An Operational Semantics of Starlog

Lunjin Lu and John G. Cleary

Department of Computer Science
University of Waikato
Hamilton, New Zealand
{lunjin,jcleary}@cs.waikato.ac.nz

Abstract

Starlog is a temporal logic programming language that supports declarative specification of reactive systems, input-output behaviour and destructive updates. This paper presents an operational semantics for Starlog. Its correctness and completeness with respect to a model semantics are proved.
Keywords: Operational Semantics; Constraint Logic Programs; Temporal Logic Programs; Stratification

1 Introduction

Starlog is a temporal logic programming (TLP for short) language. It evolved from such applications as simulation [9] and deductive databases [28]. These applications require temporal relationships between objects to be specified in a precise and direct manner. It is this requirement which dictates the design and implementation of Starlog.

While Starlog is similar in many ways to deductive databases it is intended as a full general programming language. This means in particular that two assumptions of deductive databases are untrue. The first is that the logic program can be finitely stratified on the predicate names. This is replaced by a more general ordering using integer timestamps and predicate names. The second assumption that is violated is the Datalog assumption that terms can only be of some finite depth. Starlog allows both unbounded terms and constraints. This paper is intended to provide a precise operational semantics for such a general bottom-up logic programming language.

Starlog uses the syntax of the constraint logic programs (CLP hereafter) [21]. Starlog is a CLP language with arithmetic constraints over integers and equality/disequality constraints over terms. This is different from other TLP languages that are based on a particular temporal logic [1, 2, 4, 14, 32, 47]. Unlike other CLP languages such as CLP(R) [22] and BNR Prolog [33], Starlog programs are executed bottom-up. This is suitable for its intended applications which often use the specification of a real world system to construct a temporal model of the system.

Over the past decade, a working prototype implementation of Starlog has been developed and challenging applications have been written in Starlog. This paper provides Starlog with a formal operational semantics. Its purpose is two fold. Firstly, while the present implementation works experimentally, it is important that the implementation be verified with respect to a formal semantics. Secondly, the present implementation is rather inefficient. To improve its efficiency, it is necessary to perform various semantic based program analyses, for which a formal operational semantics is a necessity.

This paper presents an operational semantics for Starlog. Its correctness and completeness with respect to its model semantics are given. The operational semantics is a bottom-up execution mechanism which deals with negative literals in the same way as positive ones.

As programs in TLP languages such as Templog [1], Tokio [2], Temporal Prolog [14], Tempura [32] and Chronology [47] can be translated into CLP programs [7, 34], our operational semantics offers a bottom-up execution mechanism for these TLP languages which use extensions of the SLD resolution as operational semantics.

Temporality is expressed in Starlog by explicitly timing truth. The operational semantics works on the class of Starlog programs that can be stratified in terms of time and predicate symbols. It generates in time order those facts whose ground instances are in the model semantics of the program. It repeatedly generates a fact using the program and transforms the program. Time-orderedness and stratification guarantee the correctness and the completeness of the operational semantics. Time-orderedness is essential in temporal applications such as simulation.

The rest of this paper is organised as follows. Section 2 introduces a subset of Starlog. The subset is the core of Starlog and is chosen to simplify the presentation of the paper. Section 3 introduces the notion of temporally stratified Starlog programs and defines its model semantics, and section 4 presents the operational semantics and gives its correctness and completeness. Section 5 concludes the paper and compares our operational semantics with related work. We assume that the reader is familiar with the terminology of constraint logic programming [21].

2 Starlog Language

Starlog was developed for specification and implementation of applications which require temporal reasoning. Starlog doesn't directly support temporal operators. However, most temporal operators can be programmed in Starlog as indicated in [7]. Moreover, Starlog allows more explicit temporal relationships to be expressed directly.

As a CLP language, Starlog can adopt any model semantics developed for the CLP scheme. This paper defines the model semantics of Starlog based on the stable model semantics [17]. Unlike other CLP languages such as [22, 10, 12, 11, 30, 5, 46], Starlog uses an explicit parameter for time. This can be thought of putting timestamps on truth values.

2.1 Syntax

Terms and constraints are formed as in other CLP(X) languages. Constraints consist of arithmetic constraints over integers and equality/disequality constraints over terms. We will use $\mathcal{D}$ to denote the underlying domain structure which interprets arithmetic constraints over integers and equality/disequality constraints over terms in the usual way [6, 8]. There is no complete constraint solver for arbitrary integer arithmetic constraints. However, there are powerful decidable subsets of integer arithmetic constraints [27, 31, 19, 39, 20, 44, 41, 18]. This paper focuses on decidable subsets of integer arithmetic constraints and assumes that $\mathcal{D}$ is satisfaction complete. The assumption, which can be relaxed, helps us to separate the issue of the completeness of the constraint solver from that of the completeness of the operational semantics itself. Substituting a decidable domain of integer arithmetic constraints for $\mathcal{D}$ will result in a particular instance of Starlog. An atom is defined to be of the form $p(s_1, \cdots, s_n)@t$ where t, called the timestamp, is a term consisting of variables, integers and arithmetic operators while each s_i is an arbitrary term. A literal is either an atom or negation of an atom. A clause is of the form $\mathbf{h} \leftarrow \delta, \mathbf{L}$ where δ is a constraint, $\mathbf{h}$ an atom and $\mathbf{L}$ a conjunction of literals. A Starlog program is a finite set of clauses. A clause without any body literal but possibly including constraints is called a fact while other clauses are called rules.

2.2 Causality

Causality is natural in temporal reasoning and is also a useful assumption which simplifies Starlog programming. Some other TLP languages also assume the causality of programs [4]. Causality means that no truth in the past is defined in terms of truth in the future. Thus, the following clause fails in Starlog.

```
retrospective_reasoning@T :-
        T=S-1000, current_finding@S.
```

Formally, a clause is causal if, in any $\mathcal{D}$-model of the clause, the timestamp of its head is no less than the timestamp of any literal in its body. Starlog implicitly adds causality constraints to program clauses.

