing clauses. The resulting flat clauses are the following:

$$\begin{aligned}&\top \to P_3^g(a),\ P_3^{gf}(x) \to \bot,\ P_3^{gg}(x) \to \bot,\ P_3(a) \to \bot,\ P_3^f(x) \to \bot,\\ &P_3^g(a) \to P_1(a),\ P_1(a) \to P_3^g(a),\\ &P_2(x) \to P_3(x) \vee P_1^f(x),\ P_3(x) \to P_2(x),\ P_1^f(x) \to P_2(x),\\ &P_1(x) \to P_2(x) \vee P_1^g(x),\ P_2(x) \to P_1(x),\ P_1^g(x) \to P_1(x).\end{aligned}$$

We will use $\mathcal{C}_1'$ throughout this paper to illustrate the presented algorithms.

For a flat clause $c$, the set $\texttt{possibilities}(c)$ contains all predicates occurring on the right-hand side of $c$. For a set $\mathcal{C} = \{c_1, \ldots, c_n\}$ of flat clauses, we define $\texttt{possibilities}(\mathcal{C}) := \{\{P_1, \ldots, P_n\} \mid \forall i \in \{1, \ldots, n\} : P_i \in \texttt{possibilities}(c_i)\}$. For example, $P_1(a) \rightarrow P_2(a) \vee P_3(a)$ has the possibilities $P_2$ and $P_3$, while $\{P_1(x) \rightarrow P_2(x) \vee P_3(x), \top \rightarrow P_0(a)\}$ has the possibilities $\{P_2, P_0\}$ and $\{P_3, P_0\}$.

In the following, we assume that any normalized set $\mathcal{C}$ of propagation rules contains at most one positive clause, which is of the form $\top \rightarrow A(a)$, and that the predicate $A$ otherwise only occurs on the left-hand side of other ground clauses. If this is not the case, we introduce a new predicate $A$, add the clause $\top \rightarrow A(a)$ to $\mathcal{C}$, and replace $\top$ by $A(a)$ in every other positive clause. It is easy to see that this modification does not affect the existence of a finite Herbrand model for $\mathcal{C}$.

For the set $\mathcal{C}'_1$ from Example 2, we simply add $\top \rightarrow A(a)$ to $\mathcal{C}'_1$ and replace the propagation rule $\top \rightarrow P_3^g(a)$ by $A(a) \rightarrow P_3^g(a)$.

## 3 Propagation Nets

We now introduce a new computational model, called *propagation net*, that will be used to decide the existence of finite Herbrand models for finite sets of propagation rules. We use notions borrowed from the theory of Petri nets [12,13].

A propagation net consists of places and transitions which are connected by directed arcs. A computation moves words from places to other places using the transitions between them. If a place has several outgoing arcs to transitions, it can choose one of them to *fire*. This means that a word from this place is transported to the transition and then distributed to all places reachable from this transition. An arc from a place to a transition can also change the word by adding a letter or removing the first letter. An arc from a transition to a place can filter out words that should not be transported to the place. The firing of a transition does not remove the word from the place but just deactivates it. The goal is to find a computation that starts with a given distribution of words among places and *terminates* in the sense that all words are deactivated.

**Definition 3.** *A* propagation net $\mathcal{N} = (P, T, \Sigma, E, I, \pi, \tau)$ *consists of*

- *a finite set* $P$ *of* places,
- *a finite set* $T$ *of* transitions,
- *a finite alphabet* $\Sigma$,
- *a set* $E \subseteq (P \times T) \cup (T \times P)$ *of* arcs,
- *an* initial marking $I : (P \cup T) \rightarrow \mathcal{P}(\Sigma^*)$ *and* $I_a : P \rightarrow \mathcal{P}(\Sigma^*)$,
- *a partial* filter function $\pi : \big(E \cap (T \times P)\big) \rightarrow \Sigma \cup \{\varepsilon\}$, *and*
- *a* successor function $\tau : \big(E \cap (P \times T)\big) \rightarrow \Sigma \cup \{f^{-1} | f \in \Sigma\} \cup \{\varepsilon\}$.

A *token* in $\mathcal{N}$ is a word over $\Sigma$. A *marking* $M$ of $\mathcal{N}$ is a pair of mappings $M : (P \cup T) \rightarrow \mathcal{P}(\Sigma^*)$ and $M_a : P \rightarrow \mathcal{P}(\Sigma^*)$ assigning to each place and each transition finite sets of tokens such that $M_a(p) \subseteq M(p)$ for every $p \in P$. $M(p)$ contains the tokens of a place $p \in P$, while $M(t)$ contains the tokens of a transition $t \in T$ in the marking $M$. The set $M_a(p)$ contains the *active* tokens of $p$ in $M$. We assume that $I$ is a proper marking in the above sense.

We say that a token $w$ *matches* the filter $\pi(t,p)$ of an arc $(t,p) \in E \cap (T \times P)$ if either (i) $\pi(t,p)$ is undefined (no restriction on $w$), (ii) $\pi(t,p) = \varepsilon$ and then $w = \varepsilon$, or (iii) $\pi(t,p) = f \in \Sigma$ and then $w$ starts with $f$.

There are two elementary operations on markings. A token $w$ is *deactivated* at $p \in P$ by removing it from $M_a(p)$, if it is in $M_a(p)$, and adding it to $M(p)$, if it is not already in $M(p)$. Note that $w$ need not be in $M(p)$ to be deactivated.

A token $w$ is *produced* at a transition $t \in T$ by adding it to $M(t)$. This operation has the side effect of also *producing* the token at all places $p \in P$ with $(t,p) \in E$. This secondary operation is executed only if $w$ matches the filter $\pi(t,p)$. If this is the case and $w \notin M(p)$, then $w$ is added to $M(p)$ and $M_a(p)$. Otherwise, the token $w$ is not added to the marking at $p$.

