ISTITUTO DI ANALISI DEI SISTEMI ED INFORMATICA
"Antonio Ruberti"
CONSIGLIO NAZIONALE DELLE RICERCHE

A. Pettorossi, M. Proietti, V. Senni

TRANSFORMATIONS OF LOGIC PROGRAMS ON INFINITE LISTS

R. 10-04, 2010 (Revised August 2010)

Alberto Pettorossi – Dipartimento di Informatica, Sistemi e Produzione, Università di Roma Tor Vergata, Via del Politecnico 1, I-00133 Roma, Italy, and Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30, I-00185 Roma, Italy.
Email : pettorossi@info.uniroma2.it. URL : http://www.iasi.cnr.it/~adp.

Maurizio Proietti – Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30, I-00185 Roma, Italy. Email : maurizio.proietti@iasi.cnr.it.
URL : http://www.iasi.cnr.it/~proietti.

Valerio Senni – Dipartimento di Informatica, Sistemi e Produzione, Università di Roma Tor Vergata, Via del Politecnico 1, I-00133 Roma, Italy.
Email : senni@info.uniroma2.it. URL : http://www.disp.uniroma2.it/users/senni.

ISSN: 1128–3378

Collana dei Rapporti dell'Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti", CNR

viale Manzoni 30, 00185 ROMA, Italy

tel. ++39-06-77161
fax ++39-06-7716461
email: iasi@iasi.cnr.it
URL: http://www.iasi.cnr.it

Abstract

We consider an extension of logic programs, called *ω-programs*, that can be used to define predicates over *infinite lists*. ω-programs allow us to specify properties of the infinite behavior of reactive systems and, in general, properties of infinite sequences of events. The semantics of ω-programs is an extension of the perfect model semantics. We present variants of the familiar unfold/fold rules which can be used for transforming ω-programs. We show that these new rules are correct, that is, their application preserves the perfect model semantics. Then we outline a general methodology based on program transformation for verifying properties of ω-programs. We demonstrate the power of our transformation-based verification methodology by proving some properties of Büchi automata and ω-regular languages.

KEYWORDS: Program Transformation, Program Verification, Infinite Lists.

1. Introduction

The problem of specifying and verifying properties of *reactive systems*, such as protocols and concurrent systems, has received much attention over the past fifty years or so. The main peculiarity of reactive systems is that they perform nonterminating computations and, in order to specify and verify the properties of these computations, various formalisms dealing with infinite sequences of events have been proposed. Among these we would like to mention: (i) Büchi automata and other classes of finite automata on infinite sequences [27], (ii) ω-languages [25], and (iii) various temporal and modal logics (see [4] for a brief overview of these logics).

Also logic programming has been proposed as a formalism for specifying computations over infinite structures, such as infinite lists or infinite trees (see, for instance, [5, 13, 14, 24]). One advantage of using logic programming languages is that they are general purpose languages and, together with a model-theoretic semantics, they also have an operational semantics. Thus, logic programs over infinite structures can be used for *specifying* infinite computations and, in fact, providing executable specifications for them. However, very few techniques which use logic programs over infinite structures, have been proposed in the literature for *verifying* properties of infinite computations. We are aware only of a recent work presented in [10], which is based on coinductive logic programming, that is, a logic programming language whose semantics is based on greatest models.

In this paper our aim is to develop a methodology based on the familiar unfold/fold transformation rules [3, 26] for reasoning about infinite structures and verifying properties of programs over such structures. In order to do so, we do not introduce a new programming language, but we consider a simple extension of logic programming on finite terms by introducing the class of the so-called *ω-programs*, which are logic programs on infinite lists. Similarly to the case of logic programs, for the class of *locally stratified* ω-programs we define the *perfect model* semantics (see [2] for a survey on negation in logic programming).

We extend to ω-programs the transformation rules for locally stratified programs presented in [8, 16, 21, 22, 23] and, in particular: (i) we introduce an *instantiation* rule which is specific for programs on infinite lists, (ii) we weaken the applicability conditions for the *negative unfolding* rule, and (iii) we consider a more powerful *negative folding* rule (see Sections 3 and 4 for more details). We prove that these rules preserve the perfect model semantics of ω-programs.

Then we extend to ω-programs the transformation-based methodology for verifying properties of programs presented in [16]. We demonstrate the power of our verification methodology through some examples. In particular, we prove: (i) the non-emptiness of the language recognized by a Büchi automaton, and (ii) the containment between languages denoted by ω-regular expressions.

The paper is structured as follows. In Section 2 we introduce the class of ω-programs and we define the perfect model semantics for locally stratified ω-programs. In Section 3 we present the transformation rules and in Section 4 we prove that they preserve the semantics of ω-programs. In Section 5 we present the transformation-based verification method and we see it in action in some examples. Finally, in Section 6 we discuss related work in the area of program transformation and program verification.

2. Programs on Infinite Lists

Let us consider a first order language $\mathcal{L}_\omega$ given by a set *Var* of variables, a set *Fun* of function symbols, and a set *Pred* of predicate symbols. We assume that *Fun* includes: (i) a *finite*, *non-*

empty set Σ of constants, (ii) the constructor $[\![\_|\_]\!]$ of the infinite lists of elements of Σ, and (iii) at least one constant not in Σ. Thus, $[\![s|t]\!]$ is an infinite list whose head is $s \in \Sigma$ and whose tail is the infinite list t. Let Σ^ω denote the set of the infinite lists of elements of Σ.

We assume that $\mathcal{L}_\omega$ is a typed language [13] with three basic types: (i) `fterm`, which is the type of the finite terms, (ii) `elem`, which is the type of the constants in Σ, and (iii) `ilist`, which is the type of the infinite lists of Σ^ω. Every function symbol in $Fun - (\Sigma \cup \{[\![\_|\_]\!]\})$, with arity $n\,(\geq 0)$, has type $(\texttt{fterm} \times \cdots \times \texttt{fterm}) \rightarrow \texttt{fterm}$, where `fterm` occurs n times to the left of $\rightarrow$. The function symbol $[\![\_|\_]\!]$ has type $(\texttt{elem} \times \texttt{ilist}) \rightarrow \texttt{ilist}$. A predicate symbol of arity $n\,(\geq 0)$ in *Pred* has type of the form $\tau_1 \times \cdots \times \tau_n$, where $\tau_1, \ldots, \tau_n \in \{\texttt{fterm}, \texttt{elem}, \texttt{ilist}\}$. For every term (or formula) t, we denote by $vars(t)$ the set of variables occurring in t.

An *ω-clause* γ is a formula of the form $A \leftarrow L_1 \wedge \ldots \wedge L_m$, with $m \geq 0$, where A is an atom and $L_1, \ldots, L_m$ are (positive or negative) literals, constructed as usual from symbols in the typed language $\mathcal{L}_\omega$, with the following extra condition: every predicate in γ has, among its arguments, *at most one* argument of type `ilist`. This condition makes it easier to prove the correctness of the positive and negative unfolding rules (see Section 3 for further details). We denote by *true* the empty conjunction of literals, and we denote by $hd(\gamma)$ and $bd(\gamma)$ the head and the body, respectively, of a clause γ. An *ω-program* is a set of ω-clauses.

Let *HU* be the Herbrand universe constructed from the set $Fun - (\Sigma \cup \{[\![\_|\_]\!]\})$ of function symbols. An interpretation for our typed language $\mathcal{L}_\omega$, called an ω-interpretation, is a function I such that: (i) I assigns to the types `fterm`, `elem`, and `ilist`, respectively, the sets *HU*, Σ, and Σ^ω (which by our assumptions are non-empty), (ii) I assigns to the function symbol $[\![\_|\_]\!]$, the function $[\![\_|\_]\!]_I$ such that, for any element $s \in \Sigma$, for any infinite list $t \in \Sigma^\omega$, $[\![s|t]\!]_I$ is the infinite list $[\![s|t]\!]$, (iii) I is an Herbrand interpretation for all function symbols in $Fun - (\Sigma \cup \{[\![\_|\_]\!]\})$, and (iv) I assigns to every n-ary predicate $p \in Pred$ of type $\tau_1 \times \ldots \times \tau_n$, a relation on $D_1 \times \cdots \times D_n$, where, for $i = 1, \ldots, n$, D_i is either *HU* or Σ or Σ^ω, if τ_i is either `fterm` or `elem` or `ilist`, respectively. We say that an ω-interpretation I is an *ω-model* of an ω-program P if for every clause $\gamma \in P$ we have that $I \models \forall X_1 \ldots \forall X_k\, \gamma$, where $vars(\gamma) = \{X_1, \ldots, X_k\}$.

A *valuation* is a function $v\colon Var \rightarrow HU \cup \Sigma \cup \Sigma^\omega$ such that: (i) if X has type `fterm` then $v(X) \in HU$, (ii) if X has type `elem` then $v(X) \in \Sigma$, and (iii) if X has type `ilist` then $v(X) \in \Sigma^\omega$. The valuation function v is extended to any term t, or literal L, or conjunction B of literals, or clause γ, by making the function v act on the variables occurring in t, or L, or B, or γ. (Obviously, $v(t) = t$ if $vars(t) = \emptyset$.)

We extend the notion of *Herbrand base* [13] to ω-programs by defining it to be the set $\mathcal{B}_\omega = \{p(v(X_1), \ldots, v(X_n)) \mid p$ is an n-ary predicate symbol in *Pred* and v is a valuation$\}$. Thus, any ω-interpretation can be identified with a subset of $\mathcal{B}_\omega$.

A *local stratification* is a function $\sigma\colon \mathcal{B}_\omega \rightarrow W$, where W is the set of countable ordinals. Given $A \in \mathcal{B}_\omega$, we define $\sigma(\neg A) = \sigma(A) + 1$. Given an ω-clause γ of the form $H \leftarrow L_1 \wedge \ldots \wedge L_m$ and a local stratification σ, we say that γ is *locally stratified* w.r.t. σ if, for $i = 1, \ldots, m$, for every valuation v, $\sigma(v(H)) \geq \sigma(v(L_i))$. An ω-program P is *locally stratified w.r.t. σ*, or σ is a *local stratification for P*, if every clause in P is locally stratified w.r.t. σ. An ω-program P is *locally stratified* if there exists a local stratification σ such that P is *locally stratified w.r.t. σ*.

A *level mapping* is a function $\ell : Pred \rightarrow \mathbb{N}$. A level mapping is extended to literals as follows: for any literal L having predicate p, if L is a positive literal, then $\ell(L) = \ell(p)$ and, if L is a negative literal then $\ell(L) = \ell(p) + 1$. An ω-clause γ of the form $H \leftarrow L_1 \wedge \ldots \wedge L_m$ is *stratified* w.r.t. ℓ if, for $i = 1, \ldots, m$, $\ell(H) \geq \ell(L_i)$. An ω-program P is *stratified* if there exists

a level mapping ℓ such that all clauses of P are stratified w.r.t. ℓ [13]. Clearly, every stratified ω-program is a locally stratified ω-program.

Similarly to the case of logic programs on finite terms, for every locally stratified ω-program P, we can construct a unique *perfect ω-model* (or *perfect model*, for short) denoted by $M(P)$ (see [2] for the case of logic programs on finite terms). Now we present an example of this construction.

Example 1. Let: (i) $\Sigma = \{a, b\}$ be the set of constants of type `elem`, (ii) H and S be variables of type `elem`, and (iii) X be a variable of type `ilist`. Let *inf_often_b* be a predicate of type `ilist`, and *member* and *last_occ* be predicates of type `elem` $\times$ `ilist`. Let us consider the following ω-program P:

1. $\mathit{inf\_often\_b}(X) \leftarrow \mathit{member}(b, X) \wedge \neg\, \mathit{last\_occ}(b, X)$
2. $\mathit{last\_occ}(S, [\![S|X]\!]) \leftarrow \neg\, \mathit{member}(S, X)$
3. $\mathit{last\_occ}(S, [\![H|X]\!]) \leftarrow \mathit{last\_occ}(S, X)$
4. $\mathit{member}(S, [\![S|X]\!]) \leftarrow$
5. $\mathit{member}(S, [\![H|X]\!]) \leftarrow \mathit{member}(S, X)$

We have that: (i) $\mathit{last\_occ}(s, w)$ holds iff there is a last, rightmost occurrence of s in w, and (ii) $\mathit{inf\_often\_b}(w)$ holds iff b occurs infinitely often in w.

Program P is stratified w.r.t. the level mapping ℓ such that $\ell(\mathit{member}) = 0$, $\ell(\mathit{last\_occ}) = 1$, and $\ell(\mathit{inf\_often\_b}) = 2$. We construct the perfect model $M(P)$ by starting from the ground atoms of level 0 (i.e., those with predicate *member*) and going up to the ground atoms of level 2 (i.e., those with predicate *inf_often_b*). For level 0 we have that, for all $w \in \{a, b\}^\omega$, $\mathit{member}(b, w) \notin M(P)$ iff $w \in a^\omega$. Then, we consider the ground atoms of level 1 (i.e., those with predicate *last_occ*). For all $w \in \{a, b\}^\omega$, $\mathit{last\_occ}(b, w) \in M(P)$ iff $w \in (a+b)^* b a^\omega$. Thus, $\mathit{last\_occ}(b, w) \notin M(P)$ iff $w \in a^\omega + (a^* b)^\omega$. Finally, we consider the ground atoms of level 2 (i.e., those with predicate *inf_often_b*). For all $w \in \{a, b\}^\omega$, $\mathit{inf\_often\_b}(w) \in M(P)$ iff (see clause 1) $\mathit{member}(b, w) \in M(P)$ and $\mathit{last\_occ}(b, w) \notin M(P)$, that is, $w \in (a^* b)^\omega$.

3. Transformation Rules

Given an ω-program P_0, a *transformation sequence* is a sequence $P_0, \ldots, P_n$, with $n \geq 0$, of ω-programs constructed as follows. Suppose that we have constructed a sequence $P_0, \ldots, P_k$, for $0 \leq k \leq n-1$. Then, the next program P_{k+1} in the sequence is derived from program P_k by applying one of the following transformation rules R1–R7.

First we have the *definition introduction* rule which allows us to introduce a new predicate definition.

R1. Definition Introduction. Let us consider m (≥ 1) clauses of the form:

$\delta_1 : \mathit{newp}(X_1, \ldots, X_d) \leftarrow B_1, \quad \ldots, \quad \delta_m : \mathit{newp}(X_1, \ldots, X_d) \leftarrow B_m$

where: (i) *newp* is a predicate symbol not occurring in $\{P_0, \ldots, P_k\}$, (ii) $X_1, \ldots, X_d$ are distinct variables occurring in $\{B_1, \ldots, B_m\}$, (iii) none of the B_i's is the empty conjunction of literals, and (iv) every predicate symbol occurring in $\{B_1, \ldots, B_m\}$ also occurs in P_0. The set $\{\delta_1, \ldots, \delta_m\}$ of clauses is said to be the *definition* of *newp*.

By *definition introduction* from program P_k we derive the new program $P_{k+1} = P_k \cup \{\delta_1, \ldots, \delta_m\}$. For $n \geq 0$, Defs_n denotes the set of clauses introduced by the definition rule during the transformation sequence $P_0, \ldots, P_n$. In particular, $\mathit{Defs}_0 = \{\}$.

In the following *instantiation* rule we assume that the set of constants of type `elem` in the language $\mathcal{L}_\omega$ is the finite set $\Sigma = \{s_1, \ldots, s_h\}$.

R2. Instantiation. Let γ: $H \leftarrow B$ be a clause in program P_k and X be a variable of type `ilist` occurring in γ. By *instantiation* of X in γ, we get the clauses:

γ_1: $(H \leftarrow B)\{X/[\![s_1|X]\!]\}$, $\ldots$, γ_h: $(H \leftarrow B)\{X/[\![s_h|X]\!]\}$

and we say that clauses $\gamma_1, \ldots, \gamma_h$ are *derived from* γ. From P_k we derive the new program $P_{k+1} = (P_k - \{\gamma\}) \cup \{\gamma_1, \ldots, \gamma_h\}$.

The *unfolding* rule consists in replacing an atom A occurring in the body of a clause by its definition in P_k. We present two unfolding rules: (1) the *positive unfolding*, and (2) the *negative unfolding*. They correspond, respectively, to the case where A or $\neg A$ occurs in the body of the clause to be unfolded.

R3. Positive Unfolding. Let $\gamma:\ H \leftarrow B_L \wedge A \wedge B_R$ be a clause in program P_k and let P'_k be a variant of P_k without variables in common with γ. Let

$\gamma_1:\ K_1 \leftarrow B_1,\ \ldots,\ \gamma_m:\ K_m \leftarrow B_m \quad (m \geq 0)$

be all clauses of program P'_k such that, for $i = 1, \ldots, m$, A is unifiable with K_i, with most general unifier ϑ_i.
By *unfolding* γ *w.r.t.* A we get the clauses $\eta_1, \ldots, \eta_m$, where for $i = 1, \ldots, m$, η_i is $(H \leftarrow B_L \wedge B_i \wedge B_R)\vartheta_i$, and we say that clauses $\eta_1, \ldots, \eta_m$ are *derived from* γ. For $i = 1, \ldots, m$, we say that clause η_i is derived by *unfolding* γ *w.r.t.* A *using* clause γ_i. From P_k we derive the new program $P_{k+1} = (P_k - \{\gamma\}) \cup \{\eta_1, \ldots, \eta_m\}$.

In rule R3, and also in the following rule R4, the most general unifier can be computed by using a unification algorithm for finite terms (see, for instance, [13]). Note that this is correct, even in the presence on infinite terms, because in any ω-program each predicate has at most one argument of type `ilist`. On the contrary, if predicates may have more than one argument of type `ilist`, in the unfolding rule it is necessary to use a unification algorithm for infinite structures [5]. For reasons of simplicity, here we do not make that extension of the unfolding rule and we stick to our assumption that every predicate has at most one argument of type `ilist`.

The *existential variables* of a clause γ are the variables occurring in the body of γ and not in its head.