2.3 Examples

Time in Starlog is discrete and positive. The following program defines a predicate `even` that is true at even time points and thus generates even numbers.

```
% even numbers program.

even@0.
even@T :-  T=S+1, not(even@S).
```

The following program generates prime numbers as a time sequence using the predicate prime. It also generates non-prime numbers by predicate mult. $T >= 2$, $T = J * K$, $K >= J$ and $J >= 2$ are constraints over integers.

```
% prime numbers program.

prime@T :-  T>=2, not(mult@T).
mult@T  :-  T=J*K, K>=J, J>=2, prime@J.
```

2.4 Normalised Programs

In the sequel, we shall only be concerned with normalised programs. A normalised program is a set of normalised clauses. A clause is normalised if its head and every atom occurring in its body are of the form $p(X_1, \cdots, X_n)@T$ where $X_1, \cdots, X_n$ and T are different variables. It is obvious that corresponding to each program, there is a semantically equivalent normalised program. The causality constraints implicit in a normalised Starlog program can be easily added.

2.5 Notations

Let $C = (h@T \leftarrow \delta, \mathbf{L})$, and t be an integer. Define $C^{\geq t} \stackrel{def}{=} (h@T \leftarrow (T \geq t), \delta, \mathbf{L})$ and $C^{<t} \stackrel{def}{=} (h@T \leftarrow (T < t), \delta, \mathbf{L})$. Let $\mathcal{P}$ be a program. $\mathcal{P}^f$ denotes the set of facts in $\mathcal{P}$ and $\mathcal{P}^r$ the set of rules in $\mathcal{P}$. $\mathcal{P} = \mathcal{P}^f \cup \mathcal{P}^r$. We sometimes write a clause as $h@T \leftarrow \delta, \wedge_{j \in J} a_j@S_j, \wedge_{k \in K} not(a_k@S_k)$ with J and K being disjoint sets of indices. $\mathcal{P}^-$ denotes the set of those rules in $\mathcal{P}^r$ which have only negative literals, and $\mathcal{P}^+$ denotes $\mathcal{P}^r \setminus \mathcal{P}^-$. Rules in $\mathcal{P}^-$ are called negative and those in $\mathcal{P}^+$ positive. Let δ be a constraint. Define $sat(\delta) \stackrel{def}{=} (\mathcal{D} \models \exists.\delta)$. $sat(\delta)$ is true iff δ is satisfiable with respect to $\mathcal{D}$. Let Q be a set of clauses. Define

$$[Q]_{\mathcal{D}} \stackrel{def}{=} \{\mu(\mathbf{h} \leftarrow \mathbf{L}) \mid (\mathbf{h} \leftarrow \delta, \mathbf{L}) \in Q \wedge (\mu \text{ is a valuation}) \wedge \mathcal{D} \models \mu(\delta)\}$$

We write $[\{S\}]_{\mathcal{D}}$ as $[S]_{\mathcal{D}}$ for simplicity.

3 Stratification and Model Semantics

This section defines the class of temporally stratified programs and their model semantics, and lays the technical ground for the operational semantics of temporally stratified programs including negation.

3.1 Temporally Stratified Programs

Stratification has been a useful notion in formulating the semantics of logic programs with negation [3, 15, 38, 37]. The idea of stratification is to disallow recursion through negation. In other words, stratification makes it impossible for a predicate to recursively invoke itself through negation. This is guaranteed by requiring that any predicate symbol occurring negatively in the body of a clause belongs to a lower stratum than the head predicate symbol and any predicate symbol occurring positively in the body belongs to a stratum no higher than the head predicate symbol.

The timestamps in Starlog programs relax the above condition for stratification in that recursion through negation is allowed provided it is through decreasing timestamps. Let $\prod$ be the set of the predicate symbols in $\mathcal{P}$ and Nat be the set of natural numbers. Let $strat$ be a function from $\prod$ to Nat. We extend $strat$ as follows. $strat(p(\tilde{s})) \stackrel{def}{=} strat(p)$ and $strat((\mathbf{h} \leftarrow \delta, \mathbf{L})) \stackrel{def}{=} strat(\mathbf{h})$.

Definition 1. A program $\mathcal{P}$ is temporally stratified if there is a function $strat : \prod \mapsto \mathsf{Nat}$ such that, for every rule $h@T \leftarrow \delta, \wedge_{j \in J} a_j@S_j, \wedge_{k \in K} not(a_k@S_k)$ in $\mathcal{P}$, for every $j \in J$, either $\mathcal{D} \models (\delta \rightarrow (T > S_j))$ or $strat(a_j) \leq strat(h)$, and for every $k \in K$, either $\mathcal{D} \models (\delta \rightarrow (T > S_k))$ or $strat(a_k) < strat(h)$.

The above definition augments the traditional predicate stratification in the literature with time stratification. Procedure calls are primarily stratified on timestamps and secondarily on predicate symbols. It ensures that recursive calls through negation in a temporally stratified program involve time decrements. Under the assumption that $\mathcal{D}$ is satisfaction complete, temporal stratifiability is decidable. An algorithm for finding a predicate stratification function $strat$ for a logic program in the literature, such as that in [45] (page 134), can be readily adapted for Starlog.

3.2 Model Semantics

We first recall the stable model semantics for logic programs [17] and then define model semantics of temporally stratified Starlog programs. Let $\mathcal{G}$ be a logic program consisting of a set of ground clauses and $\mathcal{M}$ be a set of ground atoms. Then the Gelfond-Lifschitz transformation is defined as follows.

$$GL(\mathcal{G}, \mathcal{M}) \stackrel{def}{=} \{H \leftarrow pos(\mathbf{L}) \mid (H \leftarrow \mathbf{L}) \in \mathcal{G} \wedge \mathcal{M} \models neg(\mathbf{L})\}$$

where $pos(\mathbf{L})$ is the conjunction of positive literals in $\mathbf{L}$ and $neg(\mathbf{L})$ is the conjunction of negative literals in $\mathbf{L}$. $GL(\mathcal{G}, \mathcal{M})$ is a definite logic program obtained by removing those clauses in $\mathcal{G}$ whose bodies contain the negation of an atom in $\mathcal{M}$ and deleting negative literals in other clauses in $\mathcal{G}$.