A *firing* in $\mathcal{N}$ is a triple $\mathfrak{f} = (p,w,t) \in P \times \Sigma^* \times T$ such that $(p,t) \in E$ and the concatenation $\tau(p,t)w$ is defined, i.e., if $\tau(p,t) = f^{-1}$, then $w$ begins with $f$. The result of firing $\mathfrak{f}$ in a marking $M$ is a new marking $M'$ as follows:

1. Initialize $M' := M$ and $M'_a := M_a$.
2. Deactivate the token $w$ at $p$ in $M'$.
3. Compute the *successor token* $w' := \tau(p,t)w$.
4. Produce $w'$ at $t$ in $M'$, thereby also producing $w'$ at every place reachable from $t$ by an outgoing arc whose filter matches $w'$.

If $M'$ is the result of the firing $\mathfrak{f}$ in $M$, then we write $M \xrightarrow{\mathfrak{f}} M'$. If $M(p) = M'(p)$ for all $p \in P$, this firing is called *unproductive* in $M$; otherwise, it is called *productive*. An unproductive firing only removes an active token from the marking, while a productive firing also introduces new active tokens.

Given a marking $M_0$, a *firing sequence (starting in $M_0$)* is a finite sequence $M_0 \xrightarrow{\mathfrak{f}_1} \dots \xrightarrow{\mathfrak{f}_m} M_m$ of firings. If the initial marking is not important, we denote this sequence by $\mathfrak{f}_1, \dots, \mathfrak{f}_m$. $M_m$ is called the *final marking* of this sequence. The sequence is called *terminating* if $M_m$ is *stable*, i.e., $M_{m,a}(p) = \emptyset$ for all $p \in P$. We say that $\mathcal{N}$ *terminates* if it has a terminating firing sequence that starts in $I$. Note that such a firing sequence has to end with a nonproductive firing since otherwise new active tokens would be created. Figures 1 and 2 depict a simple propagation net and the effect of a firing on the initial marking.

**Other Computational Models** There are several differences between propagation nets and Petri nets. In propagation nets, tokens are not atomic objects, but words over an alphabet $\Sigma$. Additionally, transitions do not need to be synchronized, i.e., do not require the input token to be present at every input place.

Propagation nets behave much more like two-way alternating automata on finite words [5,9,3] or trees [14,6], where places are existential states and transitions are universal states. Contrary to word automata, however, propagation nets do not read an input word, but rather write several words, i.e., the tokens that are produced. In finite trees, one can represent all these words simultaneously. But then propagation nets would represent automata on finite trees that can also accept with infinite computations, contrary to the standard definition.

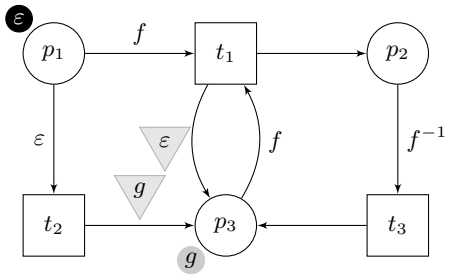


**Fig. 1.** A simple propagation net with $P = \{p_1, p_2, p_3\}$ and $T = \{t_1, t_2, t_3\}$. Edge labels denote the functions $\pi$ and $\tau$, where filters are depicted as triangles. Filled circles are the tokens of the initial marking; active tokens have a black background.

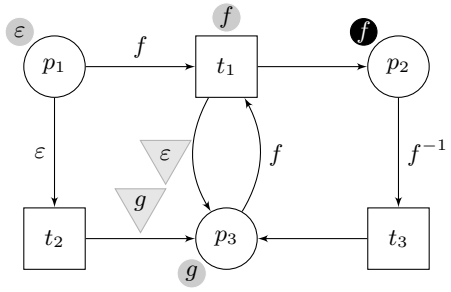


**Fig. 2.** The propagation net from Fig. 1 after firing $(p_1, \varepsilon, t_1)$. The token $f$ is produced at $t_1$ and $p_2$, but not at $p_3$ since $f$ does not match the filter $\tau(t_1, p_3) = \varepsilon$.

**From Clauses to Propagation Nets** We will now translate any normalized set $\mathcal{C}$ of propagation rules into a propagation net $\mathcal{N}_\mathcal{C}$. The goal is to express the finite Herbrand models of $\mathcal{C}$ by stable markings of $\mathcal{N}_\mathcal{C}$. We will represent terms by tokens, clauses by places, and predicates by transitions. From a clause, a token can be transferred to any of its possibilities. From a predicate, a token is then distributed to all clauses with this predicate on their left-hand side. The filter function allows to discard those terms (tokens) that are irrelevant for satisfying the clause. The successor function expresses increasing and decreasing clauses by adding or removing letters, respectively. For a flat clause, the successor function is $\varepsilon$, i.e., it leaves the term as it is. The initial marking simply consists of the active token $\varepsilon$ at $\top \to A(a)$ since this is the only clause without precondition.