R4. Negative Unfolding. Let γ: $H \leftarrow B_L \wedge \neg A \wedge B_R$ be a clause in program P_k and let P'_k be a variant of P_k without variables in common with γ. Let

$\gamma_1:\ K_1 \leftarrow B_1,\ \ldots,\ \gamma_m:\ K_m \leftarrow B_m \quad (m \geq 0)$

be all clauses of program P'_k, such that, for $i = 1, \ldots, m$, A is unifiable with K_i, with most general unifier ϑ_i. Assume that: (1) $A = K_1\vartheta_1 = \cdots = K_m\vartheta_m$, that is, for $i = 1, \ldots, m$, A is an instance of K_i, (2) for $i = 1, \ldots, m$, γ_i has no existential variables, and (3) from $\neg(B_1\vartheta_1 \vee \ldots \vee B_m\vartheta_m)$ we get a logically equivalent disjunction $D_1 \vee \ldots \vee D_r$ of conjunctions of literals, with $r \geq 0$, by first pushing $\neg$ inside and then pushing $\vee$ outside.
By *unfolding* γ *w.r.t.* $\neg A$ *using* P_k we get the clauses $\eta_1, \ldots, \eta_r$, where, for $i = 1, \ldots, r$, clause η_i is $H \leftarrow B_L \wedge D_i \wedge B_R$, and we say that clauses $\eta_1, \ldots, \eta_r$ are *derived from* γ. From P_k we derive the new program $P_{k+1} = (P_k - \{\gamma\}) \cup \{\eta_1, \ldots, \eta_r\}$.

The following *subsumption* rule allows us to remove from P_k a clause γ such that $M(P_k) = M(P_k - \{\gamma\})$.

R5. Subsumption. Let γ_1: $H \leftarrow$ be a clause in program P_k and let γ_2 in $P_k - \{\gamma_1\}$ be a variant of $(H \leftarrow B)\vartheta$, for some conjunction of literals B and substitution ϑ. Then, we say that γ_2 *is subsumed* by γ_1 and by *subsumption*, from P_k we derive the new program $P_{k+1} = P_k - \{\gamma_2\}$.

The *folding* rule consists in replacing instances of the bodies of the clauses that define an atom A by the corresponding instance of A. Similarly to the case of the unfolding rule, we have two folding rules: (1) *positive folding* and (2) *negative folding*. They correspond, respectively, to the case where folding is applied to positive or negative occurrences of literals.

R6. Positive Folding. Let γ be a clause in P_k and let $Defs'_k$ be a variant of $Defs_k$ without variables in common with γ. Let the definition of a predicate in $Defs'_k$ consist of the clause $\delta : K \leftarrow B$, where B is a non-empty conjunction of literals. Suppose that there exists a substitution ϑ such that clause γ is of the form $H \leftarrow B_L \wedge B\vartheta \wedge B_R$ and, for every variable $X \in vars(B) - vars(K)$, the following conditions hold: (i) $X\vartheta$ is a variable not occurring in $\{H, B_L, B_R\}$, and (ii) $X\vartheta$ does not occur in the term $Y\vartheta$, for any variable Y occurring in B and different from X.
By *folding* γ *using* δ we get the clause η: $H \leftarrow B_L \wedge K\vartheta \wedge B_R$, and we say that clause η is *derived from* γ. From P_k we derive the new program $P_{k+1} = (P_k - \{\gamma\}) \cup \{\eta\}$.

R7. Negative Folding. Let γ be a clause in P_k and let $Defs'_k$ be a variant of $Defs_k$ without variables in common with γ. Let the definition of a predicate in $Defs'_k$ consist of the q clauses $\delta_1 : K \leftarrow L_1, \ldots, \delta_q : K \leftarrow L_q$, with $q \geq 1$, such that, for $i = 1, \ldots, q$, L_i is a literal and δ_i has no existential variables. Suppose that there exists a substitution ϑ such that clause γ is of the form $H \leftarrow B_L \wedge (M_1 \wedge \ldots \wedge M_q)\vartheta \wedge B_R$, where, for $i = 1, \ldots, q$, if L_i is the negative literal $\neg A_i$ then M_i is A_i, and if L_i is the positive literal A_i then M_i is $\neg A_i$.
By *folding* γ *using* $\delta_1, \ldots, \delta_q$ we get the clause η: $H \leftarrow B_L \wedge \neg K\vartheta \wedge B_R$, and we say that clause η is *derived from* γ. From P_k we derive the program $P_{k+1} = (P_k - \{\gamma\}) \cup \{\eta\}$.

Note that the negative folding rule is not included in the sets of transformation rules presented in [21, 22, 23]. The negative folding rule presented in [8, 16] corresponds to our rule R7 in the case where $q = 1$.

4. Correctness of the Transformation Rules

Now let us introduce the notion of correctness of a transformation sequence w.r.t. the perfect model semantics.

Definition 4.1 (Correctness of a Transformation Sequence) *Let P_0 be a locally stratified ω-program and $P_0, \ldots, P_n$, with $n \geq 0$, be a transformation sequence. We say that $P_0, \ldots, P_n$ is* correct *if* (i) *$P_0 \cup Defs_n$ and P_n are locally stratified ω-programs and* (ii) *$M(P_0 \cup Defs_n) = M(P_n)$.*

In order to guarantee the correctness of a transformation sequence $P_0, \ldots, P_n$ (see Theorem 4.7 below) we will require that the application of the transformation rules satisfy some suitable conditions that refer to a given local stratification σ. In order to state those conditions we need the following definitions.

Definition 4.2 (σ-Maximal Atom) *Consider a clause γ: $H \leftarrow G$. An atom A in G is said to be* σ-maximal *if, for every valuation v and for every literal L in G, we have $\sigma(v(A)) \geq \sigma(v(L))$.*

Definition 4.3 (σ-Tight Clause) *A clause δ: $H \leftarrow G$ is said to be* σ-tight *if there exists a σ-maximal atom A in G such that, for every valuation v, $\sigma(v(H)) = \sigma(v(A))$.*

Definition 4.4 (Descendant Clause) *A clause η is said to be a* descendant *of a clause γ if either η is γ itself or there exists a clause δ such that η is derived from δ by using a rule in $\{\mathrm{R2}, \mathrm{R3}, \mathrm{R4}, \mathrm{R6}, \mathrm{R7}\}$, and δ is a descendant of γ.*

Definition 4.5 (Admissible Transformation Sequence) *Let P_0 be a locally stratified ω-program and let σ be a local stratification for P_0. A transformation sequence $P_0, \ldots, P_n$, with $n \geq 0$, is said to be* admissible *if:*
(1) *every clause in $Defs_n$ is locally stratified w.r.t. σ,*
(2) *for $k = 0, \ldots, n-1$, if P_{k+1} is derived from P_k by positive folding of clause γ using clause δ, then:* (2.1) *δ is σ-tight and* either (2.2.i) *the head predicate of γ occurs in P_0,* or (2.2.ii) *γ is a descendant of a clause β in P_j, with $0 < j \leq k$, such that β has been derived by positive unfolding of a clause α in P_{j-1} w.r.t. an atom which is σ-maximal in the body of α and whose predicate occurs in P_0, and*
(3) *for $k = 0, \ldots, n-1$, if P_{k+1} is derived from P_k by applying the negative folding rule thereby deriving a clause η, then η is locally stratified w.r.t. σ.*

Note that Condition (1) can always be fulfilled because the predicate introduced in program P_{k+1} by rule R1 does not occur in any of the programs $P_0, \ldots, P_k$. Conditions (2) and (3) cannot be checked in an algorithmic way for arbitrary programs and local stratification functions. In particular, the program property of being locally stratified is undecidable. However, there are significant classes of programs, such as the stratified programs, where these conditions are decidable and easy to verify.

The following Lemma 4.6 and Theorem 4.7, whose proofs can be found in the Appendix, show that: (i) when constructing an admissible transformation sequence $P_0, \ldots, P_n$, the application of the transformation rules preserves the local stratification σ for the initial program P_0 and, thus, all programs in the transformation sequence are locally stratified w.r.t. σ, and (ii) any admissible transformation sequence preserves the perfect model of the initial program.

Lemma 4.6 (Preservation of Local Stratification) *Let P_0 be a locally stratified ω-program, σ be a local stratification for P_0, and $P_0, \ldots, P_n$ be an admissible transformation sequence. Then the programs $P_0 \cup Defs_n$, $P_1, \ldots, P_n$, are all locally stratified w.r.t. σ.*

Theorem 4.7 (Correctness of Admissible Transformation Sequences) *Every admissible transformation sequence is correct.*

Now let us make a few comments on Condition (2) of Definition 4.5 and related conditions presented in the literature. Transformation sequences of stratified programs over finite terms constructed by using rules R1, R3, and R6 have been first considered in [22]. In that paper there is a sufficient condition, called (F4), for the preservation of the perfect model. Condition (F4) is like our Condition (2) except that (F4) does not require the σ-maximality of the atom w.r.t. which positive unfolding is performed. A set of transformation rules which includes also the negative unfolding rule R4, was proposed in [16] for locally stratified logic programs, and in [8] for locally stratified constraint logic programs. In [23] Condition (F4) is shown to be insufficient for the preservation of the perfect model if rule R4 is used together with rules R1, R3, and R6, as demonstrated by the following example.

Example 2. Let us consider the initial program P_0 made out of the following clauses:

$m \leftarrow$

$e \leftarrow \neg m$

$e \leftarrow e$

By rule R1 we introduce the clause

δ_1: $f \leftarrow m \wedge \neg e$

and we derive program $P_1 = P_0 \cup \{\delta_1\}$ and $Defs_1 = \{\delta_1\}$.

By rule R3 we unfold δ_1 w.r.t. m and we get the clause

δ_2: $f \leftarrow \neg e$.

We derive program $P_2 = P_0 \cup \{\delta_2\}$. Thus, Condition (F4) is satisfied. By rule R4 we unfold δ_2 w.r.t. $\neg e$ and we get

δ_3: $f \leftarrow m \wedge \neg e$.

We derive program $P_3 = P_0 \cup \{\delta_3\}$. By rule R6 we fold clause δ_3 using clause δ_1, and we get

δ_4: $f \leftarrow f$.

We derive program $P_4 = P_0 \cup \{\delta_4\}$ and $Defs_4 = \{\delta_1\}$. We have that $f \in M(P_0 \cup Defs_4)$ and $f \notin M(P_4)$. Thus, the transformation sequence $P_0, \ldots, P_4$ is not correct.

In order to guarantee the preservation of the perfect model semantics, [23] has proposed the following stronger applicability condition for negative unfolding:

Condition (NU): the negative unfolding rule R4 can be applied only if it does not increase the number of positive occurrences of atoms in the body of any derived clause.

Indeed, in the incorrect transformation sequence of Example 2 the negative unfolding does not comply with this Condition (NU).

However, Condition (NU) is very restrictive, because it forbids the unfolding of a clause w.r.t. a negative literal $\neg A$ when the body of a clause defining A contains an occurrence of a negative literal. Unfortunately, many of the correct transformation strategies proposed in [16, 8] would be ruled out if Condition (NU) is enforced. Our Condition (2) is more liberal than Condition (NU) and, in particular, it allows us to unfold w.r.t. a negative literal $\neg A$ also if the body of a clause defining A contains occurrences of negative literals. The following is an example of a correct, admissible transformation sequence which violates Condition (NU).

Example 3. Let us consider the initial program P_0 made out of the following clauses:

$even(0) \leftarrow$

$even(s(s(X))) \leftarrow even(X)$,

$odd(s(0)) \leftarrow$

$odd(s(X)) \leftarrow \neg\, odd(X)$

and the transformation sequence we now construct starting from P_0. By rule R1 we introduce the following clause

δ_1: $p \leftarrow even(X) \wedge \neg\, odd(s(X))$

and we derive $P_1 = P_0 \cup \{\delta_1\}$. By taking a local stratification function σ such that, for all ground terms t_1 and t_2, $\sigma(p) = \sigma(even(t_1)) > \sigma(odd(t_2))$, we have that δ_1 is σ-tight and $even(X)$ is a σ-maximal atom in its body. By unfolding δ_1 w.r.t. $even(X)$ we derive $P_2 = P_0 \cup \{\delta_2, \delta_3\}$, where

δ_2: $p \leftarrow \neg\, odd(s(0))$

δ_3: $p \leftarrow even(X) \wedge \neg\, odd(s(s(s(X))))$

By unfolding, clause δ_2 is removed and we derive $P_3 = P_0 \cup \{\delta_3\}$.

By unfolding δ_3 w.r.t. $\neg odd(s(s(s(X))))$ we derive $P_4 = P_0 \cup \{\delta_4\}$, where

δ_4: $p \leftarrow even(X) \wedge odd(s(s(X)))$
By unfolding δ_4 w.r.t. $odd(s(s(X)))$, we derive $P_5 = P_0 \cup \{\delta_5\}$, where
δ_5: $p \leftarrow even(X) \wedge \neg\, odd(s(X))$
By applying rule R6, we fold clause δ_5 using clause δ_1 and derive the final program $P_6 = P_0 \cup \{\delta_6\}$, where
δ_6: $p \leftarrow p$.
The transformation sequence $P_0, \ldots, P_6$ is admissible and, thus, correct. In particular, the application of rule R6 satisfies Condition (2) of Definition 4.5 because δ_1 is σ-tight and δ_5 is a descendant of δ_3 which has been derived by unfolding w.r.t. a σ-maximal atom whose predicate occurs in P_0.

Note that, $P_0, \ldots, P_6$ violates Condition (NU) because, by unfolding clause δ_3 w.r.t. the literal $\neg odd(s(s(s(X))))$, the number of positive occurrences of atoms in the body of the derived clause δ_4 is larger than that number in δ_3.

Finally, note that the incorrect transformation sequence of Example 2 is *not* an admissible transformation sequence in the sense of our Definition 4.5, because it does not comply with Condition (2). Indeed, consider any stratification σ. The atom m is not σ-maximal in $m \wedge \neg e$ because e depends on $\neg m$ and, hence, $\sigma(\neg e) > \sigma(m)$. Thus, the positive folding rule R6 is applied to the clause δ_3 which is not a descendant of any clause derived by unfolding w.r.t. a σ-maximal atom.

5. Verifying Properties of ω-Programs by Program Transformation

In this section we will outline a general method, based on the transformation rules presented in Section 3, for verifying properties of ω-programs. Then we will see our transformation-based verification method in action in the proof of: (i) the non-emptiness of the language accepted by a Büchi automaton, and (ii) the containment between ω-regular languages.

We assume that we are given an ω-program P defining a unary predicate *prop* of type `ilist`, which specifies a property of interest, and we want to check whether or not $M(P) \models \exists X\, prop(X)$. Our verification method consists of two steps.
Step 1. By using the transformation rules for ω-programs presented in Section 3 we derive a *monadic* ω-program T (see Definition 5.1 below), such that
$M(P) \models \exists X\, prop(X)$ iff $M(T) \models \exists X\, prop(X)$.
Step 2. We apply to T the decision procedure of [17] for monadic ω-programs and we check whether or not $M(T) \models \exists X\, prop(X)$.

Our verification method is an extension to ω-programs of the transformation-based method for proving properties of logic programs on finite terms presented in [16]. Furthermore, our method is more powerful than the transformation-based method for verifying CTL* properties of finite state reactive systems presented in [17]. Indeed, at Step 1 of the verification method proposed here, (i) we start from an arbitrary ω-program, instead of an ω-program which encodes the branching time temporal logic CTL*, and (ii) we use transformation rules more powerful than those in [17]. In particular, similarly to [16], the rules applied at Step 1 allow us to eliminate the existential variables from program P, while the transformation presented in [17] consists of a specialization of the initial program w.r.t. the property to be verified.

Note that there exists no algorithm which always succeeds in transforming an ω-program into a monadic ω-program. Indeed, (i) the problem of verifying whether or not, for any ω-program P and unary predicate *prop*, $M(P) \models \exists X\, prop(X)$ is undecidable, because the class of ω-programs

includes the locally stratified logic programs on finite terms, and (ii) the proof method for monadic ω-programs presented in [17] is complete. However, we believe that automatic transformation strategies can be proposed for significant subclasses of ω-programs along the lines of [19, 16].

Definition 5.1 (Monadic ω-Programs) *A* monadic *ω-clause is an ω-clause of the form $A_0 \leftarrow L_1 \wedge \ldots \wedge L_m$, with $m \geq 0$, such that*:
(i) *A_0 is an atom of the form p_0 or $q_0([\![s|X_0]\!])$, where q_0 is a predicate of type* `ilist` *and $s \in \Sigma$,*
(ii) *for $i = 1, \ldots, m$, L_i is either an atom A_i or a negated atom $\neg A_i$, where A_i is of the form p_i or $q_i(X_i)$, and q_i is a predicate of type* `ilist`*, and*
(iii) *there exists a level mapping ℓ such that, for $i = 1, \ldots, m$, if L_i is an atom and $vars(A_0) \not\supseteq vars(L_i)$, then $\ell(A_0) > \ell(L_i)$ else $\ell(A_0) \geq \ell(L_i)$.*

A monadic ω-program *is a finite set of monadic ω-clauses.*

Example 4 (Non-Emptiness of Languages Accepted by Büchi Automata) In this first application of our verification method, we will consider *Büchi automata*, which are finite automata acting on infinite words [27], and we will check whether or not the language accepted by a Büchi automaton is empty. It is well known that this verification problem has important applications in the area of model checking (see, for instance, [4]).

A *Büchi automaton* $\mathcal{A}$ is a nondeterministic finite automaton $\langle \Sigma, Q, q_0, \delta, F \rangle$, where, as usual, Σ is the input alphabet, Q is the set of states, q_0 is the initial state, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, and F is the set of final states. A *run* of the automaton $\mathcal{A}$ on an infinite input word $w = a_0\, a_1 \ldots \in \Sigma^\omega$ is an infinite sequence $\rho = \rho_0\, \rho_1 \ldots \in Q^\omega$ of states such that ρ_0 is the initial state q_0 and, for all $n \geq 0$, $\langle \rho_n, a_n, \rho_{n+1} \rangle \in \delta$. Let $Inf(\rho)$ denote the set of states that occur infinitely often in the infinite sequence ρ of states. An infinite word $w \in \Sigma^\omega$ is *accepted* by $\mathcal{A}$ if there exists a run ρ of $\mathcal{A}$ on w such that $Inf(\rho) \cap F \neq \emptyset$ or, equivalently, if there is no state ρ_m in ρ such that every state ρ_n, with $n \geq m$, is not final.

