The Horn Mu–calculus

Witold Charatonik *† David McAllester ‡ Damian Niwiński §¶
Andreas Podelski * Igor Walukiewicz §

Abstract

The Horn μ-calculus is a logic programming language allowing arbitrary nesting of least and greatest fixed points. The Horn μ-programs can naturally expresses safety and liveness properties for reactive systems. We extend the set-based analysis of classical logic programs by mapping arbitrary μ-programs into "uniform" μ-programs. Our two main results are that uniform μ-programs express regular sets of trees and that emptiness for uniform μ-programs is EXPTIME-complete. Hence we have a nontrivial decidable relaxation for the Horn μ-calculus. In a different reading, the results express a kind of robustness of the notion of regularity: alternating Rabin tree automata preserve the same expressiveness and algorithmic complexity if we extend them with pushdown transition rules (in the same way Büchi extended word automata to canonical systems).

1 Introduction

Regularity is a fundamental property of sets of words and trees. It forms the basis for the effectiveness of at least two important methods for analyzing the behavior of programs: set-based analysis and model-checking. The first method specifies the analysis problem through a logical formula with set-valued expressions (a set constraint) and computes its solutions in the form of automata over finite trees. In the second method, a property of ongoing behaviors of reactive finite-state systems is modeled as a regular set of infinite words or trees. As a fundamental first step towards the integration of the two methods, we investigate the fusion of the corresponding formalisms defining regular sets of, respectively, finite and infinite trees.

The logical formulas used in set-based analysis yield a syntactically rich formalism to define regular sets of trees; they are conceptually close to the programs from which they are directly derived. This is particularly apparent in the analysis of logic programs in [10], which uses a subclass of logic programs called uniform programs. Uniform programs, which are sets of Horn clauses in a certain form, subsume many other formalisms used in set-based analysis (e.g., in [23, 15, 12, 7, 5]) modulo simple translations.

As shown in [12, 10], the sets of finite trees defined by the least fixed point semantics of uniform programs are regular; the emptiness test is EXPTIME-complete. It is natural to ask whether also the greatest fixed point of a uniform program defines regular sets (of infinite trees). We give a positive answer not only to this question but also to its direct generalization to the full fixed point hierarchy: the sets of infinite trees defined by arbitrarily nested least and greatest fixed points of a uniform program are regular. Our first proof is not constructive; its purpose is to give a direct explanation of the expressiveness issue. To deal with algorithmic issues, we present a procedure that solves the emptiness problem for a set of infinite trees defined by a uniform program via arbitrarily nested fixed points in single-exponential time. Thus, the problem is EXPTIME-complete. A modified version of the algorithm transforms a uniform program into an equivalent alternating Rabin automaton; this yields a second, technically involved proof of the regularity.

A different reading of our results stems from the connection between set constraints and Büchi's canonical systems [3], which was observed first in [15]. Canonical systems may be viewed as a kind of pushdown automata whose input string is the initial stack contents. In this view, standard non-deterministic automata (nfa's) correspond to the special case where only pop transitions are allowed. Uniform programs generalize canonical systems to acceptors of infinite trees with alternation and with stacks containing infinite trees[1]; they extend alternating Rabin automata in the

*Max-Planck-Institut für Informatik,
D-66123 Saarbrücken, {witold;podelski}@mpi-sb.mpg.de
†on leave from Wroclaw University,
partially supported by Polish KBN grant 8T11C02913
‡AT&T Labs,
Murray Hill NJ 07974, dmac@research.att.com
§Institute of Informatics, Warsaw University,
02-097 Warsaw, Poland, {niwinski,igw}@mimuw.edu.pl,
partially supported by Polish KBN grant No. 8 T11C 002 11
¶visiting Max-Planck-Institut

[1] The tree-like stack distinguishes our notion of automaton of from the ones studied in [22, 11] which have a standard stack (and also a less general acceptance condition).

same sense that canonical systems extend nfa's. Our result thus expresses a kind of robustness of the notion of regularity wrt. expressiveness and algorithmic complexity under this extension.

Finite automata with a stack have raised interest as a kind of infinite reactive systems (then also called pushdown processes) for which various properties are decidable [18, 14, 13, 2, 4, 25, 9], and also in relation with dynamic logics (see, e.g., [11]). The decidability results seem surprising; our results may add some intuition why they are possible.

Our results have a promising potential for concrete applications to the set-based analysis of reactive programs and to the verification of infinite reactive systems such as pushdown processes. Given any infinite reactive system specified by a logic program P, we can derive a uniform program $P^{\#}$ that expresses the set-based abstraction (à la [12]) of P. Using our algorithm for the emptiness test, we can decide whether $P^{\#}$ satisfies any μ-calculus formula φ. Namely, the product construction as in [1, 9] applied to $P^{\#}$ and the Rabin tree automaton $\mathcal{A}_\varphi$ expressing φ is a uniform program with priorities (i.e., a pushdown Rabin automaton, instead of a pushdown Büchi automaton as in [9]). We thus obtain an algorithm for the abstract verification of the μ-calculus formula φ for the original program P. That is, for each input variable x of the program, we can compute a regular set that conservatively approximates the set of all values that x can take in a correct input state (correct wrt. φ).

If we specify a pushdown process by a logic program, we immediately obtain a uniform program; i.e., the abstract verification described above yields a full test of the μ-calculus property φ (in comparison, the method described in [6] applies only to CTL properties). This holds still true if we extend pushdown processes to the form that corresponds exactly to uniform programs, i.e., with tree-like stacks, nondeterministic guesses of stack contents and alternation.

2 Programs, priorities, fixed points

We define a *logic program* to be a finite set of Horn clauses, i.e., a set of expressions of the form $H \leftarrow B$ where H is a first order atom, i.e., a predicate applied to first order terms, and B is a finite set of first order atoms. It is well known that logic programs can be assigned both a least and a greatest fixed point semantics. In this section we will use an assignment of priorities to predicate symbols to define a semantics that combines least and greatest fixed points. First, however, it is useful to review the standard least and greatest fixed point semantics of logic programs.

In greatest fixed point semantics one usually works with infinite terms, e.g., the infinite term $f(f(f(\ldots)))$ where f is a monadic function symbol. Throughout the remainder of this paper we will use the phrase "ground term" to mean a possibly infinite term which does not contain variables. A ground atom is defined to be a predicate symbol of n arguments applied to n (possibly infinite) ground terms. A ground clause is an expression of the form $H \leftarrow B$ where H is a ground atom and B is a finite set of ground atoms. We say that a ground clause $H' \leftarrow B'$ is a ground instance of a Horn clause $H \leftarrow B$ if there exists a substitution σ mapping variables to ground terms such that $H' \leftarrow B'$ is the result of replacing each variable x in $H \leftarrow B$ by $\sigma(x)$. Note that under these definitions a Horn clause contains only finite terms while a ground instance of that clause may contain infinite terms.

Consider a possibly infinite term r such that each node w of r is labeled with a ground atom denoted $r(w)$. Let $C(w)$ denote the set of nodes which are children of the node w and let $r(C(w))$ denote the set of ground atoms of the form $r(s)$ for $s \in C(w)$. Given a logic program P we say that r is a P-derivation if for each node w in r we have that $r(w) \leftarrow r(C(w))$ is a ground instance of a clause in P. For any logic program P we define the greatest fixed point meaning of P, denoted $[\![P]\!]_\nu$, to be the set of all ground atoms Φ such that there exists a (possibly infinite) derivation tree whose root node is labeled with Φ. We define the least fixed point semantics of P, denoted $[\![P]\!]_\mu$, to be the set of ground atoms such that there exists a finite P-derivation whose root is labeled with Φ. Note that if a program P contains a clause of the form $p(x)$ (with an empty right hand side) then $[\![P]\!]_\mu$ contains infinite atoms — the least fixed point semantics of P over infinite atoms is different from the least fixed point meaning of P over only finite atoms. However, for any finite signature finiteness is definable, i.e., one can write clauses defining a monadic predicate F such that $F(t) \in [\![P]\!]_\mu$ if and only if t is a finite term. Since finiteness is definable (under least fixed point semantics) interpretation over infinite terms can simulate interpretation over finite terms.

Priorities We now generalize the semantics to allow arbitrary nestings of greatest and least fixed points. We define a Horn μ-program to be a logic program in which every predicate symbol p is associated with a non-negative integer $\Omega(p)$ called the priority of p. We say that a priority occurs in a program P if it equals $\Omega(p)$ for some p occurring in P. The priority of a ground atom is defined to be the priority of the predicate used in that atom. If P is a Horn μ-program then a P-derivation is defined as before. However, we now distinguish accepting from non-accepting P-derivations. Consider a P-derivation r and let w be an infinite sequence $w_0 w_1 w_2 \ldots$ of nodes in r such that w_{i+1} is a child of w_i. We will call such a sequence w an infinite path in r. Let $\mathrm{Inf}(w)$ be the set of priorities occurring infinitely often on the path, i.e., the set of priorities k such that there are infinitely many w_i where k is the priority of $r(w_i)$. We say that a path w in r is accepting if the largest element of $\mathrm{Inf}(w)$, is even. A P-derivation r is accepting if every infinite path in r is accepting. For any Horn μ-program P we define $[\![P]\!]$ to be the set of ground atomic formulas Φ such that there exists an accepting P-derivation where the root node is labeled with Φ.

