TRANSFORMATIONS OF LOGIC PROGRAMS
ON INFINITE LISTS

R. 10-04, 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.
Abstract

We consider an extension of logic programs, called $\omega$-programs, that can be used to define predicates over infinite lists. $\omega$-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 $\omega$-programs is an extension of the perfect model semantics. We present variants of the familiar unfold/fold rules which can be used for transforming $\omega$-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 $\omega$-programs. We demonstrate the power of our transformation-based verification methodology by proving some properties of Büchi automata and $\omega$-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) $\omega$-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 $\omega$-programs, which are logic programs on infinite lists. Similarly to the case of logic programs, for the class of locally stratified $\omega$-programs we define the perfect model semantics (see [2] for a survey on negation in logic programming).

We extend to $\omega$-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 $\omega$-programs.

Then we extend to $\omega$-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 $\omega$-regular expressions.

The paper is structured as follows. In Section 2 we introduce the class of $\omega$-programs and we define the perfect model semantics for locally stratified $\omega$-programs. In Section 3 we present the transformation rules and in Section 4 we prove that they preserve the semantics of $\omega$-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 $\text{Var}$ of variables, a set $\text{Fun}$ of function symbols, and a set $\text{Pred}$ of predicate symbols. We assume that $\text{Fun}$ includes: (i) a finite, non-
empty set $\Sigma$ of constants, (ii) the constructor $\lfloor \_ \rfloor$ of the infinite lists of elements of $\Sigma$, and (iii) at least one constant not in $\Sigma$. Thus, $\lfloor s|t \rfloor$ is an infinite list whose head is $s \in \Sigma$ and whose tail is the infinite list $t$. Let $\Sigma^\omega$ denote the set of the infinite lists of elements of $\Sigma$.

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

An $\omega$-clause $\gamma$ is a formula of the form $A \leftarrow L_1 \land \ldots \land L_m$, 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 $\gamma$ has, among its arguments, at most one argument of type $\text{iolist}$. 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 $\text{hd}(\gamma)$ and $\text{bd}(\gamma)$ the head and the body, respectively, of a clause $\gamma$. An $\omega$-program is a set of $\omega$-clauses.

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

A valuation is a function $v: \text{Var} \to \text{HU} \cup \Sigma \cup \Sigma^\omega$ such that: (i) if $X$ has type $\text{fterm}$ then $v(X) \in \text{HU}$, (ii) if $X$ has type $\text{elem}$ then $v(X) \in \Sigma$, and (iii) if $X$ has type $\text{iolist}$ 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 $\gamma$, by making the function $v$ act on the variables occurring in $t$, or $L$, or $B$, or $\gamma$. (Obviously, $v(t) = t$ if $\text{vars}(t) = \emptyset$.)

We extend the notion of Herbrand base [13] to $\omega$-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 $\text{Pred}$ and $v$ is a valuation}. Thus, any $\omega$-interpretation can be identified with a subset of $\mathcal{B}_\omega$.

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

A level mapping is a function $\ell: \text{Pred} \to \mathbb{N}$. 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 $\omega$-clause $\gamma$ of the form $H \leftarrow L_1 \land \ldots \land L_m$ is stratified w.r.t. $\ell$ if, for $i = 1, \ldots, m$, $\ell(H) \geq \ell(L_i)$. An $\omega$-program $P$ is stratified if there exists
a level mapping \( \ell \) such that all clauses of \( P \) are stratified w.r.t. \( \ell \) \[13\]. Clearly, every stratified \( \omega \)-program is a locally stratified \( \omega \)-program.

Similarly to the case of logic programs on finite terms, for every locally stratified \( \omega \)-program \( P \), we can construct a unique perfect \( \omega \)-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 \texttt{elem}, (ii) \( H \) and \( S \) be variables of type \texttt{elem}, and (iii) \( X \) be a variable of type \texttt{ilist}. Let \texttt{inf\_often\_b} be a predicate of type \texttt{ilist}, and \texttt{member} and \texttt{last\_occ} be predicates of type \texttt{elem \times ilist}. Let us consider the following \( \omega \)-program \( P \):

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

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

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

Finally, we consider the ground atoms of level 2 (i.e., those with predicate \texttt{inf\_often\_b}). For all \( w \in \{a, b\}^\omega \), \( \text{inf\_often\_b}(w) \in M(P) \) iff (see clause 1) \( \text{member}(b, w) \in M(P) \) and \( \text{last\_occ}(b, w) \notin M(P) \), that is, \( w \in (a^*b)^\omega \).

### 3. Transformation Rules

Given an \( \omega \)-program \( P_0 \), a **transformation sequence** is a sequence \( P_0, \ldots, P_n \), with \( n \geq 0 \), of \( \omega \)-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 \( m \ (\geq 1) \) clauses of the form:

\[
\delta_1 : \text{newp}(X_1, \ldots, X_d) \leftarrow B_1, \ldots, \delta_m : \text{newp}(X_1, \ldots, X_d) \leftarrow B_m
\]

where: (i) \texttt{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 \texttt{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 \( n \geq 0 \), \( \text{Defs}_n \) denotes the set of clauses introduced by the definition rule during the transformation sequence \( P_0, \ldots, P_n \). In particular, \( \text{Defs}_0 = \{\} \).

In the following **instantiation** rule we assume that the set of constants of type \texttt{elem} in the language \( \mathcal{L}_\omega \) is the finite set \( \Sigma = \{s_1, \ldots, s_h\} \).
R2. Instantiation. Let \( \gamma : H \leftarrow B \) be a clause in program \( P_k \) and \( X \) be a variable of type \texttt{ilist} occurring in \( \gamma \). By instantiation of \( X \) in \( \gamma \), we get the clauses:

\[
\gamma_1: \ (H \leftarrow B)\{X/\{s_1/X\}\}, \ldots , \gamma_h: \ (H \leftarrow B)\{X/\{s_h/X\}\}
\]

and we say that clauses \( \gamma_1, \ldots , \gamma_h \) are derived from \( \gamma \). 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 \( \gamma : H \leftarrow B_L \land A \land B_R \) be a clause in program \( P_k \) and let \( P'_k \) be a variant of \( P_k \) without variables in common with \( \gamma \). 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 \( \vartheta_i \).

By unfolding \( \gamma \) w.r.t. \( A \) we get the clauses \( \eta_1, \ldots , \eta_m \), where for \( i = 1, \ldots , m \), \( \eta_i \) is \( (H \leftarrow B_L \land B_i \land B_R)\vartheta_i \), and we say that clauses \( \eta_1, \ldots , \eta_m \) are derived from \( \gamma \). For \( i = 1, \ldots , m \), we say that clause \( \eta_i \) is derived by unfolding \( \gamma \) w.r.t. \( A \) using clause \( \gamma_i \). 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 \( \omega \)-program each predicate has at most one argument of type \texttt{ilist}. On the contrary, if predicates may have more than one argument of type \texttt{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 \texttt{ilist}.

The existential variables of a clause \( \gamma \) are the variables occurring in the body of \( \gamma \) and not in its head.

R4. Negative Unfolding. Let \( \gamma: H \leftarrow B_L \land \neg A \land B_R \) be a clause in program \( P_k \) and let \( P'_k \) be a variant of \( P_k \) without variables in common with \( \gamma \). 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 \( \vartheta_i \). 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 \( i = 1, \ldots , m \), \( \gamma_i \) has no existential variables, and (3) from \( \neg (B_1\vartheta_1 \lor \cdots \lor B_m\vartheta_m) \) we get a logically equivalent disjunction \( D_1 \lor \cdots \lor D_r \) of conjunctions of literals, with \( r \geq 0 \), by first pushing \( \neg \) inside and then pushing \( \lor \) outside.

By unfolding \( \gamma \) w.r.t. \( \neg A \) using \( P_k \) we get the clauses \( \eta_1, \ldots , \eta_r \), where, for \( i = 1, \ldots , r \), clause \( \eta_i \) is \( H \leftarrow B_L \land D_i \land B_R \), and we say that clauses \( \eta_1, \ldots , \eta_r \) are derived from \( \gamma \). 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 \( \gamma \) such that \( M(P_k) = M(P_k - \{\gamma\}) \).

R5. Subsumption. Let \( \gamma_1: H \leftarrow \) be a clause in program \( P_k \) and let \( \gamma_2 \) in \( P_k - \{\gamma_1\} \) be a variant of \( (H \leftarrow B)\vartheta \), for some conjunction of literals \( B \) and substitution \( \vartheta \). Then, we say that \( \gamma_2 \) is subsumed by \( \gamma_1 \) 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 $\gamma$ be a clause in $P_k$ and let $Defs'_k$ be a variant of $Defs_k$ without variables in common with $\gamma$. 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 $\vartheta$ such that clause $\gamma$ is of the form $H \leftarrow B_L \land B_\vartheta \land B_R$ and, for every variable $X \in \text{vars}(B) \setminus \text{vars}(K)$, 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* $\gamma$ using $\delta$ we get the clause $\eta: H \leftarrow B_L \land K_\vartheta \land B_R$, and we say that clause $\eta$ is *derived from* $\gamma$. From $P_k$ we derive the new program $P_{k+1} = (P_k - \{\gamma\}) \cup \{\eta\}$.