**Definition 4.** *Let $\mathcal{C}$ be a normalized set of propagation rules. The propagation net $\mathcal{N}_\mathcal{C} := (\mathcal{C}, \mathcal{P}, \mathcal{F}, E_\mathcal{C}, I_\mathcal{C}, \pi_\mathcal{C}, \tau_\mathcal{C})$ has the following components:*

- $E_\mathcal{C} := \big\{(c, P_i) \mid c = \ldots \to P_1(t_1) \vee \cdots \vee P_n(t_n) \in \mathcal{C} \text{ and } i \in \{1, \ldots, n\}\big\}$
  $\cup \big\{(P_0, c) \mid c = P_0(t_0) \to \cdots \in \mathcal{C}\big\}$
- $I_{\mathcal{C},a}(c) := I_\mathcal{C}(c) := \begin{cases} \{\varepsilon\} & \text{if } c = \top \to A(a) \\ \emptyset & \text{otherwise} \end{cases}$

– $\pi_{\mathcal{C}}\big(P_0, P_0(t_0) \to \dots\big) := \begin{cases} \varepsilon & \text{if } t_0 = a \\ \textit{undefined} & \text{if } t_0 = x \\ f & \text{if } t_0 = f(x) \end{cases}$

– $\tau_{\mathcal{C}}\big(P_0(t_0) \to P_1(t_1) \vee \dots \vee P_n(t_n), P_i\big) := \begin{cases} f & \text{if } t_0 = x,\ t_i = f(x) \\ f^{-1} & \text{if } t_0 = f(x),\ t_i = x \\ \varepsilon & \textit{otherwise} \end{cases}$

– $\tau_{\mathcal{C}}(\top \to A(a), A) := \varepsilon$

In this propagation net, every firing $(c, w, P)$ represents a possibility of $c$. Firing sequences can thus be seen as sequences of applying possibilities to tokens on the left-hand side of clauses: If $w(a)$ is a term in $P^{\mathcal{H}}$ for a Herbrand interpretation $\mathcal{H}$ and we want $\mathcal{H}$ to satisfy a clause $P(x) \to P_1(x) \vee \dots \vee P_n(x)$, then we have to find a possibility $P_i$ for which to put $w(a)$ into $P_i^{\mathcal{H}}$. If this process of satisfying clauses stops, we have found a finite Herbrand model of $\mathcal{C}$.

**Lemma 5.** *$\mathcal{C}$ has a finite Herbrand model iff $\mathcal{N}_{\mathcal{C}}$ terminates.*

*Example 6.* Consider the propagation net $\mathcal{N}_{\mathcal{C}_1'}$ for the rules from Example 2. Ignoring unproductive firings, the following is a terminating firing sequence:

$$(\top \to A(a), \varepsilon, A),\ (A(a) \to P_3^g(a), \varepsilon, P_3^g),\ (P_3^g(x) \to P_3(g(x)), \varepsilon, P_3),$$
$$(P_3(x) \to P_2(x), g, P_2),\ (P_2(x) \to P_1(x), g, P_1),\ (P_1(g(x)) \to P_1^g(x), g, P_1^g),$$
$$(P_1^g(x) \to P_1(x), \varepsilon, P_1)$$

If we abbreviate firings like $(P_1(x) \to P_2(x) \vee P_1^g(x), g, P_2)$ by $P_1(g) \to P_2(g)$ and join "adjacent" firings, the structure of this sequence becomes apparent:

$$\begin{array}{ccccc} & & P_3(g) & \longrightarrow P_2(g) \longrightarrow & P_1(g) \\ & \nearrow & & & \searrow \\ \top \ \longrightarrow A(\varepsilon) & \longrightarrow P_3^g(\varepsilon) & & & P_1^g(\varepsilon) \to P_1(\varepsilon) \end{array}$$

It is easy to read off the corresponding finite Herbrand model $\mathcal{H}$ of $\mathcal{C}_1'$:

$$A^{\mathcal{H}} = P_1^{g\mathcal{H}} = P_3^{g\mathcal{H}} = \{a\},\ P_1^{\mathcal{H}} = \{a, g(a)\},\ P_2^{\mathcal{H}} = P_3^{\mathcal{H}} = \{g(a)\},$$
$$P_1^{f\mathcal{H}} = P_3^{f\mathcal{H}} = P_3^{gf\mathcal{H}} = P_3^{gg\mathcal{H}} = \emptyset.$$

### 3.1 Behavior of Propagation Nets

Our goal is to decide termination of propagation nets $\mathcal{N}_{\mathcal{C}}$ obtained from normalized sets of propagation rules $\mathcal{C}$. We will use these propagation nets to formulate the ideas behind a decision procedure for the existence of finite Herbrand models for the clause sets.

**Termination of Propagation Nets** We first analyze what it means for $\mathcal{N}_\mathcal{C}$ to have a terminating firing sequence starting in $I_\mathcal{C}$. Any such sequence will start with the token $\varepsilon$ at $A$ and gradually distribute it to other predicates, while sometimes increasing it. There are two reasons why this might not be possible. First, it may be impossible to avoid a contradiction, i.e., a clause with $\bot$ on the right-hand side, in any firing sequence starting in $I_\mathcal{C}$. The other possibility is that every firing sequence that avoids all contradictions is forced into a cycle of creating ever longer tokens. Thus, in order for the sequence to terminate, the length of the produced tokens has to be bounded. To analyze the detailed structure of terminating firing sequences, we introduce the following notions.