Note that if all predicates have priority 0 then $[\![P]\!] = [\![P]\!]_\nu$. On the other hand, if all predicates have priority 1, then $[\![P]\!] = [\![P]\!]_\mu$. As an example of the added power of mixed priorities consider the program P consisting of the following clauses where p has priority 1 and q has priority 2.

$$p(f(x)) \leftarrow p(x)$$
$$p(x) \leftarrow q(x)$$
$$q(g(x)) \leftarrow p(x)$$

The acceptance condition implies that any infinite path in an accepting derivation must have an infinite number of occurrences of the predicate q. This implies that $[\![P]\!]$ contains all atoms of the form $p(t)$ and $q(g(t))$ where t is an infinite term constructed from the monadic functions f and g containing an infinite number of occurrences of g. Intuitively, since the predicate p has an odd index it must "terminate" in the use of a higher priority predicate. Since the higher priority q has an even index it need not terminate and can call itself — through terminating calls to p — an infinite number of times. This program expresses the statement that from any point one will eventually encounter a g — a kind of liveness property. It is possible to show that this property is not expressible under either purely least fixed point or purely greatest fixed point semantics.

In the remainder of this paper the term "program" will be used to mean Horn μ-program.

Fixed point semantics We now give an alternate characterization of the semantics $[\![P]\!]$ of a program P in terms of nested fixed point expressions. The fixed point characterization of $[\![P]\!]$ relates the parity condition (the definition of acceptance) given above to more classical formulations of fixed point logic. However, the fixed point characterization is not used in the remainder of the paper and is only provided here to strengthen the relationship with previous formalisms.

First we define the familiar T_P operator of logic programming. We define a database to be a set of ground atoms. For any database A we define $T_P(A)$ in the standard way to be the set of ground atoms Φ such that there exists a ground instance $\Phi \leftarrow B$ of a clause in P such that $B \subseteq A$. For any databases A and B and non-negative priority k we define $A[k := B]$ to be the database such that $\Phi \in A[k := B]$ if either the priority of Φ is different from k and $\Phi \in A$ or the priority of Φ equals k and $\Phi \in B$. We now define the operator T_P^k such that $T_P^k(A)$ equals $A[k := T_P(A)]$. The operator T_P^k is analogous to T_P except that T_P^k only updates predicates of priority k. We now take fixed points of the operators T_P^k separately for each value of k. More specifically, for any integer k we define $F_P^k(A)$ as follows. For negative k we define $F_P^k(A)$ to be A. For k nonnegative and even we define $F_P^k(A)$ to be the greatest set B such that $B = T_P^k(F_P^{k-1}(A[k := B]))$. In fixed point notation this can be written as

$$F_P^k(A) = \nu B.\ T_P^k(F_P^{k-1}(A[k := B]))$$

For nonnegative odd values of k the definition of F_P^k is analogous but using a least fixed point (μ) instead of a least fixed point (ν). The data base $F_P^k(A)$ is the database in which the meaning of predicates of priority greater than k is taken from A while the meaning of predicates of priority no greater than k is computed from their definition. Note that predicates of even priority are given greatest fixed point meanings while predicates of odd priority are given least fixed point meanings. We will now prove the following theorem.

Theorem 1 $[\![P]\!] = F_P^n(\emptyset)$ where n is the largest priority of any predicate in P.

This theorem is proved by formulating an appropriate induction on the priority k in the operators F_P^k. More specifically, for any program P, integer k, and database A we define a k-P-A-derivation to be a tree r where each node w is labeled with a ground atom $r(w)$ and has children $C(w)$ and where for each node w we either have that the priority of $r(w)$ is greater than k and $r(w) \in A$ or the priority of $r(w)$ is no greater than k and $r(w) \leftarrow r(C(w))$ is a ground instance of a clause in P. A k-P-A-derivation tree is called accepting if every infinite path in that tree is accepting, i.e., the largest priority occurring infinitely often on that path is even. It is possible to show that $\Phi \in F_P^k(A)$ if and only if there exists an accepting k-P-A-derivation of Φ. The claim is proved by induction on k. See the proof of Theorem 6.1 in [20] (or Theorem 3.2 in [21]) for a similar argument. Theorem 1 follows by taking $k = n$.

The case for k negative is trivial — for k negative we have $\Phi \in F_P^k(A)$ if and only if $\Phi \in A$ and there is a k-P-A-derivation of Φ if and only if $\Phi \in A$.

3 Uniform Programs

Let $[\![P,\ p]\!]$ denote the set of terms t such that $p(t) \in [\![P]\!]$. A set of ground terms is said to be definable by a Horn μ-program if it can be written as $[\![P,\ p]\!]$ for some P and p. The fact that the mu-level (all predicate have priority 1) corresponds to classical logic programs which are known to be r.e.-complete, immediately raises the concern that the language is overly-expressive — there appears to be no hope of computing properties of sets defined by Horn μ-programs.

Our main results say that the set-based relaxation for classical logic programs of [12, 10] can be generalized to a decidable relaxation for Horn μ-programs.

We will call a program Q a relaxation of a program P if $[\![P,\ p]\!] \subseteq [\![Q,\ p]\!]$ for any predicate symbol p occurring in P. If we want to prove that some set $[\![P,\ p]\!]$ is empty, it suffices to construct a relaxation Q of P and show that $[\![Q,\ p]\!]$ is empty. To show that some set $[\![P,\ p]\!]$ contains all terms it suffices to show that the complement is empty.

For any Horn μ-program P we now define the set-based relaxation, denoted $P^{\#}$, as follows. For each clause $H[x_1,\ \ldots,\ x_n] \leftarrow B[x_1,\ \ldots,\ x_n]$ containing variables $x_1, \ldots, x_n$ the following clauses are included in $P^{\#}$.

$$\tau_1(x_1) \leftarrow B[x_1,\ \ldots,\ x_n]$$

$$\vdots$$

$$\tau_n(x_n) \leftarrow B[x_1,\ \ldots,\ x_n]$$

$$H[y_1,\ \ldots,\ y_k] \leftarrow \tau_{h_1}(y_1),\ \ldots,\ \tau_{h_k}(y_k)$$

Here τ_i is a fresh predicate symbol of priority 0 representing the "type" of the variable x_i. The atom $H[y_1,\ \ldots,\ y_k]$ is the result of replacing each occurrence of a variable x_i in H by a fresh variable y_j so that $H[y_1,\ \ldots,\ y_k]$ is linear, i.e., no variable occurs more than once. The predicate τ_{h_j} in the body of the last clause equals τ_i where x_i is the variable replaced by y_j. For example, if P contains the clause $P(x,\ x,\ y) \leftarrow Q(x,\ y)$ then $P^{\#}$ contains the clauses $\tau_1(x) \leftarrow P(x,\ y)$, $\tau_2(y) \leftarrow Q(x,\ y)$ and $P(x_1,\ x_2,\ y) \leftarrow \tau_1(x_1),\ \tau_1(x_2),\ \tau_2(y)$. It is easy to see that any accepting P-derivation can be converted to an accepting $P^{\#}$-derivation. Hence $P^{\#}$ is a relaxation of P. The second main result of this paper states, in essence, that for any set $[\![P,\ p]\!]$ one can effectively determine, in single exponential time in the size of P, whether the set $[\![P^{\#},\ p]\!]$ is empty.

A clause of the form

$$p(f(x_1,\ \ldots,\ x_n)) \leftarrow q_1(x_1),\ \ldots,\ q_n(x_n)$$

will be called a reduction clause. A clause of the form

$$p(x) \leftarrow q(t)$$

will be called an expansion clause. A program will be called uniform if it consists entirely of reduction and expansion clauses. Any pair of the form $[\![P^{\#},\ p]\!]$ can be converted in linear time into a pair $[\![Q,\ q]\!]$ where Q is uniform and $[\![P^{\#},\ p]\!]$ is empty if and only if $[\![Q,\ q]\!]$ is empty. Our second main result can now be rephrased as the statement that if Q is uniform then the emptiness of $[\![Q,\ q]\!]$ can be decided in single exponential time.

Pushdown processes. Uniform programs can easily represent *pushdown processes* studied in, e.g., [14, 13, 2, 4, 25]. The clause $p(a(x)) \leftarrow q(x)$ expresses a *pop* transition from state p to state q under the condition that the first letter of the stack is a; the clause $p(x) \leftarrow q(x)$ is an ε-transition leaving the stack x invariant; the clause $p(x) \leftarrow q(a(x))$ is a *push* transition.[2] The computation trees (*pushdown trees* of [25]) become derivation trees in our sense. The issue of an *input* is absent in [25] (or, an input tree is always the starting stack symbol). The uniform programs extend pushdown processes in several directions. They take nontrivial inputs, permit free variables in bodies of clauses, and allow functional symbols of arbitrary arity.

4 Regularity of Uniform Programs

A clause of the form

$$p(x) \leftarrow q_1(x),\ \ldots,\ q_n(x)$$

will be called an intersection clause. Any program consisting of reduction, expansion, and intersection clauses can be efficiently converted to a uniform program — intersection clauses can be represented by expansion and reduction clauses. However, it is far less clear that expansion clauses can be replaced by intersection clauses. A program consisting entirely of reduction and intersection clauses will be called an alternating automaton. A program consisting entirely of reduction clauses will be called a nondeterministic automaton.

It is straightforward to see that the above concept of nondeterministic automaton coincides with a classical nondeterministic tree automaton with the *parity acceptance condition*, as considered in [8]. These automata are known to coincide in expressive power with Rabin automata and Muller automata [17, 8]. Also, an alternating automaton in our sense can be easily presented in the classical setting and it is known that this class of automaton has the same expressive power as the others — each of these kinds of automata can express exactly the same class of languages of possibly infinite terms. The languages in this class are called regular. Since uniform programs can express any set expressible by an alternating automaton, uniform programs can express any regular set. Our first main result is that the converse also holds — uniform programs can express exactly the class of regular sets.