The *language accepted* by $\mathcal{A}$ is the subset of Σ^ω, denoted $\mathcal{L}(\mathcal{A})$, of the infinite words accepted by $\mathcal{A}$. In order to check whether or not the language $\mathcal{L}(\mathcal{A})$ is empty, we construct an ω-program which defines a unary predicate *accepting_run* such that:

(α) $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\exists X\, accepting\_run(X)$

The predicate *accepting_run* is defined by the following formulas:

(1) $accepting\_run(X) \equiv_{def} run(X) \wedge \neg\, rejecting(X)$

(2) $run(X) \equiv_{def} \exists S\, (occ(0, X, S) \wedge initial(S)) \wedge$
$\forall N\, \forall S_1\, \forall S_2\, (nat(N) \wedge occ(N, X, S_1) \wedge occ(s(N), X, S_2) \rightarrow \exists A\, tr(S_1, A, S_2)))$

(3) $rejecting(X) \equiv_{def} \exists M (nat(M) \wedge \forall N \forall S (geq(N, M) \wedge occ(N, X, S) \rightarrow \neg\, final(S)))$

where, for all $n \geq 0$, for all $\rho = \rho_0\, \rho_1 \ldots \in Q^\omega$, for all $q, q_1, q_2 \in Q$, for all $a \in \Sigma$,
(i) $occ(s^n(0), \rho, q)$ iff $\rho_n = q$,
(ii) $initial(q)$ iff $q = q_0$,
(iii) $nat(s^n(0))$ iff $n \geq 0$,
(iv) $tr(q_1, a, q_2)$ iff $\langle q_1, a, q_2 \rangle \in \delta$,
(v) $geq(s^n(0), s^m(0))$ iff $n \geq m$, and
(vi) $final(q)$ iff $q \in F$.

By (α) and (1)–(3) above, $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff there exists an infinite sequence $\rho = \rho_0\, \rho_1 \ldots \in Q^\omega$ of states such that: (i) ρ_0 is the initial state q_0, (ii) for all $n \geq 0$, there exists $a \in \Sigma$ such that $\langle \rho_n, a, \rho_{n+1} \rangle \in \delta$ (see (2)), and (iii) there exists no state ρ_m, with $m \geq 0$, in ρ such that, for all $n \geq m$, $\rho_n \notin F$ (see (3)).

Now we introduce an ω-program $P_{\mathcal{A}}$ defining the predicates *accepting_run*, *run*, *rejecting*, *nat*, *occ*, and *geq*. In particular, clause 1 corresponds to formula (1), clauses 2–4 correspond to formula (2), and clauses 5 and 6 correspond to formula (3). (Actually, clauses 1–6 can be derived from formulas (1)–(3) by applying the Lloyd-Topor transformation [13].) In program $P_{\mathcal{A}}$ any infinite sequence $\rho_0\rho_1\ldots$ of states is represented by the infinite list $[\![\rho_0, \rho_1, \ldots]\!]$ of constants.

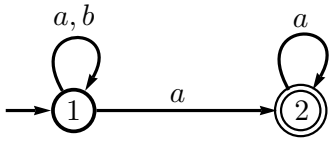
Given a Büchi automaton $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F\rangle$, the encoding ω-program $P_{\mathcal{A}}$ consists of the following clauses (independent of $\mathcal{A}$):

1. $\mathit{accepting\_run}(X) \leftarrow \mathit{run}(X) \wedge \neg\, \mathit{rejecting}(X)$
2. $\mathit{run}(X) \leftarrow \mathit{occ}(0, X, S) \wedge \mathit{initial}(S) \wedge \neg\, \mathit{not\_a\_run}(X)$
3. $\mathit{not\_a\_run}(X) \leftarrow \mathit{nat}(N) \wedge \mathit{occ}(N,X,S_1) \wedge \mathit{occ}(s(N),X,S_2) \wedge \neg\, \mathit{exists\_tr}(S_1,S_2)$
4. $\mathit{exists\_tr}(S_1, S_2) \leftarrow \mathit{tr}(S_1, A, S_2)$
5. $\mathit{rejecting}(X) \leftarrow \mathit{nat}(M) \wedge \neg\, \mathit{exists\_final}(M, X)$
6. $\mathit{exists\_final}(M, X) \leftarrow \mathit{geq}(N, M) \wedge \mathit{occ}(N, X, S) \wedge \mathit{final}(S)$
7. $\mathit{nat}(0) \leftarrow$
8. $\mathit{nat}(s(N)) \leftarrow \mathit{nat}(N)$
9. $\mathit{occ}(0, [\![S|X]\!], S) \leftarrow$
10. $\mathit{occ}(s(N), [\![S|X]\!], R) \leftarrow \mathit{occ}(N, X, R)$
11. $\mathit{geq}(N, 0) \leftarrow$
12. $\mathit{geq}(s(N), s(M)) \leftarrow \mathit{geq}(N, M)$

together with the clauses (depending on $\mathcal{A}$) which define the predicates *initial*, *tr*, and *final*, where: for all states $s, s_1, s_2 \in Q$, for all symbols $a \in \Sigma$, (i) $\mathit{initial}(s)$ holds iff s is q_0, (ii) $\mathit{tr}(s_1,a,s_2)$ holds iff $\langle s_1,a,s_2\rangle \in \delta$, and (iii) $\mathit{final}(s)$ holds iff $s \in F$.

The ω-program $P_{\mathcal{A}}$ is locally stratified w.r.t. the stratification function σ defined as follows: for every atom A in $\mathcal{B}_\omega$, $\sigma(A) = 0$, except that: for every element n in $\{s^k(0) \mid k \geq 0\}$, for every infinite list ρ in Q^ω, (i) $\sigma(\mathit{rejecting}(\rho)) = \sigma(\mathit{not\_a\_run}(\rho)) = \sigma(\mathit{nat}(n)) = 1$, and (ii) $\sigma(\mathit{run}(\rho)) = \sigma(\mathit{accepting\_run}(\rho)) = 2$.

Now, let us consider a Büchi automaton $\mathcal{A}$ such that:
$\Sigma = \{a, b\}$, $Q = \{1, 2\}$, $q_0 = 1$, $\delta = \{\langle 1, a, 1\rangle, \langle 1, b, 1\rangle, \langle 1, a, 2\rangle, \langle 2, a, 2\rangle\}$, $F = \{2\}$
which can be represented by the following graph:



For this automaton $\mathcal{A}$, program $P_{\mathcal{A}}$ consists of clauses 1–12 and the following clauses 13–18 that encode the initial state (clause 13), the transition relation (clauses 14–17), and the final state (clause 18):

13. $\mathit{initial}(1) \leftarrow$
14. $\mathit{tr}(1, a, 1) \leftarrow$ 15. $\mathit{tr}(1, b, 1) \leftarrow$ 16. $\mathit{tr}(1, a, 2) \leftarrow$ 17. $\mathit{tr}(2, a, 2) \leftarrow$
18. $\mathit{final}(2) \leftarrow$

In order to check whether or not $\mathcal{L}(\mathcal{A}) = \emptyset$ we proceed in two steps as indicated at the beginning of this Section 5. In the first step we use the rules of Section 3 for transforming the ω-program $P_{\mathcal{A}}$ into a monadic ω-program T. This transformation aims at the elimination of the existential variables from clauses 1–6, with the objective of deriving unary predicates of type `ilist`. We start from clause 6 and, by instantiation of the variable X of type `ilist`, we get:

19. $\mathit{exists\_final}(M, [\![1|X]\!]) \leftarrow \mathit{geq}(N, M) \wedge \mathit{occ}(N, [\![1|X]\!], S) \wedge \mathit{final}(S)$

20. $\mathit{exists\_final}(M, [\![2|X]\!]) \leftarrow \mathit{geq}(N, M) \wedge \mathit{occ}(N, [\![2|X]\!], S) \wedge \mathit{final}(S)$

By some unfolding and subsumption steps, from clauses 19 and 20 we get:

21. $\mathit{exists\_final}(0, [\![1|X]\!]) \leftarrow \mathit{occ}(N, X, S) \wedge \mathit{final}(S)$
22. $\mathit{exists\_final}(s(M), [\![1|X]\!]) \leftarrow \mathit{geq}(N, M) \wedge \mathit{occ}(N, X, S) \wedge \mathit{final}(S)$
23. $\mathit{exists\_final}(0, [\![2|X]\!]) \leftarrow$
24. $\mathit{exists\_final}(s(M), [\![2|X]\!]) \leftarrow \mathit{geq}(N, M) \wedge \mathit{occ}(N, X, S) \wedge \mathit{final}(S)$

Note that clauses 21–24 are descendants of clauses derived by unfolding clauses 19 and 20 w.r.t. the σ-maximal atom $\mathit{geq}(N, M)$. By rule R1, we introduce:

25. $\mathit{new}_1(X) \leftarrow \mathit{occ}(N, X, S) \wedge \mathit{final}(S)$

This clause is σ-tight by taking, for every infinite list ρ of states, $\sigma(\mathit{new}_1(\rho)) = 0$. By folding clause 21 using clause 25, and folding clauses 22 and 24 using clause 6 (indeed, without loss of generality, we may assume that clauses 1–6 have been introduced by rule R1), we get:

26. $\mathit{exists\_final}(0, [\![1|X]\!]) \leftarrow \mathit{new}_1(X)$
27. $\mathit{exists\_final}(s(M), [\![1|X]\!]) \leftarrow \mathit{exists\_final}(M, X)$
28. $\mathit{exists\_final}(s(M), [\![2|X]\!]) \leftarrow \mathit{exists\_final}(M, X)$

By instantiation of the variable X and by some unfolding and subsumption steps, from clause 25 we get:

29. $\mathit{new}_1([\![1|X]\!]) \leftarrow \mathit{occ}(N, X, S) \wedge \mathit{final}(S)$
30. $\mathit{new}_1([\![2|X]\!]) \leftarrow$

Note that clause 29 is a descendant of clause 25, that has been unfolded w.r.t. the σ-maximal atom $\mathit{occ}(N, X, S)$. By folding clause 29 using clause 25 we get:

31. $\mathit{new}_1([\![1|X]\!]) \leftarrow \mathit{new}_1(X)$

At this point we have obtained the definitions of the predicates *exists_final* and new_1 (that is, clauses 23, 26–28, 30, and 31) that do not have existential variables.

Now the transformation of program $P_{\mathcal{A}}$ proceeds by performing on clauses 1–5 a sequence of transformation steps, which is similar to the one we have performed above on clause 6 for eliminating its existential variables. By doing so, we get:

32. $\mathit{accepting\_run}([\![1|X]\!]) \leftarrow \neg\, \mathit{not\_a\_run}(X) \wedge \mathit{new}_1(X) \wedge \neg\, \mathit{rejecting}(X)$
33. $\mathit{run}([\![1|X]\!]) \leftarrow \neg\, \mathit{not\_a\_run}(X)$
34. $\mathit{not\_a\_run}([\![1|X]\!]) \leftarrow \mathit{not\_a\_run}(X)$
35. $\mathit{not\_a\_run}([\![2|X]\!]) \leftarrow \mathit{new}_2(X)$
36. $\mathit{not\_a\_run}([\![2|X]\!]) \leftarrow \mathit{not\_a\_run}(X)$
37. $\mathit{new}_2([\![1|X]\!]) \leftarrow$
38. $\mathit{rejecting}([\![1|X]\!]) \leftarrow \neg\, \mathit{new}_1(X)$
39. $\mathit{rejecting}([\![1|X]\!]) \leftarrow \mathit{rejecting}(X)$
40. $\mathit{rejecting}([\![2|X]\!]) \leftarrow \mathit{rejecting}(X)$

The final ω-program T obtained from program $P_{\mathcal{A}}$, consists of clauses 30–40 and it is a monadic ω-program.

Now, in the second step of our verification method, we check whether or not the formula $\exists X\, \mathit{accepting\_run}(X)$ holds in $M(T)$ by applying the proof method of [17]. We construct the tree depicted in Figure 1, where the literals occurring in the two lowest levels are the same (see the two rectangles) and, thus, we have detected an infinite loop. According to the conditions given in Definition 6 of [17], this tree is a proof of $\exists X\, \mathit{accepting\_run}(X)$. The run $\rho = 12^{\omega}$ is a witness for X and corresponds to the accepted word a^{ω}. Thus, $\mathcal{L}(\mathcal{A}) \neq \emptyset$.

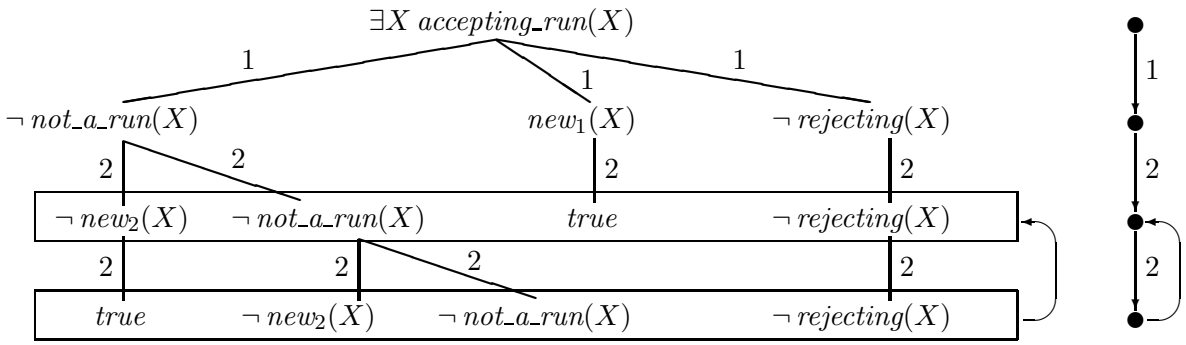


Figure 1: Proof of $\exists X\ accepting\_run(X)$ w.r.t. the monadic ω-program T. On the right we have shown the infinite loop and the associated accepting run 122^ω (that is, 12^ω).

Example 5 (Containment Between ω-Regular Languages) In this second application of our verification method, we will consider regular sets of infinite words over a finite alphabet Σ [27]. These sets are denoted by ω-regular expressions whose syntax is defined as follows:

$e ::= a \mid e_1 e_2 \mid e_1 + e_2 \mid e^*$ with $a \in \Sigma$ (regular expressions)

$f ::= e^\omega \mid e_1 e_2^\omega \mid f_1 + f_2$ (ω-regular expressions)

Given a regular (or an ω-regular) expression r, by $\mathcal{L}(r)$ we indicate the set of all words in Σ^* (or Σ^ω, respectively) denoted by r. In particular, given a regular expression e, we have that $\mathcal{L}(e^\omega) = \{w_0 w_1 \ldots \in \Sigma^\omega \mid \text{for } i \geq 0, w_i \in \mathcal{L}(e) \subseteq \Sigma^*\}$.

Now we introduce an ω-program, called P_f, which defines the predicate $\omega\text{-}acc$ such that for any ω-regular expression f, for any infinite word w, $\omega\text{-}acc(f, w)$ holds iff $w \in \mathcal{L}(f)$. Any infinite word $a_0 a_1 \ldots \in \Sigma^\omega$ is represented by the infinite list $[\![a_0, a_1, \ldots]\!]$ of symbols in Σ. The ω-program P_f is made out of the following clauses:

1. $acc(E, [E]) \leftarrow symb(E)$
2. $acc(E_1 E_2, X) \leftarrow app(X_1, X_2, X) \wedge acc(E_1, X_1) \wedge acc(E_2, X_2)$
3. $acc(E_1 + E_2, X) \leftarrow acc(E_1, X)$
4. $acc(E_1 + E_2, X) \leftarrow acc(E_2, X)$
5. $acc(E^*, [\,]) \leftarrow$
6. $acc(E^*, X) \leftarrow app(X_1, X_2, X) \wedge acc(E, X_1) \wedge acc(E^*, X_2)$
7. $\omega\text{-}acc(F_1 + F_2, X) \leftarrow \omega\text{-}acc(F_1, X)$
8. $\omega\text{-}acc(F_1 + F_2, X) \leftarrow \omega\text{-}acc(F_2, X)$
9. $\omega\text{-}acc(E^\omega, X) \leftarrow \neg\, new_1(E, X)$
10. $\omega\text{-}acc(E_1 E_2^\omega, X) \leftarrow prefix(X, N, X_1) \wedge acc(E_1, X_1) \wedge \omega\text{-}acc1(E_2^\omega, X_1, X)$
11. $new_1(E, X) \leftarrow nat(M) \wedge \neg\, new_2(E, M, X)$
12. $new_2(E, M, X) \leftarrow geq(N, M) \wedge prefix(X, N, V) \wedge acc(E^*, V)$
13. $\omega\text{-}acc1(E, [\,], X) \leftarrow \omega\text{-}acc(E, X)$
14. $\omega\text{-}acc1(E, [H|T], [\![H|X]\!]) \leftarrow \omega\text{-}acc1(E, T, X)$
15. $geq(N, 0) \leftarrow$
16. $geq(s(N), s(M)) \leftarrow geq(N, M)$
17. $nat(0) \leftarrow$
18. $nat(s(N)) \leftarrow nat(N)$
19. $prefix(X, 0, [\,]) \leftarrow$
20. $prefix([\![S|X]\!], s(N), [S|Y]) \leftarrow prefix(X, N, Y)$

21. $app([\,], Y, Y) \leftarrow$
22. $app([S|X], Y, [S|Z]) \leftarrow app(X, Y, Z)$

together with the clauses defining the predicate *symb*, where $symb(a)$ holds iff $a \in \Sigma$. We have that $prefix(X, N, Y)$ holds iff Y is the list of the $N\,(\geq 0)$ leftmost symbols of the infinite list X. Clauses 1–6 stipulate that, for any finite word w and regular expression e, $acc(e, w)$ holds iff $w \in \mathcal{L}(e)$. Analogously, clauses 7–14 stipulate that, for any infinite word w and ω-regular expression f, $\omega\text{-}acc(f, w)$ holds iff $w \in \mathcal{L}(f)$. In particular, clauses 9, 11, and 12 correspond to the following definition:

$$\omega\text{-}acc(E^\omega, X) \equiv_{def} \forall M(nat(M) \rightarrow \exists N \exists V(geq(N, M) \wedge prefix(X, N, V) \wedge acc(E^*, V)))$$

The ω-program P_f is stratified and, thus, it is locally stratified.