**Definition 7.** *Let $P \in \mathcal{X} \subseteq \mathcal{P}$ and $w = fw' \in \mathcal{F}^+$. A* $(P, \mathcal{X}, w)$-replacement sequence *is a firing sequence of $\mathcal{N}_\mathcal{C}$ starting in $M_0$ and ending in $M_m$ such that*

- *$M_0$ only contains the token $w$ at $P$ and the active token $w$ at all clauses with $P(x)$ or $P(f(x))$ on the left-hand side,*
- *$M_m$ only contains tokens with the suffix $w$,*
- *$w \in M_m(Q)$ iff $Q \in \mathcal{X}$, and*
- *if $w' \in M_{m,a}(c)$, then $w' = w$ and $c = Q(f(x)) \to Q^f(x)$.*

*A* $(P, \varepsilon)$-replacement sequence *is a firing sequence starting in $M_0$ and ending in $M_m$ such that*

- *$M_0$ only contains the token $\varepsilon$ at $P$ and the active token $\varepsilon$ at all clauses with $P(x)$ or $P(a)$ on the left-hand side, and*
- *$M_m$ is stable.*

*The* height *of a replacement sequence is the maximal number $|w'| - |w|$ for any token $w'$ in $M_m$.*

Every terminating firing sequence starting in $I_\mathcal{C}$ consists of the firing $(\top \to A(a), \varepsilon, A)$ and an $(A, \varepsilon)$-replacement sequence. Thus, our goal is to decide the existence of such replacement sequences. If there is an $(A, \varepsilon)$-replacement sequence of height 0, then only the token $\varepsilon$ is produced in this sequence. Deciding the existence of such sequences is easy (see Alg. 2). If the height of an $(A, \varepsilon)$-replacement sequence is larger than 0, it contains other replacement sequences of smaller height, as explained in the following.

The sequence has to produce a token $w = fw' \neq \varepsilon$ at a predicate $P$, and then $w$ is contained in the final marking at all clauses with $P(x)$ or $P(f(x))$ on the left-hand side. We can extract a $(P, \mathcal{X}, w)$-replacement sequence as follows: Starting from the token $w$ at all clauses with $P(x)$ or $P(f(x))$ on the left-hand side, we extract all firings that deactivate these tokens and the tokens produced from these firings, except firings of the form $(Q(f(x)) \to Q^f(x), w, Q^f)$. The extracted firings form the replacement sequence and the set $\mathcal{X}$ consists of all predicates $Q$ at which $w$ was produced in this sequence.

*Example 8.* The terminating firing sequence from Example 6 mainly consists of an $(A, \varepsilon)$-replacement sequence. The firing $(P_3^g(x) \to P_3(g(x)), \varepsilon, P_3)$ produces

the token $g$ at all clauses with $P_3(x)$ or $P_3(g(x))$ on the left-hand side, which is the starting point of a replacement sequence. The corresponding $(P_3, \{P_3, P_2, P_1\}, g)$-replacement sequence is

$$(P_3(x) \to P_2(x), g, P_2),\ (P_2(x) \to P_1(x), g, P_1),$$
$$(P_2(x) \to P_3(x) \vee P_1^f(x), g, P_3),\ (P_1(x) \to P_2(x) \vee P_1^g(x), g, P_2).$$

If a longer token $w'$ is produced in such a sequence at $Q \in \mathcal{P}$, we can use the same procedure to extract a $(Q, \mathcal{Y}, w')$-replacement sequence of smaller height. We continue this until the height of the replacement sequences is 0. Thus, every terminating firing sequence is decomposed into nested replacement sequences.

To decide termination of $\mathcal{N}_\mathcal{C}$, we construct all possible replacement sequences, starting with height 0. These can be used to build replacement sequences of increasing heights, until we can construct an $(A, \varepsilon)$-replacement sequence.

**Replacement Sequences of Height 0** To construct replacement sequences of height 0 for a predicate $P$, we define the set $\texttt{possibilities}(P)$ to contain all possibilities of the set of all flat clauses with $P(x)$ on the left-hand side. Such a possibility $\{Q_1, \ldots, Q_n\}$ represents one way of firing all these flat clauses. Afterwards, we have to consider the possibilities of the reached predicates $Q_1, \ldots, Q_n$ and repeat this process until no new predicates are reached.

Since we want to find replacement sequences of height 0, we must prevent this process to reach predicates of the form $P^f$ with $(P, f) \in \mathcal{D}(\mathcal{C})$. Thus, we define $\texttt{possibilities}(P^f(x) \to P(f(x))) := \emptyset$ and extend the set $\texttt{possibilities}(P^f)$ to also consider this increasing clause. Thus, $\texttt{possibilities}(P^f) = \emptyset$, which indicates that we have no way of dealing with the token $w$ at $P^f$.

*Example 9.* The $(P_3, \{P_3, P_2, P_1\}, g)$-replacement sequence from Example 8 can be constructed as follows: For $P_3$, we have the possibility $\{P_2\}$, i.e., the firing $(P_3(x) \to P_2(x), g, P_2)$. $P_2$ has the possibilities $\{P_1, P_3\}$ and $\{P_1, P_1^f\}$. The first one yields $(P_2(x) \to P_1(x), g, P_1)$ and $(P_2(x) \to P_3(x) \vee P_1^f(x), g, P_3)$. The second possibility would lead to the active token $g$ at $P_1^f$, which we disallow. Finally, for $P_1$ we choose the unproductive firing $(P_1(x) \to P_2(x) \vee P_1^g(x), g, P_2)$.

It is easy to see that a $(P, \mathcal{X}, w)$-replacement sequence can be changed into a $(P, \mathcal{X}, w')$-replacement sequence by substituting the suffix $w$ by $w'$ in every token in the sequence. Thus, the token $w$ is not necessary to describe the replacement sequence. Similarly, it is not important which firings are used to deactivate tokens, only which predicates are reached. We are thus only interested in so-called *shortcuts* $(P, \mathcal{X})$ with $P \in \mathcal{X} \subseteq \mathcal{P}$ for which a $(P, \mathcal{X}, w)$-replacement sequence exists. There may be several possibilities for $P$, and thus several replacement sequences and several shortcuts $(P, \mathcal{X}_1)$, $(P, \mathcal{X}_2)$, ... representing them.