**R7. Negative Folding.** Let $\gamma$ be a clause in $P_k$ and let $Defs'_k$ be a variant of $Defs_k$ without variables in common with $\gamma$. 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 $\delta_i$ has no existential variables. Suppose that there exists a substitution $\vartheta$ such that clause $\gamma$ is of the form $H \leftarrow B_L \land (\bigwedge_{i=1}^q M_i \land \bigwedge_{i=1}^q \neg M_i) \vartheta \land B_R$, 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* $\gamma$ using $\delta_1, \ldots, \delta_q$ we get the clause $\eta: H \leftarrow B_L \land \neg K_\vartheta \land B_R$, and we say that clause $\eta$ is *derived from* $\gamma$. 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 $\omega$-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) $P_0 \cup Defs_0$ and $P_n$ are locally stratified $\omega$-programs and (ii) $M(P_0 \cup Defs_0) = M(P_n)$.

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 $\sigma$. In order to state those conditions we need the following definitions.

**Definition 4.2 ($\sigma$-Maximal Atom)** Consider a clause $\gamma: H \leftarrow G$. An atom $A$ in $G$ is said to be $\sigma$-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 ($\sigma$-Tight Clause)** A clause $\delta: H \leftarrow G$ is said to be $\sigma$-tight if there exists a $\sigma$-maximal atom $A$ in $G$ such that, for every valuation $v$, $\sigma(v(H)) = \sigma(v(A))$. 

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

Definition 4.5 (Admissible Transformation Sequence) Let $P_0$ be a locally stratified $\omega$-program and let $\sigma$ 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 $\text{Defs}_n$ is locally stratified w.r.t. $\sigma$,
2. for $k = 0, \ldots, n-1$, if $P_{k+1}$ is derived from $P_k$ by positive folding of clause $\gamma$ using clause $\delta$, then:
   2.1 $\delta$ is $\sigma$-tight and either (2.2.i) the head predicate of $\gamma$ occurs in $P_0$, or (2.2.ii) $\gamma$ is a descendant of a clause $\beta$ in $P_j$ with $0 < j \leq k$, such that $\beta$ has been derived by positive unfolding of a clause $\alpha$ in $P_{j-1}$ w.r.t. an atom which is $\sigma$-maximal in the body of $\alpha$ 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 $\eta$, then $\eta$ is locally stratified w.r.t. $\sigma$.

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 $\sigma$ for the initial program $P_0$ and, thus, all programs in the transformation sequence are locally stratified w.r.t. $\sigma$, 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 $\omega$-program, $\sigma$ be a local stratification for $P_0$, and $P_0, \ldots, P_n$ be an admissible transformation sequence. Then the programs $P_0 \cup \text{Defs}_n, P_1, \ldots, P_n$, are all locally stratified w.r.t. $\sigma$.

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 $\sigma$-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:

\[
\begin{align*}
m & \leftarrow \\
e & \leftarrow \neg m \\
e & \leftarrow e
\end{align*}
\]

By rule R1 we introduce the clause

\[
\delta_1: f \leftarrow m \land \neg e
\]

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

By rule R3 we unfold $\delta_1$ w.r.t. $m$ and we get the clause

\[
\delta_2: f \leftarrow \neg e.
\]

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

\[
\delta_3: f \leftarrow m \land \neg e.
\]

We derive program $P_3 = P_0 \cup \{\delta_3\}$. By rule R6 we fold clause $\delta_3$ using clause $\delta_1$, and we get

\[
\delta_4: f \leftarrow f.
\]

We derive program $P_4 = P_0 \cup \{\delta_4\}$. Thus, the transformation sequence $P_0$, ..., $P_4$ 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:

\textbf{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:

\[
\begin{align*}
even(0) & \leftarrow \\
even(s(s(X))) & \leftarrow even(X), \\
odd(s(0)) & \leftarrow \\
odd(s(X)) & \leftarrow \neg odd(X)
\end{align*}
\]

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

\[
\delta_1: p \leftarrow even(X) \land \neg odd(s(X))
\]

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

\[
\begin{align*}
\delta_2: p & \leftarrow \neg odd(s(0)) \\
\delta_3: p & \leftarrow even(X) \land \neg odd(s(s(X))))
\end{align*}
\]

By unfolding, clause $\delta_2$ is removed and we derive $P_3 = P_0 \cup \{\delta_3\}$.

By unfolding $\delta_3$ w.r.t. $\neg odd(s(s(X))))$ we derive $P_4 = P_0 \cup \{\delta_4\}$, where
By applying rule R6, we fold clause $\delta_1$ and derive the final program $P_6 = P_0 \cup \{\delta_6\}$, where $\delta_6: p \leftarrow p$.

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 $\delta_1$ is $\sigma$-tight and $\delta_5$ is a descendant of $\delta_3$ which has been derived by unfolding w.r.t. a $\sigma$-maximal atom whose predicate occurs in $P_0$.

Note that, $P_0, \ldots, P_5$ violates Condition (NU) because, by unfolding clause $\delta_3$ w.r.t. the literal $\neg \text{odd}(s(s(s(X))))$, the number of positive occurrences of atoms in the body of the derived clause $\delta_4$ is larger than that number in $\delta_3$.

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 $\sigma$. The atom $m$ is not $\sigma$-maximal in $m \land \neg e$ because $e$ depends on $\neg m$ and, hence, $\sigma(\neg e) > \sigma(m)$. Thus, the positive folding rule R6 is applied to the clause $\delta_3$ which is not a descendant of any clause derived by unfolding w.r.t. a $\sigma$-maximal atom.

5. Verifying Properties of $\omega$-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 $\omega$-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 $\omega$-regular languages.

We assume that we are given an $\omega$-program $P$ defining a unary predicate $prop$ of type $\text{ilist}$, 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 $\omega$-programs presented in Section 3 we derive a monadic $\omega$-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 $\omega$-programs and we check whether or not $M(T) \models \exists X \ prop(X)$.

Our verification method is an extension to $\omega$-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 $\omega$-program, instead of an $\omega$-program which encodes the branching time temporal logic $\text{CTL}^*$, 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 $\omega$-program into a monadic $\omega$-program. Indeed, (i) the problem of verifying whether or not, for any $\omega$-program $P$ and unary predicate $prop$, $M(P) \models \exists X \ prop(X)$ is undecidable, because the class of $\omega$-programs
includes the locally stratified logic programs on finite terms, and (ii) the proof method for monadic \( \omega \)-programs presented in [17] is complete. However, we believe that automatic transformation strategies can be proposed for significant subclasses of \( \omega \)-programs along the lines of [19, 16].

**Definition 5.1 (Monadic \( \omega \)-Programs)** A monadic \( \omega \)-clause is an \( \omega \)-clause of the form \( A_0 \leftarrow L_1 \land \ldots \land L_m \), 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 \( \text{ilist} \) 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 \( \text{ilist} \), and

(iii) there exists a level mapping \( \ell \) such that, for \( i = 1, \ldots, m \), if \( L_i \) is an atom and \( \text{vars}(A_0) \not\supseteq \text{vars}(L_i) \), then \( \ell(A_0) > \ell(L_i) \) else \( \ell(A_0) \geq \ell(L_i) \).

A monadic \( \omega \)-program is a finite set of monadic \( \omega \)-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 \( A \) is a nondeterministic finite automaton \( \langle \Sigma, Q, q_0, \delta, F \rangle \), where, as usual, \( \Sigma \) 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 \( A \) 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 \( \rho_0 \) is the initial state and, for all \( n \geq 0 \), \( \langle \rho_n, a_n, \rho_{n+1} \rangle \in \delta \). Let \( \text{Inf}(\rho) \) denote the set of states that occur infinitely often in the infinite sequence \( \rho \) of states. An infinite word \( w \in \Sigma^\omega \) is accepted by \( A \) if there exists a run \( \rho \) of \( A \) on \( w \) such that \( \text{Inf}(\rho) \cap F \neq \emptyset \) or, equivalently, if there is no state \( \rho_m \) in \( \rho \) such that every state \( \rho_n \), with \( n \geq m \), is not final.

The language accepted by \( A \) is the subset of \( \Sigma^\omega \), denoted \( \mathcal{L}(A) \), of the infinite words accepted by \( A \). In order to check whether or not the language \( \mathcal{L}(A) \) is empty, we construct an \( \omega \)-program which defines a unary predicate \( \text{accepting-run}(X) \) such that:

(a) \( \mathcal{L}(A) \neq \emptyset \) iff \( \exists X \text{ accepting-run}(X) \)

The predicate \( \text{accepting-run}(X) \) is defined by the following formulas:

(1) \( \text{accepting-run}(X) \equiv_{\text{def}} \text{run}(X) \land \neg \text{rejecting}(X) \)

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

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

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) \( \text{occ}(s^n(0), \rho, q) \) iff \( \rho_n = q \),

(ii) \( \text{initial}(q) \) iff \( q = q_0 \),

(iii) \( \text{nat}(s^n(0)) \) iff \( n \geq 0 \),

(iv) \( \text{tr}(q_1, a, q_2) \) iff \( \langle q_1, a, q_2 \rangle \in \delta \),

(v) \( \text{geq}(s^n(0), s^n(0)) \) iff \( n \geq m \), and

(vi) \( \text{final}(q) \) iff \( q \in F \).