Now, let us consider the ω-regular expressions $f_1 \equiv_{def} a^\omega$ and $f_2 \equiv_{def} (b^*a)^\omega$. The following two clauses:

23. $expr_1(X) \leftarrow \omega\text{-}acc(a^\omega, X)$
24. $expr_2(X) \leftarrow \omega\text{-}acc((b^*a)^\omega, X)$

together with program P_f, define the predicates $expr_1$ and $expr_2$ such that, for every infinite word w, $expr_1(w)$ holds iff $w \in \mathcal{L}(f_1)$ and $expr_2(w)$ holds iff $w \in \mathcal{L}(f_2)$. If we introduce the following clause:

25. $not\_contained(X) \leftarrow expr_1(X) \wedge \neg\, expr_2(X)$

we have that $\mathcal{L}(f_1) \subseteq \mathcal{L}(f_2)$ iff $M(P_f \cup \{23, 24, 25\}) \not\models \exists X\, not\_contained(X)$. By performing a sequence of transformation steps which is similar to the one we have performed in Example 4, from program $P_f \cup \{23, 24, 25\}$ we get the following monadic ω-program T:

26. $not\_contained([\![a|X]\!]) \leftarrow \neg new_3(X) \wedge new_4(X)$
27. $new_3([\![a|X]\!]) \leftarrow new_3(X)$
28. $new_3([\![b|X]\!]) \leftarrow$
29. $new_4([\![a|X]\!]) \leftarrow new_4(X)$
30. $new_4([\![b|X]\!]) \leftarrow new_5(X)$
31. $new_5([\![a|X]\!]) \leftarrow new_4(X)$
32. $new_5([\![b|X]\!]) \leftarrow new_5(X)$
33. $new_5([\![b|X]\!]) \leftarrow \neg\, new_6(X)$
34. $new_6([\![a|X]\!]) \leftarrow$
35. $new_6([\![b|X]\!]) \leftarrow new_6(X)$

By using the proof method for monadic ω-programs of [17] we have that:
$M(T) \not\models \exists X\, not\_contained(X)$
and, thus, $\mathcal{L}(f_1) \subseteq \mathcal{L}(f_2)$.

6. Related Work and Conclusions

There have been various proposals for extending logic programming languages to infinite structures (see, for instance, [5, 13, 14, 24]). In order to provide the semantics of infinite structures, these languages introduce new concepts, such as *complete Herbrand interpretations*, *rational trees*, and *greatest models*. Moreover, the operational semantics of these languages requires an extension of SLDNF-resolution by means of equational reasoning and new inference rules, such as the so-called *coinductive hypothesis* rule.

On the contrary, the semantics of ω-programs we consider in this paper is very close to the usual perfect model semantics for logic programs on finite terms, and we do not define any

new operational semantics. Indeed, the main objective of this paper is *not* to provide a new model for computing over infinite structures, but to present a methodology, based on unfold/fold transformation rules, for reasoning about such structures and proving their properties.

Very little work has been done for applying transformation techniques to logic languages that specify the (possible infinite) computations of reactive systems. Notable exceptions are [28] and [7], where the unfold/fold transformation rules have been studied in the context of *guarded Horn clauses* (GHC) and *concurrent constraint programs* (CCP). However, GHC and CCP programs are *definite* programs and do not manipulate terms denoting infinite lists.

The transformation rules presented in this paper extend to ω-programs the rules for general programs proposed in [8, 16, 21, 22, 23]. In Sections 3 and 4 we discuss in detail the relationship of the rules in those papers with our rules here.

In Section 5 we have used our transformation rules for extending to infinite lists a verification methodology proposed in [16] and, as an example, we have shown how to verify properties of the infinite behaviour of Büchi automata and properties of ω-regular languages. This extends our previous work (see [17]), as already illustrated at the beginning of Section 5.

The verification methodology based on transformations we have proposed here, is very general and it can be applied to the proof of properties of infinite state reactive systems and, thus, it goes beyond the capabilities of finite state model checkers. The focus of our paper has been the proposal of correct transformation rules, that is, rules which preserve the perfect model, while the automation of the verification methodology itself is left for future work. This automation requires the design of suitable transformation strategies that can be defined by adapting to ω-programs some strategies already developed in the case of logic programs on finite terms (see, for instance, [19, 16]).

Many other papers use logic programming, possibly with constraints, for specifying and verifying properties of finite or infinite state reactive systems (see, for instance, [1, 6, 9, 11, 12, 15, 20]), but they do not consider terms which explicitly represent infinite structures. As we have seen in the examples of Section 5, infinite lists are very convenient for specifying those properties and the use of infinite lists avoids ingenious encodings which would have been otherwise required.

A. Proofs for Section 4

We start off by showing that admissible transformation sequences preserve the local stratification σ for the initial program P_0 as stated in the following lemma.

Lemma 4.6 (Preservation of Local Stratification)
Suppose that P_0 is a locally stratified ω-program, σ is a local stratification for P_0, and $P_0, P_1, \ldots, P_n$ is an admissible transformation sequence. Then the programs $P_0 \cup \mathit{Defs}_n$, $P_1, \ldots, P_n$ are locally stratified w.r.t. σ.

Proof. Since $P_0, \ldots, P_n$ is an admissible transformation sequence, every definition in Defs_n is locally stratified w.r.t. σ (see Point (1) of Definition 4.5). Since, by hypothesis, P_0 is locally stratified w.r.t. σ, also $P_0 \cup \mathit{Defs}_n$ is locally stratified w.r.t. σ.
Now we will prove that, for $k = 0, \ldots, n$, P_k is locally stratified w.r.t. σ by induction on k.

Basis ($k = 0$). By hypothesis P_0 is locally stratified w.r.t. σ.

Step. We assume that P_k is locally stratified w.r.t. σ and we show that P_{k+1} is locally stratified w.r.t. σ. We proceed by cases depending on the transformation rule which is applied to derive P_{k+1} from P_k.

Case 1. Program P_{k+1} is derived by definition introduction (rule R1). We have that $P_{k+1} = P_k \cup \{\delta_1, \ldots, \delta_m\}$, where P_k is locally stratified w.r.t. σ by the inductive hypothesis. Since $P_0, \ldots, P_n$ is an admissible transformation sequence, $\{\delta_1, \ldots, \delta_m\}$ is locally stratified w.r.t. σ (see Point (1) of Definition 4.5). Thus, P_{k+1} is locally stratified w.r.t. σ.

Case 2. Program P_{k+1} is derived by instantiation (rule R2). We have that $P_{k+1} = (P_k - \{\gamma\}) \cup \{\gamma_1, \ldots, \gamma_h\}$, where γ is the clause $H \leftarrow B$ and, for $i = 1, \ldots, h$, γ_i is the clause $(H \leftarrow B)\{X/[\![s_i|X]\!]\}$.
Take any $i \in \{1, \ldots, h\}$. Let $L\{X/[\![s_i|X]\!]\}$ be a literal in the body of γ_i. Let v be any valuation and v' be the valuation such that $v'(X) = [\![s_i|v(X)]\!]$ and $v'(Y) = v(Y)$ for every variable Y different from X. We have:

$$\begin{array}{lll} \sigma(v(H\{X/[\![s_i|X]\!]\})) = \sigma(v'(H)) & & \text{(definition of } v') \\ \qquad \geq \sigma(v'(L)) & & (\gamma \text{ is locally stratified w.r.t. } \sigma) \\ \qquad = \sigma(v(L\{X/[\![s_i|X]\!]\})) & & \text{(definition of } v') \end{array}$$

Thus, γ_i is locally stratified w.r.t. σ. Hence, P_{k+1} is locally stratified w.r.t. σ.

Case 3. Program P_{k+1} is derived by positive unfolding (rule R3). We have that $P_{k+1} = (P_k - \{\gamma\}) \cup \{\eta_1, \ldots, \eta_m\}$, where γ is a clause in P_k of the form $H \leftarrow G_L \wedge A \wedge G_R$ and clauses $\eta_1, \ldots, \eta_m$ are derived by unfolding γ w.r.t. A. Since, by the induction hypothesis, $(P_k - \{\gamma\})$ is locally stratified w.r.t. σ, it remains to show that, for $i = 1, \ldots, m$, clause η_i is locally stratified w.r.t. σ. For $i = 1, \ldots, m$, η_i is of the form $(H \leftarrow G_L \wedge B_i \wedge G_R)\vartheta_i$, where γ_i: $K_i \leftarrow B_i$ is a clause in a variant of P_k such that γ_i has no variable in common with γ and $A\vartheta_i = K_i\vartheta_i$. Take any valuation v and let v' be a valuation such that, for every variable X occurring in γ or γ_i, $v'(X) = v(X\vartheta_i)$.

Let $G_L \wedge B_i \wedge G_R$ be the conjunction of s (≥ 0) literals $L_1, \ldots, L_s$. Without loss of generality, we assume that $G_L \wedge G_R$ is $L_1 \wedge \ldots \wedge L_r$ and B_i is $L_{r+1} \wedge \ldots \wedge L_s$, with $0 \leq r \leq s$.
For $j = 1, \ldots, r$, we have:

$$\begin{array}{lll} \sigma(v(H\vartheta_i)) = \sigma(v'(H)) & & \text{(definition of } v') \\ \qquad \geq \sigma(v'(L_j)) & & (\gamma \text{ is locally stratified w.r.t. } \sigma) \\ \qquad = \sigma(v(L_j\vartheta_i)) & & \text{(definition of } v') \end{array}$$

For $j = r+1, \ldots, s$, we have:

$$\begin{array}{lll} \sigma(v(H\vartheta_i)) & = \sigma(v'(H)) & \text{(definition of } v') \\ & \geq \sigma(v'(A)) & (\gamma \text{ is locally stratified w.r.t. } \sigma) \\ & = \sigma(v'(K_i)) & \text{(definition of } v' \text{ and because } A\vartheta_i = K_i\vartheta_i) \\ & \geq \sigma(v'(L_j)) & (\gamma_i \text{ is locally stratified w.r.t. } \sigma) \\ & = \sigma(v(L_j\vartheta_i)) & \text{(definition of } v') \end{array}$$

Thus, the clause η_i is locally stratified w.r.t. σ.

Case 4. Program P_{k+1} is derived by negative unfolding (rule R4). We have that $P_{k+1} = (P_k - \{\gamma\}) \cup \{\eta_1, \ldots, \eta_r\}$, where γ is a clause in P_k of the form $H \leftarrow G_L \wedge \neg A \wedge G_R$ and clauses $\eta_1, \ldots, \eta_r$ are derived by negative unfolding γ w.r.t. $\neg A$. Since, by the inductive hypothesis, $(P_k - \{\gamma\})$ is locally stratified w.r.t. σ, it remains to show that, for $j = 1, \ldots, r$, clause η_j is locally stratified w.r.t. σ.

Let γ_1: $K_1 \leftarrow B_1$, ..., γ_m: $K_m \leftarrow B_m$ be the clauses in a variant of P_k such that, for $i = 1, \ldots, m$, $A = K_i\vartheta_i$ for some substitution ϑ_i. Then, for $j = 1, \ldots, r$, η_j is of the form $H \leftarrow L_{j1} \wedge \ldots \wedge L_{js}$ and, by construction, for $p = 1, \ldots, s$, L_{jp} is a literal such that either (Case a) L_{jp} is an atom that occurs positively in $G_L \wedge G_R$, or (Case b) L_{jp} is a negated atom that occurs in $G_L \wedge G_R$, or (Case c) L_{jp} is an atom M and $\neg M$ occurs in $B_i\vartheta_i$, for some $i \in \{1, \ldots, m\}$, or (Case d) L_{jp} is a negated atom $\neg M$ and M is an atom that occurs positively in $B_i\vartheta_i$, for some $i \in \{1, \ldots, m\}$.

Take any $j \in \{1, \ldots, h\}$. Take any $p \in \{1, \ldots, s\}$. Take any valuation v. In Cases (a) and (b) we have $\sigma(v(H)) \geq \sigma(v(L_{jp}))$ because, by the inductive hypothesis, γ is locally stratified w.r.t. σ. In Case (c) we have:

$$\begin{array}{lll} \sigma(v(H)) & > \sigma(v(A)) & (\gamma \text{ is locally stratified w.r.t. } \sigma \text{ and } \neg A \text{ occurs in the body of } \gamma) \\ & = \sigma(v(K_i\vartheta_i)) & (A = K_i\vartheta_i) \\ & > \sigma(v(L_{jp})) & (\gamma_i \text{ is locally stratified w.r.t. } \sigma) \end{array}$$

In Case (d) we have:

$$\begin{array}{lll} \sigma(v(H)) & \geq \sigma(v(A)) + 1 & (\gamma \text{ is locally stratified w.r.t. } \sigma \text{ and } \neg A \text{ occurs in the body of } \gamma) \\ & = \sigma(v(K_i\vartheta_i)) + 1 & (A = K_i\vartheta_i) \\ & \geq \sigma(v(L_{jp})) + 1 & (\gamma_i \text{ is locally stratified w.r.t. } \sigma) \end{array}$$

Thus, η_j is locally stratified w.r.t. σ. Hence, P_{k+1} is locally stratified w.r.t. σ.

Case 5. Program P_{k+1} is derived by subsumption (rule R5). P_{k+1} is locally stratified w.r.t. σ by the inductive hypothesis because $P_{k+1} \subseteq P_k$.

Case 6. Program P_{k+1} is derived by positive folding (rule R6). We have that $P_{k+1} = (P_k - \{\gamma\}) \cup \{\eta\}$, where η is a clause of the form $H \leftarrow B_L \wedge K\vartheta \wedge B_R$ derived by positive folding of clause γ of the form $H \leftarrow B_L \wedge B\vartheta \wedge B_R$ using a clause δ of the form $K \leftarrow B \in \mathit{Defs}_k$. We have to show that η is locally stratified w.r.t. σ, that is, for every valuation v, $\sigma(v(H)) \geq \sigma(v(K)\vartheta)$.

Take any valuation v. By the inductive hypothesis, since γ is locally stratified w.r.t. σ, we have that: (α) for every literal L occurring in $B_L \wedge B\vartheta \wedge B_R$, we have $\sigma(v(H)) \geq \sigma(v(L))$.

By the applicability conditions of rule R6, clause δ is the unique clause defining the predicate of its head and, by the hypothesis that the transformation sequence is admissible, this definition is σ-tight (see Point (2) of Definition 4.5). Thus, for every valuation v', we have that: (1) for every L in B, $\sigma(v'(K)) \geq \sigma(v'(L))$, and (2) there exists an atom A in B such that $\sigma(v'(K)) = \sigma(v'(A))$.

Let the valuation v' be defined as follows: for every variable X, $v'(X) = v(X\vartheta)$. Then, we have that: (β.1) for every L in B, $\sigma(v(K\vartheta)) \geq \sigma(v(L\vartheta))$, and ($\beta$.2) there exists an atom A in B such

that $\sigma(v(K\vartheta)) = \sigma(v(A\vartheta))$. Thus, from (α), $(\beta.1)$, and $(\beta.2)$, we get that $\sigma(v(H)) \geq \sigma(v(K\vartheta))$. Hence, η is locally stratified w.r.t. σ.

Case 7. Program P_{k+1} is derived by negative folding (rule R7). We have that $P_{k+1} = (P_k - \{\gamma\}) \cup \{\eta\}$ and, by the hypothesis that the transformation sequence is admissible, η is locally stratified w.r.t. σ (see Point (3) of Definition 4.5). ∎

In the rest of this Appendix we will consider:
(i) a local stratification $\sigma : \mathcal{B}_\omega \to W$,
(ii) an ω-program P_0 which is locally stratified w.r.t. σ, and
(iii) an admissible transformation sequence $P_0, \ldots, P_n$.

Definition A.1 (Old and New Predicates, Old and New Literals) *Each predicate occurring in P_0 is called an* old predicate *and each predicate introduced by rule* R1 *is called a* new predicate. *An* old literal *is a literal with an old predicate. A* new literal *is a literal with a new predicate.*

Thus, the new predicates are the ones which occur in the heads of the clauses of $Defs_n$.

Without loss of generality, we will assume that the admissible transformation sequence $P_0, \ldots, P_n$ is of the form $P_0, \ldots, P_d, \ldots, P_n$, with $0 \leq d \leq n$, where:

(1) the sequence $P_0, \ldots, P_d$, with $d \geq 0$, is constructed by applying d times the definition introduction rule, and
(2) the sequence $P_d, \ldots, P_n$, is constructed by applying any rule, except the definition introduction rule R1.

Thus, $P_d = P_0 \cup Defs_n$. In order to prove the correctness of the admissible transformation sequence $P_0, \ldots, P_n$ (see Proposition A.16 below) we will show that $M(P_d) = M(P_n)$. In order to prove Proposition A.16, we introduce the notion of a *proof tree* which is the proof-theoretic counterpart of the perfect model semantics (see Theorem A.4 below). A proof tree for an atom $A \in \mathcal{B}_\omega$ and a locally stratified ω-program P is constructed by transfinite induction as indicated in the following definition.

Definition A.2 (Proof Tree for Atoms and Negated Atoms) *Let A be an atom in $\mathcal{B}_\omega$, let P be a locally stratified ω-program, and let σ be a local stratification for P. Let $PT_{<A}$ denote the set of proof trees for H and P, where $H \in \mathcal{B}_\omega$ and $\sigma(H) < \sigma(A)$.*
A proof tree *for A and P is a finite tree T such that*:
(i) *the root of T is labeled by A,*
(ii) *a node N of T has children labeled by $L_1, \ldots, L_r$ iff N is labeled by an atom $H \in \mathcal{B}_\omega$ and there exist a clause $\gamma \in P$ and a valuation v such that $v(\gamma)$ is $H \leftarrow L_1 \wedge \ldots \wedge L_r$, and*
(iii) *every leaf of T is either labeled by the empty conjunction true or by a negated atom $\neg H$, with $H \in \mathcal{B}_\omega$, such that there is no proof tree for H and P in $PT_{<A}$.*

Let A be an atom in $\mathcal{B}_\omega$ and P be a locally stratified ω-program.
A proof tree *for $\neg A$ and P exists iff there are no proof trees for A and P. There exists at most one proof tree for $\neg A$ and P and, when it exists, it consists of the single root node labeled by $\neg A$.*

Remark A.3. (i) *For any $A \in \mathcal{B}_\omega$ if there is a proof tree for A and P, then there is no proof tree for $\neg A$ and P.*
(ii) *In any proof tree if a node H is an ancestor of a node A then $\sigma(H) \geq \sigma(A)$.*

The following theorem, whose proof is omitted, shows that proof trees can be used for defining a semantics equivalent to the perfect model semantics.