A set $\mathcal{M}$ of atoms is a stable model of $\mathcal{G}$ if $\mathcal{M}$ is the least model of $GL(\mathcal{G}, \mathcal{M})$. If $\mathcal{G}$ is locally stratified then $\mathcal{G}$ has a unique stable model which is also the least model of $\mathcal{G}$. For locally stratified programs, the stable model semantics coincides with the perfect model semantics [36, 35] and the well-founded semantics [16].

Let $\mathcal{P}$ be a temporally stratified Starlog program. $[\mathcal{P}]_{\mathcal{D}}$ is a locally stratified logic program. Therefore, $[\mathcal{P}]_{\mathcal{D}}$ has one stable model which is also the least model of $[\mathcal{P}]_{\mathcal{D}}$. We take the unique stable model of $[\mathcal{P}]_{\mathcal{D}}$ as the canonical model semantics of $\mathcal{P}$, denoted as $CM(\mathcal{P})$. The operational semantics of temporally stratified Starlog programs computes a representation of $CM(\mathcal{P})$ in time order. A representation of $CM(\mathcal{P})$ is a set $\mathcal{F}$ of facts such that $[\mathcal{F}]_{\mathcal{D}} = CM(\mathcal{P})$.

Let $\mathbf{f}$ be a fact. Elements of $[\mathbf{f}]_{\mathcal{D}}$ are called ground instances of $\mathbf{f}$. Note that ground instances of $\mathbf{f}$ has an empty body and can be thought of as ground atoms. Let $\mathcal{M}$ be a set of ground atoms. We say that $\mathbf{f}$ is contained in $\mathcal{M}$ if all ground instances of $\mathbf{f}$ are in $\mathcal{M}$, i.e., $[\mathbf{f}]_{\mathcal{D}} \subseteq \mathcal{M}$.

3.3 Approximate Success Time

Let $\mathcal{P}$ be the current program. A key step in the operational semantics is to determine the minimum timestamp that an atom in $CM(\mathcal{P})$ has. Because of negation, the minimum timestamp can not be determined by the facts in $\mathcal{P}$ alone. Rules have to be taken into account. For example, the prime numbers program doesn't contain any fact and yet $\mathsf{prime}@T \leftarrow T \geq 2, T < 4$ is in its model semantics.

As we require the operational semantics to generate facts in time order, we naturally expect that such a fact be generated from a clause that has the smallest success time where the success time m_C of a clause C is defined as follows. Let C be $h@T \leftarrow \delta, \mathbf{L}$. Then

$$\mathsf{m}_C \stackrel{def}{=} \min\{t \mid \mathcal{D} \models \mu(\delta \wedge (T = t)) \wedge CM(\mathcal{P}) \models \mu(\mathbf{L}) \wedge (\mu \text{ is a valuation})\}$$

m_C is not computable but serves as a useful reference. The operational semantics uses a conservative approximation $\hat{\mathsf{m}}_C$ to m_C to choose a program clause from which the next fact is generated.

$$\hat{\mathsf{m}}_C \stackrel{def}{=} min\{t \mid sat(\delta \wedge (T = t))\}$$

$\hat{\mathsf{m}}_C$ is a conservative approximation to m_C in that $\hat{\mathsf{m}}_C$ never exceeds m_C. $\hat{\mathsf{m}}_C$ is determined by C alone. This is in contrast to m_C which also depends on other clauses in $\mathcal{P}$. $\hat{\mathsf{m}}_C$ is computable as $\mathcal{D}$ is satisfaction complete. If C is a fact then $\hat{\mathsf{m}}_C = \mathsf{m}_C$. We define approximate success time for a set of clauses as the minimum of the approximate success times of the clauses in the set.

3.4 Extracting Facts from Negative Rules

Time-orderedness requires that each time a fact is generated it has the smallest success time. If the smallest approximate success time of the program happens to be that of a fact in the program then the situation is simple. It is also the smallest success time of any fact in the model semantics of the program.

The situation becomes more complicated when the approximate success time of a rule is smaller than those of the facts in the program. In this case, there is a possibility that facts with success times smaller than those of the facts in the program can be generated by rules in the program, as shown later. We will develop a method to extract such facts from the program. The method is based on a few properties of temporally stratified programs that are detailed below. The following lemma states that if the success time of a fact in the model semantics of the program is smaller than those of the facts in the program then a negative rule in the program derives the fact.

Lemma 2. *Let $\mathcal{P}$ be a program and $a@t$ be a ground atom such that $a@t \in CM(\mathcal{P})$ and $t < \hat{\mathsf{m}}_{\mathcal{P}f}$. Then there are a negative rule $C = (h@T \leftarrow \delta, \wedge_{k\in K} not(\mathbf{a}_k))$ and a valuation ν such that $\mathcal{D} \models \nu(\delta)$, $\nu(\mathbf{a}_k) \notin CM(\mathcal{P})$ and $\nu(T) \leq t$.*

Proof. By contradiction. Assume there were no negative rule $C = (h@T \leftarrow \delta, \wedge_{k\in K} not(\mathbf{a}_k))$ and valuation ν such that $\mathcal{D} \models \nu(\delta)$, $\nu(\mathbf{a}_k) \notin CM(\mathcal{P})$ and $\nu(T) \leq t$. Then, $t' > t$ for each fact $h'@t'$ in $GL([\mathcal{P}]_{\mathcal{D}}, CM(\mathcal{P}))$. By the causality requirement, $a@t \notin CM(\mathcal{P})$ as $CM(\mathcal{P})$ is the least Herbrand model of $GL([\mathcal{P}]_{\mathcal{D}}, CM(\mathcal{P}))$. □

According to lemma 2, if all negative rules have an approximate time no less than $\hat{\mathsf{m}}_{\mathcal{P}f}$ then the success time of the next fact to generate is $\hat{\mathsf{m}}_{\mathcal{P}f}$.