*Example 10.* The $(P_3, \{P_3, P_2, P_1\}, g)$-replacement sequence shown in Example 8 yields the shortcut $(P_3, \{P_3, P_2, P_1\})$. We can also find replacement sequences for $P_1$ and $P_2$, represented by the shortcuts $(P_1, \{P_1, P_2, P_3\})$ and $(P_2, \{P_1, P_2, P_3\})$.

**Replacement Sequences of Larger Height** If we have shortcuts for all replacement sequences of height 0, we can construct replacement sequences of height 1 as follows. Such a sequence will contain firings of increasing clauses $P^f(x) \to P(f(x))$ w.r.t. some token $w$. This firing produces the token $fw$ at all clauses having $P(x)$ or $P(f(x))$ on the left-hand side. This is a possible starting point for a $(P, \mathcal{X}, fw)$-replacement sequence of height 0.

If we have already computed a shortcut $(P, \mathcal{X})$, there is a firing sequence that deactivates the token $fw$ and distributes it to all predicates of $\mathcal{X}$. This leaves us to consider the tokens that were created at decreasing clauses. These clauses must be of the form $Q(f(x)) \to Q^f(x)$ for $Q \in \mathcal{X}$ since the token begins with $f$ and is distributed only to predicates in $\mathcal{X}$. We then simply fire these decreasing clauses, which gets us back to the original token $w$.

Thus, when looking for replacement sequences of height 1, we can use shortcuts as possibilities for the predicates $P^f$. Each shortcut $(P, \mathcal{X})$ yields a possibility $\{Q^f \mid Q \in \mathcal{X} \cap \mathcal{D}^f(\mathcal{C})\}$ for the increasing clause $P^f(x) \to P(f(x))$. If there is at least one shortcut $(P, \mathcal{X})$, then $\texttt{possibilities}(P^f)$ can now be non-empty. With this new definition of `possibilities`, we can compute shortcuts for replacement sequences of height 1, similar to the construction of replacement sequences of height 0. These yield more possibilities, which lead to shortcuts for replacement sequences of height 2, and so on.

The following procedure implements the computation of all possibilities for a predicate $P$ w.r.t. a set $\mathcal{R}$ of previously computed shortcuts.

**Algorithm 1** ($\texttt{possibilities}(\mathcal{C}, \mathcal{R}, P)$)**.**

**Input:** a normalized set $\mathcal{C}$ of propagation rules, a set $\mathcal{R}$ of shortcuts, and a predicate $P$
**Output:** the set of possibilities for $P$ w.r.t. $\mathcal{C}$ and $\mathcal{R}$
  **if** $P = Q^f$ with $(Q, f) \in \mathcal{D}(\mathcal{C})$ **then**
    $\mathcal{L} \leftarrow \{\{Q_1^f, \ldots, Q_n^f\} \mid (Q, \mathcal{X}) \in \mathcal{R},\ \{Q_1, \ldots, Q_n\} = \mathcal{X} \cap \mathcal{D}^f(\mathcal{C})\}$
  **else** $\mathcal{L} \leftarrow \{\emptyset\}$
  **for all** $P(x) \to P_1(x) \vee \cdots \vee P_n(x) \in \mathcal{C}$ **do**
    $\mathcal{L} \leftarrow \{\mathcal{Y} \cup \{P_l\} \mid \mathcal{Y} \in \mathcal{L},\ l \in \{1, \ldots, n\}\}$
  **return** $\mathcal{L}$

For example, if we have the shortcut $(P_1, \{P_1, P_2, P_3\})$ from Example 10, then $\texttt{possibilities}(\mathcal{C}_1', \mathcal{R}, P_1^f)$ is $\{\{P_1^f, P_3^f, P_2\}\}$ instead of $\emptyset$.

**Replacement Sequences for $\varepsilon$** To construct a replacement sequence for $\varepsilon$, we can use the same approach as above, but we also have to consider the ground clauses of $\mathcal{C}$. Since we only want to decide the existence of such a replacement sequence, we need not compute any shortcuts.

We call a predicate $P \in \mathcal{P}$ *good* if there is a $(P, \varepsilon)$-replacement sequence. All other predicates are *bad*. To decide whether $A$ is good, we construct the set $\mathcal{B}$ of all bad predicates using the following procedure. The idea is that a predicate is bad whenever all its possibilities contain a bad predicate. This is similar to the emptiness test for looping automata on infinite trees [15].

**Algorithm 2** (`isTerminating`$(\mathcal{C}, \mathcal{R})$)**.**

**Input:** a normalized set $\mathcal{C}$ of propagation rules and a set $\mathcal{R}$ of shortcuts
**Output:** `true` iff $A$ is good w.r.t. $\mathcal{R}$

$\mathcal{B}_0 \leftarrow \emptyset$, $k \leftarrow 0$
**repeat**
  $\mathcal{B}_{k+1} \leftarrow \mathcal{B}_k$
    $\cup \{P \in \mathcal{P} \mid \exists P(x) \to P_1(x) \vee \cdots \vee P_n(x) \in \mathcal{C} : \{P_1, \ldots, P_n\} \subseteq \mathcal{B}_k\}$
    $\cup \{P \in \mathcal{P} \mid \exists P(a) \to P_1(a) \vee \cdots \vee P_n(a) \in \mathcal{C} : \{P_1, \ldots, P_n\} \subseteq \mathcal{B}_k\}$
    $\cup \{P^f \in \mathcal{P} \mid (P, f) \in \mathcal{D}(\mathcal{C}),\ \forall (P, \mathcal{X}) \in \mathcal{R}\ \exists Q \in \mathcal{X} \cap \mathcal{D}^f(\mathcal{C}) : Q^f \in \mathcal{B}_k\}$
  $k \leftarrow k + 1$
**until** $\mathcal{B}_k = \mathcal{B}_{k-1}$
**return** $A \notin \mathcal{B}_k$

*Example 11.* Consider the set $\mathcal{C}_1'$ from Example 2 and assume that no shortcuts are available. The predicates $P_1^f$, $P_1^g$, $P_3^f$, $P_3^g$, $P_3^{gf}$, and $P_3^{gg}$ are immediately bad. Because of the clause $A(a) \to P_3^g(a)$, $A$ is also bad. With the shortcuts computed in Example 10, the predicates $P_3^g$ and $A$ are no longer bad. This means that there is an $(A, \varepsilon)$-replacement sequence of height 1, as already seen in Example 6.