Theorem A.4 (Proof Tree and Perfect Model) *Let P be a locally stratified ω-program. For every $A \in \mathcal{B}_\omega$, there exists a proof tree for A and P iff $A \in M(P)$.*

In order to show that $M(P_d) = M(P_n)$, we will use Theorem A.4 and we will show that, given any atom $A \in \mathcal{B}_\omega$, there exists a proof tree for A and P_d iff there exists a proof tree for A and P_n.

In the following, we will use suitable *measures* which we now introduce.

Definition A.5 (Three Measures: *size*, *weight*, μ) (i) *For any proof tree T, size(T) denotes the number of nodes in T labeled by atoms in $\mathcal{B}_\omega$.*

(ii) *For any atom $A \in \mathcal{B}_\omega$, the ordinal $\sigma(A)$ is said to be the* stratum *of A.*
For any ordinal $\alpha \in W$, for any proof tree T, weight(α, T) is the number of nodes of T whose label is an atom with stratum α. (Recall that true, that is, the empty conjunction of literals, is not an atom.)

(iii) *For any atom $A \in \mathcal{B}_\omega$, we define*:
$$\textit{min-weight}(A) =_{def} \min\{\textit{weight}(\alpha, T) \mid \sigma(A){=}\alpha \text{ and } T \text{ is a proof tree for } A \text{ and } P_d\}.$$

(iv) *For any atom $A \in \mathcal{B}_\omega$ such that there exists at least a proof tree for A and P_d, we define*:
$$\mu(A) =_{def} \langle \sigma(A), \textit{min-weight}(A) \rangle \quad \textit{if } A \textit{ is an old atom}$$
$$\mu(A) =_{def} \langle \sigma(A), \textit{min-weight}(A) - 1 \rangle \quad \textit{if } A \textit{ is a new atom}$$

(v) *For any atom $A \in \mathcal{B}_\omega$ such that there exists no proof tree for A and P_d, we define*:
$$\mu(\neg A) =_{def} \langle \sigma(A), 0 \rangle.$$

Remark A.6. (i) *If A is an old atom then min-weight$(A){>}0$ else min-weight$(A){\geq}0$.*
(ii) *For any atom $A \in \mathcal{B}_\omega$, $\mu(A)$ is undefined if there is no proof tree for A and P_d.*

Now we extend μ to conjunctions of literals. First, we introduce the binary operation $\oplus : (W \times \mathbb{N})^2 \to (W \times \mathbb{N})$, where W is the set of countable ordinals and $\mathbb{N}$ is the set of natural numbers, defined as follows:

$$\langle \alpha_1, m_1 \rangle \oplus \langle \alpha_2, m_2 \rangle = \begin{cases} \langle \alpha_1,\, m_1 \rangle & \textit{if } \alpha_1 > \alpha_2 \\ \langle \alpha_1,\, m_1{+}m_2 \rangle & \textit{if } \alpha_1 = \alpha_2 \\ \langle \alpha_2,\, m_2 \rangle & \textit{if } \alpha_1 < \alpha_2 \end{cases}$$

or equivalently,

$$\langle \alpha_1, m_1 \rangle \oplus \langle \alpha_2, m_2 \rangle = \langle \max(\alpha_1, \alpha_2),\ \textit{if } \alpha_1{=}\alpha_2 \textit{ then } m_1{+}m_2 \textit{ else } (\textit{if } \alpha_1{>}\alpha_2 \textit{ then } m_1 \textit{ else } m_2) \rangle$$

Given a conjunction of literals $L_1 \wedge \ldots \wedge L_r$ such that, for $i = 1, \ldots, r$, with $r \geq 1$, there is a proof tree for L_i and P_d, we define:

$$\mu(L_1 \wedge \ldots \wedge L_r) =_{def} \mu(L_1) \oplus \cdots \oplus \mu(L_r)$$

For *true*, which is the empty conjunction of literals, we define:

$$\mu(\textit{true}) =_{def} \langle 0, 0 \rangle$$

Note that the definition of $\mu(\textit{true})$ is consistent with the fact that *true* is the neutral element for $\wedge$ and, thus, $\mu(\textit{true})$ should be the neutral element for $\oplus$, which is $\langle 0, 0 \rangle$.

The following lemma follows from the definition of the measure μ. Recall that a new predicate can only be defined in terms of old predicates.

Lemma A.7 (Properties of μ for a Definition in P_d) *Let $\delta \in P_d$ be a σ-tight clause introduced by the definition rule* R1 *with $m=1$, that is, δ is the only clause defining the head predicate of δ in P_d. Let v be a valuation and $v(\delta)$ be of the form: $K \leftarrow L_1 \wedge \ldots \wedge L_q$. We have that: $\mu(K) = \mu(L_1) \oplus \ldots \oplus \mu(L_q)$.*

Proof. Without loss of generality, we may assume that L_1 is an atom and $\sigma(K) = \sigma(L_1)$ because δ is σ-tight and, thus, L_1 is σ-maximal. We have that:
Thus

$\mu(K) = \langle \sigma(K),$ *min-weight*$(K) - 1 \rangle$.

Now, *min-weight*$(K) = \{$by definition of *min-weight*$\} =$

$= \min\{$*weight*$(\sigma(K), T_K)\}$ where T_K is a proof tree for K and $P_d =$

$= \{$by definition of *weight* (see also Figure 2)$\} =$

$= \big(\min \sum_{i=1,\ldots,q}$ *weight*$(\sigma(K), T_i)\big) + 1$ where for $i=1,\ldots,q$, T_i is a proof tree for L_i and $P_d =$

$= \{$by definition of *weight* and Remark A.3$\} =$

$= \big(\min \sum_{i=1,\ldots,q \ \wedge \ \sigma(L_i)=\sigma(K)}$ *weight*$(\sigma(K), T_i)\big) + 1 = \{$by $\min \sum = \sum \min\} =$

$= \big(\sum_{i=1,\ldots,q \ \wedge \ \sigma(L_i)=\sigma(K)}$*min-weight*$(L_i)\big) + 1 = \{$by σ-tightness$\} =$

$= \big(\sum_{i=1,\ldots,q \ \wedge \ \sigma(L_i)=\sigma(L_1)}$*min-weight*$(L_i)\big) + 1.$

Thus,

$\mu(K) = \langle \sigma(L_1), \ \sum_{i=1,\ldots,q \ \wedge \ \sigma(L_i)=\sigma(L_1)}$*min-weight*$(L_i) \rangle = \{$by definition of $\oplus\} =$

$= \mu(L_1) \oplus \ldots \oplus \mu(L_q)$. ∎

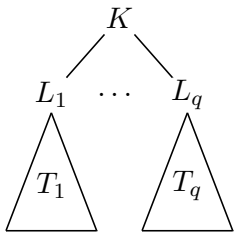


Figure 2: A proof tree for K and P_d. There is a valuation v and a clause $\delta \in P_d$ such that $v(\delta)$ is of the form: $K \leftarrow L_1 \wedge \ldots \wedge L_q$. For $i = 1, \ldots, q$, T_i is a proof tree for L_i and P_d.

Let $>$ denote the usual greater-than relation on $\mathbb{N}$. Let $>_{lex}$ denote the lexicographic ordering over $W \times \mathbb{N}$.

Let π_1 and π_2 denote, respectively, the first and second projection function on pairs. Given a pair $A = \langle a, b \rangle$ by A_1 we denote a and by A_2 we denote b.

Lemma A.8 (Properties of $\oplus$) (i) $\oplus$ *is an associative, commutative binary operator.*
(ii) *For every $A, B, C \in W \times \mathbb{N}$ and $\mathcal{R} \in \{\geq_{lex}, >_{lex}\}$, we have that:*

(ii.1)	$A \oplus B \geq_{lex} A$	
(ii.2)	*if* $A \geq_{lex} B$	*then* $A \oplus C \geq_{lex} B \oplus C$
(ii.3)	*if* $A >_{lex} B$, $A_1 \geq C_1$, *and* $A_2 > 0$	*then* $A \oplus C >_{lex} B \oplus C$
(ii.4)	*if* $A \ \mathcal{R} \ B$ *and* $A_1 > C_1$	*then* $A \ \mathcal{R} \ B \oplus C$
(ii.5)	*if* $A \ \mathcal{R} \ B \oplus C$	*then* $A \ \mathcal{R} \ B$ *and* $A \ \mathcal{R} \ C$

Proof. (i) It follows immediately from the definition.

(ii.1) By cases. If $A_1 > B_1$ then $A \oplus B = A \geq_{lex} A$. If $A_1 = B_1$ then $A \oplus B = \langle A_1, A_2 + B_2 \rangle \geq_{lex} A$. If $B_1 > A_1$ then $A \oplus B = B >_{lex} A$.

(ii.2) Let us consider the following two pairs:

$(\alpha) =_{def} A \oplus C = \langle \max(A_1, C_1),$ *if* $A_1 = C_1$ *then* $A_2 + C_2$ *else if* $A_1 > C_1$ *then* A_2 *else* $C_2 \rangle$

and

$(\beta) =_{def} B \oplus C = \langle \max(B_1, C_1),$ *if* $B_1 = C_1$ *then* $B_2 + C_2$ *else if* $B_1 > C_1$ *then* B_2 *else* $C_2 \rangle$.

We have to show that $(\alpha) \geq_{lex} (\beta)$.

Since $A \geq_{lex} B$, there are two cases. Case (1): $A = B$, and Case (2): $A >_{lex} B$. Case (2) consists of two subcases: Case (2.1): $A_1 > B_1$, and Case (2.2): $A_1 = B_1$ and $A_2 > B_2$.

In Case (1) we have that $(\alpha) = (\beta)$. Thus, we get $(\alpha) \geq_{lex} (\beta)$ as desired.

In Case (2.1) we consider two subcases: Case (2.1.1): $A_1 > B_1$ and $B_1 \geq C_1$, and Case (2.1.2): $A_1 > B_1$ and $B_1 < C_1$.

In Case (2.1.1) we have that $\max(A_1, C_1) > \max(B_1, C_1)$ and thus, we get that $(\alpha) \geq_{lex} (\beta)$.

In Case (2.1.2) (β) reduces to $\langle C_1, C_2 \rangle$ and, since $A_1 > B_1 \geq C_1$, we get that $(\alpha) \geq_{lex} (\beta)$.

In Case (2.2) since $A_1 = B_1$, (β) reduces to

$\langle \max(A_1, C_1),$ *if* $A_1 = C_1$ *then* $B_2 + C_2$ *else if* $A_1 > C_1$ *then* B_2 *else* $C_2 \rangle$

and, since $A_2 > B_2$, we get that $(\alpha) \geq_{lex} (\beta)$.

(ii.3) Let us consider again the two pairs:

$(\alpha) =_{def} A \oplus C = \langle \max(A_1, C_1),$ *if* $A_1 = C_1$ *then* $A_2 + C_2$ *else if* $A_1 > C_1$ *then* A_2 *else* $C_2 \rangle$

and

$(\beta) =_{def} B \oplus C = \langle \max(B_1, C_1),$ *if* $B_1 = C_1$ *then* $B_2 + C_2$ *else if* $B_1 > C_1$ *then* B_2 *else* $C_2 \rangle$.

We have to show $(\alpha) >_{lex} (\beta)$.

Since $A >_{lex} B$ there are two cases. Case (1): $A_1 > B_1$ and $A_1 \geq C_1$ and $A_2 > 0$. Case (2): $A_1 = B_1$ and $A_2 > B_2$ and $A_1 \geq C_1$ and $A_2 > 0$.

For Case (1) we consider two subcases: Case (1.1) $A_1 = C_1$ and Case (1.2) $A_1 > C_1$.

In Case (1.1) we have that (α) reduces to $\langle C_1, A_2 + C_2 \rangle$ and

(β) reduces to $\langle C_1,$ *if* $B_1 = C_1$ *then* $B_2 + C_2$ *else if* $B_1 > C_1$ *then* B_2 *else* $C_2 \rangle$

and since $A_1 > B_1$ and $A_1 = C_1$, we get that (β) further reduces to $\langle C_1, C_2 \rangle$ and, since $A_2 > 0$, we get that $(\alpha) >_{lex} (\beta)$.

In Case (1.2) we have that (α) reduces to $\langle A_1, \ldots \rangle$ and (β) reduces to $\langle \max(B_1, C_1), \ldots \rangle$, and since $A_1 > B_1$ and $A_1 > C_1$ we get that $(\alpha) >_{lex} (\beta)$.

For Case (2) we consider two subcases: Case (2.1) $A_1 = B_1 = C_1$ and Case (2.2) $A_1 = B_1 > C_1$.

In Case (2.1) we have that (α) reduces to $\langle A_1, A_2 + C_2 \rangle$ and (β) reduces to $\langle A_1, B_2 + C_2 \rangle$, and since in Case (2) we have that $A_2 > B_2$, we get that $(\alpha) >_{lex} (\beta)$.

In Case (2.2) we have that (α) reduces to $\langle A_1, A_2 \rangle$ and (β) reduces to $\langle B_1, B_2 \rangle$, and since $A_1 = B_1$ and in Case (2) we have that $A_2 > B_2$, we get that $(\alpha) >_{lex} (\beta)$.

(ii.4) We have that:

$B \oplus C = \langle \max(B_1, C_1),$ *if* $B_1 = C_1$ *then* $B_2 + C_2$ *else if* $B_1 > C_1$ *then* B_2 *else* $C_2 \rangle$

We reason by cases. Case (1): we assume $A = B$ and $A_1 > C_1$ and we show $A \geq_{lex} B \oplus C$. Case (2): we assume $A >_{lex} B$ and $A_1 > C_1$ and we show $A >_{lex} B \oplus C$.

Case (1). Since $A = B$, from $A_1 > C_1$ we get that $B_1 > C_1$ and thus, $B \oplus C = B$. Thus, $A \geq_{lex} B \oplus C$.

Case (2). There are two subcases: (2.1) $A_1 > B_1$ and $A_1 > C_1$, and (2.2) ($A_1 = B_1$ and $A_2 > B_2$) and $A_1 > C_1$.

Case (2.1). We have that: $A_1 > \max(B_1, C_1)$ and thus, $A \geq_{lex} B \oplus C$.
Case (2.2). Since $A_1 = B_1$ and $A_1 > C_1$, we have that: $B \oplus C = \langle B_1, B_2 \rangle =_{def} B$. Since $A_1 = B_1$ and $A_2 > B_2$ we get $A >_{lex} B$, and thus, $A >_{lex} B \oplus C$.

(ii.5) We have that:
$B \oplus C = \langle \max(B_1, C_1),$ *if* $B_1 = C_1$ *then* $B_2 + C_2$ *else if* $B_1 > C_1$ *then* B_2 *else* $C_2 \rangle$
We reason by cases: Case (1) $A = B \oplus C$, and Case (2) $A >_{lex} B \oplus C$. In order to show Point (ii.5) in Case (1) we have to show $A \geq_{lex} B$ and $A \geq_{lex} C$, and in Case (2) we have to show $A >_{lex} B$ and $A >_{lex} C$.
Case (1) Assume $A = B \oplus C$.
Case (1.1): $B_1 = C_1$. Thus, $A_1 = B_1 = C_1$ and $A_2 = B_2 + C_2$. Thus, $A \geq_{lex} B$ and $A \geq_{lex} C$.
Case (1.2): $B_1 > C_1$. Thus, $A_1 = B_1$ and $A_2 = B_2$. Thus, $A \geq_{lex} B$ and $A \geq_{lex} C$.
Case (1.3): $B_1 < C_1$. Like Case (1.2), by interchanging B and C.
Case (2) Assume $A >_{lex} B \oplus C$.
Case (2.1): $A_1 > \max(B_1, C_1)$. We get: $A >_{lex} B$ and $A >_{lex} C$.
Case (2.2): $A_1 = \max(B_1, C_1)$.
Case (2.2.1): $B_1 = C_1$. We have: $A_1 = B_1 = C_1$ and, since $A >_{lex} B \oplus C$, we have: $A_2 > B_2 + C_2$. Thus, we get $A >_{lex} B$ and $A >_{lex} C$.
Case (2.2.2): $B_1 > C_1$. Thus, $A_1 = \max(B_1, C_1) = B_1$. Since $A >_{lex} B \oplus C$ and $A_1 = \pi_1(B \oplus C)$, we have: $A_2 > \pi_2(B \oplus C)$, that is,
$A_2 >$ *if* $B_1 = C_1$ *then* $B_2 + C_2$ *else if* $B_1 > C_1$ *then* B_2 *else* C_2, that is,
$A_2 > B_2$.
Thus, we get $A >_{lex} B$ and, since $B_1 > C_1$, we also get $A >_{lex} C$.
Case (2.2.3): $B_1 < C_1$. Like Case (2.2.2), by interchanging B and C. ∎

Notation A.9. By $\overline{L}$ we will denote the negative literal $\neg L$, if L is a positive literal, and the positive literal A, if L is the negative literal $\neg A$.

Lemma A.10. *For all atoms $A \in \mathcal{B}_\omega$, literals $L_1, \ldots, L_m$, which are either atoms in $\mathcal{B}_\omega$ or negation of atoms in $\mathcal{B}_\omega$, if for $i = 1, \ldots, m$, $\sigma(A) \geq \sigma(L_i)$ then $\mu(\overline{A}) \geq_{lex} \mu(\overline{L}_1) \oplus \cdots \oplus \mu(\overline{L}_m)$.*

Proof. The proof is by induction on m by recalling that the $\oplus$ is associative and commutative. We do the induction step. The base case can be proved similarly to Cases (1) and (2.1) below.

We assume that $\mu(\overline{A}) \geq_{lex} \mu(\overline{L}_1) \oplus \cdots \oplus \mu(\overline{L}_j)$, for some $j \geq 1$, and we show that $\mu(\overline{A}) \geq_{lex} \mu(\overline{L}_1) \oplus \cdots \oplus \mu(\overline{L}_j) \oplus \mu(\overline{L}_{j+1})$.

By definition, $\mu(\overline{A}) = \langle \sigma(A), 0 \rangle$. Let $\mu(\overline{L}_1) \oplus \cdots \oplus \mu(\overline{L}_j) = \langle \beta, w_1 \rangle$, for some $\beta \in W$ and $w_1 \in \mathbb{N}$. Thus, the induction hypothesis can be stated as follows: $\langle \sigma(A), 0 \rangle \geq_{lex} \langle \beta, w_1 \rangle$.

We have the following two cases.