When $\hat{\mathsf{m}}_{\mathcal{P}f} > \hat{\mathsf{m}}_{\mathcal{P}-}$, a fact can be extracted from a negative rule, as is stated in the following lemma.

Lemma 3. *Let $\mathcal{P}$ be a program such that $\hat{\mathsf{m}}_{\mathcal{P}f} > \hat{\mathsf{m}}_{\mathcal{P}-}$ and $C = (h@T \leftarrow \delta, \wedge_{k\in K} not(a_k@S_k))$ be a negative rule in $\mathcal{P}$ such that $\hat{\mathsf{m}}_C = \hat{\mathsf{m}}_{\mathcal{P}-}$ and $strat(C) \leq strat(C')$ for any other negative rule C' in $\mathcal{P}$ with $\hat{\mathsf{m}}_{C'} = \hat{\mathsf{m}}_C$. Then $[h@T \leftarrow \delta \wedge (T = \hat{\mathsf{m}}_{\mathcal{P}-})]_{\mathcal{D}} \subseteq CM(\mathcal{P})$.*

Proof. First consider the simple case where C is the only one negative rule whose approximate success time is equal to $\hat{\mathsf{m}}_{\mathcal{P}-}$. Let ν be an arbitrary valuation such that $\mathcal{D} \models \nu(\delta \wedge (T = \hat{\mathsf{m}}_{\mathcal{P}-}))$. The temporal stratification and causality requirements ensure that for each $k \in K$, either (a) $\nu(S_k) < \hat{\mathsf{m}}_{\mathcal{P}-}$ or (b) $(\nu(S_k) = \hat{\mathsf{m}}_{\mathcal{P}-}) \wedge (strat(a_k) < strat(h))$. In the case (a), $\nu(a_k@S_k) \notin CM(\mathcal{P})$ by lemma 2. In the case (b), we also have $\nu(a_k@S_k) \notin CM(\mathcal{P})$ as shown in the following. We have $\mathsf{Pred}(a_k) \neq \mathsf{Pred}(h)$ where $\mathsf{Pred}(a)$ is the predicate symbol of the atom a. If $\mathsf{Pred}(a_k)$ is not defined by any positive rule then every clause for $\mathsf{Pred}(a_k)$ has an approximate success time greater than $\hat{\mathsf{m}}_{\mathcal{P}-}$ because C is the only negative rule whose approximate success time is equal to $\hat{\mathsf{m}}_{\mathcal{P}-}$. This implies $\nu(a_k@S_k) \notin CM(\mathcal{P})$. Now suppose that $\mathsf{Pred}(a_k)$ be defined by a positive rule $a_k@S \leftarrow \delta', \mathbf{L}$. Either $\mathbf{L}$ contains a positive call to a predicate q other than $\mathsf{Pred}(h)$, implying $\nu(a_k@S_k) \notin CM(\mathcal{P})$ because the approximate success time of any negative rule for q is greater than $\hat{\mathsf{m}}_{\mathcal{P}-}$, or every positive literal in $\mathbf{L}$ is a call to $\mathsf{Pred}(h)$, also implying $\nu(a_k@S_k) \notin CM(\mathcal{P})$ because such a call must involve a time decrement to satisfy the temporal stratification requirement. Thus,$\nu(a_k@S_k) \notin CM(\mathcal{P})$ in the case (b). So, $(\mathcal{D} \models \nu(\delta \wedge (T = \hat{\mathsf{m}}_{\mathcal{P}-}))) \rightarrow \nu(a_k@S_k) \notin CM(\mathcal{P})$ for any valuation ν, which implies $[h@T \leftarrow \delta \wedge (T = \hat{\mathsf{m}}_{\mathcal{P}-})]_{\mathcal{D}} \subseteq GL([\mathcal{P}]_{\mathcal{D}}, CM(\mathcal{P}))$.

Now suppose that there be more than one negative rules whose approximate success times equal $\hat{\mathsf{m}}_{\mathcal{P}-}$. Each such a clause defines a predicate in a stratum. Consider a negative rule $C = (h@T \leftarrow \delta, \wedge_{k\in K} not(a_k@S_k))$ that has the lowest stratum among these negative rules. The same reasoning as in the above paragraph leads to $[h@T \leftarrow \delta \wedge (T = \hat{\mathsf{m}}_{\mathcal{P}-})]_{\mathcal{D}} \subseteq GL([\mathcal{P}]_{\mathcal{D}}, CM(\mathcal{P}))$.

The lemma follows because $CM(\mathcal{P})$ is a model of $GL([\mathcal{P}]_{\mathcal{D}}, CM(\mathcal{P}))$. □

4 Operational Semantics

This section presents an operational semantics for temporally stratified programs.

Let $\mathcal{P}$ be the program. The operational semantics enumerates a representation of $CM(\mathcal{P})$ in time order. It repeatedly generates a fact $\mathbf{f}$ and then uses $\mathbf{f}$ to transform the current program $\mathcal{P}$ into a new program $\mathcal{P}'$. The transformation is done in such a way that any ground fact in $CM(\mathcal{P})$ is either a ground instance of $\mathbf{f}$ or in $CM(\mathcal{P}')$, and that any fact in $CM(\mathcal{P}')$ is also in $CM(\mathcal{P})$. The operational semantics keeps track of a timer. The timer records the minimum timestamp that the next fact to generate could have. The operational semantics ensures that all the facts with a timestamp smaller than the timer and in the model semantics of the original program has been generated and that all other facts in the model semantics of the original program are in the model semantics of the current program.