By (a) and (1)–(3) above, \( \mathcal{L}(A) \neq \emptyset \) iff there exists an infinite sequence \( \rho = \rho_0 \rho_1 \ldots \in Q^\omega \) of states such that: (i) \( \rho_0 \) 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 \( \rho_m \), with \( m \geq 0 \), in \( \rho \) such that, for all \( n \geq m \), \( \rho_n \not\in F \) (see (3)).
Now we introduce an $\omega$-program $P_A$ 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 $P_A$ 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 $A = \langle \Sigma, Q, q_0, \delta, F \rangle$, the encoding $\omega$-program $P_A$ consists of the following clauses (independent of $A$):

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

Together with the clauses (depending on $A$) 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) $initial(s)$ holds iff $s$ is $q_0$, (ii) $tr(s_1, a, s_2)$ holds iff $\langle s_1, a, s_2 \rangle \in \delta$, and (iii) $final(s)$ holds iff $s \in F$.

The $\omega$-program $P_A$ is locally stratified w.r.t. the stratification function $\sigma$ defined as follows: for every atom $A$ in $B_\omega$, $\sigma(A) = 0$, except that: for every element $n$ in $\{sk(0) \mid k \geq 0\}$, for every infinite list $\rho$ in $Q_\omega$, (i) $\sigma(occ(\rho)) = \sigma(not\_a\_run(\rho)) = \sigma(occ(\rho)) = 1$, and (ii) $\sigma(\rho) = \sigma(accepting\_run(\rho)) = 2$.

Now, let us consider a Büchi automaton $A$ 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:

```
1   a   2
\_   \_   \_
```

For this automaton $A$, program $P_A$ 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. $initial(1)$
14. $tr(1, a, 1)$
15. $tr(1, b, 1)$
16. $tr(1, a, 2)$
17. $tr(2, a, 2)$
18. $final(2)$

In order to check whether or not $L(A) = \emptyset$ 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 $\omega$-program $P_A$ into a monadic $\omega$-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. $\neg exists\_final(M, [1|X]) \leftarrow \neg geq(N, M) \land occ(N, [1|X], S) \land final(S)$
20. \textit{exists\_final}(M, [2|X]) \leftarrow \textit{geo}(N, M) \land \textit{occ}(N, [2|X], S) \land \textit{final}(S)

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

21. \textit{exists\_final}(0, [1|X]) \leftarrow \textit{occ}(N, X, S) \land \textit{final}(S)

22. \textit{exists\_final}(s(M), [1|X]) \leftarrow \textit{geo}(N, M) \land \textit{occ}(N, X, S) \land \textit{final}(S)

23. \textit{exists\_final}(0, [2|X]) \leftarrow \textit{final}(S)

24. \textit{exists\_final}(s(M), [2|X]) \leftarrow \textit{geo}(N, M) \land \textit{occ}(N, X, S) \land \textit{final}(S)

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

25. \textit{new\_1}(X) \leftarrow \textit{occ}(N, X, S) \land \textit{final}(S)

This clause is \(\sigma\)-tight by taking, for every infinite list \(\rho\) of states, \(\sigma(\textit{new\_1}(\rho)) = 0\). By folding clause 25 using clause 21, 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. \textit{exists\_final}(0, [1|X]) \leftarrow \textit{new\_1}(X)

27. \textit{exists\_final}(s(M), [1|X]) \leftarrow \textit{exists\_final}(M, X)

28. \textit{exists\_final}(s(M), [2|X]) \leftarrow \textit{exists\_final}(M, X)

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

29. \textit{new\_1}([1|X]) \leftarrow \textit{occ}(N, X, S) \land \textit{final}(S)

30. \textit{new\_1}([2|X]) \leftarrow

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

31. \textit{new\_1}([1|X]) \leftarrow \textit{new\_1}(X)

At this point we have obtained the definitions of the predicates \textit{exists\_final} and \textit{new\_1} (that is, clauses 23, 26–28, 30, and 31) that do not have existential variables.

Now the transformation of program \(P_A\) 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. \textit{accepting\_run}([1|X]) \leftarrow \neg \textit{not\_a\_run}(X) \land \textit{new\_1}(X) \land \neg \textit{rejecting}(X)

33. \textit{run}([1|X]) \leftarrow \neg \textit{not\_a\_run}(X)

34. \textit{not\_a\_run}([1|X]) \leftarrow \neg \textit{not\_a\_run}(X)

35. \textit{not\_a\_run}([2|X]) \leftarrow \textit{new\_2}(X)

36. \textit{not\_a\_run}([2|X]) \leftarrow \neg \textit{not\_a\_run}(X)

37. \textit{new\_2}([1|X]) \leftarrow

38. \textit{rejecting}([1|X]) \leftarrow \neg \textit{new\_1}(X)

39. \textit{rejecting}([1|X]) \leftarrow \textit{rejecting}(X)

40. \textit{rejecting}([2|X]) \leftarrow \textit{rejecting}(X)

The final \(\omega\)-program \(T\) obtained from program \(P_A\), consists of clauses 30–40 and it is a monadic \(\omega\)-program.

Now, in the second step of our verification method, we check whether or not the formula \(\exists X \textit{accepting\_run}(X)\) 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 \(\exists X \textit{accepting\_run}(X)\). The run \(\rho = 12^\omega\) is a witness for \(X\) and corresponds to the accepted word \(a^\omega\). Thus, \(L(A) \neq \emptyset\).
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:

\[
e := a \mid e_1 e_2 \mid e_1 + e_2 \mid e^* \text{ with } a \in \Sigma \quad \text{(regular expressions)}
\]

\[
f := e^\omega \mid e_1 e_2^\omega \mid f_1 + f_2 \quad \text{(ω-regular expressions)}
\]

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

Now we introduce an ω-program, called \( P_f \), which defines the predicate ω-acc such that for any ω-regular expression \( f \), for any infinite word \( w \), ω-acc(\( f, w \)) holds iff \( w \in L(f) \). 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 \( \Sigma \). The ω-program \( P_f \) is made out of the following clauses:

1. \( \text{acc}(E,[E]) \leftarrow \text{symb}(E) \)
2. \( \text{acc}(E_1 E_2, X) \leftarrow \text{app}(X_1, X_2, X) \land \text{acc}(E_1, X_1) \land \text{acc}(E_2, X_2) \)
3. \( \text{acc}(E_1 + E_2, X) \leftarrow \text{acc}(E_1, X) \)
4. \( \text{acc}(E_1 + E_2, X) \leftarrow \text{acc}(E_2, X) \)
5. \( \text{acc}(E^\omega, [\[]) \leftarrow \text{acc}(E^\omega, X) \leftarrow \text{app}(X_1, X_2, X) \land \text{acc}(E, X_1) \land \text{acc}(E^\omega, X_2) \)
6. \( \text{acc}(F_1 + F_2, X) \leftarrow \text{acc}(F_1, X) \)
7. \( \text{acc}(F_1 + F_2, X) \leftarrow \text{acc}(F_2, X) \)
8. \( \text{acc}(E^\omega, X) \leftarrow \text{app}(X, E) \)
9. \( \text{acc}(E^\omega, X) \leftarrow \text{app}(X, E) \leftarrow \text{acc}(E, X) \land \text{acc}(E^\omega, X) \)
10. \( \text{acc}(E_1 E_2^\omega, X) \leftarrow \text{app}(X_1, X_2, X) \land \text{acc}(E_1, X_1) \land \text{acc}(E_2^\omega, X_2) \)
11. \( \text{new}_1(E, X) \leftarrow \text{nat}(M) \land \neg \text{new}_2(E, M, X) \)
12. \( \text{new}_2(E, M, X) \leftarrow \text{geq}(N, M) \land \text{prefix}(X, N, V) \land \text{acc}(E^\omega, V) \)
13. \( \text{geq}(N, 0) \leftarrow \)
14. \( \text{geq}(s(N), s(M)) \leftarrow \text{geq}(N, M) \)
15. \( \text{nat}(0) \leftarrow \)
16. \( \text{nat}(s(N)) \leftarrow \text{nat}(N) \)
17. \( \text{prefix}(X, 0, [\[]) \leftarrow \)
18. \( \text{prefix}([S|X], s(N), [S|Y]) \leftarrow \text{prefix}(X, N, Y) \)
19. \( \text{app}([\[]], Y, Y) \leftarrow \)
20. \( \text{app}([S|X], Y, [S|Z]) \leftarrow \text{app}(X, Y, Z) \)