Case (1). Assume that $\overline{L}_{j+1}$ is a positive literal, say B. Let $\mu(B)$ be $\langle \sigma(B), w_2 \rangle$, for some $w_2 \in W$. Since $\sigma(A) \geq \sigma(L_{j+1}) > \sigma(B)$, by Lemma A.8 (ii.4) we get that $\mu(\overline{A}) \geq_{lex} \mu(\overline{L}_1) \oplus \cdots \oplus \mu(\overline{L}_j) \oplus \mu(B)$.

Case (2). Assume that $\overline{L}_{j+1}$ is a negative literal, say $\neg B$. Let $\mu(\neg B)$ be $\langle \sigma(B), 0 \rangle$. By hypothesis, we have $\sigma(A) \geq \sigma(L_{j+1}) = \sigma(B)$. We have three subcases.
Case (2.1). $\sigma(B) > \beta$. By induction hypothesis we have that $\langle \sigma(A), 0 \rangle \geq_{lex} \langle \beta, w_1 \rangle$. We also have that $\langle \beta, w_1 \rangle \oplus \langle \sigma(B), 0 \rangle = \langle \sigma(B), 0 \rangle$ and $\langle \sigma(A), 0 \rangle \geq_{lex} \langle \sigma(B), 0 \rangle$. Thus, we get $\langle \sigma(A), 0 \rangle \geq_{lex} \langle \beta, w_1 \rangle \oplus \langle \sigma(B), 0 \rangle$.
Case (2.2). $\sigma(B) = \beta$. By induction hypothesis we have that $\langle \sigma(A), 0 \rangle \geq_{lex} \langle \beta, w_1 \rangle$. We also have that $\langle \beta, w_1 \rangle \oplus \langle \sigma(B), 0 \rangle = \langle \beta, w_1 \rangle$. Thus, we get $\langle \sigma(A), 0 \rangle \geq_{lex} \langle \beta, w_1 \rangle \oplus \langle \sigma(B), 0 \rangle$.

Case (2.3). $\sigma(B) < \beta$. As Case (2.2). ∎

Now we introduce the notion of a *μ-consistent* proof tree which will be used in Proposition A.16 below. This notion is a generalization of the one of a *rank-consistent* proof tree introduced in [26].

Definition A.11 (σ-max Derived Clause) *We say that a clause γ in a program P_k of the sequence $P_d, \ldots, P_n$ is a* σ-max derived *clause if γ is a descendant of a clause β in P_j, with $d < j \leq k$, such that β has been derived by unfolding a clause α in P_{j-1} w.r.t. an old σ-maximal atom. (Recall that, by definition, a clause is a descendant of itself.)*

Definition A.12 (μ-consistent Proof Tree) *Let A be an atom in $\mathcal{B}_\omega$ and P_k be a program in the transformation sequence $P_d, \ldots, P_n$. We say that a proof tree T for A and P_k is* μ-consistent *if for all atoms H, all literals $L_1, \ldots, L_r$ which are the children of H in T, where $H \leftarrow L_1 \wedge \ldots \wedge L_r$ is a clause $v(\gamma)$ for some valuation v and some clause $\gamma \in P_k$, we have that:*
if *H has a new predicate and γ is not σ-max derived* then $\mu(H) \geq_{lex} \mu(L_1) \oplus \cdots \oplus \mu(L_r)$
else $\mu(H) >_{lex} \mu(L_1) \oplus \cdots \oplus \mu(L_r)$.

The proof tree for the negated atom $\neg A$ and P_k, if any, is μ-consistent. (Recall that this proof tree, if it exists, consists of the single root node labeled by $\neg A$.)

Let us consider the following ordering on $\mathcal{B}_\omega$ which will be used in the proof of Proposition A.16.

Definition A.13 (Ordering $\succ$) *Given any two atoms $A_1, A_2 \in \mathcal{B}_\omega$, we write $A_1 \succ A_2$ if either*
(i) *$\mu(A_1) >_{lex} \mu(A_2)$, or*
(ii) *$\mu(A_1) = \mu(A_2)$ and A_1 is a new atom and A_2 is an old atom.*
By abuse of notation, given any two atoms $A_1, A_2 \in \mathcal{B}_\omega$, we write $A_1 \succ \neg A_2$ if $\sigma(A_1) > \sigma(A_2)$ (that is, $\sigma(A_1) \geq \sigma(A_2)$).

We have that $\succ$ is a well-founded ordering on B_ω.

Lemma A.14. *Let T be a μ-consistent proof tree for an atom A and a program P. Then, for every atom B and literal L which is a child of B in T, we have $B \succ L$.*

Proof. Let $L_1, \ldots, L_r$ be the children of B in T, for some $\gamma \in P$ and valuation v such that $v(\gamma)$ is $B \leftarrow L_1 \wedge \ldots \wedge L_r$, and let L be the literal L_i. If L_i is the negated atom $\neg A_i$ then, since P is locally stratified w.r.t. σ, we have $\sigma(B) > \sigma(A_i)$ and $B \succ L_i$. Let us now consider the case where L_i is positive.

If the predicate of B is old then, by μ-consistency of T, $\mu(B) >_{lex} \mu(L_1) \oplus \cdots \oplus \mu(L_r)$. By Lemma A.8 (ii.1), $\mu(L_1) \oplus \cdots \oplus \mu(L_r) \geq_{lex} \mu(L_i)$ and, thus, $\mu(B) >_{lex} \mu(L_i)$. By definition of $\succ$, we have that $B \succ L_i$.

If the predicate of B is new and γ is σ-max derived then, by μ-consistency of T, $\mu(B) >_{lex} \mu(L_i)$ and, thus, $B \succ L_i$.

Finally, if the predicate of B is new and γ is *not* σ-max derived then γ is a descendant of a clause that has not been derived by folding and, thus, the predicate of L_i is old. By μ-consistency, $\mu(B) \geq_{lex} \mu(L_i)$ and, since the predicate of B is new and the one of L_i is old, we have $B \succ L_i$. ∎

Lemma A.15. *Consider the locally stratified ω-program P_d of the admissible transformation sequence $P_0, \ldots, P_d, \ldots, P_n$, where: (1) $P_0, \ldots, P_d$ is constructed by using rule* (R1)*, and (2) $P_d, \ldots, P_n$ is constructed by applying rules* (R2)–(R7)*. If there exists a proof tree for A and P_d then there exists a μ-consistent proof tree for A and P_d.*

Proof. Let us consider a proof tree T for A and P_d such that *min-weight*(A) = *weight*$(\sigma(A), T)$. We want to show that T is μ-consistent. That tree T can be depicted as in Figure 3. That tree has been constructed by using at the top the clause γ and a valuation v such that $v(\gamma)$ is of the form $A \leftarrow L_1 \wedge \ldots \wedge L_n$.

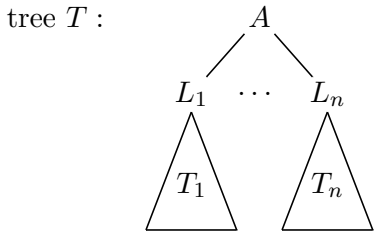


Figure 3: A proof tree T for A and P_d such that *min-weight*(A) = *weight*$(\sigma(A), T)$. There is a valuation v and a clause $\gamma \in P_d$ such that $v(\gamma)$ is of the form: $A \leftarrow L_1 \wedge \ldots \wedge L_n$. For $i = 1, \ldots, n$, T_i is a μ-consistent proof tree for L_i and P_d.

By induction on *size*(T), we may assume that $T_1, \ldots, T_n$ are μ-consistent proof trees. Since γ is locally stratified, we also have that for $i = 1, \ldots, n$, $\sigma(A) \geq \sigma(L_i)$.
(Recall that if L_i, for some $i \in \{1, \ldots, n\}$, is a negated atom, then T_i consists of the single node L_i and T_i is μ-consistent.)

In order to prove the lemma we have to show the following two points:
(P1) if A is a new atom then $\mu(A) \geq_{lex} \mu(L_1) \oplus \wedge \ldots \wedge \oplus \mu(L_n)$, and
(P2) if A is an old atom then $\mu(A) >_{lex} \mu(L_1) \oplus \wedge \ldots \wedge \oplus \mu(L_n)$.

(Note that $A \leftarrow L_1 \wedge \ldots \wedge L_n$ is not an instance of a σ-max derived clause belonging to P_d, because no such a clause exists in P_d and, thus, if Points (P1) and (P2) hold then the proof tree T is μ-consistent.)

Now, let us consider the following two cases: (1) A is a new atom, and (2) A is an old atom.

Case (1): A is a new atom. We have that:
$\mu(A) = \langle \sigma(A),\ \textit{min-weight}(A) - 1 \rangle = \langle \sigma(A), \sum_{i=1,\ldots,n \ \wedge \ \sigma(L_i)=\sigma(A)} \textit{min-weight}(L_i) \rangle$.

Now, we consider two subcases.
Case (1.1): for $i = 1, \ldots, n$, $\sigma(A) > \sigma(L_i)$. In this case we have that:
$\langle \sigma(A), \sum_{i=1,\ldots,n \ \wedge \ \sigma(L_i)=\sigma(A)} \textit{min-weight}(L_i) \rangle =$
$= \langle \sigma(A), 0 \rangle >_{lex} \mu(L_1) \oplus \ldots \oplus \mu(L_n)$.
This last inequality holds because $\pi_1(\mu(L_1) \oplus \ldots \oplus \mu(L_n)) = \max\{\sigma(L_i) \mid i = 1, \ldots, n\} < \sigma(A)$ because for $i = 1, \ldots, n$, $\sigma(A) > \sigma(L_i)$.
Case (1.2): there exists $i \in \{1, \ldots, n\}$ such that $\sigma(A) = \sigma(L_i)$. In this case we have that:
$\langle \sigma(A), \sum_{i=1,\ldots,n \ \wedge \ \sigma(L_i)=\sigma(A)} \textit{min-weight}(L_i) \rangle = \mu(L_1) \oplus \ldots \oplus \mu(L_n)$,
because $\mu(L_p) \oplus \mu(L_q) = \mu(L_p)$, whenever $\pi_1(\mu(L_p)) > \pi_1(\mu(L_q))$. This concludes the proof of Case (1) and of Point (P1).

Case (2): A is an old atom. We have that:
$\mu(A) = \langle \sigma(A),\ \textit{min-weight}(A) \rangle =$
$= \{\text{the proof tree } T \text{ for } A \text{ and } P_d \text{ is such that } \textit{min-weight}(A) = \textit{weight}(\sigma(A), T)\} =$
$= \langle \sigma(A),\ \big(\sum_{i=1,\ldots,n \ \wedge \ \sigma(L_i)=\sigma(A)} \textit{min-weight}(L_i) \big) + 1 \rangle$.
Let M be the subset of $\{1, \ldots, n\}$ such that for all $j \in M$, $\sigma(L_j) = \sigma(A)$. We have that:

$\langle \sigma(A),\ \big(\sum_{i=1,\ldots,n\ \wedge\ \sigma(L_i)=\sigma(A)}\ \textit{min-weight}(L_i)\big)+1\rangle =$
$= \langle \sigma(A),\ \big(\sum_{j\in M}\ \textit{min-weight}(L_j)\big)+1\rangle >_{lex} \mu(L_1)\oplus\ldots\oplus\mu(L_n)$.
This last inequality holds because $\sum_{j\in M}\ \textit{min-weight}(L_j) = \pi_2(\mu(L_1)\oplus\ldots\oplus\mu(L_n))$. This concludes the proof of Case (2), of Point (P2), and of the lemma. ∎

Proposition A.16. *Let P_0 be a locally stratified ω-program, σ be a local stratification for P_0, and $P_0,\ldots,P_d,\ldots,P_n$ be an admissible transformation sequence where*: (1) *$P_0,\ldots,P_d$ is constructed by using rule* (R1)*, and* (2) *$P_d,\ldots,P_n$ is constructed by applying rules* (R2)–(R7)*. Then, for every atom $A\in\mathcal{B}_\omega$, we have that, for $k=d,\ldots,n$:*

(*Soundness*) *if there exists a proof tree for A and P_k, then there exists a proof tree for A and P_d, and*

(*Completeness*) *if there exists a μ-consistent proof tree for A and P_d, then there exists a μ-consistent proof tree for A and P_k.*

Proof. We prove the (*Soundness*) and (*Completeness*) properties by induction on k.
Clearly they hold for $k=d$.
Now, let us assume, by induction, that:

(IndHyp) the (*Soundness*) and (*Completeness*) properties hold for k, with $d\leq k<n$.

We have to show that they hold for $k+1$.

In order to prove that the (*Soundness*) and (*Completeness*) properties hold for $k+1$, it is sufficient to prove that:

(S) for every atom $A\in\mathcal{B}_\omega$, if there exists a proof tree for A and P_{k+1} then there exists a proof tree for A and P_k, and
(C) for every atom $A\in\mathcal{B}_\omega$, if there exists a μ-consistent proof tree for A and P_k then there exists a μ-consistent proof tree for A and P_{k+1}.

We proceed by complete induction on the ordinal $\sigma(A)$ associated with the atom A. The inductive hypotheses (IS) and (IC) for (S) and (C), respectively, are as follows:

(IS) for every atom $A'\in\mathcal{B}_\omega$ such that $\sigma(A')<\sigma(A)$, if there exists a proof tree for A' and P_{k+1} then there exists a proof tree for A' and P_k,

and

(IC) for every atom $A'\in\mathcal{B}_\omega$ such that $\sigma(A')<\sigma(A)$, if there exists a μ-consistent proof tree for A' and P_k then there exists a μ-consistent proof tree for A' and P_{k+1}.

By the inductive hypotheses (IS) and (IC), we have that:

(ISC) for every atom $A'\in\mathcal{B}_\omega$ such that $\sigma(A')<\sigma(A)$ (and thus, $A\succ A'$), there exists a proof tree T' for A' and P_k iff there exists a proof tree U' for A' and P_{k+1}.

Proof of (S). Given a proof tree U for A and P_{k+1} we have to prove that there exists a proof tree T for A and P_k. The proof is by complete induction on $\textit{size}(U)$. The inductive hypothesis is:

(Isize) for every atom $A'\in\mathcal{B}_\omega$, for every proof tree U' for A' and P_{k+1}, if $\textit{size}(U')<\textit{size}(U)$ then there exists a proof tree T' for A' and P_k.

Let η be a clause in P_{k+1} and v be a valuation. Let $v(\eta)$ be a clause of the form $A \leftarrow L_1 \wedge \ldots \wedge L_r$ used at the root of U. We proceed by considering the following cases: *either* (Case 1) η belongs to P_k *or* (Case 2) η does not belong to P_k and it has been derived from a clause in P_k by applying a transformation rule among R2, R3, R4, R6, and R7. These two cases are mutually exclusive and exhaustive because rule R5 removes a clause.

We have that, for $i = 1, \ldots, r$, there is a proof tree T_i for L_i and P_k. Indeed, (i) if L_i is an atom then, by induction on (Isize), there exists a proof tree T_i for L_i and P_k, and (ii) if L_i is a negated atom $\neg A_i$ then, by the fact that program P_{k+1} is locally stratified w.r.t. σ and by the inductive hypothesis (ISC), there is no proof tree for A_i and P_k and hence, by definition, there is a proof tree T_i for L_i and P_k.

Case 1. A proof tree T for A and P_k can be constructed by using $v(\eta)$ and the proof trees $T_1, \ldots, T_r$ for $L_1, \ldots, L_r$, respectively, and P_k.

Case 2.1 (P_{k+1} is derived from P_k by using rule R2.) Clause η is derived by instantiating a variable X in a clause $\gamma \in P_k$. We have that γ is a clause of the form $\widetilde{A} \leftarrow \widetilde{L}_1 \wedge \ldots \wedge \widetilde{L}_r$ and η is of the form $(\widetilde{A} \leftarrow \widetilde{L}_1 \wedge \ldots \wedge \widetilde{L}_r)\{X/[\![s|X]\!]\}$ for some $s \in \Sigma$. Thus, $v(\widetilde{A}\{X/[\![s|X]\!]\}) = A$ and, for $i = 1, \ldots, r$, $v(\widetilde{L}_i\{X/[\![s|X]\!]\}) = L_i$.

Let v' be the valuation such that $v'(X) = v([\![s|X]\!])$ and $v'(Y) = v(Y)$ for every variable Y different from X. Then $v'(\gamma) = v(\eta)$ and a proof tree T for A and P_k can be constructed from $T_1, \ldots, T_r$ by using $v'(\gamma)$ at the root of T.

Case 2.2 (P_{k+1} is derived from P_k by using rule R3.) Clause η is derived by unfolding a clause $\gamma \in P_k$ w.r.t. a positive literal, say $\widetilde{K}$, in its body using clause γ_i. Recall that clauses γ and γ_i are assumed to have no variables in common (see rule R3). Without loss of generality, we may assume that: (i) η is of the form $(\widetilde{A} \leftarrow \widetilde{L}_1 \wedge \ldots \wedge \widetilde{L}_r)\vartheta_i$, (ii) γ is of the form $\widetilde{A} \leftarrow \widetilde{K} \wedge \widetilde{L}_{q+1} \wedge \ldots \wedge \widetilde{L}_r$, with $0 \leq q \leq r$, and (iii) γ_i is of the form $\widetilde{H} \leftarrow \widetilde{L}_1 \wedge \ldots \wedge \widetilde{L}_q$, where ϑ_i is an (idempotent and without identity bindings) most general unifier of $\widetilde{K}$ and $\widetilde{H}$.

Let v' be the valuation such that:(i) $v'(X) = v(X\vartheta_i)$ for every variable X in the domain of ϑ_i, and (ii) $v'(Y) = v(Y)$ for every variable Y not in the domain of ϑ_i. For this choice of v' we have that $v'(\widetilde{K}) = \{\text{by definition of } v'\} = v(\widetilde{K}\vartheta_i) = \{\text{since } \widetilde{K}\vartheta_i = \widetilde{H}\vartheta_i\} = v(\widetilde{H}\vartheta_i) = \{\text{by definition of } v'\} = v'(\widetilde{H})$.

For instance, given γ: $p(X) \leftarrow q(X,Y) \wedge s(X,Y,W)$ and γ_i: $q(Z,a) \leftarrow r(Z)$, by unfolding γ w.r.t. $q(X,Y)$ using γ_i, we get a most general unifier $\vartheta_i = \{Z/X, Y/a\}$ and the clause η: $p(X) \leftarrow r(X) \wedge s(X,a,W)$. Thus, if $v(\eta) = p(b) \leftarrow r(b) \wedge s(b,a,c)$, we have $v'(X) = b$, $v'(W) = c$, $v'(Z) = b$, and $v'(Y) = a$.