Theorem 2 (Regularity) If P is uniform then the set $[\![P,\ p]\!]$ is regular, i.e., recognizable by a Rabin automaton.

[2] These three kinds of clauses correspond to the three kinds of rules in canonical systems, which Büchi called, respectively, contraction, neutration and expansion [3].

The proof gives a *non-constructive* method of converting a uniform program to an equivalent alternating automaton. The fundamental intuition is that expansion clauses can be "short-circuited" by intersection clauses. For example, consider the expansion clause $p(x) \leftarrow q(f(x,\ g(x,\ y)))$. Suppose the program also contains the reduction clauses $r(g(x,\ y)) \leftarrow h(x),\ w(y)$ and $q(f(x,\ u)) \leftarrow s(x),\ r(u)$. If one can determine that there exists a term t such that $w(t) \in [\![P]\!]$ then from these clauses we can infer the short-circuit intersection clause $p(x) \leftarrow s(x),\ h(x)$. Given enough of these intersection clauses it is possible to enumerate all the ways of short circuiting the expansion clauses and the expansion clauses become redundant.

In the case of purely least fixed point programs or purely greatest fixed point programs this basic intuition can be used as the foundation for a constructive algorithm for converting uniform programs to alternating automata. In the case of the Horn μ-calculus things are more complex. One problem is that the meaning of Horn μ-programs is not invariant under the addition of derived clauses. If the program contains $p(x) \leftarrow q(x)$ and $q(x) \leftarrow r(x)$ the introduction of the derived clause $p(x) \leftarrow r(x)$ may allow derivation trees which satisfy the parity condition by virtue of the fact that uses of the predicate q have been eliminated from certain derivation paths. It is possible, however, to formulate a nonstandard notion of derived clause such that the program remains invariant under the addition of derived clauses.

For each predicate p occurring in P and each priority k at least as large as the priority of p we introduce a new predicate p^k with priority k. We say that the intersection clause $p(x) \leftarrow B[x]$ is *derivable* from P if every predicate in the body $B[x]$ is one of the newly introduced predicates p^k and there exists a derivation tree r such that the root node of r is labeled with $p(a)$ where a is a fresh constant (not occurring in P), every infinite path in r is accepting (the largest priority occurring infinitely often is even), and for every node w in r either $r(w) \leftarrow r(C(w))$ is a ground instance of a clause in P (as in normal P-derivation) or $r(w)$ is an atom $q(a)$ where $B[x]$ contains $q^k(x)$ where k is largest priority occurring on the path from the root to the node w in the tree r. Now let P' consist of all the reduction clauses in P, plus all intersection clauses derivable from P, plus all clauses of the form $q^k(x) \leftarrow q(x)$. The program P' is an alternating automaton and hence $[\![P',\ p]\!]$ is a regular set for any predicate p occurring in P'. Furthermore, it possible to show that for any predicate p occurring in P we have that $[\![P,\ p]\!] = [\![P',\ p]\!]$.

Although we have not shown how to construct the derivable intersection clauses, we can bound the size of the resulting alternating automaton P'. If m is a number of predicate letters used in Ψ and n the maximum priority of these predicates, then Ψ' uses at most mn new predicates, but perhaps $2^{O(mn)}$ new clauses. Note that we can further transform Ψ' into a *nondeterministic* automaton by the construction of [19] that depends exponentially only on the number of *states* (that is, predicate letters in our setting). Thus we can obtain a nondeterministic automaton of the size $2^{|\Psi|^{0(1)}}$ equivalent to the program Ψ

5 The second main theorem

Theorem 3 If $\mathcal{P}$ is uniform then the emptiness of $[\![\mathcal{P}, p]\!]$ can be determined in $2^{|\mathcal{P}|^{O(1)}}$ time.

Outline of the proof We do not know of any direct way of constructing the set of derivable intersection clauses used in the proof of Theorem 5. Instead, we will eliminate the expansion clauses at the cost of expanding the term signature. The idea is to annotate a term with some hints pointing to reduction rules that can be used instead of an expansion rule. Two operators on terms will be in order: an operator $A(t)$ of annotating ordinary terms, and an operator $D(\bar{t})$ of deleting all annotations from an annotated term. We will write an alternating program $\mathcal{P}'$ such that $p(A(t)) \in [\![\mathcal{P}']\!]$, whenever $p(t) \in [\![\mathcal{P}]\!]$, and, if $p(\bar{t}) \in [\![\mathcal{P}']\!]$, for an annotated term $\bar{t}$, then $p(D(\bar{t})) \in [\![\mathcal{P}]\!]$. This way the emptiness problem for $[\![\mathcal{P}, p]\!]$ will be reduced to the emptiness problem for $[\![\mathcal{P}', p]\!]$. The theorem will follow since $\mathcal{P}'$ will be an alternating automaton of size polynomial in the size of $\mathcal{P}$.

In order to explain the construction and properties of the alternating program $\mathcal{P}'$, we find it convenient to use games of possibly infinite duration between two players whom we call *Prover* and *Opponent*. We first introduce a *program game* in which winning strategies for Prover correspond to derivations in $\mathcal{P}$. Then we define what we call a *flow game* which can be viewed as an intermediate construction between $\mathcal{P}$ and $\mathcal{P}'$. The key technical property is that winning strategies for both players can be transferred from program game to flow game, so that the nonemptiness of $\mathcal{P}$ is equivalent to the existence of a winning strategy for Prover in the flow game. Then we introduce the aforementioned annotated terms over an extended signature. These terms are closely related to the flow game: the winning strategies for Prover can be directly encoded by annotated terms. Conversely, from a suitably well formed annotated term, Prover can always read some strategy (not necessarily winning). Finally, we construct the alternating automaton $\mathcal{P}'$ which accepts precisely those annotated terms which encode winning strategies for Prover in the flow game. Thus, $[\![\mathcal{P}']\!]$ is nonempty iff Prover can win the flow game. But the latter is equivalent to the nonemptiness of $\mathcal{P}$, which completes the proof.

Games We recall the concept of a game of possibly infinite duration played by two players, Ms. 0 and Ms. 1, on ranked graphs called *arenas*. An arena is a bipartite graph whose nodes are partitioned into the sets V_0 and V_1 of *positions* of Ms. 0 and Ms. 1, respectively. There is also a finitely valued function $\Omega : V_0 \cup V_1 \to \mathbb{N}$ called the *priority function* [8, 16]. The players move a token alternately, so that a *play* is recorded as a path in the arena. If a player cannot move, the other player wins. If a play is infinite then Ms. i wins if the highest priority occurring infinitely often during the play *modulo* 2 is i. A strategy for Ms. i tells how to prolongate a finite path ending in V_i by one edge. Such a strategy is *winning* (for Ms. i) if any maximal play consistent with the strategy is won by Ms. i.

It is known that any game of such form is determined, i.e., at every position one of the players has a strategy to win. Moreover, a winning strategy can be assumed *positional* (or memoryless), i.e., depending only on the last position of an actual play. Such a strategy for Ms. i can be presented as a partial function from V_i to V_{1-i}, defined for all positions from which Ms. i can win. (It does not depend on the initial position of the play.)

We shall consider games related to a uniform program $\mathcal{P}$ in which one of the players called *Prover* wants to show that a ground atom is accepted by $\mathcal{P}$.

In what follows we fix a uniform program $\mathcal{P}$ with a priority function Ω. Let Pred stand for the set of predicate letters appearing in $\mathcal{P}$. Let $GTerms$ stand for the set of (possibly infinite) ground terms over the function symbols appearing in $\mathcal{P}$.

The first game is just rephrasing the semantics of programs with priorities in game theoretic terms.

Definition 1 (Program game) We define *program game* $\mathcal{G}(\mathcal{P}, \Omega) = \langle V_p, V_o, E, \Omega_g \rangle$ by the following clauses.

- The set V_p of positions of Prover consists of atoms $p(t)$ for $p \in \mathsf{Pred}$ and t a ground term.
- The set V_o of positions of Opponent consists of the pairs (R, σ) where R is a rule of $\mathcal{P}$ and $\sigma : \mathsf{Var} \to GTerms$ is a substitution.
- For every rule R of $\mathcal{P}$, say $R \equiv p(\tau) \leftarrow q_1(\tau_1), \ldots, q_k(\tau_k)$, there is an edge from $p(t)$ to $\langle R, \sigma \rangle$ if $\sigma(\tau) = t$. From the position $\langle R, \sigma \rangle$ there are edges to $q_i(\sigma(\tau_i))$ for $i = 1, \ldots, k$.
- $\Omega_g(p(t)) = \Omega(p)$ and $\Omega_g(v) = 0$ for $v \in V_o$.

The following is an easy consequence of definitions.

Proposition 2 Prover has a winning strategy from a position $p(t)$ iff the ground atom $p(t)$ is accepted by $\mathcal{P}$.

We now define a game that can be viewed as an intermediate step between the program $\mathcal{P}$ and an alternating automaton $\mathcal{P}'$ we are going to construct in the next section. It is motivated by the following situation that can occur in a program game defined above. Suppose that Prover wishes to prove $p(t)$, and to this end selects a rule R; in other words, she moves from position $p(t)$ to (R, σ). Let R be an expansion rule, say $p(x) \leftarrow q(\tau)$. Then Opponent has only one possibility, and from the subsequent position Prover will have to prove $q(\sigma(\tau))$. Note that the term $\sigma(\tau)$ "expands" the term t (i.e., t is a subtree of $\sigma(\tau)$). Such situation is avoided in *flow game* defined below. While using rule $p(x) \leftarrow q(\tau)$, Prover may offer to Opponent several possibilities, but none of them will expand the tree t. (The play always "flows down" in a tree.) Roughly speaking, Prover will divide the job of showing $q(\sigma(\tau))$ into several tasks, proving separately correctness of substitution σ, and correctness of a *pattern* τ.