Figure 1: Proof of \( \exists X \text{ accepting}_\text{run}(X) \) w.r.t. the monadic ω-program \( T \). On the right we have shown the infinite loop and the associated accepting run \( 122^\omega \) (that is, \( 12^\omega \)).
21. \( \text{app}([], Y, Y) \leftarrow \)
22. \( \text{app}([S|X], Y, [S|Z]) \leftarrow \text{app}(X, Y, Z) \)

Together with the clauses defining the predicate \( \text{symb} \), where \( \text{symb}(a) \) holds iff \( a \in \Sigma \). Note that:

(i) \( \text{prefix}(X, N, Y) \) holds iff \( Y \) is the list of the \( N (\geq 0) \) leftmost symbols of the infinite list \( X \), and

(ii) \( \omega\text{-app}(X, Y, Z) \) holds iff the concatenation of the finite list \( X \) and the infinite list \( Y \) is the infinite list \( Z \). Clauses 1–6 stipulate that, for any finite word \( w \) and regular expression \( e \), \( \text{acc}(e, w) \) holds iff \( w \in L(e) \). Analogously, clauses 7–12 stipulate that, for any infinite word \( w \) and \( \omega\)-regular expression \( f \), \( \omega\text{-acc}(f, w) \) holds iff \( w \in L(f) \). In particular, clauses 9, 11, and 12 correspond to the following definition:

\[
\omega\text{-acc}(E^\omega, X) \equiv \exists M(\text{nat}(M) \rightarrow \exists N \exists V (\text{geq}(N, M) \land \text{prefix}(X, N, V) \land \text{acc}(E^*, V)))
\]

The \( \omega \)-program \( P_f \) is stratified and, thus, it is locally stratified.

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

23. \( \text{expr}_1(X) \leftarrow \omega\text{-acc}(a^\omega, X) \)
24. \( \text{expr}_2(X) \leftarrow \omega\text{-acc}((b^*a)\omega, X) \)

Together with program \( P_f \), define the predicates \( \text{expr}_1 \) and \( \text{expr}_2 \) such that, for every infinite word \( w \), \( \text{expr}_1(w) \) holds iff \( w \in L(f_1) \) and \( \text{expr}_2(w) \) holds iff \( w \in L(f_2) \). If we introduce the following clause:

25. \( \text{not-contained}(X) \leftarrow \text{expr}_1(X) \land \neg \text{expr}_2(X) \)

we have that \( L(f_1) \subseteq L(f_2) \) iff \( M(P_f \cup \{23, 24, 25\}) \not\models \exists X \text{not-contained}(X) \). 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 \( \omega \)-program \( T \):

26. \( \text{not-contained}(\llbracket a|X] \rrbracket) \leftarrow \neg \text{new}_3(X) \land \text{new}_4(X) \)
27. \( \text{new}_3(\llbracket a|X] \rrbracket) \leftarrow \text{new}_3(X) \)
28. \( \text{new}_3(\llbracket b|X] \rrbracket) \leftarrow \)
29. \( \text{new}_4(\llbracket a|X] \rrbracket) \leftarrow \text{new}_4(X) \)
30. \( \text{new}_4(\llbracket b|X] \rrbracket) \leftarrow \text{new}_5(X) \)
31. \( \text{new}_5(\llbracket a|X] \rrbracket) \leftarrow \text{new}_4(X) \)
32. \( \text{new}_5(\llbracket b|X] \rrbracket) \leftarrow \text{new}_5(X) \)
33. \( \text{new}_5(\llbracket b|X] \rrbracket) \leftarrow \neg \text{new}_6(X) \)
34. \( \text{new}_6(\llbracket a|X] \rrbracket) \leftarrow \)
35. \( \text{new}_6(\llbracket b|X] \rrbracket) \leftarrow \text{new}_6(X) \)

By using the proof method for monadic \( \omega \)-programs of [17] we have that:

\( M(T) \not\models \exists X \text{not-contained}(X) \)

and, thus, \( L(f_1) \subseteq L(f_2) \).

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 $\omega$-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 $\omega$-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 $\omega$-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 $\omega$-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 $\sigma$ 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 $\omega$-program, $\sigma$ is a local stratification for $P_0$, and $P_0, P_1, \ldots, P_n$ is an admissible transformation sequence. Then the programs $P_0 \cup \text{Defs}_n$, $P_1, \ldots, P_n$ are locally stratified w.r.t. $\sigma$.

Proof. Since $P_0, P_1, \ldots, P_n$ is an admissible transformation sequence, every definition in $\text{Defs}_n$ is locally stratified w.r.t. $\sigma$ (see Point (1) of Definition 4.5). Since, by hypothesis, $P_0$ is locally stratified w.r.t. $\sigma$, also $P_0 \cup \text{Defs}_n$ is locally stratified w.r.t. $\sigma$.

Now we will prove that, for $k = 0, \ldots, n$, $P_k$ is locally stratified w.r.t. $\sigma$ by induction on $k$.

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

**Step.** We assume that $P_k$ is locally stratified w.r.t. $\sigma$ and we show that $P_{k+1}$ is locally stratified w.r.t. $\sigma$. 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. $\sigma$ by the inductive hypothesis. Since $P_0, P_1, \ldots, P_n$ is an admissible transformation sequence, $\{\delta_1, \ldots, \delta_m\}$ is locally stratified w.r.t. $\sigma$ (see Point (1) of Definition 4.5). Thus, $P_{k+1}$ is locally stratified w.r.t. $\sigma$.

**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 $\gamma$ is the clause $H \leftarrow B$ and, for $i = 1, \ldots, h$, $\gamma_i$ is the clause $(H \leftarrow B_i)[X/\llbracket s_i|X\rrbracket]$. Take any $i \in \{1, \ldots, h\}$. Let $L\{X/\llbracket s_i|X\rrbracket\}$ be a literal in the body of $\gamma_i$. Let $v$ be any valuation and $v'$ be the valuation such that $v'(X) = \llbracket s_i|v(X)\rrbracket$ and $v'(Y) = v(Y)$ for every variable $Y$ different from $X$. We have:

$$
\sigma(v(H\{X/\llbracket s_i|X\rrbracket\})) = \sigma(v'(H)) \quad \text{(definition of $v'$)}
$$

$$
\geq \sigma(v'(L)) \quad \text{($\gamma$ is locally stratified w.r.t. $\sigma$)}
$$

$$
= \sigma(v(\overline{v(L\{X/\llbracket s_i|X\rrbracket\})})) \quad \text{(definition of $v'$)}
$$

Thus, $\gamma_i$ is locally stratified w.r.t. $\sigma$. Hence, $P_{k+1}$ is locally stratified w.r.t. $\sigma$.

**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 $\gamma$ 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 $\gamma$ w.r.t. $A$. Since, by the induction hypothesis, $(P_k - \{\gamma\})$ is locally stratified w.r.t. $\sigma$, it remains to show that, for $i = 1, \ldots, m$, clause $\eta_i$ is locally stratified w.r.t. $\sigma$. For $i = 1, \ldots, m$, $\eta_i$ is of the form $(H \leftarrow G_L \wedge B_i \wedge G_R)\vartheta_i$, where $\gamma_i$: $K_i \leftarrow B_i$ is a clause in a variant of $P_k$ such that $\gamma_i$ has no variable in common with $\gamma$ 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 $\gamma$ or $\gamma_i$, $v'(X) = v(X\vartheta_i)$.

Let $G_L \wedge B_i \wedge G_R$ be the conjunction of $s \geq 0$ 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:

$$
\sigma(v(H\vartheta_i)) = \sigma(v'(H)) \quad \text{(definition of $v'$)}
$$

$$
\geq \sigma(v'(L_{kj})) \quad \text{($\gamma$ is locally stratified w.r.t. $\sigma$)}
$$

$$
= \sigma(v(L_{kj}\vartheta_i)) \quad \text{(definition of $v'$)}
$$
For \( j = r + 1, \ldots, s \), we have:

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

Thus, the clause \( \eta_i \) is locally stratified w.r.t. \( \sigma \).

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

Let \( \gamma_1: K_1 \leftarrow B_1, \ldots, \gamma_m: K_m \leftarrow B_m \) be the clauses in a variant of \( P_k \) such that, for \( i = 1, \ldots, m \), \( A = K_i\vartheta_i \) for some substitution \( \vartheta_i \). Then, for \( j = 1, \ldots, r \), \( \eta_j \) is of the form \( H \leftarrow L_{j_1} \land \ldots \land L_{j_p} \) 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 \( G_L \land G_R \), or (Case b) \( L_{jp} \) is a negated atom that occurs in \( G_L \land G_R \), 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, \( \gamma \) is locally stratified w.r.t. \( \sigma \). In Case (c) we have:

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

In Case (d) we have:

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

Thus, \( \eta_j \) is locally stratified w.r.t. \( \sigma \). Hence, \( P_{k+1} \) is locally stratified w.r.t. \( \sigma \).