## 4 Deciding Termination

We can now formulate our main algorithm that decides whether $\mathcal{N}_\mathcal{C}$ terminates. It computes shortcuts representing replacement sequences of increasing height. The sets $\mathcal{R}_i$ are used to store all shortcuts computed so far. In each iteration, the algorithm checks whether these shortcuts already suffice to prove termination of $\mathcal{N}_\mathcal{C}$ using `isTerminating`$(\mathcal{C}, \mathcal{R}_i)$ (Alg. 2). If not, shortcuts for the next height are computed. If there are no new shortcuts, the algorithm stops and returns `false`, indicating that $\mathcal{N}_\mathcal{C}$ does not terminate.

**Algorithm 3 (Main algorithm).**

**Input:** a normalized set $\mathcal{C}$ of propagation rules
**Output:** `true` iff $\mathcal{N}_\mathcal{C}$ terminates

$\mathcal{R}_0 \leftarrow \emptyset$, $i \leftarrow 0$
**repeat**
  **if** `isTerminating`$(\mathcal{C}, \mathcal{R}_i)$ **then return** `true`
  $\mathcal{R}_{i+1} \leftarrow$ `nextShortcuts`$(\mathcal{C}, \mathcal{R}_i)$
  $i \leftarrow i + 1$
**until** $\mathcal{R}_i = \mathcal{R}_{i-1}$
**return** `false`

The procedure `nextShortcuts`$(\mathcal{C}, \mathcal{R})$ implements the computation of the shortcuts representing replacement sequences of the next height. It uses a set $\mathcal{T}$ of triples of the form $(P, R_P, V_P)$, where $R_P$ is the set of predicates reached so far starting from $P$, and $V_P \subseteq R_P$ contains the predicates that were already *visited*,

i.e., for which all possibilities have been considered. Visiting $Q$ corresponds to firing all clauses starting with $Q(x)$.

The computation of shortcuts for $P$ starts with the triple $(P, \{P\}, \emptyset)$. In each step, we choose a triple $(P, R_P, V_P) \in \mathcal{T}$ that still contains an unvisited predicate $Q \in R_P \setminus V_P$ and consider its possibilities. For each $\mathcal{Y} \in \texttt{possibilities}(\mathcal{C}, \mathcal{R}, Q)$, we add $(P, R_P \cup \mathcal{Y}, V_P \cup \{Q\})$ to $\mathcal{T}$ since the predicates from $\mathcal{Y}$ have been reached and $Q$ has just been visited. The original triple is removed from $\mathcal{T}$.

We continue this process until there are no more unvisited predicates. A triple $(P, R_P, R_P)$ then yields the shortcut $(P, R_P)$. We restrict the starting triples $(P, \{P\}, \emptyset)$ to satisfy $(P, f) \in \mathcal{D}(\mathcal{C})$ for some $f \in \mathcal{F}$ since only such predicates can be reached by an increasing clause.

**Algorithm 4 ($\texttt{nextShortcuts}(\mathcal{C}, \mathcal{R})$).**

**Input:** a normalized set $\mathcal{C}$ of propagation rules and a set $\mathcal{R}$ of shortcuts
**Output:** a set $\mathcal{R}'$ of shortcuts for the next height
  $\mathcal{T} \leftarrow \{(P, \{P\}, \emptyset) \mid r \in \mathcal{F},\ (P, r) \in \mathcal{D}(\mathcal{C})\}$
  **while** there is $(P, R_P, V_P) \in \mathcal{T}$ with $R_P \setminus V_P \neq \emptyset$ **do**
    $\mathcal{T} \leftarrow \mathcal{T} \setminus \{(P, R_P, V_P)\}$
    **choose** $Q$ from $R_P \setminus V_P$
    **for all** $\mathcal{Y} \in \texttt{possibilities}(\mathcal{C}, \mathcal{R}, Q)$ **do**
      $\mathcal{T} \leftarrow \mathcal{T} \cup \{(P, R_P \cup \mathcal{Y}, V_P \cup \{Q\})\}$
  **return** $\{(P, R_P) \mid (P, R_P, R_P) \in \mathcal{T}\}$

*Example 12.* Consider the set $\mathcal{C}_1'$ from Example 2. We describe the computation of $\texttt{nextShortcuts}(\mathcal{C}_1', \emptyset)$, which was already illustrated in Example 9. It starts with the triples $(P_1, \{P_1\}, \emptyset)$, $(P_3, \{P_3\}, \emptyset)$, $(P_3^f, \{P_3^f\}, \emptyset)$, and $(P_3^g, \{P_3^g\}, \emptyset)$, but we consider here only the first one.