It will be now necessary to keep track of the highest priority seen so far in course of a play (unless reset). This value will be essential for determining the priority of a position. Therefore, the positions of Prover will be now decorated by two parameters ranging over $\mathrm{Rg}(\Omega)$, denoted $clock$ and $prio$, respectively. Here and below we use $\mathrm{Rg}(\Omega)$ for the range of the function Ω.

Definition 3 (Patterns) We extend the term signature by the set $Aconst$ of *assumption constants* of the form c_W, for $W \subseteq \mathsf{Pred} \times \mathrm{Rg}(\Omega)$. A *pattern* is a finite term over the new signature. We use $Patterns$ for the set of patterns.

Definition 4 (Flow game) For $\mathcal{P}$ and Ω as above we define a *flow game* $\mathcal{FG}(\mathcal{P}, \Omega) = \langle V_p, V_o, E, \Omega_{fg} \rangle$ by the following clauses.

- The set V_p of positions of Prover consists of triples of the form $\langle p(t), clock, prio \rangle$, where t is a term or a pattern, and $clock, prio \in \mathrm{Rg}(\Omega)$.
- The set V_o of positions of Opponent contains quadruples of the form $\langle R, clock, \sigma, \rho \rangle$, where R is a rule of $\mathcal{P}$, $clock \in \mathrm{Rg}(\Omega)$, σ is a substitution of terms or patterns for variables, and ρ is a substitution of assumption constants for variables. The set V_o also contains a distinguished position $ProverWon$.
- For every reduction rule R, say $R \equiv p(f(x_1, \ldots, x_k)) \leftarrow q_1(x_1), \ldots, q_k(x_k)$, there is an edge from $\langle p(f(t_1, \ldots, t_k)), clock, prio \rangle$ to $\langle R, clock, \sigma, \emptyset \rangle$, where $\sigma(x_i) = t_i$, for $i = 1, \ldots, k$. From the position $\langle R, clock, \sigma, \emptyset \rangle$, there are edges to the positions $\langle q_i(t_i), \max(clock, \Omega(q_i)), \Omega(q_i) \rangle$, for $i = 1, \ldots, k$.
- For every expansion rule R, say $R \equiv p(x) \leftarrow q(\tau)$, there is an edge from $\langle p(t), clock, prio \rangle$ to any position $\langle R, clock, \sigma, \rho \rangle$, provided $\sigma(y)$ is a term, for $y \neq x$. From the position $\langle R, clock, \sigma, \rho \rangle$, there is

an edge to the position $\langle q(\rho(\tau)), \Omega(q), \Omega(q)\rangle$ (note the absence of *max* operator here!), and to the position $\langle r(\sigma(y)), \max(clock, j), j\rangle$, for every (r, j) such that $(r, j) \in W$, for $c_W = \rho(y)$.

- There is an edge from a position $\langle p(c_W), clock, prio\rangle$ to $ProverWon$, provided $(p, clock) \in W$. There is no edge out from $ProverWon$.
- $\Omega_{fg}(\langle p(t), clock, prio\rangle) = prio$, and $\Omega_{fg}(v) = 0$ for $v \in V_o$.

We will show that both games are equivalent in a strong sense. In what follows, a finite path (in the game graph) is called a j-path if the highest priority encountered on the path is j.

Lemma 5 If Prover has a winning strategy in the game $\mathcal{G}(\mathcal{P}, \Omega)$ from a position $p_0(t_0)$ then she has a winning strategy in the game $\mathcal{FG}(\mathcal{P}, \Omega)$ from the position $\langle p_0(t_0), 0, 0\rangle$.

Proof
Let $\mathcal{S}$ be a positional winning strategy in $\mathcal{G} = \mathcal{G}(\mathcal{P}, \Omega)$. We will show how Prover can win from the position $\langle p_0(t_0), 0, 0\rangle$ in the game $\mathcal{FG} = \mathcal{FG}(\mathcal{P}, \Omega)$. We assume that while playing in $\mathcal{FG}$ Prover simulates the actual play by moving a token in the game graph of $\mathcal{G}$. We shall refer to this sequence of moves in $\mathcal{G}$ as to a *simulating play*. Then Prover will be able to take advantage of strategy $\mathcal{S}$: she will, in a sense, transfer it to $\mathcal{FG}$. Moreover, the following invariant will be kept during the play. Whenever Prover has to move from some vertex $\langle p(t), clock, prio\rangle$ then in the simulating play in the game $\mathcal{G}$ she is in a position $p(t')$ such that the following is satisfied.

- The strategy $\mathcal{S}(p(t'))$ is defined for the position $p(t')$.
- If t is a term then $t' = t$.
- If t is a pattern then there is a term τ and two substitutions $\widetilde{\sigma} : \mathsf{Var} \to GTerms$ and $\widetilde{\rho} : \mathsf{Var} \to Aconst$ such that $t = \widetilde{\rho}(\tau)$ and $t' = \widetilde{\sigma}(\tau)$. Moreover, for every variable y, if $\widetilde{\sigma}(y)$ is defined and if there is a j-path consistent with $\mathcal{S}$ from $p(t')$ to $r(\widetilde{\sigma}(y))$ then $(r, \max(clock, j)) \in W$, where W is the subscript in $\widetilde{\rho}(y) = c_W$.

This invariant guarantees that whenever the play in $\mathcal{FG}$ reaches a position $\langle p(c_W), clock, prio\rangle$, this position is winning for Prover, i.e. there is an edge from there to $ProverWon$. (At position $ProverWon$ Opponent looses as he cannot make any move.) Indeed, by the invariant, the simulating play in $\mathcal{G}$ is at this moment in a position $p(t')$ and the invariant is satisfied. In this case τ is just a variable, say x, $\widetilde{\rho}(x) = c_W$, and $\widetilde{\sigma}(x) = t'$. Clearly there is a $\Omega(p)$–path (of length 1) from $p(t')$ to $p(t')$ in $\mathcal{G}$. So, by the invariant, $(p, \max(clock, \Omega(p))) \in W$. But, by definition of the game $\mathcal{FG}$, we have $\max(clock, \Omega(p)) = clock$, and consequently the position $\langle p(c_W), clock, prio\rangle$ is winning for Prover.

We now describe how Prover should play in $\mathcal{FG}$ starting from the position $\langle p_0(t_0), 0, 0\rangle$. The simulating play in $\mathcal{G}$ starts from $p_0(t_0)$. Clearly these positions satisfy the invariant since t_0 is a term.

Next suppose that the plays are in positions $\langle p(t), clock\ prio\rangle$ and $p(t')$ in $\mathcal{FG}$ and $\mathcal{G}$, respectively. The invariant guarantees that the value of $\mathcal{S}(p(t'))$ is defined. We will show how Prover should move in $\mathcal{FG}(\mathcal{P}, \Omega)$ keeping the invariant satisfied; she will use the value of $\mathcal{S}(p(t'))$ as an advice.

If $\mathcal{S}(p(t'))$ is some position with a reduction rule,

$$\langle p(f(x_1, \ldots, x_k)) \leftarrow q_1(x_1), \ldots, q_k(x_k), \sigma\rangle$$

then Prover moves in $\mathcal{FG}(\mathcal{P}, \Omega)$ to the, uniquely determined, position with the same reduction rule. It has the form

$$\langle p(f(x_1, \ldots, x_k)) \leftarrow q_1(x_1), \ldots, q_k(x_k), clock, \sigma', \emptyset\rangle$$

From there, Opponent can move to one of the positions $\langle q_i(\sigma'(x_i)), \max(clock, \Omega(q_i)), \Omega(q_i)\rangle$, for $i = 1, \ldots, k$. Now, the simulating play in $\mathcal{G}$ should be updated accordingly. Prover does it by simulating the move of Opponent of choosing the position $q_i(\sigma(x_i))$ in $\mathcal{G}$. The pair of actual positions becomes

$$\langle q_i(\sigma'(x_i)), \max(clock, \Omega(q)), \Omega(q)\rangle \quad \text{and} \quad q_i(\sigma(x_i))$$

It is not difficult to see that the invariant is satisfied.

The other case is when $\mathcal{S}(p(t'))$ is a position with an expansion rule: $\langle p(x) \leftarrow q(s), \sigma\rangle$. In this case Prover chooses the position

$$\langle p(x) \leftarrow q(s), clock, \sigma', \rho'\rangle \tag{1}$$

Here $\sigma'(x) = t$ and $\sigma'(y) = \sigma(y)$ for $y \neq x$. Additionally, $\rho'(z) = c_W$ for W containing all the pairs (r, j) such that there is a j-path consistent with $\mathcal{S}$ from $q(\sigma(s))$ to $r(\sigma(z))$.

From the position (1) Opponent may choose to move to the position $\langle q(\rho'(s)), \Omega(q), \Omega(q)\rangle$. In this case, in the game $\mathcal{G}$ Prover simulates this choice of Opponent by moving to $q(\sigma(s))$. The obtained pair of positions is $\langle q(\rho'(s)), \Omega(q), \Omega(q)\rangle$ and $q(\sigma(s))$. To see that it satisfies our invariant, take s for τ, take σ for $\widetilde{\sigma}$ and take ρ' for $\widetilde{\rho}$ in the description of the invariant.