As the operational semantics discards generated facts, a newly generated fact must be propagated through each rule that matches the fact. The propagation results in several clauses which replace the rule through which the fact is propagated. Let $C = (h@T \leftarrow \delta, \wedge_{j\in J} a_j@S_j, \wedge_{k\in K} not(a_k@S_k))$ be a rule in $\mathcal{P}^r$, $\mathbf{f} = (\mathbf{h}' \leftarrow \delta')$ be the generated fact, and t be the value of the timer when $\mathbf{f}$ is generated. Then C is replaced by a set of clauses obtained by (1) replacing each $a_l@S_l$ in the body of C with $(a_l@S_l \wedge (S_l \geq t) \vee (a_l@S_l = \rho_l(\mathbf{h}')\wedge\rho_l(\delta')))$ where ρ_l is a renaming substitution, (2) converting the resulting body into its disjunctive normal form and throwing away conjuncts with unsatisfiable constraints, and (3) for each remaining conjunct, producing a clause with $h@T$ as its head and the conjunct as its body. Formally,

$$prop((h@T \leftarrow \delta, \wedge_{j\in J} a_j@S_j, \wedge_{k\in K} not(a_k@S_k)), (\mathbf{h}' \leftarrow \delta'), t)$$
$$\stackrel{def}{=} \{h@T \leftarrow \gamma, \mathbf{L} \mid (\gamma, \mathbf{L}) \in KK \wedge sat(\gamma)\}$$

with $KK = DNF\begin{pmatrix} \delta, \bigwedge_{j\in J}(S_j \geq t \wedge a_j@S_j \vee (a_j@S_j = \rho_j(\mathbf{h}')) \wedge \rho_j(\delta')), \\ \bigwedge_{k\in K} not(S_k \geq t \wedge a_k@S_k \vee (a_k@S_k = \rho_k(\mathbf{h}')) \wedge \rho_k(\delta')) \end{pmatrix}$
where $DNF(F)$ is the disjunctive normal form of F.

The definition of $prop(C, \mathbf{f}, t)$ doesn't distinguish atoms that match with the generated fact from those that do not. If $a_l@S_l$ doesn't match with $\mathbf{h}'$ then $(a_l@S_l \wedge (S_l \geq t) \vee (a_l@S_l = \rho_l(\mathbf{h}')) \wedge \rho_l(\delta')$ is equivalent to $a_l@S_l \wedge (S_l \geq t)$ and the effect is to strengthen the constraint part of C.

Let a be an atom. We define $\tau_{a,\mathcal{P}}$ as the minimum of the approximate success times of the clauses in $\mathcal{P}$ whose heads have the same predicate symbol as a.

Algorithm 4 *Given a temporally stratified program $\mathcal{P}$ and its predicate stratification function strat, the algorithm enumerates in time order a representation of $CM(\mathcal{P})$.*

- *Initialisation.*

$$t := 0$$
$$\mathcal{P} := \{\mathbf{h} \leftarrow \delta, \mathbf{L} \mid (\mathbf{h} \leftarrow \delta, \mathbf{L}) \in \mathcal{P} \wedge sat(\delta)\}$$

– *While* $(\mathcal{P}^- \cup \mathcal{P}^f) \neq \emptyset$ *do*

(Ia) $t := \hat{\mathsf{m}}_{(\mathcal{P}^f \cup \mathcal{P}^-)}$

(Ib) $\mathcal{P} := \mathcal{P}^f \cup \mathcal{P}^+ \cup \bigcup_{C \in \mathcal{P}^-} extr\_t(C, \mathcal{P})$ *where,*

$$extr\_t((h@T \leftarrow \delta, \wedge_{k\in K} not(a_k@S_k)), \mathcal{P}) \stackrel{\text{def}}{=} \{h@T \leftarrow \gamma, \mathbf{L} \mid (\gamma, \mathbf{L}) \in KK \wedge sat(\gamma)\}$$

with $KK = DNF(\delta, \bigwedge_{k\in K}((S_k < \tau_{a_k,\mathcal{P}}) \vee ((S_k \geq \tau_{a_k,\mathcal{P}}) \wedge not(a_k@S_k))))$.

(Ic) If $t \neq \hat{\mathsf{m}}_{\mathcal{P}^f}$ *then* $\mathcal{P} := \mathcal{P} \cup \{h@T \leftarrow \delta, (T = t)\}$, *assuming that* $C = (h@T \leftarrow \delta, \mathbf{L})$ *is a negative rule in* $\mathcal{P}$ *such that* $\hat{\mathsf{m}}_C = t$ *and* $strat(C) \leq strat(C')$ *for any other negative rule* C' *in* $\mathcal{P}$ *with* $\hat{\mathsf{m}}_{C'} = t$.

(II) Choose a fact $\mathbf{f} = (h'@T' \leftarrow \delta')$ *from* $\mathcal{P}^f$ *such that* $sat((T' = t) \wedge \delta')$.

(III) $\mathcal{P} := \mathcal{P}^f \setminus \{\mathbf{f}\} \cup \bigcup_{C \in \mathcal{P}^r} prop(C, \mathbf{f}, t)$

□

We sometimes write step (Ib) as $\mathcal{P} := extr\_t(\mathcal{P})$ with $extr\_t(\mathcal{P}) \stackrel{def}{=} \mathcal{P}^f \cup \mathcal{P}^+ \cup \bigcup_{C\in\mathcal{P}^-} extr\_t(C, \mathcal{P})$. We also write step (Ic) as $\mathcal{P} := extr\_p(\mathcal{P}, t)$. Define $extr(\mathcal{P}, t) \stackrel{def}{=} extr\_p(extr\_t(\mathcal{P}), t)$.

Given a program $\mathcal{P}$ and a predicate stratification function *strat* that can be obtained by a stratification algorithm [45] (page 134), the operational semantics first initialises the timer to 0 and removes the clauses with unsatisfiable body constraints and then repeatedly generates a fact and transforming the current program into a new program as follows.