Now, since $v'(\widetilde{K}) = v'(\widetilde{H})$, given the proof trees $T_1, \ldots, T_r$ for $L_1, \ldots, L_r$, respectively, and P_k, we can construct a proof tree T for A and P_k as follows. Let K denote $v'(\widetilde{K})$. (i) We first construct a proof tree T_K for K and P_k from $T_1, \ldots, T_q$ by using clause $v'(\gamma_i)$ at the root of T_K, and then, (ii) we construct T from $T_K, T_{q+1}, \ldots, T_r$ by using clause $v'(\gamma)$ at the root of T.

Case 2.3 (P_{k+1} is derived from P_k by using rule R4.) Clause η is derived by unfolding a clause $\gamma \in P_k$ w.r.t. a negative literal, say $\neg\widetilde{K}$, in its body. Recall that we have assumed that $v(\eta)$ is of the form $A \leftarrow L_1 \wedge \ldots \wedge L_r$. Without loss of generality, we may assume that:
(i) there exist m substitutions $\vartheta_1, \ldots, \vartheta_m$ and m clauses $\gamma_1, \ldots, \gamma_m$ in P_k such that, for $i = 1, \ldots, m$, ϑ_i is a most general unifier of $\widetilde{K}$ and $hd(\gamma_i)$, $\widetilde{K} = hd(\gamma_i)\vartheta_i$, and $v(\gamma_i\vartheta_i)$ is of the form $K \leftarrow B_i$, and
(ii) $v(\gamma)$ is of the form $A \leftarrow \neg K \wedge L_{m+1} \wedge \ldots \wedge L_r$, with $0 \leq m \leq r$, (note that, by Condition (1) of rule R4, γ is not instantiated by the negative unfolding). Thus, $v(\eta) = A \leftarrow L_1 \wedge \ldots \wedge L_r$,

is derived from $A \leftarrow \neg(B_1 \vee \ldots \vee B_m) \wedge L_{m+1} \wedge \ldots \wedge L_r$ by pushing $\neg$ inside and pushing $\vee$ outside.

Now, let us assume by absurdum that there exists a proof tree U_K for K and P_{k+1}. Then, there exists a valuation v' such that the children of the root of U_K are labeled by the literals $M_1, \ldots, M_s$, where $v'(bd(\gamma_i\vartheta_i)) = M_1 \wedge \ldots \wedge M_s$, for some i, with $1 \leq i \leq m$. Since γ_i has no existential variables, without loss of generality we take $v'(X) = v(X)$, for every variable X. By the definition of the negative unfolding rule, there exist $j \in \{1, \ldots, s\}$ and $h \in \{1, \ldots, m\}$ such that $M_j = \overline{L}_h$. By hypothesis, there exists a proof tree for L_h and P_k and, thus, U_K is not a proof tree for K and P_{k+1}. This is a contradiction and, thus, we have that there is no proof tree for K and P_{k+1}. Since $\sigma(K) < \sigma(A)$, by the inductive hypothesis (ISC), we have that there is no proof tree for K and P_k. Hence, there is a proof tree $T_{\neg K}$ for $\neg K$ and P_k. Thus, we can construct a proof tree T for A and P_k from $T_{\neg K}, T_{m+1}, \ldots, T_r$ by using clause $v(\gamma)$ at the root of T.

Case 2.4 (P_{k+1} is derived from P_k by using rule R6.) Let us assume that clause η of the form $\widetilde{A} \leftarrow \widetilde{L}_1 \wedge \widetilde{L}_2 \wedge \ldots \wedge \widetilde{L}_r$ is derived by positive folding from a clause $\gamma \in P_k$ of the form $\widetilde{A} \leftarrow \widetilde{M}'_1 \wedge \ldots \wedge \widetilde{M}'_s \wedge \widetilde{L}_2 \wedge \ldots \wedge \widetilde{L}_r$ using a clause $\delta \in \mathit{Defs}_k$ of the form $\widetilde{K} \leftarrow \widetilde{M}_1 \wedge \ldots \wedge \widetilde{M}_s$. Without loss of generality, we may assume that $\widetilde{L}_1 = \widetilde{K}\vartheta$, where ϑ is a substitution such that, for $i = 1, \ldots, s$, $\widetilde{M}_i\vartheta = \widetilde{M}'_i$. Thus, the literal L_1 in the body of $v(\eta)$ is $v(\widetilde{K}\vartheta)$. We have that $\delta \in P_d$ and the definition of the head predicate of δ in P_d consists of clause δ only.

By induction on k, we have that the (*Soundness*) property holds for k. We know that there is a proof tree for L_1 and P_k. Hence, by Conditions (i) and (ii) of rule R6, there exists a proof tree for L_1 and P_d, for some valuation v' such that $v'(\delta)$ is of the form $L_1 \leftarrow M_1 \wedge \ldots \wedge M_s$ (note that if $X \in \mathit{vars}(\eta)$ then $v'(X) = v(X)$).

By induction on k, we have that the (*Soundness*) and (*Completeness*) properties hold for k. Thus, there are proof trees $U_1, \ldots, U_s$ for $M_1, \ldots, M_s$, respectively, and P_k.

Finally, by induction on (Isize), we know that there exist the proof trees $T_2, \ldots, T_r$ for $L_2, \ldots, L_r$, respectively, and P_k. As a consequence, we can construct a proof tree T for A and P_k from $U_1, \ldots, U_s, T_2, \ldots, T_r$ by using clause $v(\gamma)$ at the root of T.

Case 2.5 (P_{k+1} is derived from P_k by using rule R7.) Clause η is derived by negative folding from a clause $\gamma \in P_k$ using clauses $\delta_1, \ldots, \delta_m$ in Defs_k. Thus, we have that: (i) $v(\gamma)$ is of the form $A \leftarrow N_1 \wedge \ldots \wedge N_m \wedge L_2 \wedge \ldots \wedge L_r$, (ii) for $i = 1, \ldots, m$, $v(\delta_i)$ is of the form $K \leftarrow B_i$, where either N_i is a positive literal A_i and B_i is $\neg A_i$, or N_i is a negative literal $\neg A_i$ and B_i is A_i, and (iii) $v(\eta)$ is of the form $A \leftarrow \neg K \wedge L_2 \wedge \ldots \wedge L_r$. Thus, $L_1 = \neg K$.

By the inductive hypothesis (ISC), there exists a proof tree for L_1 and P_k and, since $L_1 = \neg K$, there is no proof tree for K and P_k. By induction on k, we have that the (*Completeness*) holds for k and, therefore, there exists no proof tree for K and P_d. We have that $\{\delta_1, \ldots, \delta_m\} \subseteq P_d$ and the clauses defining the head predicate of $\delta_1, \ldots, \delta_m$ in P_d are $\{\delta_1, \ldots, \delta_m\}$. Thus, there are no proof trees for $B_1, \ldots, B_m$ and P_d.

By induction on k, the (*Soundness*) property holds for k and, therefore, there are no proof trees for $B_1, \ldots, B_m$ and P_k. Thus, there are proof trees $U_1, \ldots, U_m$ for $N_1, \ldots, N_m$, respectively, and P_k. Finally, by induction on (Isize), we have that there are the proof trees $T_2, \ldots, T_r$ for $L_2, \ldots, L_r$, respectively, and P_k. We can construct a proof tree T for A and P_k from $U_1, \ldots, U_m, T_2, \ldots, T_r$ by using clause $v(\gamma)$ at the root of T.

Proof of (C). Given a μ-consistent proof tree T for A and P_k, we prove that there exists a μ-consistent proof tree U for A and P_{k+1}.

The proof is by well-founded induction on $\succ \subseteq \mathcal{B}_\omega \times \mathcal{B}_\omega$. The inductive hypothesis is:

(Iμ) for every atom $A' \in \mathcal{B}_\omega$ such that $A \succ A'$, if there exists a μ-consistent proof tree T' for A' and P_k then there exists a μ-consistent proof tree U' for A' and P_{k+1}.

Let γ be a clause in P_k and v be a valuation such that $v(\gamma)$ is the clause of the form $A \leftarrow L_1 \wedge \ldots \wedge L_r$ used at the root of T. We consider the following cases: *either* (Case 1) γ belongs to P_{k+1} *or* (Case 2) γ does not belong to P_{k+1} because it has been replaced by zero or more clauses derived by applying a transformation rule among R2–R7.

Case 1. By the μ-consistency of T and Lemma A.14, for $i = 1, \ldots, r$, we have $A \succ L_i$. Hence, by the inductive hypotheses (Iμ) and (ISC), there exists a μ-consistent proof tree U_i for L_i and P_{k+1}. A μ-consistent proof tree U for A and P_{k+1} is constructed by using $v(\gamma)$ at the root of U and the proof trees $U_1, \ldots, U_r$ for $L_1, \ldots, L_r$, respectively, and P_{k+1}.

Case 2.1 (P_{k+1} is derived from P_k by using rule R2.) Suppose that by instantiating a variable X of clause γ in P_k we derive clauses $\gamma_1, \ldots, \gamma_h$ in P_{k+1}. For $i = 1, \ldots, h$, γ_i is $\gamma\{X/[\![s_i|X]\!]\}$, with $s_i \in \Sigma$. Hence, there exist $i \in \{1, \ldots, h\}$ and a valuation v' such that $v(\gamma) = v'(\gamma_i)$. By the μ-consistency of T and Lemma A.14, for $i = 1, \ldots, r$, we have $A \succ L_i$. Hence, by the inductive hypotheses (Iμ) and (ISC), for $i = 1, \ldots, r$, there exists a μ-consistent proof tree U_i for L_i and P_{k+1}. A proof tree U for A and P_{k+1} is constructed by using $v'(\gamma_i)$ at the root of U and the proof trees $U_1, \ldots, U_r$ for $L_1, \ldots, L_r$, respectively, and P_{k+1}.

The proof tree U is μ-consistent because: (i) by (Iμ), we have that $U_1, \ldots, U_r$ are μ-consistent, (ii) γ_i is σ-max derived iff γ is σ-max derived, and (iii) since T is μ-consistent, we have that if γ is not σ-max derived then $\mu(A) \geq_{lex} \mu(L_1) \oplus \ldots \oplus \mu(L_r)$ else $\mu(A) >_{lex} \mu(L_1) \oplus \ldots \oplus \mu(L_r)$.

Case 2.2 (P_{k+1} is derived from P_k by using rule R3.) Suppose that by unfolding γ w.r.t. an atom B in its body we derive clauses $\eta_1, \ldots, \eta_m$ in P_{k+1}. Without loss of generality, we assume that B is the leftmost literal in the body of γ. Hence, there exists a clause γ_i in (a variant of) P_k such that: (i) $v(\gamma_i)$ is of the form $L_1 \leftarrow M_1 \wedge \ldots \wedge M_q$, (ii) $v(\eta_i)$ is $A \leftarrow M_1 \wedge \ldots \wedge M_q \wedge L_2 \wedge \ldots \wedge L_r$, and (iii) $v(\gamma_i)$ is the clause which is used for constructing the children of L_1 in T. By the μ-consistency of T and Lemma A.14, for $i = 1, \ldots, q$, we have $A \succ M_i$ and, for $i = 2, \ldots, r$, we have $A \succ L_i$. Hence, by the inductive hypotheses (Iμ) and (ISC), for $i = 1, \ldots, q$, there exists a μ-consistent proof tree V_i for M_i and P_{k+1} and, for $i = 2, \ldots, r$, there exists a μ-consistent proof tree U_i for L_i and P_{k+1}. A proof tree U for A and P_{k+1} is constructed by using $v(\eta_i)$ at the root of U and the proof trees $V_1, \ldots, V_q, U_2, \ldots, U_r$ for $M_1, \ldots, M_q, L_2, \ldots, L_r$, respectively, and P_{k+1}.

It remains to show that the proof tree U is μ-consistent. There are two cases: (a) and (b).
Case (a): in this first case we assume that A is new *and* η_i is not σ-max derived.
Since T is μ-consistent we get $\mu(A) \geq_{lex} \mu(L_1) \oplus \mu(L_2) \oplus \ldots \oplus \mu(L_r)$ and $\mu(L_1) \geq_{lex} \mu(M_1) \oplus \ldots \oplus \mu(M_q)$. By Lemma A.8 (ii.2), we get $\mu(A) \geq_{lex} \mu(M_1) \oplus \ldots \oplus \mu(M_q) \oplus \mu(L_2) \oplus \ldots \oplus \mu(L_r)$.
Case (b): in this second case, we assume that A is old *or* η_i is σ-max derived. We have two subcases (b.1) and (b.2).
Subcase (b.1): A is old. Since T is μ-consistent, we get that $\mu(A) >_{lex} \mu(L_1) \wedge \ldots \wedge \mu(L_r)$ and $\mu(L_1) \geq_{lex} \mu(M_1) \wedge \ldots \wedge \mu(M_q)$. By Lemma A.8 (ii.2) we get $\mu(A) >_{lex} \mu(M_1) \wedge \ldots \wedge \mu(M_q)$.

Subcase (b.2): η_i is σ-max derived. We may assume that A is new, because in Subcase (b.1) we have considered that A is old. Now we consider two subcases of this Subcase (b.2).
Subcase (b.2.1): η_i is σ-max derived, A is new, and γ is σ-max derived, and
Subcase (b.2.2): η_i is σ-max derived, A is new, and γ is not σ-max derived.

Subcase (b.2.1). Since T is μ-consistent we get $\mu(A) >_{lex} \mu(L_1) \oplus \mu(L_2) \oplus \ldots \oplus \mu(L_r)$ and $\mu(L_1) \geq_{lex} \mu(M_1) \oplus \ldots \oplus \mu(M_q)$. By Lemma A.8 (ii.2), we get $\mu(A) >_{lex} \mu(M_1) \oplus \ldots \oplus \mu(M_q) \oplus \mu(L_2) \oplus \ldots \oplus \mu(L_r)$.
Subcase (b.2.2). Since T is μ-consistent and L_1 is old, we get: (†1) $\mu(L_1) >_{lex} \mu(M_1) \oplus \ldots \oplus \mu(M_q)$, and (†2) $\pi_2(\mu(L_1)) > 0$. Since η_i is σ-maximal derived, we have that, for $i = 2, \ldots, r$, $\sigma(L_1) \geq \sigma(L_j)$. Thus, (†3) $\sigma(L_1) \geq \pi_1(\mu(L_2) \oplus \ldots \oplus \mu(L_r))$. From (†1), (†2), and (†3), by Lemma A.8 (ii.3), we get: (†4) $\mu(L_1) \oplus \mu(L_2) \oplus \ldots \oplus \mu(L_r) >_{lex} \mu(M_1) \oplus \ldots \oplus \mu(M_q) \oplus \mu(L_2) \oplus \ldots \oplus \mu(L_r)$. Since T is μ-consistent, we have that $\mu(A) \geq_{lex} \mu(L_1) \oplus \ldots \oplus \mu(L_r)$, and by (†4) we get: $\mu(A) >_{lex} \mu(M_1) \oplus \ldots \oplus \mu(M_q) \oplus \mu(L_2) \oplus \ldots \oplus \mu(L_r)$, as desired.

This concludes the proof that U is a μ-consistent proof tree.

Case 2.3 (P_{k+1} is derived from P_k by using rule R4.) Suppose that we unfold γ w.r.t. a negated atom in its body and we derive clauses $\eta_1, \ldots, \eta_s$ in P_{k+1}. Without loss of generality, we assume that we unfold γ w.r.t. the leftmost literal in its body. Let $\gamma_1, \ldots, \gamma_m$ be all clauses in (a variant of) P_k whose heads are unifiable with the leftmost literal in the body of γ. We may assume that, for $i = 1, \ldots, m$, $v(\gamma_i)$ is of the form $A_1 \leftarrow B_i$, where $L_1 = \neg A_1$ and B_i is a conjunction of literals. Since there is no proof tree for A_1 and P_k, for $i = 1, \ldots, m$, there exists a literal R_i in B_i such that there is no proof tree for R_i and P_k. By definition, there is a proof tree for $\overline{R}_i$ and P_k. Moreover, (i) $A \succ \neg A_1$ because by hypothesis the proof tree T is μ-consistent, and (ii) $\sigma(\neg A_1) \geq \sigma(\overline{R}_i)$, because P_k is locally stratified w.r.t. σ.

Now we have two cases: (i) R_i is an atom, and (ii) R_i is a negated atom, say $\neg C_i$. In Case (i) we have that $\sigma(A) > \sigma(A_1) \geq \sigma(R_i)$ and, thus, $A \succ \overline{R}_i$. In Case (ii) we have that $\sigma(A) > \sigma(A_1) \geq \sigma(\neg C_i)$ and, thus, $\sigma(A) > \sigma(C_i) = \sigma(\overline{R}_i)$ and $\mu(A) > \mu(\overline{R}_i)$. Hence, $A \succ \overline{R}_i$. Thus, in both cases $A \succ \overline{R}_i$.

Since $A \succ \overline{R}_i$, by the inductive hypotheses (Iμ) and (ISC), we have that, for $i = 1, \ldots, m$, there exists a μ-consistent proof tree V_i for $\overline{R}_i$ and P_{k+1}. By the μ-consistency of T, for $i = 2, \ldots, r$, there exists a μ-consistent proof tree U_i for L_i and P_{k+1}. By the definition of rule R4, there exists a clause η_p among the clauses $\eta_1, \ldots, \eta_s$ derived from γ, such that $v(\eta_p)$ is of the form $A \leftarrow \overline{R}_1 \wedge \ldots \wedge \overline{R}_m \wedge L_2 \wedge \ldots \wedge L_r$. (To see this, recall that by pushing $\neg$ inside and $\vee$ outside, from $\neg((C_1 \wedge C_2) \vee (D_1 \wedge D_2))$ we get $(\overline{C}_1 \wedge \overline{D}_1) \vee (\overline{C}_1 \wedge \overline{D}_2) \vee (\overline{C}_2 \wedge \overline{D}_1) \vee (\overline{C}_2 \wedge \overline{D}_2)$.)

A proof tree U for A and P_{k+1} is constructed by using $v(\eta_p)$ at the root of U and the proof trees $V_1, \ldots, V_m, U_2, \ldots, U_r$ for $\overline{R}_1, \ldots, \overline{R}_m, L_2, \ldots, L_r$, respectively, and P_{k+1}.