From the position (1), Opponent may also move to $\langle r(\sigma'(y)), \max(clock, j), j\rangle$ for some $(r, j) \in W$, $\rho'(y) = c_W$. In this case, we know that in the game $\mathcal{G}$ there is a

$\max(\Omega(p), j)$-path consistent with $\mathcal{S}$ from $p(t')$ to $r(\sigma(y))$. This is because there is a $\Omega(p)$-path from $p(t')$ to $q(\sigma(s))$ (it has length 2), and a j-path from $q(\sigma(s))$ to $r(\sigma(y))$. In the game $\mathcal{G}$, Prover can simulate a part of the game leading to the position $r(\sigma(y))$. We claim that the obtained two positions:

$$\langle r(\sigma'(y)), \max(\mathit{clock}, j), j\rangle \quad \text{and} \quad r(\sigma(y))$$

satisfy the invariant. Indeed, if $y \neq x$ then the two obtained positions satisfy the invariant as $\sigma'(y) = \sigma(y)$ is a term. If $y = x$ then we need to use our assumption. Let τ be a term and $\widetilde{\sigma}$ and $\widetilde{\rho}$ be the substitutions given by the invariant for the positions $\langle p(t), \mathit{clock}, \mathit{prio}\rangle$ and $p(t')$. We want to show that this choice of substitutions also works for the positions $\langle r(\sigma'(x)), \max(\mathit{clock}, j), j\rangle$ and $r(\sigma(x))$. We have $\sigma'(x) = t$ and $\sigma(x) = t'$ so indeed $\sigma'(x) = \widetilde{\rho}(\tau)$ and $\sigma(x) = \widetilde{\sigma}(\tau)$. Let y be a variable such that $\widetilde{\sigma}(y)$ is defined and there is a j'-path consistent with $\mathcal{S}$ from $r(\widetilde{\sigma}(y))$ to $r'(t')$. Then there is a $\max(\Omega(p), j, j')$-path consistent with $\mathcal{S}$ from $p(t')$ to $r'(\sigma(y))$. By our invariant for the positions $\langle p(t), \mathit{clock}, \mathit{prio}\rangle$ and $p(t')$, we know that $(r', \max(\mathit{clock}, \Omega(p), j, j')) \in W$. Since, by definition of $\mathcal{FG}$, $\max(\mathit{clock}, \Omega(p)) = \mathit{clock}$, we have $(r', \max(\mathit{clock}, j, j')) \in W$, and we are done.

We have shown how Prover can play in $\mathcal{FG}$ keeping the invariant satisfied. The above description can be easily formalized as a strategy function for Prover (it may not be positional). The invariant guarantees that whenever a play terminates, it is won by Prover. If a play is infinite, the simulating play in $\mathcal{G}$ is also infinite. Now, it follows from the construction that the highest priority encountered between any two positions of the play in $\mathcal{FG}$ is the same as the highest priority encountered between the corresponding positions in simulating play. Since the latter is consistent with the strategy $\mathcal{S}$ (again, by invariant), it is winning for Prover in $\mathcal{G}$, and so is the play in $\mathcal{FG}$. □

Lemma 6 If Opponent has a winning strategy in the game $\mathcal{G}(\mathcal{P}, \Omega)$ from a position $p_0(t_0)$ then Opponent has a winning strategy in the game $\mathcal{FG}(\mathcal{P}, \Omega)$ from the position $\langle p_0(t_0), 0, 0\rangle$.

Proof

Let $\mathcal{OS}$ be a positional winning strategy for Opponent in $\mathcal{G} = \mathcal{G}(\mathcal{P}, \Omega)$. We will show how Opponent can win from the position $\langle p_0(t_0), 0, 0\rangle$ in the game $\mathcal{FG} = \mathcal{FG}(\mathcal{P}, \Omega)$. The argument will be analogous to the one in the previous lemma. We assume that while playing in $\mathcal{FG}$ Opponent additionally performs a sequence of moves in $\mathcal{G}$, simulating by this the actual play in $\mathcal{FG}$. Again, we refer to the path in $\mathcal{G}$ created in that way as to a simulating play. Moreover, whenever the plays simultaneously reach positions $\langle p(t), \mathit{clock}\ \mathit{prio}\rangle$ and $p(t')$ in $\mathcal{FG}$ and $\mathcal{G}$, respectively (these are positions of Prover), the following invariant will hold.

- The strategy $\mathcal{OS}$ is defined for all successor positions of $p(t')$.
- If t is a term then $t' = t$.
- If t is a pattern then there exist a term τ, and two substitutions $\widetilde{\sigma} : \mathsf{Var} \to \mathit{GTerms}$ and $\widetilde{\rho} : \mathsf{Var} \to \mathit{Aconst}$, such that $t = \widetilde{\rho}(\tau)$ and $t' = \widetilde{\sigma}(\tau)$. Moreover, for every variable y, if $\widetilde{\sigma}(y) = c_W$ and $(r, \max(\mathit{clock}, j)) \in W$ then there is no j-path consistent with $\mathcal{OS}$ from $p(t')$ to $r(\widetilde{\sigma}(y))$.

This invariant guarantees that whenever the play in $\mathcal{FG}$ reaches a position $\langle p(c_W), \mathit{clock}, \mathit{prio}\rangle$, Prover cannot move to the position $\mathit{ProverWon}$. Indeed, suppose that the simulating play in $\mathcal{G}$ reaches simultaneously a position $p(t')$ and the invariant is satisfied. In this case τ is just a variable, say x, $\widetilde{\rho}(x) = c_W$ and $\widetilde{\sigma}(x) = t'$. Clearly there is a $\Omega(p)$–path (of length 1) from $p(t')$ to $p(t')$ in $\mathcal{G}$. By definition of the game $\mathcal{FG}$, we have $\max(\mathit{clock}, \Omega(p)) = \mathit{clock}$. So, by the invariant, $(p, \mathit{clock}) \notin W$, and form the position $\langle p(c_W), \mathit{clock}, \mathit{prio}\rangle$ there is no edge to $\mathit{ProverWon}$ in $\mathcal{FG}$.

We now describe how Opponent should play in $\mathcal{FG}$ starting from the position $\langle p_0(t_0), 0, 0\rangle$. The simulating play in $\mathcal{G}$ starts from $p_0(t_0)$. Clearly these positions satisfy the invariant.

Now suppose the plays are in positions $\langle p(t), \mathit{clock}\ \mathit{prio}\rangle$ and $p(t')$ in $\mathcal{FG}$ and $\mathcal{G}$, respectively. We will show how Opponent should move in $\mathcal{FG}(\mathcal{P}, \Omega)$ from the *subsequent* positions, keeping the invariant satisfied.

Suppose Prover chooses a node with a reduction rule:

$$\langle p(f(x_1, \ldots, x_k)) \leftarrow q_1(x_1), \ldots, q_k(x_k), \mathit{clock}, \sigma', \emptyset\rangle \tag{2}$$

In this case Opponent updates the play in $\mathcal{G}$ by simulating a move of Prover of choosing the position

$$\langle p(f(x_1, \ldots, x_k)) \leftarrow q_1(x_1), \ldots, q_k(x_k), \sigma\rangle$$

By the invariant, the strategy $\mathcal{OS}$ is defined for this position. Let its value be $q_i(\sigma(x_i))$. Opponent transfers this move back to $\mathcal{FG}$ by moving from the position (2) to the position $\langle q_i(\sigma(x_i)), \max(\mathit{clock}, \Omega(q_i)), \Omega(q_i)\rangle$. It is not difficult to check that the obtained two positions satisfy the invariant.

The other case is when Prover chooses a node with an expansion rule $\langle p(x) \leftarrow q(s), \mathit{clock}, \sigma', \rho'\rangle$. In this case Opponent updates the play in $\mathcal{G}$ by simulating a move of Prover to the position $\langle p(x) \leftarrow q(s), \sigma\rangle$, where $\sigma(x) = t'$ and $\sigma(y) = \sigma'(y)$, for $y \neq x$. Note that in $\mathcal{G}$ Opponent has now only one possibility, he should move to $q(\sigma(s))$.

Suppose there is a variable y with $\rho'(y) = c_W$ and $(r, j) \in W$, such that there is a j-path consistent with $\mathcal{OS}$ from $q(\sigma(s))$ to $r(\sigma(y))$. In this case Opponent chooses position $\langle r(\sigma'(y)), \max(\mathit{clock}, j), j\rangle$. Finally, he updates the play in $\mathcal{G}$ by simulating a sequence of moves leading to $r(\sigma(y))$. It can be checked that the invariant is satisfied.

If there is no variable with the property mentioned above, Opponent decides to move to the position $\langle q(\rho'(s)), \Omega(q), \Omega(q)\rangle$ instead. Next, he updates the play in $\mathcal{G}$ by moving to the position $q(\sigma(s))$. It follows from the requirement on ρ' that the invariant is satisfied for $\tau = s$, $\widetilde{\rho} = \rho'$ and $\widetilde{\sigma} = \sigma$.

The concluding argument is the same as in the previous lemma. □

Annotated terms In this subsection we extend the term signature and introduce the concept of annotated terms. These terms can be seen as encodings of strategies for Prover in the game $\mathcal{FG}$. Then we present an alternating automaton $\mathcal{P}'$ recognising those annotated terms that represent winning strategies. The results of the previous subsection will allow us to reduce the emptiness problem for $\mathcal{P}$ to the emptiness problem for $\mathcal{P}'$.