**Case 5.** Program \( P_{k+1} \) is derived by subsumption (rule R5). \( P_{k+1} \) is locally stratified w.r.t. \( \sigma \) 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 \( P_{k+1} = (P_k \setminus \{\gamma\}) \cup \{\eta\} \), where \( \eta \) is a clause of the form \( H \leftarrow B_L \land K\vartheta \land B_R \) derived by positive folding of clause \( \gamma \) of the form \( H \leftarrow B_L \land B\vartheta \land B_R \) using a clause \( \delta \) of the form \( K \leftarrow B \in \text{Def}_f \). We have to show that \( \eta \) is locally stratified w.r.t. \( \sigma \), that is, for every valuation \( v \), \( \sigma(v(H)) \geq \sigma(v(K)\vartheta) \).

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

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

Hence, \( \eta \) is locally stratified w.r.t. \( \sigma \).

**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, \( \eta \) is locally stratified w.r.t. \( \sigma \) (see Point (3) of Definition 4.5).

In the rest of this Appendix we will consider:

(i) a local stratification \( \sigma : B_\omega \rightarrow W \),
(ii) an \( \omega \)-program \( P_0 \) which is locally stratified w.r.t. \( \sigma \), 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 \( \text{Defs}_n \).

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, \( P_d = P_0 \cup \text{Defs}_n \). In order to prove the correctness of the admissible transformation sequence \( P_0, \ldots, P_d \) (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 \( A \in B_\omega \) and a locally stratified \( \omega \)-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 \( B_\omega \), let \( P \) be a locally stratified \( \omega \)-program, and let \( \sigma \) be a local stratification for \( P \). Let \( PT_{<A} \) denote the set of proof trees for \( H \) and \( P \), where \( H \in B_\omega \) 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 \( H \in B_\omega \) and there exist a clause \( \gamma \in P \) and a valuation \( v \) such that \( v(\gamma) \) is \( H \leftarrow L_1 \land \ldots \land L_r \), and
(iii) every leaf of \( T \) is either labeled by the empty conjunction true or by a negated atom \( \neg H \), with \( H \in B_\omega \), such that there is no proof tree for \( H \) and \( P \) in \( PT_{<A} \).

Let \( A \) be an atom in \( B_\omega \) and \( P \) be a locally stratified \( \omega \)-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 \( A \in B_\omega \) 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 $\omega$-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, $\mu$)** (i) For any proof tree $T$, $\text{size}(T)$ 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 $T$, $\text{weight}(\alpha, T)$ is the number of nodes of $T$ whose label is an atom with stratum $\alpha$. (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:

$$\text{min-weight}(A) = \min \{ \text{weight}(\alpha, T) \mid \sigma(A) = \alpha \text{ and } T \text{ is a proof tree for } A \text{ and } P_d \}.$$ 

(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:

$$\mu(A) = \begin{cases} \sigma(A), \text{min-weight}(A) & \text{if } A \text{ is an old atom} \\ \sigma(A), \text{min-weight}(A) - 1 & \text{if } A \text{ is a new atom} \end{cases}$$

(v) For any atom $A \in \mathcal{B}_\omega$ such that there exists no proof tree for $A$ and $P_d$, we define:

$$\mu(\lnot A) = \sigma(A), 0.$$ 

**Remark A.6.** (i) If $A$ is an old atom then $\text{min-weight}(A) > 0$ else $\text{min-weight}(A) \geq 0$.

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

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

or equivalently,

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

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

$$\mu(L_1 \land \ldots \land L_r) = \mu(L_1) \oplus \cdots \oplus \mu(L_r)$$

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

$$\mu(\text{true}) = (0, 0)$$

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

The following lemma follows from the definition of the measure $\mu$. Recall that a new predicate can only be defined in terms of old predicates.
Lemma A.7 (Properties of $\mu$ for a Definition in $P_d$) Let $\delta \in P_d$ be a $\sigma$-tight clause introduced by the definition rule R1 with $m = 1$, that is, $\delta$ is the only clause defining the head predicate of $\delta$ in $P_d$. Let $v$ be a valuation and $v(\delta)$ be of the form: $K \leftarrow L_1 \land \ldots \land L_q$. 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 $\delta$ is $\sigma$-tight and, thus, $L_1$ is $\sigma$-maximal. We have that:

Thus

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

Now, $\text{min-weight}(K) = \{\text{by definition of min-weight}\} = \min\{\text{weight}(\sigma(K), T_K)\}$ where $T_K$ is a proof tree for $K$ and $P_d = \{\text{by definition of weight (see also Figure 2)}\} = \{\text{by definition of weight and Remark A.3}\} = \{\text{by min}\sum = \sum\text{min}\} = \{\text{by } \sigma\text{-tightness}\} = \{\text{by } \sigma\text{-tightness}\} = \{\text{by } \sigma\text{-tightness}\} = \{\text{by } \sigma\text{-tightness}\} = \{\text{by } \sigma\text{-tightness}\}. \quad \blacksquare$

![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: $K \leftarrow L_1 \land \ldots \land L_q$. For $i = 1, \ldots, q$, $T_i$ is a proof tree for $L_i$ and $P_d$.](image)

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

Let $\pi_1$ and $\pi_2$ 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 $R \in \{>_{\text{lex}}, >_{\text{lex}}\}$, we have that:

(iii.1) $A \oplus B \geq_{\text{lex}} A$

(iii.2) if $A \geq_{\text{lex}} B$ then $A \oplus C \geq_{\text{lex}} B \oplus C$

(iii.3) if $A >_{\text{lex}} B$, $A_1 \geq C_1$, and $A_2 > 0$ then $A \oplus C >_{\text{lex}} B \oplus C$

(iii.4) if $A R B$ and $A_1 \geq C_1$ then $A \oplus B \oplus C$

(iii.5) if $A \not\text{R} B \oplus C$ then $A \not\text{R} B$ and $A \not\text{R} C$
(ii.1) By cases. If $A > B_1$ 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:

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

and

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

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): $A > B_1$, 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 $A_1 \geq C_1$, 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) $\beta$ 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 $A_1 = B_1$, $\beta$ reduces to

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

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

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

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

and

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

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 $\alpha$ reduces to $\langle C_1, A_2 + C_2 \rangle$ and

$\beta$ reduces to $\langle C_1, \text{ if } B_1 = C_1 \text{ then } B_2 + C_2 \text{ else if } B_1 > C_1 \text{ then } B_2 \text{ else } C_2 \rangle$

and since $A_1 > B_1$ and $A_1 = C_1$, we get that $\beta$ 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 $\alpha$ reduces to $\langle A_1, \ldots \rangle$ and $\beta$ 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 $\alpha$ reduces to $\langle A_1, A_2 + C_2 \rangle$ and $\beta$ 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 $\alpha$ reduces to $\langle A_1, A_2 \rangle$ and $\beta$ 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:

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

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 $A_1 > C_1$. 

Proof. (i) It follows immediately from the definition.
Case (2.1). We have that: $A_1 > \max(B_1, C_1)$ and thus, $A \geq_{\text{lex}} B \oplus C$.

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

(ii.5) We have that:

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

We reason by cases: Case (1) $A = B \oplus C$, and Case (2) $A >_{\text{lex}} B \oplus C$. In order to show Point (ii.5) in Case (1) we have to show $A \geq_{\text{lex}} B$ and $A \geq_{\text{lex}} C$, and in Case (2) we have to show $A >_{\text{lex}} B$ and $A >_{\text{lex}} C$.

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, $A \geq_{\text{lex}} B$ and $A \geq_{\text{lex}} C$.

Case (1.2): $B_1 > C_1$. Thus, $A_1 = B_1$ and $A_2 = B_2$. Thus, $A \geq_{\text{lex}} B$ and $A \geq_{\text{lex}} C$.

Case (1.3): $B_1 < C_1$. Like Case (1.2), by interchanging $B$ and $C$.

Case (2) Assume $A >_{\text{lex}} B \oplus C$.

Case (2.1): $A_1 > \max(B_1, C_1)$. We get: $A >_{\text{lex}} B$ and $A >_{\text{lex}} C$.

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 $A >_{\text{lex}} B \oplus C$, we have: $A_2 = B_2 + C_2$.

Thus, we get $A >_{\text{lex}} B$ and $A >_{\text{lex}} C$.

Case (2.2.2): $B_1 > C_1$. Thus, $A_1 = \max(B_1, C_1) = B_1$. Since $A >_{\text{lex}} B \oplus C$ and $A_1 = \pi_1(B \oplus C)$, we have: $A_2 > \pi_2(B \oplus C)$, that is,

$A_2 > \text{ if } B_1 = C_1 \text{ then } B_2 + C_2 \text{ else } \text{if } B_1 > C_1 \text{ then } B_2 \text{ else } C_2$, that is,

$A_2 > B_2$.

Thus, we get $A >_{\text{lex}} B$ and, since $B_1 > C_1$, we also get $A >_{\text{lex}} C$.

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 $A \in B_\omega$, literals $L_1, \ldots, L_m$, which are either atoms in $B_\omega$ or negation of atoms in $B_\omega$, if for $i = 1, \ldots, m$, $\sigma(A) \geq \sigma(L_i)$ then $\mu(\overline{A}) \geq_{\text{lex}} \mu(L_1) \oplus \cdots \oplus \mu(L_m)$.