The possibilities $\{P_2\}$ and $\{P_1^g\}$ for $P_1$ yield the triples $(P_1, \{P_1, P_2\}, \{P_1\})$ and $(P_1, \{P_1, P_1^g\}, \{P_1\})$. Since there is no shortcut $(P_1, \mathcal{X})$, the set of possibilities for $P_1^g$ is empty and the second triple is removed. $P_2$ has the possibilities $\{P_3, P_1\}$ and $\{P_1^f, P_1\}$. One of the resulting triples is simply removed, leaving us with $(P_1, \{P_1, P_2, P_3\}, \{P_1, P_2\})$. Finally, $P_3$ is visited, resulting in $(P_1, \{P_1, P_2, P_3\}, \{P_1, P_2, P_3\})$, and thus in the shortcut $(P_1, \{P_1, P_2, P_3\})$.

In the following, we show that the computed shortcuts actually represent replacement sequences. More precisely, the shortcuts computed in the $i$-th iteration of the main loop of Alg. 3 represent all replacement sequences of height at most $i - 1$.

**Lemma 13.** *Let $i \geq 1$ be such that $\mathcal{R}_i$ was computed by Alg. 3, $(P, \mathcal{X}) \in \mathcal{R}_i$, and $w \in \mathcal{F}^+$. Then there is a $(P, \mathcal{X}, w)$-replacement sequence of height $\leq i - 1$.*

On the other hand, every replacement sequence of $\mathcal{N}_\mathcal{C}$ of height at most $i$ corresponds to a shortcut computed in the $i + 1$-th iteration of the algorithm. However, this shortcut does not need to have the same set $\mathcal{X}$ of reached predicates, but only a subset of it. The reason for this is that firings can always be applied, regardless of whether they are necessary to deactivate some token

or not. This means that replacement sequences might contain irrelevant firings. However, Alg. 3 computes shortcuts in such a way that only necessary firings are considered, i.e., only possibilities for predicates that were already reached.

**Lemma 14.** *Consider the variant of Alg. 3 that never returns, but simply computes the sets $\mathcal{R}_i$ for all $i \geq 0$. Let $P \in \mathcal{D}^f(\mathcal{C})$. If there is a $(P, \mathcal{X}, fw)$-replacement sequence of height $\leq i$, then $(P, \mathcal{X}') \in \mathcal{R}_{i+1}$ for some $\mathcal{X}' \subseteq \mathcal{X}$.*

These results can be used to show that the algorithm is correct. If Alg. 3 returns `true`, then Lemma 13 allows us to construct a terminating firing sequence from the computed shortcuts. On the other hand, if there is such a sequence, Lemma 14 shows that Alg. 3 computes enough shortcuts to detect its existence.

**Theorem 15.** *Termination of propagation nets of the form $\mathcal{N}_\mathcal{C}$ for normalized sets $\mathcal{C}$ of propagation rules can be decided in time exponential in the size of $\mathcal{C}$.*

*Proof (Sketch).* We have $\mathcal{R}_{i-1} \subseteq \mathcal{R}_i$ after every step of Alg. 3. Since there are only exponentially many possible shortcuts and $\texttt{nextShortcuts}(\mathcal{C}, \mathcal{R}_i)$ takes at most exponential time, the overall runtime is also exponential. □

**Corollary 16.** *The existence of finite Herbrand models for finite sets of propagation rules can be decided in* ExpTime.

*Proof.* This follows from Theorem 5 and the reductions of Sects. 2 and 3. □

If all the clauses of $\mathcal{C}$ are *deterministic*, i.e., have at most one possibility, the propagation net $\mathcal{N}_\mathcal{C}$ is called *deterministic*. Then all places of $\mathcal{N}_\mathcal{C}$ have at most one outgoing arc and the algorithm runs in time polynomial in the size of $\mathcal{C}$. For every additional nondeterministic clause in the set $\mathcal{C}$, the runtime of the algorithm increases by an exponential factor due to the computation of all possibilities and all shortcuts in $\texttt{possibilities}(\mathcal{C}, \mathcal{R}, P)$ and $\texttt{nextShortcuts}(\mathcal{C}, \mathcal{R})$.

## 5 Hardness

To conclude the complexity analysis, we present a reduction from linear language equations to finite sets of propagation rules. The equations are of the form

$$S_0 \cup S_1 X_1 \cup \cdots \cup S_n X_n = T_0 \cup T_1 X_1 \cup \cdots \cup T_n X_n$$

for finite sets $S_0, \ldots, S_n, T_0, \ldots, T_n$ of words over an alphabet $\Sigma$. A *solution* assigns finite sets of words to the variables $X_i$ such that the equation holds. Deciding whether such an equation has a solution is ExpTime-complete [1].

We can transform such equations into *flat linear language inclusions*

$$L_0 X_0 \subseteq L_1 X_1 \cup \cdots \cup L_n X_n$$

for $L_0, \ldots, L_n \subseteq \Sigma \cup \{\varepsilon\}$. By *flat* we mean that all coefficients contain only words of length at most 1. This can be achieved in polynomial time.

*Example 17.* Consider the equation $\{rs\}\cup\{s\}Y\cup X = \{r\}Y\cup\{s\}X\cup\{\varepsilon\}$.[2] If we abbreviate $\{r\}$ by $r$ and introduce a new variable $Z$, we can equivalently write this problem using the flat equations $rZ \cup sY \cup X = rY \cup sX \cup \varepsilon$ and $Z = s$. These are then split into the following flat linear language inclusions:

$$\begin{aligned}\mathcal{I}_1 := \{&rZ \subseteq rY \cup sX \cup \varepsilon,\ sY \subseteq rY \cup sX \cup \varepsilon,\ X \subseteq rY \cup sX \cup \varepsilon,\ Z \subseteq s\\ &rY \subseteq rZ \cup sY \cup X,\ sX \subseteq rZ \cup sY \cup X,\ \varepsilon \subseteq rZ \cup sY \cup X,\ s \subseteq Z\}.\end{aligned}$$

To solve a finite set $\mathcal{I}$ of such inclusions, we translate $\mathcal{I}$ into a finite set $\mathcal{C}_\mathcal{I}$ of propagation rules that express the same restrictions as the inclusions. We will treat each $r \in \Sigma$ as a unary function symbol, each variable $X$ occurring in $\mathcal{I}$ as a unary predicate. The intention behind $\mathcal{C}_\mathcal{I}$ is that a finite Herbrand model $\mathcal{H}$ of $\mathcal{C}_\mathcal{I}$ represents a solution $\theta$ of $\mathcal{I}$ with $\theta(X) = \{w \mid w(a) \in X^\mathcal{H}\}$.