For simplicity of notation we fix a variable x together with a sequence of *reduction variables* $RVar = x_1, x_2, \ldots$, and a sequence of *expansion variables* $EVar = y_1, y_2, \ldots$ We assume that $x \notin RVar \cup EVar$ and $RVar \cap EVar = \emptyset$. Without loss of generality we can assume that every reduction rule in the program $\mathcal{P}$ is of the form $p(f(x_1, \ldots, x_n)) \leftarrow q_1(x_1), \ldots, q_n(x_n)$ (i.e., it uses an initial segment of $RVar$). Similarly we assume that every expansion rule in $\mathcal{P}$ is of the form $p(x) \leftarrow q(\tau)$ where x is the distinguished variable, and the term τ using x and some initial segment of $EVar$. We also assume that $RVar$ and $EVar$ are finite and bounded by the size of $\mathcal{P}$.

Let k be the number of triples of the form $(p, \mathit{clock}, \mathit{prio})$, where $p \in \mathsf{Pred}$ and $\mathit{clock}, \mathit{prio} \in \mathrm{Rg}(\Omega)$. We call such triples *indexed predicates*. We let $\overline{p}, \overline{q}, \ldots$ range over indexed predicates.

Definition 7 (Annotated terms) An *annotated term* is a (possibly infinite) term over the signature consisting of functional symbols of the program $\mathcal{P}$ and the following symbols:

- the symbol Π of arity $k + 1$,
- for every rule $p(x) \leftarrow q(\tau)$ of $\mathcal{P}$ a symbol $\mathit{Exp}_{q(\tau)}$ of arity greater by one than the number of expansion variables in τ,
- for every rule $p(f(x_1, \ldots, x_n)) \leftarrow q_1(x_1), \ldots, q_n(x_n)$ of $\mathcal{P}$, a constant $\mathit{Red}_{q_1,\ldots,q_n}$,
- a binary operation cons, a constant nil and a constant c_w, for every $w \in \mathsf{Pred} \times \mathrm{Rg}(\Omega)$. (These are used to encode assumption constants $c_W \in \mathit{Aconst}$ as list of c_w constants.)

We define the *deletion operator* $D(t)$ on annotated terms by:

$$\begin{aligned} D(\Pi(\mathit{cons}(t, t'), \ldots)) &= \mathit{cons}(t, t') \\ D(\Pi(f(t_1, \ldots, t_n), \ldots)) &= f(D(t_1), \ldots, D(t_n)) \\ &\quad \text{for } f \neq \mathit{cons} \end{aligned}$$

We think of annotated terms as of encodings of strategies for Prover in the game $\mathcal{FG}$. For every indexed predicate $\overline{p} = \langle p, \mathit{clock}, \mathit{prio}\rangle$ Prover can read from a term $\Pi(t, h_1, \ldots, h_k)$ which move to make in the position $\langle p(D(t)), \mathit{clock}, \mathit{prio}\rangle$ of $\mathcal{FG}$. For this, Prover looks at the hint $h_{\overline{p}}$ (we assume a bijection between the $\overline{p}$'s and numbers $1, \ldots, k$). She expects to find there a constant $\mathit{Red}_{\overline{q}}$ or a term $\mathit{Exp}_{q(\tau)}(s, s_1, \ldots, s_n)$. These describe a move Prover should make. To encode a strategy in this way, an annotated term should be structured in a proper way. This is captured by the notion of well formed annotated term.

An annotated term t *is well formed* if it is of the form $\Pi(s, h_1, \ldots, h_k)$ and satisfies the following conditions.

- $s = f(s_1, \ldots, s_n)$ where f is a symbol from the original signature and $s_1, \ldots, s_n$ are well formed; or $s = \mathit{cons}(c_{w_1}, \mathit{cons}(\ldots \mathit{cons}(c_{w_i}, \mathit{nil}) \ldots))$, for some $w_1, \ldots, w_i \in \mathsf{Pred} \times \mathrm{Rg}(\Omega)$
- For every indexed predicate $\overline{p} = (p, \mathit{clock}, \mathit{prio})$ and the corresponding argument $h_{\overline{p}}$ of $\Pi(s, h_1, \ldots, h_k)$ one of the following holds:
  - $h_{\overline{p}} = \mathit{Red}_{q_1,\ldots,q_n}$; $p(f(x_1, \ldots, x_n)) \leftarrow q_1(x_1), \ldots, q_n(x_n)$ is a rule of $\mathcal{P}$; and $s = f(s_1, \ldots, s_n)$
  - $h_{\overline{p}} = \mathit{Exp}_{q(\tau)}(s', s'_1, \ldots, s'_n)$; $p(x) \leftarrow q(\tau)$ is a rule of $\mathcal{P}$; $s', s'_1, \ldots, s'_n$ are well formed; and $D(s') = \rho(\tau)$ for some substitution $\rho : \mathsf{Var} \to \mathit{Aconst}$.

Intuitively, a well formed annotated term is an annotated term that suggests only legal moves for Prover. It is not difficult to write a program consisting only of reduction and intersection rules that accepts exactly well formed annotated terms.

We now explain how strategies can be encoded into annotated terms. For the rest of this section, we fix a winning positional strategy $\mathcal{S}$ for Prover in the game $\mathcal{FG}$. For every ground term or pattern t we define the annotated term $A(t)$.

$$\begin{aligned} A(c_W) &= \Pi(\overline{c_W}, h_1, \ldots, h_k) \\ A(f(t_1, \ldots, t_n)) &= \Pi(f(A(t_1), \ldots, A(t_n)), h_1, \ldots, h_k) \end{aligned}$$

where $\overline{c_W} = cons(c_{w_1}, cons(\dots cons(c_{w_i}, nil)\dots))$ for $\{w_1, \dots, w_i\} = W$ and, for every $\overline{p}$, the term $h_{\overline{p}}$ on the position corresponding to $\overline{p}$ is:

- $h_{\overline{p}} = Red_{q_1,\dots,q_n}$ if $\mathcal{S}(\langle p(t), clock, prio\rangle) = \langle p(f(x_1,\dots,x_n)) \leftarrow q_1(x_1),\dots,q_n(x_k), clock, \sigma, \emptyset\rangle$

- $h_{\overline{p}} = Exp_{q(\tau)}(A(\rho(\tau)), A(\sigma(y_1)),\dots,A(\sigma(y_n)))$ if $\mathcal{S}(\langle p(t), clock, prio\rangle) = \langle P(x) \leftarrow Q(\tau), clock, \sigma, \rho\rangle$ and $y_1,\dots,y_n$ are all the variables other than x in τ.

- if $\mathcal{S}(\langle p(t), clock, prio\rangle)$ is not defined then $h_{\overline{p}}$ is any term such that $A(t)$ is well formed.

We are now going to construct a program $\mathcal{P}'$. It is intended to accept $\overline{p}(\overline{t})$, for $\overline{p} = (p, 0, 0)$, precisely when $\overline{t}$ encodes some winning strategy for Prover from the position $\langle p(D(\overline{t})), 0, 0\rangle$ in the game $\mathcal{FG}$. The predicates of $\mathcal{P}'$ will be indexed predicates and some additional predicates introduced for technical reasons. These technical predicates will have priority 0 so we will be able to ignore them when deciding whether a path of a derivation in $\mathcal{P}'$ is accepting or not. This is why the nodes of a derivation labelled with atoms containing indexed predicates are called *real* and the other nodes are called *bureaucratic*.

The form of $\mathcal{P}'$ may seem slightly unusual but please recall that we are allowed to use only reduction and intersection rules. To understand the idea behind the construction of $\mathcal{P}'$ it may be helpful to imagine a derivation in $\mathcal{P}'$ of $(p_0, 0, 0)(A(t_0))$ for some predicate $p \in \mathsf{Pred}$ and some ground term t_0 such that $\mathcal{S}$ is defined for the position $\langle p_0(t_0), 0, 0\rangle$. We are after the following property of such a derivation.

> For every real node $\overline{p}(\overline{t})$ of the derivation, the strategy $\mathcal{S}$ is defined for $\langle p(D(\overline{t})), clock, prio\rangle$. Every path of the derivation from $\overline{p}(\overline{t})$ goes through some bureaucratic nodes and either it ends with an axiom or it reaches a real node $\overline{p}'(\overline{t}')$. We have that $\langle p'(D(\overline{t}')), clock', prio'\rangle$ is a successor of $\mathcal{S}(\langle p(D(\overline{t})), clock, prio\rangle)$ in $\mathcal{FG}$. (3)

A derivation satisfying this property very closely matches the strategy $\mathcal{S}$. In particular the derivation is accepting since $\mathcal{S}$ is winning.

We start the presentation of $\mathcal{P}'$ with definitions of auxiliary predicates $\mathrm{NA}_{r(z),j,\tau}$ (NA stands for "not assumed") for every $r \in \mathsf{Pred}$, $z \in EVar \cup x$, $j \in \mathrm{Rg}(\Omega)$ and τ a term occurring in the right hand side of some expansion rule $P(x) \leftarrow Q(\tau')$ of $\mathcal{P}$ (i.e., τ is a subterm of τ').

$$\mathrm{NA}_{r(z),j,z'}(z'') \leftarrow \qquad \text{for } z \neq z'$$
$$\mathrm{NA}_{r(z),j,z}(z'') \leftarrow notin_{(r,j)}(z'')$$
$$\mathrm{NA}_{r(z),j,\tau}(\Pi(z'',\dots)) \leftarrow \mathrm{NA}_{r(z),j,\tau}(z'')$$
$$\mathrm{NA}_{r(z),j,f(\tau_1,\dots,\tau_n)}(f(z_1,\dots,z_n)) \leftarrow \mathrm{NA}_{r(z),j,\tau_1}(z_1),\dots,\mathrm{NA}_{r(z),j,\tau_n}(z_n)$$
$$f \text{ a symbol from the original signature}$$

In the above, the predicate $notin_{(r,j)}(t)$ holds if t is a list $cons(c_{w_1}, cons(\dots cons(c_{w_i}, nil)\dots))$ and $(r,j) \notin \{w_1,\dots,w_n\}$. A predicate $\mathrm{NA}_{r(z),j,\tau}(\overline{t})$ holds if there is a substitution ρ with $\rho(\tau) = D(\overline{t})$ such that $\rho(z) = c_W$ and $(r,j) \notin W$ (it holds also for some other terms but it is irrelevant for the proof). In short, $\mathrm{NA}_{r(z),j,\tau}(\overline{t})$ says that (r,j) is not assumed for z in $D(\overline{t})$.