Step (Ia) sets the timer to the minimum time at which the body constraint of any fact contained in $CM(\mathcal{P})$ can be satisfied. The minimum time is determined by $\mathcal{P}^f$ and $\mathcal{P}^-$ according to lemmas 2 and 3 and is no less than the previous value of the timer. Step (Ib) extracts positive information from negative rules in $\mathcal{P}^-$ using time stratification as follows. Each negative literal $not(a@S)$ in a negative rule is replaced by

$$S < \tau_{a,\mathcal{P}} \vee (S \geq \tau_{a,\mathcal{P}}) \wedge not(a@S)$$

Each such rule is then normalised, resulting in a set of clauses which replace the original negative rule. Step (Ic) extracts a fact from a negative rule using predicate symbol stratification. It ensures that there is always a fact to choose in step (II). Step (II) generates a fact $\mathbf{f}$. Through invocation of $prop(C, \mathbf{f}, t)$, step (III) replaces an atom $a_l@S_l$ with $(a_l@S_l \wedge (S_l \geq t) \vee [a_l@S_l = \rho_l(\mathbf{h}') \wedge \rho_l(\delta')])$. The first disjunct allows the further solution to $a_l@S_l$ to be considered while the second propagates its solution provided by $\mathbf{f}$.

The operational semantics is indeterministic. There might be several facts that can be extracted from negative rules in the current program at step (Ic). There might also a number of facts whose success time is equal to the current value of the timer. The operational semantics indeterministically chooses one at these steps.

Example 1. This example illustrates the operational semantics using the even numbers program. The following is the normalised even numbers program completed with causality constraints.

```
% even numbers program.

even@T :-  T=0.
even@T :-  T=S+1,S>=0, not(even@S).
```

The program is temporally stratified. The configuration after initialisation is illustrated in Figure 1(a).

As there is no positive literal in the body of either clause. $\mathcal{P}^r = \mathcal{P}^-$ throughout the execution of the program. The first iteration is as follows. The approximate success time of $\mathcal{P}^- \cup \mathcal{P}^f$ is 0. So, $t = 0$ after step (Ia). In step (Ib), the goal $not(even@S)$ in the only rule in $\mathcal{P}^-$ is replaced by $(S < 0) \vee (S >= 0) \wedge not(even@S)$. Normalising the resulting clause gives rise to two clauses: (a)

```
even@T :- T=S+1,S>=0,S<0.
```

and (b)

```
even@T :- T=S+1,S>=0, not(even@S).
```

(a) is thrown away as it has an unsatisfiable constraint part while (b) is a rule and replaces the original rule. Thus, after step (Ib), $\mathcal{P}^-$ contains one rule (b) and $\mathcal{P}^f$ contains one fact: (c)

```
even@T :- T=0.
```

Step (Ic) doesn't change the configuration. Step (II) selects (c) from $\mathcal{P}^f$. Step (III) removes (c) and propagates it through (b), resulting in the following rules

```
even@T :- T=S+1,S>=0,S<>0, S<0.
even@T :- T=S+1,S>=0,S<>0, S>=0, not(even@S).
```

The first clause is discarded as its body constraint is unsatisfiable. The configuration after the first iteration is illustrated in figure 1(b).

Now consider the second iteration. $\mathcal{P}^f$ is empty while $\mathcal{P}^-$ contains one rule. 2 is the approximate success time of $\mathcal{P}^-$. So, $t = 2$ after step (Ia). Step (Ib) extracts from the rule the following two clauses: (d)

```
 even@T :- T=S+1,S>=0,S<>0,S<2.
```

and (e)

```
 even@T :- T=S+1,S>=2,not(even@S).
```

(d) and (e) replace the original rule. Step (Ic) doesn't change the configuration. Step (II) generates (d). In step (III), (d) is removed and propagated through (e), giving rise to the following clauses

```
even@T :- T=S+1,S>=2,S<>2,S<2.
even@T :- T=S+1,S>=2,S<>2,S>=2, not(even@S).
```

The first clause is discarded and the second replaces (e). The configuration after the second iteration is illustrated in figure 1(c).
□

We now present the correctness and the completeness of the operational semantics. It is obvious that the timestamps of generated facts are in ascending order. In the sequel, we will denote the original program by $\mathcal{P}$, the current context after the i^{th} iteration by $(t_i, \mathcal{P}_i)$ and the generated fact during the i^{th} iteration by $\mathbf{f}_i$. Thus, the sequence of configurations obtained during the execution of $\mathcal{P}$ is $(t_0, \mathcal{P}_0), \cdots, (t_i, \mathcal{P}_i), \cdots$ where $(t_0, \mathcal{P}_0)$ is the initial configuration and the sequence of generated facts are $\mathbf{f}_1, \cdots, \mathbf{f}_i, \cdots$. Note that t_{i+1} and $\mathbf{f}_{i+1}$ are determined by $(t_i, \mathcal{P}_i)$.

The following lemma shows that the model semantics of the current program can only contain ground facts whose timestamps are no less than the minimum time at which the body constraint of a clause is satisfiable.

Lemma 5. *Let $a@t$ be a ground atom. If $a@t \in CM(\mathcal{P}_i)$ then $t \geq \tau_{a,\mathcal{P}_i}$.*

Proof. Let $a = p(\tilde{s})$ and the set of the clauses for p in $\mathcal{P}_i$ be $\{p(\tilde{x})@T \leftarrow \delta_o, \mathbf{L}_o \mid 1 \leq o \leq m\}$. Let $\mu = \{\tilde{x} \mapsto \tilde{s}, T \mapsto t\}$. Since $p(\tilde{s})@t \in CM(\mathcal{P}_i)$, we have $\mathcal{D} \models \mu(\delta_o)$ and $CM(\mathcal{P}_i) \models \mu(\mathbf{L}_o)$ for some $1 \leq o \leq m$. $t \geq \tau_{a,\mathcal{P}_i}$ since $\mathcal{D} \models (\mu(\delta_o) \rightarrow (\mu(T) \geq \tau_{a,\mathcal{P}_i}))$. □

The following lemma shows that steps (Ib) and (Ic) preserve the meaning of the program.