To express an inclusion $L_0X_0 \subseteq L_1X_1 \cup \cdots \cup L_nX_n$ by clauses, we use the following idea. The clauses have to restrict the interpretation of the variables such that every word $w \in \Sigma^*$ occurring on the left-hand side of the inclusion also occurs on the right-hand side. For each word $w$ occurring in $L_0X_0$, we make a case analysis based on the first letter of $w$. We create one clause for the case $w = \varepsilon$, and one clause for every possible first letter of $w$.

*Example 18.* Consider the inclusion $rZ \subseteq rY \cup sX \cup \varepsilon$ from Example 17. Every word $w$ on its left-hand side has to begin with $r$, so the case analysis can be narrowed to one case. The corresponding clause is $Z(x) \to Y(x)$. Note that the terms $sX$ and $\varepsilon$ can never be responsible for this inclusion to be satisfied, and thus they are not represented in the clause.

Consider now another inclusion $X \subseteq rY \cup sX \cup \varepsilon$, which has to be split according to $s$, $r$, and $\varepsilon$. For the case that a word $w$ on the left-hand side begins with $r$, we introduce the clause $X(r(x)) \to Y(x)$. Similarly, for $s$ we obtain $X(s(x)) \to X(x)$. The case $w = \varepsilon$ is expressed by the clause $X(a) \to A(a)$, where $A$ is a special predicate that is always interpreted as $\{a\}$.

**Theorem 19.** *Deciding the existence of finite Herbrand models for finite sets of propagation rules is* ExpTime*-hard.*

## 6 Summary and Conclusions

Viewed from a different perspective, Alg. 3 and the reduction from Sect. 5 yield a new ExpTime-algorithm for deciding solvability of linear language equations. While the original decision procedure [1] constructs a tree automaton of exponential size and uses a linear-time emptiness test, our algorithm constructs a polynomial-size propagation net and uses an algorithm that is worst-case exponential, but exhibits a better behavior if the constructed set of propagation rules contains few nondeterministic clauses.

[2] This equation is equivalent to the $\mathcal{FL}_0$-unification problem $\forall r.\forall s.A \sqcap \forall s.Y \sqcap X \equiv^? \forall r.Y \sqcap \forall s.X \sqcap A$, where $A$ is a constant and $X, Y$ are variables (see [4] for details).

In future work, we want to modify the algorithm to actually compute solutions to the language equations and analyze the usefulness of these solutions; it may be desirable to output minimal solutions w.r.t. some order. We also want to implement the algorithm and compare it with an implementation of the naive tree automaton construction. To this end, we will have to design optimizations to our algorithm.

Another interesting open question is whether the presented approach can be applied to finite sets of arbitrary clauses with unary predicates, unary function symbols and constants. The formalism of propagation nets is certainly powerful enough to reflect this change, but the decision procedure also has to be adapted.

**Acknowledgement** We would like to thank Prof. Franz Baader for helpful discussions and comments.

## References

1. Baader, F., Narendran, P.: Unification of concept terms in description logics. J. Symb. Comput. 31(3), 277–305 (2001)
2. Baumgartner, P., Fuchs, A., de Nivelle, H., Tinelli, C.: Computing finite models by reduction to function-free clause logic. J. Appl. Log. 7(1), 58–74 (2009)
3. Birget, J.: State-complexity of finite-state devices, state compressibility and incompressibility. Math. Syst. Theory 26(3), 237–269 (1993)
4. Borgwardt, S., Morawska, B.: Finding finite Herbrand models. LTCS-Report 11-04, TU Dresden (2011), see http://lat.inf.tu-dresden.de/research/reports.html.
5. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. J. ACM 28(1), 114–133 (1981)
6. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata (2007)
7. Dreben, B., Goldfarb, W.D.: The Decision Problem: Solvable Classes of Quantificational Formulas. Addison-Wesley (1979)
8. Joyner Jr., W.H.: Resolution strategies as decision procedures. J. ACM 23(3), 398–417 (1976)
9. Ladner, R.E., Lipton, R.J., Stockmeyer, L.J.: Alternating pushdown and stack automata. SIAM J. Comput. 13(1), 135–155 (1984)
10. Leitsch, A.: The Resolution Calculus. Springer (1997)
11. Peltier, N.: Model building with ordered resolution: Extracting models from saturated clause sets. J. Symb. Comput. 36(1-2), 5–48 (2003)
12. Petri, C.A.: Kommunikation mit Automaten. Ph.D. thesis, Uni Bonn (1962)
13. Reisig, W.: Petri Nets: An Introduction. Springer (1985)
14. Slutzki, G.: Alternating tree automata. Theor. Comput. Sci. 41, 305–318 (1985)
15. Vardi, M.Y., Wolper, P.: Automata theoretic techniques for modal logics of programs (extended abstract). In: Proc. STOC'84. pp. 446–456. ACM (1984)
16. Zhang, J.: Constructing finite algebras with FALCON. J. Autom. Reasoning 17, 1–22 (1996)