For every indexed predicate $\overline{p} = (p, clock, prio)$ we will have in $\mathcal{P}'$ the following two rules with $\overline{p}$ in the head

$$\overline{p}(z) \leftarrow in_{(p,clock)}(z)$$
$$\overline{p}(z) \leftarrow \{\mathrm{OK}^{\overline{p}}_{q(x_i)}(z) : q \in \mathsf{Pred}, x_i \in RVar\}, \mathrm{OK}^{\overline{p}}_{Exp}(z), \{\mathrm{OK}^{\overline{p}}_{r(x),j}(z) : r \in \mathsf{Pred}, j \in \mathrm{Rg}(\Omega)\}, \{\mathrm{OK}^{\overline{p}}_{r(y_i),j}(z) : r \in \mathsf{Pred}, j \in \mathrm{Rg}(\Omega), y_i \in EVar\}$$

The predicate $in_{(p,clock)}(t)$ holds if t is a list $cons(c_{w_1}, cons(\dots cons(c_{w_i}, nil)\dots))$ and $(p, clock) \in \{w_1,\dots,w_n\}$. So the first rule says that a term is accepted if what is to be proved about it is assumed in it. The second rule says that $\overline{p}(\overline{t})$ holds if $\overline{t}$ passes all the $\mathrm{OK}^{\overline{p}}$ tests. Predicate $OK^{\overline{p}}(\overline{t})$ with different subscripts decides whether or not to initiate checking of the predicate in the subscript (as we describe below).

The rules for $\mathrm{OK}^{\overline{p}}_{q(x_i)}$ for each $q \in \mathsf{Pred}$ and $x_i \in RVar$ are.

$$\mathrm{OK}^{\overline{p}}_{q(x_i)}(\Pi(\dots, Exp_{q(\tau)}(\dots),\dots)) \leftarrow \qquad \text{for arbitrary } q(\tau)$$
$$\mathrm{OK}^{\overline{p}}_{q(x_i)}(\Pi(\dots, Red_{q_1,\dots,q_n},\dots)) \leftarrow \qquad \text{if } q \neq q_i$$
$$\mathrm{OK}^{\overline{p}}_{q(x_i)}(\Pi(f(z_1,\dots,z_n),\dots)) \leftarrow (q, \max(clock, \Omega(q)), \Omega(q))(z_i)$$
$$f \text{ a symbol from the original signature}$$

When we write $\Pi(\dots, Exp_{q(\tau)}(\dots),\dots)$ above we mean that $Exp_{q(\tau)}(\dots)$ stands on the position corresponding to $\overline{p}$. We will use the same convention below. We also make an extensive use of ellipsises to show which variables are irrelevant.

The intention is that a predicate $\mathrm{OK}^{\overline{p}}_{q(x_i)}(\overline{t})$ holds if the rule on the position corresponding to $\overline{p}$ is not a reduction rule or it is a reduction rule that does not assume q at the position i. Otherwise $\overline{t}$ must be of the form $\Pi(f(\overline{t}_1,\dots,\overline{t}_n),\dots)$ and $\mathrm{OK}^{\overline{p}}_{q(x_i)}(\overline{t})$ requires $\overline{q}(\overline{t}_i)$ to hold for an appropriate $\overline{q}$.

The rules for $\mathrm{OK}^{\overline{p}}_{Exp}$ are.

$$\mathrm{OK}^{\overline{p}}_{Exp}(\Pi(\dots, Red_{\vec{q}},\dots)) \leftarrow \qquad \text{for arbitrary } \vec{q}$$
$$\mathrm{OK}^{\overline{p}}_{Exp}(\Pi(\dots, Exp_{q(\tau)}(z,\dots),\dots)) \leftarrow (q, \Omega(q), \Omega(q))(z)$$
$$\text{for arbitrary } q(\tau)$$

The predicate $\mathrm{OK}^{\overline{p}}_{Exp}(\overline{t})$ holds if a reduction rule is suggested in $\overline{t}$ on the position corresponding to $\overline{p}$. Otherwise there is an expansion rule at this position and the predicate requires the proof of the main assumption of this rule.

The rules for $\mathrm{OK}^{\overline{p}}_{r(x),j}$ for every $r \in \mathsf{Pred}$ and $j \in \mathrm{Rg}(\Omega)$ are.

$$\mathrm{OK}^{\overline{p}}_{r(x),j}(\Pi(\dots, Red_{\vec{q}}, \dots)) \leftarrow \qquad \text{for arbitrary } \vec{q}$$

$$\mathrm{OK}^{\overline{p}}_{r(x),j}(\Pi(\dots, Exp_{q(\tau)}(z, \dots), \dots)) \leftarrow \mathrm{NA}_{r(x),j,\tau}(z)$$
$$\text{for arbitrary } q(\vec{\tau})$$

$$\mathrm{OK}^{\overline{p}}_{r(x),j}(z) \leftarrow (r, \max(clock, j), j)(z)$$

The predicate $\mathrm{OK}^{\overline{p}}_{r(x),j}(\overline{t})$ checks whether it is suggested in $\overline{t}$ to use an expansion rule and in this use (r, j) is assumed about $\overline{t}$. If it is so then it requires $\overline{r}(\overline{t})$ to hold for appropriate $\overline{r}$. Otherwise it accepts.

The rules for $\mathrm{OK}^{\overline{p}}_{r(y_i),j}$ for every $r \in \mathsf{Pred}$, $y_i \in EVar$ and $j \in \mathrm{Rg}(\Omega)$ are.

$$\mathrm{OK}^{\overline{p}}_{r(y_i),j}(\Pi(\dots, Red_{\vec{q}}, \dots)) \leftarrow \qquad \text{for arbitrary } \vec{q}$$

$$\mathrm{OK}^{\overline{p}}_{r(y_i),j}(\Pi(\dots, Exp_{q(\tau)}(z, \dots), \dots)) \leftarrow \mathrm{NA}_{r(y_i),j,\tau}(z)$$
$$\text{for arbitrary } q(\tau)$$

$$\mathrm{OK}^{\overline{p}}_{r(y_i),j}(\Pi(\dots, Exp_{q(\tau)}(z, z_1, \dots, z_n) \dots) \leftarrow$$
$$(r, \max(clock, j), j)(z_i) \qquad \text{for arbitrary } q(\tau)$$

The predicate $\mathrm{OK}^{\overline{p}}_{r(y_i),j}(\overline{t})$ checks whether it is suggested in $\overline{t}$ to use an expansion rule and in this use (r, j) is assumed about a term being substituted for y_i. If it is so then it requires $\overline{r}(\overline{t}_i)$ to hold for appropriate $\overline{r}$ and $\overline{t}_i$. Otherwise it accepts.

Finally we need to define the priority of each predicate in our program $\mathcal{P}'$. We let $\Omega'(p, clock, prio) = prio$ and $\Omega'(q) = 0$ for every other predicate.

To see how the program $\mathcal{P}'$ works, let us try to construct a derivation of $(p, clock, prio)(A(t))$ for some predicate $p \in \mathsf{Pred}$ and some ground term t such that $\mathcal{S}$ is defined for the position $\langle p(t), clock, prio\rangle$. In what follows we denote $(p, clock, prio)$ by $\overline{p}$.

First, suppose that

$$\mathcal{S}(\langle p(t), clock, prio\rangle) =$$
$$\langle p(f(x_1, \dots, x_n)) \leftarrow q_1(x_1), \dots, q_n(x_n), clock, \sigma, \emptyset\rangle$$

In this case the term $A(t)$ has the form $\Pi(f(A(t_1), \dots, A(t_n)), h_1, \dots, h_k)$ with the term $h_{\overline{p}}$ on the position corresponding to $\overline{p}$ being $Red_{q_1,\dots,q_n}$. The predicate $\mathrm{OK}^{\overline{p}}_{Exp}(A(t))$ is true because h_p is a Red constant. For the same reasons all $\mathrm{OK}^{\overline{p}}_{r(x),j}(A(t))$ and $\mathrm{OK}^{\overline{p}}_{r(y_i),j}(A(t))$ predicates are true. Also a predicate $\mathrm{OK}^{\overline{p}}_{q(x_i)}(A(t))$ holds if $q_i \neq q$. Finally each predicate $\mathrm{OK}^{\overline{p}}_{q_i(x_i)}(A(t))$ requires the predicate $(q, \max(\Omega(q), clock), \Omega(q))(A(t_i))$ to hold. Hence in this case the property (3) is satisfied.