Lemma 6. *$CM(\mathcal{P}_i) = CM(extr(\mathcal{P}_i, t_{i+1}))$.*

Proof. Let $Q = extr\_t(\mathcal{P}_i)$ and $R = extr\_p(Q, t_{i+1})$. We first prove $CM(\mathcal{P}_i) = CM(Q)$ by showing $GL([extr\_t(C, \mathcal{P}_i)]_{\mathcal{D}}, CM(\mathcal{P}_i)) = GL([C]_{\mathcal{D}}, CM(\mathcal{P}_i))$ for each $C \in \mathcal{P}_i^-$. Let $C = (h@T \leftarrow \delta, \wedge_{k\in K} not(a_k@S_k))$.

($\subseteq$) Let $\mu(h@T \leftarrow) \in GL([extr\_t(C, \mathcal{P}_i)]_{\mathcal{D}}, CM(\mathcal{P}_i))$. Then $extr\_t(C, \mathcal{P}_i)$ contains a $C' = (h@T \leftarrow \delta, \wedge_{k\in K}\mathbf{b}_k)$ such that $\mathcal{D} \models \mu(\delta)$ and $CM(\mathcal{P}_i) \models_{\mathcal{D}} \mathbf{b}_k$. $\mathbf{b}_k$ is either $S_k < \tau_{a_k,\mathcal{P}_i}$ or $(S_k \geq \tau_{a_k,\mathcal{P}_i}) \wedge not(a_k@S_k)$. It can be shown that $\mu(a_k@S_k) \notin CM(\mathcal{P}_i)$ in both cases, implying $\mu(h@T \leftarrow) \in GL([C]_{\mathcal{D}}, CM(\mathcal{P}_i))$.

($\supseteq$) Let $\mu(h@T \leftarrow) \in GL([C]_{\mathcal{D}}, CM(\mathcal{P}_i))$. Then $\mathcal{D} \models \mu(\delta)$ and $\mu(a_k@S_k) \notin CM(\mathcal{P}_i)$ for each $k \in K$. Let $C' = (h@T \leftarrow \delta, \wedge_{k\in K} not(\mathbf{b}_k))$ with $\mathbf{b}_k$ being $S_k < \tau_{a_k,\mathcal{P}_i}$ if $\mathcal{D} \models \mu(S_k < \tau_{a_k,\mathcal{P}_i})$ or being $(S_k \geq \tau_{a_k,\mathcal{P}_i}) \wedge not(a_k@S_k)$ otherwise. Then we have $\mu(h@T \leftarrow) \in GL([C']_{\mathcal{D}}, CM(\mathcal{P}_i))$ implying $\mu(h@T \leftarrow) \in GL([extr\_t(C, \mathcal{P}_i)]_{\mathcal{D}}, CM(\mathcal{P}_i))$.

It remains to prove $CM(Q) = CM(R)$). $t_{i+1} = \hat{\mathsf{m}}_{(Q^f \cup Q^-)}$ because, for any $C = (h@T \leftarrow \delta, \wedge_{k\in K} not(a_k@S_k))$ in $\mathcal{P}_i$, there is C' in Q such that $\hat{\mathsf{m}}_{C'} = \hat{\mathsf{m}}_C$ where $C' = (h@T \leftarrow \delta, \wedge_{k\in K}\mathbf{b}_k)$ with $\mathbf{b}_k$ being $S_k < \tau_{a_k,\mathcal{P}_i}$ if $\mathcal{D} \models (\delta, (S_k < \tau_{a_k,\mathcal{P}_i}))$ and $\mathbf{b}_k$ being $(S_k \geq \tau_{a_k,\mathcal{P}_i}) \wedge not(a_k@S_k)$ otherwise. If $t_{i+1} = \hat{\mathsf{m}}_{Q^f}$ then $R = Q$ and hence $CM(Q) = CM(R)$. Otherwise, Q contains a $C = (h@T \leftarrow \delta, \wedge_{k\in K} not(a_k@S_k))$ such that $\hat{\mathsf{m}}_C = t_{i+1}$, $strat(C) \leq strat(C'')$ for any other negative rule C'' in Q with $\hat{\mathsf{m}}_{C''} = t_{i+1}$, and $R = Q \cup \{h@T \leftarrow \delta \wedge (T = t_{i+1})\}$. By lemma 3, we have $[h@T \leftarrow \delta \wedge (T = t_{i+1})]_{\mathcal{D}} \subseteq CM(Q)$ and $(\mathcal{D} \models \mu(\delta \wedge (T = t_{i+1}))) \rightarrow \mu(a_k@S_k) \notin CM(Q)$ for any valuation μ. So, $GL([R]_{\mathcal{D}}, CM(Q)) = GL([Q]_{\mathcal{D}}, CM(Q))$ implying that $CM(Q)$ is the least Herbrand model of $GL([R]_{\mathcal{D}}, CM(Q))$. Therefore, $CM(Q) = CM(R)$. □

The following lemma states that each cycle of iteration is correct and complete with respect to the model semantics.

Lemma 7. *$CM(\mathcal{P}_i) = [\mathbf{f}_{i+1}]_{\mathcal{D}} \cup CM(\mathcal{P}_{i+1})$.*

Proof. Let $R = extr(\mathcal{P}_i, t_{i+1})$. By lemma 6, it suffices to prove $CM(R) = [\mathbf{f}_{i+1}]_{\mathcal{D}} \cup CM(\mathcal{P}_{i+1})$ by proving (1) $CM(R) \supseteq [\mathbf{f}_{i+1}]_{\mathcal{D}} \cup CM(\mathcal{P}_{i+1})$ and (2) $CM(R) \subseteq [\mathbf{f}_{i+1}]_{\mathcal{D}} \cup CM(\mathcal{P}_{i+1})$.