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

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: $\langle \sigma(A), 0 \rangle \geq_{\text{lex}} \langle \beta, w_1 \rangle$.

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 $\mu(\overline{A}) \geq_{\text{lex}} \mu(\overline{L}_1) \oplus \cdots \oplus \mu(\overline{L}_j) \oplus \mu(B)$.

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 $\langle \sigma(A), 0 \rangle \geq_{\text{lex}} \langle \beta, w_1 \rangle$. We also have that $\langle \beta, w_1 \rangle \oplus \langle \sigma(B), 0 \rangle = \langle \sigma(B), 0 \rangle$ and $\langle \sigma(A), 0 \rangle \geq_{\text{lex}} \langle \sigma(B), 0 \rangle$. Thus, we get $\langle \sigma(A), 0 \rangle \geq_{\text{lex}} \langle \beta, w_1 \rangle \oplus \langle \sigma(B), 0 \rangle$.

Case (2.2). $\sigma(B) = \beta$. By induction hypothesis we have that $\langle \sigma(A), 0 \rangle \geq_{\text{lex}} \langle \beta, w_1 \rangle$. We also have that $\langle \beta, w_1 \rangle \oplus \langle \sigma(B), 0 \rangle = \langle \beta, w_1 \rangle$. Thus, we get $\langle \sigma(A), 0 \rangle \geq_{\text{lex}} \langle \beta, w_1 \rangle \oplus \langle \sigma(B), 0 \rangle$. 


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

Now we introduce the notion of a \( \mu \)-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 (\( \sigma \)-max Derived Clause)** We say that a clause \( \gamma \) in a program \( P_k \) of the sequence \( P_d, \ldots, P_n \) is a \( \sigma \)-max derived clause if \( \gamma \) is a descendant of a clause \( \beta \) in \( P_j \), with \( d < j \leq k \), such that \( \beta \) has been derived by unfolding a clause \( \alpha \) in \( P_{j-1} \) w.r.t. an old \( \sigma \)-maximal atom. (Recall that, by definition, a clause is a descendant of itself.)

**Definition A.12 (\( \mu \)-consistent Proof Tree)** Let \( A \) be an atom in \( B_\omega \) 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 \( \mu \)-consistent if for all atoms \( H \), all literals \( L_1, \ldots, L_r \) which are the children of \( H \) in \( T \), where \( H \leftarrow L_1 \land \ldots \land L_r \) 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 \( \gamma \) is not \( \sigma \)-max derived then \( \mu(H) \geq_{\text{lex}} \mu(L_1) \oplus \cdots \oplus \mu(L_r) \)
- Else \( \mu(H) >_{\text{lex}} \mu(L_1) \oplus \cdots \oplus \mu(L_r) \).

The proof tree for the negated atom \( \neg A \) and \( P_k \), if any, is \( \mu \)-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 \( B_\omega \) which will be used in the proof of Proposition A.16.

**Definition A.13 (Ordering \( \succ \))** Given any two atoms \( A_1, A_2 \in B_\omega \), we write \( A_1 \succ A_2 \) if either

1. \( \mu(A_1) >_{\text{lex}} \mu(A_2) \), or
2. \( \mu(A_1) = \mu(A_2) \) and \( A_1 \) is a new atom and \( A_2 \) is an old atom.

By abuse of notation, given any two atoms \( A_1, A_2 \in B_\omega \), 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 \( B_\omega \).

**Lemma A.14.** Let \( T \) be a \( \mu \)-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 \( B \leftarrow L_1 \land \ldots \land L_r \), and let \( L \) be the literal \( L_i \). If \( L_i \) is the negated atom \( \neg A \), then, since \( P \) is locally stratified w.r.t. \( \sigma \), 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 \( \mu \)-consistency of \( T \), \( \mu(B) >_{\text{lex}} \mu(L_1) \oplus \cdots \oplus \mu(L_r) \). By Lemma A.8 (ii.1), \( \mu(L_1) \oplus \cdots \oplus \mu(L_r) \geq_{\text{lex}} \mu(L_i) \) and, thus, \( \mu(B) >_{\text{lex}} \mu(L_i) \). By definition of \( \succ \), we have that \( B \succ L_i \).

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

Finally, if the predicate of \( B \) is new and \( \gamma \) is not \( \sigma \)-max derived then \( \gamma \) is a descendant of a clause that has not been derived by folding and, thus, the predicate of \( L_i \) is old. By \( \mu \)-consistency, \( \mu(B) \geq_{\text{lex}} \mu(L_i) \) 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 \( \omega \)-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 \( \mu \)-consistent proof tree for \( A \) and \( P_d \).
Proof. Let us consider a proof tree $T$ for $A$ and $P_d$ such that $\text{min-weight}(A) = \text{weight}(\sigma(A), T)$. We want to show that $T$ is $\mu$-consistent. That tree $T$ can be depicted as in Figure 3. That tree has been constructed by using at the top the clause $\gamma$ and a valuation $v$ such that $v(\gamma)$ is of the form $A \leftarrow L_1 \land \ldots \land L_n$.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{proof_tree.png}
\caption{A proof tree $T$ for $A$ and $P_d$ such that $\text{min-weight}(A) = \text{weight}(\sigma(A), T)$. There is a valuation $v$ and a clause $\gamma \in P_d$ such that $v(\gamma)$ is of the form: $A \leftarrow L_1 \land \ldots \land L_n$. For $i = 1, \ldots, n$, $T_i$ is a $\mu$-consistent proof tree for $L_i$ and $P_d$.}
\end{figure}

By induction on $\text{size}(T)$, we may assume that $T_1, \ldots, T_n$ are $\mu$-consistent proof trees. Since $\gamma$ 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 $\mu$-consistent.)

In order to prove the lemma we have to show the following two points:

1. If $A$ is a new atom then $\mu(A) \geq \text{lex} \, \mu(L_1) \oplus \ldots \oplus \mu(L_n)$, and
2. If $A$ is an old atom then $\mu(A) > \text{lex} \, \mu(L_1) \oplus \ldots \oplus \mu(L_n)$.

(Note that $A \leftarrow L_1 \land \ldots \land L_n$ is not an instance of a $\sigma$-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 $\mu$-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:

\[
\mu(A) = \langle \sigma(A), \text{min-weight}(A) - 1 \rangle = \langle \sigma(A), \sum_{i=1, \ldots, n} \land \sigma(L_i) = \sigma(A) \, \text{min-weight}(L_i) \rangle.
\]

Now, we consider two subcases.

Case (1.1): for $i = 1, \ldots, n$, $\sigma(A) > \sigma(L_i)$. In this case we have that:

\[
\langle \sigma(A), \sum_{i=1, \ldots, n} \land \sigma(L_i) = \sigma(A) \, \text{min-weight}(L_i) \rangle =
\langle (\sigma(A), 0) \rangle > \text{lex} \, \mu(L_1) \oplus \ldots \oplus \mu(L_n).
\]

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:

\[
\langle \sigma(A), \sum_{i=1, \ldots, n} \land \sigma(L_i) = \sigma(A) \, \text{min-weight}(L_i) \rangle = \mu(L_1) \oplus \ldots \oplus \mu(L_n),
\]

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:

\[
\mu(A) = \langle \sigma(A), \text{min-weight}(A) \rangle =
\{\text{the proof tree } T \text{ for } A \text{ and } P_d \text{ is such that } \text{min-weight}(A) = \text{weight}(\sigma(A), T)\} =
\langle \sigma(A), \sum_{i=1, \ldots, n} \land \sigma(L_i) = \sigma(A) \, \text{min-weight}(L_i) \rangle + 1.
\]

Let $M$ be the subset of $\{1, \ldots, n\}$ such that for all $j \in M$, $\sigma(L_j) = \sigma(A)$. We have that:
Given a proof tree includes the proof of Case (2), of Point (P2), and of the lemma. This last inequality holds because \( \sum_{j \in M} \text{min-weight}(L_j) + 1 \geq \text{lex} \mu(L_1) \oplus \ldots \oplus \mu(L_n). \) This concludes the proof of Case (2), of Point (P2), and of the lemma.

**Proposition A.16.** Let \( P_0 \) be a locally stratified \( \omega \)-program, \( \sigma \) 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 \( k = d, \ldots, n \):

(\text{Soundness}) if there exists a proof tree for \( A \) and \( P_k \), then there exists a proof tree for \( A \) and \( P_d \), and

(\text{Completeness}) if there exists a \( \mu \)-consistent proof tree for \( A \) and \( P_k \), then there exists a \( \mu \)-consistent proof tree for \( A \) and \( P_k \).

**Proof.** We prove the (\text{Soundness}) and (\text{Completeness}) properties by induction on \( k \).

Clearly they hold for \( k = d \).

Now, let us assume, by induction, that:

\[(\text{IndHyp}) \text{ the (Soundness) and (Completeness) properties hold for } k, \text{ with } d \leq k < n.\]

We have to show that they hold for \( k+1 \).