The other case is when

$$\mathcal{S}(\langle p(t), clock, prio\rangle) = \langle p(x) \leftarrow q(\tau), clock, \sigma, \rho\rangle$$

Here we have that $A(t)$ has the form $\Pi(s, h_1, \dots, h_k)$ with the term $h_{\overline{p}}$ being $Exp_{q(\tau)}(A(\rho(\tau)), A(\sigma(y_1)), \dots, A(\sigma(y_n)))$. Predicates $\mathrm{OK}^{\overline{p}}_{q(x_i)}(A(t))$ hold because $h_{\overline{p}}$ starts with Exp constant. The predicate $\mathrm{OK}^{\overline{p}}_{Exp}(A(t))$ requires the proof of $(q, \Omega(q), \Omega(q))(A(t))$. A predicate $\mathrm{OK}^{\overline{p}}_{r(x),j}(A(t))$ is true if $(r, j) \notin W$ for W the subscript in $\rho(x) = c_W$; otherwise $\mathrm{OK}^{\overline{p}}_{r(x),j}(A(t))$ requires $(r, \max(clock, j), j)(A(t))$ to be proved. Similarly, a predicate $\mathrm{OK}^{\overline{p}}_{r(y_i),j}(A(t))$ is true if $(r, j) \notin W$ for W the subscript in $\rho(y_i) = c_W$; otherwise $\mathrm{OK}^{\overline{p}}_{r(y_i),j}(A(t))$ requires $(r, \max(clock, j), j)(\sigma(y_i))$.

The above considerations show that the property (3) holds. This property allows us to construct accepting derivations as stated in the next lemma.

Lemma 8 If Prover has a winning strategy from $\langle p(t), 0, 0\rangle$ in $\mathcal{FG}$ then $(p, 0, 0)(A(t)) \in [\![\mathcal{P}']\!]$. $\square$

Now let us consider the converse of this lemma. We call a derivation *minimal* if it always chooses a rule with empty body whenever it is possible and always chooses to prove $NA_{r(z),j,\tau}(\overline{t})$ whenever the predicate holds. It is clear that if there is an accepting derivation of an atom then there is a minimal accepting derivation of this atom. It is also easy to check that this derivation is unique.

A simple examination of the rules of $\mathcal{P}'$ shows a property analogous to the property (3).

Lemma 9 Suppose $\overline{p}(\overline{t}) \in [\![\mathcal{P}']\!]$ for some triple $\overline{p} = (p, clock, prio)$ and some well formed annotated term $\overline{t}$. Take the minimal accepting derivation of $\overline{p}(\overline{t})$. Each path of this derivation goes through bureaucratic nodes and ends in an axiom or a real node. Let $\overline{p}_1(\overline{t}_1), \dots, \overline{p}_n(\overline{t}_n)$ be all the real nodes such that there is no real node between them and the root. We have:

> the node $\langle p(D(\overline{t})), clock\ prio\rangle$ in the game $\mathcal{FG}$ has a successor v whose only successors are $\langle p_i(D(\overline{t}_i)), clock, prio\rangle$ for $i = 1, \dots, n$. $\square$

Suppose $\overline{p}_0(\overline{t}_0) \in [\![\mathcal{P}']\!]$. The above lemma allows us to construct a strategy tree for Prover from the position $\langle p_0(D(t_0)), clock_0, prio_0\rangle$. This strategy is winning because every infinite path of the strategy tree corresponds to a path of the minimal accepting derivation. We get

Lemma 10 If $\overline{p}(\overline{t}) \in [\![\mathcal{P}']\!]$ for $\overline{p} = (p, clock, prio)$ and $\overline{t}$ a well formed annotated term then Prover has a winning strategy from $\langle p(D(t)), clock, prio\rangle$. $\square$

Finally, let us add to $\mathcal{P}'$ rules of the form

$$p(z) \leftarrow (p,0,0)(z), \mathrm{WF}(z)$$

for every predicate $p \in \mathsf{Pred}$. The predicate $\mathrm{WF}(\bar{t})$ holds if t is a well formed annotated term. As we have already mentioned, it is easy to write a polynomial size alternating automaton for $\mathrm{WF}(z)$. Now we are ready to prove the main result.

Theorem 11
For every predicate $p \in \mathsf{Pred}$: $[\![\mathcal{P}, p]\!]$ is nonempty iff $[\![\mathcal{P}', p]\!]$ is nonempty

Proof
If $p(t) \in [\![\mathcal{P}]\!]$ then by Lemma 5 there is a winning strategy in the game $\mathcal{FG}$ from the position $\langle p(t), 0, 0\rangle$. From the general theory of games [24, 26] it follows that there is a positional winning strategy defined for this position. By Lemma 8 $(p,0,0)(A(t)) \in [\![\mathcal{P}']\!]$. As $A(t)$ is a well formed annotated term we have $p(A(t)) \in [\![\mathcal{P}']\!]$.

If $p(\bar{t}) \in [\![\mathcal{P}']\!]$ then $\bar{t}$ is well formed and $(p,0,0)(\bar{t}) \in [\![\mathcal{P}']\!]$. By Lemma 10 there is a winning strategy for Prover from the position $\langle p(D(\bar{t}), 0, 0\rangle$ in $\mathcal{FG}$. So there is no winning strategy for Opponent from $\langle p(D(\bar{t})), 0, 0\rangle$. By Lemma 6 there is no winning strategy for Opponent from $p(D(\bar{t})$ in $\mathcal{G}$. By the determinacy of the game $\mathcal{G}$, there is a strategy for Prover from this position. By Proposition 2, $p(D(\bar{t})) \in [\![\mathcal{P}]\!]$. □

References

[1] O. Bernholtz, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In *Computer Aided Verification, Proc. 6th Int. Workshop*, volume 818 of *LNCS*, pages 142–155, Stanford, California, June 1994. Springer-Verlag. Full version available from authors.

[2] A. Bouajjani, J. Esparza, and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model Checking. In *CONCUR'97*. LNCS 1243, 1997.

[3] J. R. Büchi. Regular canonical systems. *Archiv Mathematische Logik und Grundlagenforschung*, 6:91–111, 1964. Reprint in Saunders Mac Lane, Dirk Siefkes, editors, The collected works of J. Richard Büchi, Springer-Verlag, 1990.

[4] O. Burkart and B. Steffen. Composition, Decomposition and Model-Checking of Pushdown Processes. *Nordic Journal of Computing*, 2, 1995.

[5] W. Charatonik and A. Podelski. Co-definite set constraints. In T. Nipkow, editor, *Proceedings of the 9th International Conference on Rewriting Techniques and Applications*, volume 1379 of *LNCS*, pages 211–225. Springer-Verlag, 1998.

[6] W. Charatonik and A. Podelski. Set-based analysis of reactive infinite-state systems. In B. Steffen, editor, *Proceedings of the First International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1384 of *LNCS*, pages 358–375. Springer-Verlag, 1998.

[7] P. Devienne, J.-M. Talbot, and S. Tison. Solving classes of set constraints with tree automata. In G. Smolka, editor, *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming - CP97*, volume 1330 of *LNCS*, pages 68–83. Springer-Verlag, October 1997.

[8] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proc. FOCS 91*, 1991.

[9] A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. Electronic Notes in Theoretical Computer Science 9, www.elsevier.nl/locate/entcs, 13 pages, 1997.

[10] T. Frühwirth, E. Shapiro, M. Vardi, and E. Yardeni. Logic programs as types for logic programs. In *Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 300–309, July 1991.

[11] D. Harel and D. Raz. Deciding emptiness for stack automata on infinite trees. *Information and Computation (formerly Information and Control)*, 113:278–299, 1994.

[12] N. Heintze and J. Jaffar. A finite presentation theorem for approximating logic programs. In *Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 197–209, January 1990.

[13] Y. Hirschfeld, M. Jerrum, and F. Moller. A polynomial-time algorithm for deciding equivalence of normed context-free processes. In *35th Annual Symposium on Foundations of Computer Science*, pages 623–631. IEEE Computer Society Press, November 1994.

[14] H. Hüttel and C. Stirling. Actions speak louder than words: Proving bisimilarity for context-free processes. In *Proceedings, 6th Annual Symposium on Logic in Computer Science*, pages 376–386. IEEE Computer Society Press, July 1991.

[15] N. D. Jones and S. S. Muchnick. Flow analysis and optimization of lisp-like structures. In *Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 244–256, January 1979.

[16] R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65:149–184, 1993.

[17] A. W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In A. Skowron, editor, *Fifth Symposium on Computation Theory*, volume 208 of *LNCS*, pages 157–168, 1984.

[18] D. E. Muller and P. E. Schup. The theory of ends, pushdown automata, and second-order logic. *Theoretical Comput. Sci.*, 37, 1985.

[19] D. E. Muller and P. E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Comput. Sci.*, 141:69–107, 1995.

[20] D. Niwiński. Fixed points vs. infinite generation. In *LICS '88*, pages 402–409, 1988.

[21] D. Niwiński. Fixed points characterization of infinite behavior of finite state systems. *Theoret. Comput. Sci.*, 189:1–69, 1997.

[22] W. Peng and S. Purushothaman. Empty stack pushdown omega-tree automata. In J.-C. Raoult, editor, *Colloquium on Trees in Algebra and Programming*, volume 581 of *LNCS*, pages 248–264. Springer-Verlag, February 1992.

[23] J. C. Reynolds. Automatic computation of data set definitions. *Information Processing*, 68:456–461, 1969.

[24] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3. 1997.

[25] I. Walukiewicz. Pushdown processes: Games and model checking. In *CAV'96*, volume 1102 of *LNCS*, pages 62–74, 1996.

[26] W. Zielonka. Infinite games on fintely coloured graphs with applications to automata on infinte trees. To appear in TCS, 1997.