In order to show that U is μ-consistent we need to consider two cases. In the first case, we assume that A is old or η is σ-max derived. Thus, in this case, also γ is σ-max derived. By μ-consistency of T, we have $\mu(A) >_{lex} \mu(L_1) \oplus \cdots \oplus \mu(L_r)$. By local stratification of P_k and by Lemma A.10, $\mu(L_1) \geq_{lex} \mu(\overline{R}_1) \oplus \cdots \oplus \mu(\overline{R}_m)$. Therefore, by Lemma A.8 (ii.2), $\mu(A) >_{lex} \mu(\overline{R}_1) \oplus \cdots \oplus \mu(\overline{R}_m) \oplus \mu(L_2) \oplus \cdots \oplus \mu(L_r)$ and U is μ-consistent.

In the second case, A is new and η is not σ-max derived. As a consequence, also γ is not σ-max derived. By μ-consistency of T we have $\mu(A) \geq_{lex} \mu(L_1) \oplus \cdots \oplus \mu(L_r)$. By local stratification of P_k and by Lemma A.10, $\mu(L_1) \geq_{lex} \mu(\overline{R}_1) \oplus \cdots \oplus \mu(\overline{R}_m)$ and, by Lemma A.8 (ii.2), $\mu(A) \geq_{lex} \mu(\overline{R}_1) \oplus \cdots \oplus \mu(\overline{R}_m) \oplus \mu(L_2) \oplus \cdots \oplus \mu(L_r)$. Therefore, U is μ-consistent.

Case 2.4 (P_{k+1} is derived from P_k by using rule R5.) Suppose that the clause γ is removed from P_k by subsumption. Hence, there exists a clause γ_1 in $P_k - \{\gamma\}$ and a valuation v' such that $v'(\gamma_1)$ is of the form $A \leftarrow$. The clause γ_1 belongs to P_{k+1} and, therefore, a proof tree U for A and P_{k+1} can be constructed by using $v'(\gamma_1)$ at the root of U. The proof tree U consists of the root A with the single child *true*. Now we prove that the proof tree U is μ-consistent, that is, $\mu(A) >_{lex} \mu(true)$. We have to prove that $\mu(A) >_{lex} \langle 0, 0 \rangle$. We have the following three cases:

(a), (b.1), and (b.2).
Case (a). A is an old atom. In this case we have that $\mu(A) >_{lex} \langle 0,0\rangle$, because, as stated in Remark A.6, for any old atom B, we have that $\mathit{min\text{-}weight}(B) > 0$.
Case (b). A is a new atom. Since A is new, there is a valuation v' and a clause δ in P_d such that $v'(\delta)$ is of the form $A \leftarrow G$, for some goal G. Now, let us consider the following two subcases.
Case (b.1) G is of the form: $G_L \wedge B \wedge G_R$ and B is an old atom. By (1) the hypothesis that T is a μ-consistent proof tree for A in P_k, (2) the (*Soundness*) property, and (3) Lemma A.15, we have that there exists a μ-consistent proof tree T_d for A and P_d where B is a child of A. By μ-consistency of T_d, we have that $\mu(A) \geq_{lex} \mu(B)$. Since $\mu(B) =_{def} \langle \sigma(B), \mathit{min\text{-}weight}(B)\rangle$ and, since B is an old atom, by Remark A.6, we have that $\mathit{min\text{-}weight}(B) > 0$. Thus, we get that $\mu(A) >_{lex} \langle 0,0\rangle$.
Case (b.2) G is of the form: $G_L \wedge \neg B \wedge G_R$ and B is an old atom. Since δ is locally stratified, $\sigma(A) > \sigma(B)$ and, thus, $\sigma(A) > 0$. Hence, $\mu(A) =_{def} \langle \sigma(A), \mathit{min\text{-}weight}(A) - 1\rangle >_{lex} \langle 0,0\rangle$.
This concludes the proof tree U is μ-consistent.

Case 2.5 (P_{k+1} is derived from P_k by using rule R6.) Let us assume that clause η of the form $\widetilde{A} \leftarrow \widetilde{K}\vartheta \wedge \widetilde{L}_{q+1} \wedge \ldots \wedge \widetilde{L}_r$ is derived by positive folding from a clause $\gamma \in P_k$ of the form $\widetilde{A} \leftarrow \widetilde{L}_1 \wedge \ldots \wedge \widetilde{L}_q \wedge \widetilde{L}_{q+1} \wedge \ldots \wedge \widetilde{L}_r$ using a clause $\delta \in \mathit{Defs}_k$ of the form $\widetilde{K} \leftarrow \widetilde{L}'_1 \wedge \ldots \wedge \widetilde{L}'_q$ and where ϑ is a substitution such that, for $i = 1, \ldots, q$, $\widetilde{L}'_i\vartheta = \widetilde{L}_i$. We have that $\delta \in P_d$ and the definition of the head predicate of δ in P_d consists of clause δ only.

Thus, there is a valuation v such that $v(\widetilde{A}) = A$ and in the proof tree T for A and P_k the children of A are the nodes $L_1, \ldots, L_q, L_{q+1}, \ldots, L_r$ such that for $i = 1, \ldots, q$, $L_i = v(\widetilde{L}'_i)$ and for $i = q+1, \ldots, r$, $L_i = v(\widetilde{L}_i)$. By the induction hypothesis (IndHyp) there exist proof trees for $v'(\widetilde{L}'_1), \ldots, v'(\widetilde{L}'_q)$ and P_k, for some valuation v' such that, for $i = 1, \ldots, q$, $v'(\widetilde{L}'_i\vartheta) = v(\widetilde{L}_i)$. Let K be $v'(\widetilde{K}\vartheta)$.

Since $\delta \in P_d$ and $M(P_d) \models \delta$, by Theorem A.4 and Definition A.12, there is a μ-consistent proof tree for K and P_d. By induction hypothesis, the (*Completeness*) property holds for k and, thus, we have that there exists a μ-consistent proof tree for K and P_k. By the hypothesis that the transformation sequence $P_0, \ldots, P_d, \ldots, P_n$ is admissible and by Condition (2) of Definition 4.5, either A is old or γ is σ-max derived. Thus, by the μ-consistency of the proof tree T, we have that $\mu(A) >_{lex} \mu(L_1) \oplus \cdots \oplus \mu(L_q) \oplus \mu(L_{q+1}) \oplus \cdots \oplus \mu(L_r)$.

Since δ is a clause in Defs_k, by Lemma A.7 we have that $\mu(K) = \mu(L_1) \oplus \cdots \oplus \mu(L_q)$ and, thus, $\mu(A) >_{lex} \mu(K) \oplus \mu(L_{q+1}) \oplus \cdots \oplus \mu(L_r)$.

Moreover, by Lemma A.8 (ii.5), $\mu(A) >_{lex} \mu(K)$. Thus, $A \succ K$ and, by the inductive hypothesis (Iμ), there exists a μ-consistent proof tree U_K for K and P_{k+1}. By the μ-consistency of T and Lemma A.14, for $i = q+1, \ldots, r$, we have $A \succ L_i$. Hence, by the inductive hypotheses (Iμ) and (ISC), for $i = q+1, \ldots, r$, there exists a μ-consistent proof tree U_i for L_i and P_{k+1}. A proof tree U for A and P_{k+1} is constructed by using $v'(\eta)$ at the root of U and the proof trees $U_K, U_{q+1}, \ldots, U_r$ for $K, L_{q+1}, \ldots, L_r$, respectively, and P_{k+1}. The proof tree U is μ-consistent because, as we have shown above, $\mu(A) >_{lex} \mu(K) \oplus \mu(L_{q+1}) \oplus \cdots \oplus \mu(L_r)$.

Case 2.6 (P_{k+1} is derived from P_k by using rule R7.) Suppose that we fold γ using clauses $\delta_1, \ldots, \delta_q$, belonging to (a variant of) Defs_k, and we derive a clause η in P_{k+1}. Without loss of generality, by the definition of rule R7 and the commutativity of $\wedge$, we may assume that (i) $v(\gamma)$ is of the form $A \leftarrow L_1 \wedge \ldots \wedge L_q \wedge L_{q+1} \wedge \ldots \wedge L_r$, (ii) for $i = 1, \ldots, q$, $v(\delta_i)$ is of the form $K \leftarrow M_i$, where $M_i = A_i$, if $L_i = \neg A_i$, and $M_i = \neg A_i$, if $L_i = A_i$, and (iii) $v(\eta)$ is of the form $A \leftarrow \neg K \wedge L_{q+1} \wedge \ldots \wedge L_r$. By the inductive hypothesis, the (*Soundness*) and (*Completeness*) properties hold for k and, therefore, for $i = 1, \ldots, q$, there is no proof tree for

M_i and P_d. Since $M(P_d) \models K \leftrightarrow M_1 \vee \ldots \vee M_q$, there is no proof tree for K and P_d. By the inductive hypothesis, the (*Soundness*) property holds for k and, thus, we have that there is no proof tree for K and P_k. By the hypothesis that the transformation sequence $P_0, \ldots, P_d, \ldots, P_n$ is admissible and by Condition (3) of Definition 4.5, $\sigma(A) > \sigma(K)$. Hence, by the inductive hypothesis (IS), there is no proof tree for K and P_{k+1}, that is, there is a proof tree $U_{\neg K}$ for $\neg K$ and P_{k+1}. By the μ-consistency of T and Lemma A.14, for $i = q+1, \ldots, r$, we have $A \succ L_i$. Hence, by the inductive hypotheses (Iμ) and (ISC), there exists a μ-consistent proof tree U_i for L_i and P_{k+1}. A proof tree U for A and P_{k+1} is constructed by using $v(\eta)$ at the root of U and the proof trees $U_{\neg K}, U_{q+1}, \ldots, U_r$ for $\neg K, L_{q+1}, \ldots, L_r$, respectively, and P_{k+1}.

In order to show that U is μ-consistent we need to consider two cases.

In the first case, we assume that A is old or γ is σ-max derived. Thus, in this case, also η is σ-max derived. By μ-consistency of T, we have $\mu(A) >_{lex} \mu(L_1) \oplus \cdots \oplus \mu(L_q) \oplus \mu(L_{q+1}) \oplus \cdots \oplus \mu(L_r)$. By Lemma A.8 (ii.5), we have that $\mu(A) >_{lex} \mu(L_{q+1}) \oplus \cdots \oplus \mu(L_r)$. Since the transformation sequence $P_0, \ldots, P_n$ is admissible, clause η is locally stratified and, thus, $\sigma(A) > \sigma(K)$. Hence, $\pi_1(\mu(A)) = \{\text{by definition of } \mu\} = \sigma(A) > \sigma(K) = \{\text{by definition of } \mu\} = \pi_1(\mu(\neg K))$. Therefore, by Lemma A.8 (ii.4), we have that: $\mu(A) >_{lex} \mu(\neg K) \oplus \mu(L_{q+1}) \oplus \cdots \oplus \mu(L_r)$. Thus, U is μ-consistent.

In the second case, A is new and γ is not σ-max derived. As a consequence, also η is not σ-max derived. By μ-consistency of T we have $\mu(A) \geq_{lex} \mu(L_1) \oplus \cdots \oplus \mu(L_q) \oplus \mu(L_{q+1}) \oplus \cdots \oplus \mu(L_r)$. And, by Lemma A.8 (ii.5), $\mu(A) \geq_{lex} \mu(L_{q+1}) \oplus \cdots \oplus \mu(L_r)$. Since $\pi_1(\mu(A)) > \pi_1(\mu(\neg K))$ (see the first case), by Lemma A.8 (ii.4), we have that: $\mu(A) \geq_{lex} \mu(\neg K) \oplus \mu(L_{q+1}) \oplus \cdots \oplus \mu(L_r)$. Thus, U is μ-consistent. This completes the proof. ∎

The correctness of admissible transformation sequences, that is, Theorem 4.7 of Section 4, follows immediately from Theorem A.4 and Proposition A.16 because: (i) $P_d = P_0 \cup \mathit{Defs}_n$, and (ii) a μ-consistent proof tree is a proof tree.

Acknowledgements

We thank Hirohisa Seki for stimulating conversations on the topics of this paper. We also thank John Gallagher for his comments and the anonymous referees of ICLP 2010 for their constructive criticism.

We also acknowledge the financial support of: (i) PRIN 2008 (Progetto di Ricerca di Interesse Nazionale) Project no. 9M932N-003, and (ii) the GNCS Group of INdAM (Istituto Nazionale di Alta Matematica) under the grant 'Contributo Progetto 2009'.

References

[1] Abadi, M. and Manna, Z. 1989. Temporal logic programming. *Journal of Symbolic Computation 8,* 3, 277–295.

[2] Apt, K. R. and Bol, R. N. 1994. Logic programming and negation: A survey. *Journal of Logic Programming 19, 20*, 9–71.

[3] Burstall, R. M. and Darlington, J. 1977. A transformation system for developing recursive programs. *Journal of the ACM 24,* 1 (January), 44–67.

[4] Clarke, E. M., Grumberg, O., and Peled, D. 1999. *Model Checking.* MIT Press.

[5] Colmerauer, A. 1982. Prolog and infinite trees. In *Logic Programming*, K. L. Clark and S.-Å. Tärnlund, Eds. Academic Press, 231–251.

[6] Delzanno, G. and Podelski, A. 2001. Constraint-based deductive model checking. *International Journal on Software Tools for Technology Transfer 3,* 3, 250–270.

[7] Etalle, S., Gabbrielli, M., and Meo, M. C. 2001. Transformations of CCP programs. *ACM Transactions on Programming Languages and Systems 23,* 3, 304–395.

[8] Fioravanti, F., Pettorossi, A., and Proietti, M. 2004. Transformation rules for locally stratified constraint logic programs. In *Program Development in Computational Logic*, K.-K. Lau and M. Bruynooghe, Eds. Lecture Notes in Computer Science 3049. Springer-Verlag, 292–340.

[9] Fribourg, L. and Olsén, H. 1997. A decompositional approach for computing least fixed-points of Datalog programs with Z-counters. *Constraints 2,* 3/4, 305–335.

[10] Gupta, G., Bansal, A., Min, R., Simon, L., and Mallya, A. 2007. Coinductive logic programming and its applications. In *Proceedings ICLP'07, Porto, Portugal*, V. Dahl and I. Niemelä, Eds. Lecture Notes in Computer Science 4670. Springer, 27–44.

[11] Jaffar, J., Santosa, A. E., and Voicu, R. 2004. A CLP proof method for timed automata. In *The 25th IEEE International Real-Time Systems Symposium*, J. Anderson and J. Sztipanovits, Eds. IEEE Computer Society, 175–186.

[12] Leuschel, M. and Massart, T. 2000. Infinite state model checking by abstract interpretation and program specialization. In *Proceedings LOPSTR'99, Venezia, Italy*, A. Bossi, Ed. Lecture Notes in Computer Science 1817. Springer, 63–82.

[13] Lloyd, J. W. 1987. *Foundations of Logic Programming.* Springer, Berlin. 2nd Edition.

[14] Min, R. and Gupta, G. 2009. Coinductive logic programming with negation. In *Proc. LOPSTR'09, Coimbra, Portugal*, D. De Schreye, Ed. Lecture Notes in Computer Science 6037. Springer, 97–112.

[15] Nilsson, U. and Lübcke, J. 2000. Constraint logic programming for local and symbolic model-checking. In *Proceedings CL 2000, London, UK*, J. W. Lloyd, Ed. Lecture Notes in Artificial Intelligence 1861. Springer-Verlag, 384–398.

[16] Pettorossi, A. and Proietti, M. 2000. Perfect model checking via unfold/fold transformations. In *Proceedings CL 2000, London, UK*, J. W. Lloyd, Ed. Lecture Notes in Artificial Intelligence 1861. Springer-Verlag, 613–628.

[17] Pettorossi, A., Proietti, M., and Senni, V. 2009. Deciding full branching time logic by program transformation. In *Proc. LOPSTR'09, Coimbra, Portugal*, D. De Schreye, Ed. Lecture Notes in Computer Science 6037. Springer, 5–21.

[18] Pettorossi, A., Proietti, M., and Senni, V. 2010. Transformations of logic programs on infinite lists. Technical Report, IASI-CNR, Roma, Italy.

[19] Proietti, M. and Pettorossi, A. 1995. Unfolding-definition-folding, in this order, for avoiding unnecessary variables in logic programs. *Theoretical Computer Science 142,* 1, 89–124.

[20] Ramakrishna, Y. S., Ramakrishnan, C. R., Ramakrishnan, I. V., Smolka, S. A., Swift, T., and Warren, D. S. 1997. Efficient model checking using tabled resolution. In *Proceedings CAV'97*. Lecture Notes in Computer Science 1254. Springer-Verlag, 143–154.

[21] Roychoudhury, A., Kumar, K. N., Ramakrishnan, C. R., and Ramakrishnan, I. V. 2002. Beyond Tamaki-Sato style unfold/fold transformations for normal logic programs. *International Journal on Foundations of Computer Science 13,* 3, 387–403.

[22] Seki, H. 1991. Unfold/fold transformation of stratified programs. *Theoretical Computer Science 86*, 107–139.

[23] Seki, H. 2009. On inductive and coinductive proofs via unfold/fold transformations. In *Proc. LOPSTR'09, Coimbra, Portugal*, D. De Schreye, Ed. Lecture Notes in Computer Science 6037. Springer, 82–96.

[24] Simon, L., Mallya, A., Bansal, A., and Gupta, G. 2006. Coinductive logic programming. In *Proceedings ICLP'06, Seattle, USA*, S. Etalle and M. Truszczyński, Eds. Lecture Notes in Computer Science 4079. Springer, 330–345.

[25] Staiger, L. 1997. ω-Languages. In *Handbook of Formal Languages*, G. Rozenberg and A. Salomaa, Eds. Vol. 3. Springer, Berlin, 339–387.

[26] Tamaki, H. and Sato, T. 1984. Unfold/fold transformation of logic programs. In *Proceedings ICLP'84*, S.-Å. Tärnlund, Ed. Uppsala University, Uppsala, Sweden, 127–138.

[27] Thomas, W. 1990. Automata on infinite objects. In *Handbook of Theoretical Computer Science*, J. Van Leeuwen, Ed. Vol. B. Elsevier, Amsterdam, 135–191.

[28] Ueda, K. and Furukawa, K. 1988. Transformation rules for GHC programs. In *Proceedings International Conference on Fifth Generation Computer Systems, ICOT, Tokyo, Japan.* 582–591.