In order to prove that the (\text{Soundness}) and (\text{Completeness}) properties hold for \( k+1 \), it is sufficient to prove that:

(\text{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

(\text{C}) for every atom \( A \in \mathcal{B}_\omega \), if there exists a \( \mu \)-consistent proof tree for \( A \) and \( P_k \) then there exists a \( \mu \)-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:

(\text{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

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

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

(\text{ISC}) for every atom \( A' \in \mathcal{B}_\omega \) such that \( \sigma(A') < \sigma(A) \) (and thus, \( A > A' \)), 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 \( \text{size}(U) \). The inductive hypothesis is:

(\text{Isize}) for every atom \( A' \in \mathcal{B}_\omega \), for every proof tree \( U' \) for \( A' \) and \( P_{k+1} \), if \( \text{size}(U') < \text{size}(U) \) then there exists a proof tree \( T' \) for \( A' \) and \( P_k \).
Let $\eta$ 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) $\eta$ belongs to $P_k$ or (Case 2) $\eta$ 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. $\sigma$ 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 $\eta$ is derived by instantiating a variable $X$ in a clause $\gamma \in P_k$. We have that $\gamma$ is a clause of the form $A \leftarrow L_1 \wedge \ldots \wedge L_r$ and $\eta$ is of the form $(A \leftarrow L_1 \wedge \ldots \wedge L_r)[X/\llbracket s \rrbracket[X]]$ for some $s \in \Sigma$. Thus, $v(A[\llbracket s \rrbracket[X]]) = A$ and, for $i = 1, \ldots, r$, $v(L_i[\llbracket s \rrbracket[X]]) = L_i$.

Let $v'$ be the valuation such that $v'(X) = v(\llbracket s \rrbracket[X])$ 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 $\eta$ is derived by unfolding a clause $\gamma \in P_k$ w.r.t. a positive literal, say $\bar{K}$, in its body using clause $\gamma_i$. Recall that clauses $\gamma$ and $\gamma_i$ are assumed to have no variables in common (see rule R3). Without loss of generality, we may assume that: (i) $\eta$ is of the form $(A \leftarrow L_1 \wedge \ldots \wedge L_r)\bar{\theta}_i$, (ii) $\gamma$ is of the form $\bar{K} \leftarrow \bar{K} \wedge L_{q+1} \wedge \ldots \wedge L_r$, with $0 \leq q \leq r$, and (iii) $\gamma_i$ is of the form $\bar{H} \leftarrow \bar{L}_1 \wedge \ldots \wedge \bar{L}_q$, where $\bar{\theta}_i$ is an (idempotent and without identity bindings) most general unifier of $\bar{K}$ and $\bar{H}$.

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

For instance, given $\gamma$: $p(X) \leftarrow q(X,Y) \wedge s(X,Y,W)$ and $\gamma_i$: $q(Z,a) \leftarrow r(Z)$, by unfolding $\gamma$ w.r.t. $q(X,Y)$ using $\gamma_i$, we get a most general unifier $\bar{\theta}_i = \{Z/X,Y/a\}$ and the clause $\eta$: $p(X) \leftarrow r(X) \wedge s(X,a,W)$. 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 $v'(\bar{K}) = v'(\bar{H})$, 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 $v'(\bar{K})$. (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 $\eta$ is derived by unfolding a clause $\gamma \in P_k$ w.r.t. a negative literal, say $\neg \bar{K}$, 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 $\bar{\theta}_1, \ldots, \bar{\theta}_m$ and $m$ clauses $\gamma_1, \ldots, \gamma_m$ in $P_k$ such that, for $i = 1, \ldots, m$, $\bar{\theta}_i$ is a most general unifier of $\bar{K}$ and $\text{hd}(\gamma_i)$, $\bar{K} = \text{hd}(\gamma_i)\bar{\theta}_i$, and $v(\gamma_i\bar{\theta}_i)$ 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, $\gamma$ is not instantiated by the negative unfolding). Thus, $v(\eta) = A \leftarrow L_1 \wedge \ldots \wedge L_r$,
is derived from $A \leftarrow \neg(B_1 \lor \ldots \lor B_m) \land L_{m+1} \land \ldots \land L_r$ by pushing $\neg$ inside and pushing $\lor$ 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 $v'(bd(\gamma_i, \vartheta)) = M_1 \land \ldots \land M_s$, for some $i$, with $1 \leq i \leq m$. Since $\gamma_i$ 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 $T_{-K}$ for $\neg K$ and $P_k$. Thus, we can construct a proof tree $T$ for $A$ and $P_k$ from $T_{-K}, T_{m+1}, \ldots, T_r$ 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 $\eta$ of the form $A \leftarrow L_1 \land L_2 \land \ldots \land L_r$ is derived by positive folding from a clause $\gamma \in P_k$ of the form $A \leftarrow M_1' \land \ldots \land M_s' \land L_2 \land \ldots \land L_r$ using a clause $\delta \in \text{Defs}_k$ of the form $K \leftarrow M_1 \land \ldots \land M_s$. Without loss of generality, we may assume that $\overline{L}_1 = K \vartheta$, where $\vartheta$ is a substitution such that, for $i = 1, \ldots, s$, $M_i \vartheta = M_i'$. Thus, the literal $L_1$ in the body of $v(\eta)$ is $v(K \vartheta)$. We have that $\delta \in P_d$ and the definition of the head predicate of $\delta$ in $P_d$ consists of clause $\delta$ 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 $L_1 \leftarrow M_1 \land \ldots \land M_s$ (note that if $X \in \text{vars}(\eta)$ 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 $\eta$ is derived by negative folding from a clause $\gamma \in P_k$ using clauses $\delta_1, \ldots, \delta_m$ in $\text{Defs}_k$. Thus, we have that: (i) $v(\gamma)$ is of the form $A \leftarrow N_1 \land \ldots \land N_m \land L_2 \land \ldots \land L_r$, (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 $A \leftarrow \neg K \land L_2 \land \ldots \land L_r$. 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 $\mu$-consistent proof tree $T$ for $A$ and $P_k$, we prove that there exists a $\mu$-consistent proof tree $U$ for $A$ and $P_{k+1}$. 
The proof is by well-founded induction on $\succ \subseteq B_\omega \times B_\omega$. The inductive hypothesis is:

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

Let $\gamma$ be a clause in $P_k$ and $v$ be a valuation such that $v(\gamma)$ is the clause of the form $A \leftarrow L_1 \land \ldots \land L_r$ used at the root of $T$. We consider the following cases: either (Case 1) $\gamma$ belongs to $P_{k+1}$ or (Case 2) $\gamma$ 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 $\mu$-consistency of $T$ and Lemma A.14, for $i = 1, \ldots, r$, we have $A \succ L_i$. Hence, by the inductive hypotheses $(I\mu)$ and (ISC), there exists a $\mu$-consistent proof tree $U_i$ for $L_i$ and $P_{k+1}$. A $\mu$-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 $\gamma$ in $P_k$ we derive clauses $\gamma_1, \ldots, \gamma_h$ in $P_{k+1}$. For $i = 1, \ldots, h$, $\gamma_i$ is $\gamma \{X/\llbracket s_i \rrbracket X\}$, 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 $\mu$-consistency of $T$ and Lemma A.14, for $i = 1, \ldots, r$, we have $A \succ L_i$. Hence, by the inductive hypotheses $(I\mu)$ and (ISC), for $i = 1, \ldots, r$, there exists a $\mu$-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 $\mu$-consistent because: (i) by $(I\mu)$, we have that $U_1, \ldots, U_r$ are $\mu$-consistent, (ii) $\gamma_i$ is $\sigma$-max derived iff $\gamma$ is $\sigma$-max derived, and (iii) since $T$ is $\mu$-consistent, we have that if $\gamma$ is not $\sigma$-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 $\gamma$ 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 $\gamma$. Hence, there exists a clause $\gamma_i$ in (a variant of) $P_k$ such that: (i) $v(\gamma_i)$ is of the form $L_1 \leftarrow M_1 \land \ldots \land M_q$, (ii) $v(\eta_i)$ is $A \leftarrow M_1 \land \ldots \land M_q \land L_2 \land \ldots \land L_r$, and (iii) $v(\gamma_i)$ is the clause which is used for constructing the children of $L_1$ in $T$. By the $\mu$-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 $(I\mu)$ and (ISC), for $i = 1, \ldots, q$, there exists a $\mu$-consistent proof tree $V_i$ for $M_i$ and $P_{k+1}$ and, for $i = 2, \ldots, r$, there exists a $\mu$-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 $\mu$-consistent. There are two cases: (a) and (b).

Case (a): in this first case we assume that $A$ is new and $\eta_i$ is not $\sigma$-max derived. Since $T$ is $\mu$-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 $\eta_i$ is $\sigma$-max derived. We have two subcases (b.1) and (b.2).

Subcase (b.1): $A$ is old. Since $T$ is $\mu$-consistent, we get that $\mu(A) >_{lex} \mu(L_1) \land \ldots \land \mu(L_r)$ and $\mu(L_1) >_{lex} \mu(M_1) \land \ldots \land \mu(M_q)$. By Lemma A.8 (ii.2) we get $\mu(A) >_{lex} \mu(M_1) \land \ldots \land \mu(M_q)$.

Subcase (b.2): $\eta_i$ is $\sigma$-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): $\eta_i$ is $\sigma$-max derived, $A$ is new, and $\gamma$ is $\sigma$-max derived, and Subcase (b.2.2): $\eta_i$ is $\sigma$-max derived, $A$ is new, and $\gamma$ is not $\sigma$-max derived.
Subcase (b.2.1). Since $T$ is $\mu$-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 $\mu$-consistent and $L_1$ is old, we get: (i) $(\mu(L_1) >_{lex} \mu(M_1) \oplus \ldots \oplus \mu(M_q))$, and (ii) $\tau_2(\mu(L_1)) > 0$. Thus, $\sigma_i > \sigma(L_j)$. Therefore, $(\mu(L_1) \geq \sigma_1(\mu(L_2) \oplus \ldots \oplus \mu(L_r))).$ From (i), (ii), and (iii), by Lemma A.8 (ii.3), we get: $\mu(L_1) \oplus \mu(L_2) \oplus \ldots \oplus \mu(L_r) >_{lex} \mu(L_1) \oplus \ldots \oplus \mu(M_q) \oplus \mu(L_2) \oplus \ldots \oplus \mu(L_r)$. Since $T$ is $\mu$-consistent, we have that $\mu(A) >_{lex} \mu(L_1) \oplus \ldots \oplus \mu(L_r)$, and by (iv) 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 $\mu$-consistent proof tree.

Case 2.3 ($P_{k+1}$ is derived from $P_k$ by using rule R4.) Suppose that we unfold $\gamma$ 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 $\gamma$ 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 $\gamma$. 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) $A \not\leftarrow \neg A_1$ because by hypothesis the proof tree $T$ is $\mu$-consistent, and (ii) $\sigma(\neg A_1) > \sigma(\overline{R_i})$, because $P_k$ is locally stratified w.r.t. $\sigma$.

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 $\sigma(A) > \sigma(A_i) \geq \sigma(R_i)$ and, thus, $A > \overline{R_i}$. In Case (ii) we have that $\sigma(A) > \sigma(A_i) \geq \sigma(\neg C_i)$ and, thus, $\sigma(A) > \sigma(C_i) = \sigma(\overline{R_i})$ and $\mu(A) > \mu(\overline{R_i})$. Hence, $A > \overline{R_i}$. Thus, in both cases $A > \overline{R_i}$.

Since $A > \overline{R_i}$, by the inductive hypotheses (I$\mu$) and (ISC), we have that, for $i = 1, \ldots, m$, there exists a $\mu$-consistent proof tree $V_i$ for $\overline{R_i}$ and $P_{k+1}$. By the $\mu$-consistency of $T$, for $i = 2, \ldots, r$, there exists a $\mu$-consistent proof tree $U_i$ for $L_i$ and $P_{k+1}$. By the definition of rule R4, there exists a clause $\eta_p$ among the clauses $\eta_1, \ldots, \eta_s$ derived from $\gamma$, such that $v(\eta_p)$ is of the form $A \leftarrow \overline{R_1} \land \ldots \land \overline{R_m} \land L_2 \land \ldots \land L_r$. (To see this, recall that by pushing $\neg$ inside and $\lor$ outside, from $\neg((C_1 \land C_2) \lor (D_1 \land D_2))$ we get $(\overline{C_1} \lor \overline{C_2}) \lor (\overline{D_1} \lor \overline{D_2})$.)

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 $\mu$-consistent we need to consider two cases. In the first case, we assume that $A$ is old or $\eta$ is $\sigma$-max derived. Thus, in this case, also $\gamma$ is $\sigma$-max derived. By $\mu$-consistency of $T$, we have $\mu(A) >_{lex} \mu(L_1) \oplus \ldots \oplus \mu(L_r).$ By local stratification of $P_k$ and by Lemma A.10, $\mu(L_1) \geq_{lex} \mu(\overline{R_1}) \oplus \ldots \oplus \mu(\overline{R_m}).$ Therefore, by Lemma A.8 (ii.2), $\mu(A) >_{lex} \mu(\overline{R_1}) \oplus \ldots \oplus \mu(\overline{R_m}) \oplus \mu(L_2) \oplus \ldots \oplus \mu(L_r)$ and $U$ is $\mu$-consistent.

In the second case, $A$ is new and $\eta$ is not $\sigma$-max derived. As a consequence, also $\gamma$ is not $\sigma$-max derived. By $\mu$-consistency of $T$ we have $\mu(A) \geq_{lex} \mu(L_1) \oplus \ldots \oplus \mu(L_r).$ By local stratification of $P_k$ and by Lemma A.10, $\mu(L_1) \geq_{lex} \mu(\overline{R_1}) \oplus \ldots \oplus \mu(\overline{R_m})$ and, by Lemma A.8 (ii.2), $\mu(A) \geq_{lex} \mu(\overline{R_1}) \oplus \ldots \oplus \mu(\overline{R_m}) \oplus \mu(L_2) \oplus \ldots \oplus \mu(L_r).$ Therefore, $U$ is $\mu$-consistent.

Case 2.4 ($P_{k+1}$ is derived from $P_k$ by using rule R5.) Suppose that the clause $\gamma$ is removed from $P_k$ by subsumption. Hence, there exists a clause $\gamma_1$ in $P_k - \{\gamma\}$ and a valuation $v'$ such that $v'(\gamma_1)$ is of the form $A \leftarrow \gamma$. The clause $\gamma_1$ 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 $\mu$-consistent, that is, $\mu(A) >_{lex} \mu(true)$. We have to prove that $\mu(A) >_{lex} (0,0)$. We have the following three cases:
Thus, $\mu$ (the form $K$ of generality, by the definition of rule R7 and the commutativity of $\mu$). (i) $U_i \delta$ transformation sequence. Proof tree for thus, we have that there exists a $\mu$. Because, as we have shown above, $K_i$ for $\delta$ $A$ children of $\mu$.

2.5 (b.2) Case $\tilde{\mu}$ (T $\tilde{\mu}$ is of the form G. Moreover, by Lemma A.8 (ii.5). Then, there is no proof tree for $\tilde{\mu}$ ($\tilde{\mu}$ and, thus, $\sigma(A) > 0$). Hence, $\mu(A) = \text{def} \langle \sigma(A), \text{min-weight}(A) - 1 \rangle > \text{lex}(0, 0)$.

This concludes the proof tree $U$ is $\mu$-consistent.

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

Thus, there is a valuation $v$ such that $v(A) = A$ 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 $i = 1, \ldots, q$, $L_i = v(\tilde{L}_i)$ and for $q+1, \ldots, r$, $L_i = v(\tilde{L}_i)$. By the induction hypothesis (IndHyp) there exist proof trees for $v'(\tilde{L}_1), \ldots, v'(\tilde{L}_q)$ and $P_k$, for some valuation $v'$ such that, for $i = 1, \ldots, q$, $v'(\tilde{L}_i) \eta = v(\tilde{L}_i)$. Let $K$ be $v'(\tilde{K} \tilde{\eta})$.

Since $\delta \in P_d$ and $M(P_d) = \delta$, by Theorem A.4 and Definition A.12, there is a $\mu$-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 $\mu$-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 $\gamma$ is $\sigma$-max derived. Thus, by the $\mu$-consistency of the proof tree $T$, we have that $\mu(A) > \text{lex} \mu(L_1) \oplus \cdots \oplus \mu(L_q) \oplus \mu(L_{q+1}) \oplus \cdots \oplus \mu(L_r)$.

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

Moreover, by Lemma A.8 (ii.5), $\mu(A) > \text{lex} \mu(K)$. Thus, $A \succ K$ and, by the inductive hypothesis (1)$\mu$, there exists a $\mu$-consistent proof tree $U_K$ for $K$ and $P_{k+1}$. By the $\mu$-consistency of $T$ and Lemma A.14, for $i = q+1, \ldots, r$, we have $A \succ L_i$. Hence, by the inductive hypotheses (1)$\mu$ and (ISC), for $i = q+1, \ldots, r$, there exists a $\mu$-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 $\mu$-consistent because, as we have shown above, $\mu(A) > \text{lex} \mu(K) \oplus \mu(L_{q+1}) \oplus \cdots \oplus \mu(L_r)$.

Case 2.6 ($P_{k+1}$ is derived from $P_k$ by using rule R7.) Suppose that we fold $\gamma$ using clauses $\delta_1, \ldots, \delta_q$, belonging to (a variant of) $\text{Defs}_{\tilde{L}_q}$, and we derive a clause $\eta$ in $P_{k+1}$. Without loss of generality, by the definition of rule R7 and the commutativity of $\land$, we may assume that (i) $v(\gamma)$ is of the form $A \leftarrow L_1 \land \ldots \land L_q \land L_{q+1} \land \ldots \land L_r$, (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 $A \leftarrow \neg K \land L_{q+1} \land \ldots \land L_r$. 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
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 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’.