(1). $CM(R) \supseteq [\mathbf{f}_{i+1}]_{\mathcal{D}}$ since $\mathbf{f}_{i+1} \in R$. So, it suffices to show $CM(R) \supseteq CM(\mathcal{P}_{i+1})$. $[\mathcal{P}_{i+1}]_{\mathcal{D}}$ is locally stratified and hence $CM(\mathcal{P}_{i+1})$ is the least Herbrand model of $[\mathcal{P}_{i+1}]_{\mathcal{D}}$. Therefore, it reduces to prove that $CM(R)$ is a model of $[\mathcal{P}_{i+1}]_{\mathcal{D}}$. $CM(R)$ is a model of $[\mathcal{P}_i^f \setminus \{\mathbf{f}_{i+1}\}]_{\mathcal{D}}$. So, it remains to prove that $CM(R)$ is a model of $[prop(C, \mathbf{f}_{i+1}, t_{i+1})]_{\mathcal{D}}$ for each $C \in R^r$. Let C be $h@T \leftarrow \delta, \wedge_{j\in J} a_j@S_j, \wedge_{k\in K} not(a_k@S_k)$ and $\mathbf{f}_{i+1}$ be $\mathbf{h}' \leftarrow \delta'$. Then $prop(C, \mathbf{f}_{i+1}, t_{i+1})$ is $h@T \leftarrow \delta, \wedge_{j\in J}\mathbf{b}_j, \wedge_{k\in K} not(\mathbf{b}_k)$ where $\mathbf{b}_l$ is $a_l@S_l \wedge (S_l \geq t_{i+1}) \vee (a_l@S_l = \rho_l(\mathbf{h}')) \wedge \rho_l(\delta')$. Let ν be an arbitrary valuation such that $\mathcal{D} \models \nu(\delta)$, (i) $CM(R) \models_{\mathcal{D}} \nu(\mathbf{b}_j)$ for $j \in J$, and (ii) $CM(R) \not\models_{\mathcal{D}} \nu(\mathbf{b}_k)$ for $k \in K$. (i) implies $\nu(a_j@S_j) \in CM(R)$ because $[\mathbf{f}_{i+1}]_{\mathcal{D}} \subseteq CM(R)$ and (ii) implies $\nu(a_k@S_k) \notin CM(R)$. So, $\nu(h@T) \in CM(R)$ because $CM(R)$ is a model of $[C]_{\mathcal{D}}$. This completes the proof of (1).

(2). Let $W = [\mathbf{f}_{i+1}]_{\mathcal{D}} \cup CM(\mathcal{P}_{i+1})$. It reduces to proving that W is a model of $[R]_{\mathcal{D}}$ because R is locally stratified and $CM(R)$ is the least Herbrand model of $[R]_{\mathcal{D}}$. As $R^f \subseteq \mathcal{P}_{i+1}^f \cup \{\mathbf{f}_{i+1}\}$, it suffices to prove W is a model of $[C]_{\mathcal{D}}$ for every $C \in R^r$. Let C be $h@T \leftarrow \delta, \wedge_{j\in J} a_j@S_j, \wedge_{k\in K} not(a_k@S_k)$ and $\mathbf{f}_{i+1}$ be $\mathbf{h}' \leftarrow \delta'$. Then $prop(C, \mathbf{f}_{i+1}, t_{i+1})$ is $h@T \leftarrow \delta, \wedge_{j\in J}\mathbf{b}_j, \wedge_{k\in K} not(\mathbf{b}_k)$ where $\mathbf{b}_l = a_l@S_l \wedge (S_l \geq t_{i+1}) \vee ((a_l@S_l = \rho_l(\mathbf{h}')) \wedge \rho_l(\delta'))$. Let ν be any valuation such that $\mathcal{D} \models \nu(\delta)$, (iii) $W \models \nu(a_j@S_j)$ for $j \in J$ and (iv) $W \not\models \nu(a_k@S_k)$ for $k \in K$. (iii) implies $CM(\mathcal{P}_{i+1}) \models_{\mathcal{D}} \nu(\mathbf{b}_j)$ for otherwise, $\nu(a_j@S_j) \notin CM(\mathcal{P}_{i+1})$, $\nu(a_j@S_j) \notin [\mathbf{f}_{i+1}]_{\mathcal{D}}$ and hence $\nu(a_j@S_j) \notin W$. (iv) implies $CM(\mathcal{P}_{i+1}) \not\models_{\mathcal{D}} \nu(\mathbf{b}_k)$ for otherwise, either $CM(\mathcal{P}_{i+1}) \models_{\mathcal{D}} \nu(a_k@S_k \wedge (S_k \geq t_{i+1}))$ or $\nu(a_k@S_k) \in [\mathbf{f}_{i+1}]_{\mathcal{D}}$, contradicting $\nu(a_k@S_k) \notin W$. So, $\nu(h@T) \in CM(\mathcal{P}_{i+1}) \subseteq W$ as $CM(\mathcal{P}_{i+1}) \models [prop(C, \mathbf{f}_{i+1}, t_{i+1})]_{\mathcal{D}}$ and hence $W \models [C]_{\mathcal{D}}$. This completes the proof of (2). □

The following theorem establishes the correctness of the operational semantics, that is, every generated fact is contained in the model semantics of the program.

Theorem 8. *$[\mathbf{f}_i]_{\mathcal{D}} \subseteq CM(\mathcal{P})$ for each $i > 0$.*

Proof. We have $CM(\mathcal{P}) = [\{\mathbf{f}_1, \cdots, \mathbf{f}_i\}]_{\mathcal{D}} \cup CM(\mathcal{P}_{i+1})$ by repeatedly applying lemma 7. So, $[\mathbf{f}_i]_{\mathcal{D}} \subseteq CM(\mathcal{P})$ for each $i > 0$. □

The following theorem states that the operational semantics is complete in the sense that any ground atom in the model semantics of the original program is a ground instance of a generated fact or in the model semantics of the current program and that any ground atom in the model semantics of the original program with a timestamp smaller than the current value of the timer is a ground instance of a generated fact.

Theorem 9. *Let $\mathbf{f}$ be a fact. If $[\mathbf{f}]_{\mathcal{D}} \subseteq CM(\mathcal{P})$ then*

(a) $[\mathbf{f}]_{\mathcal{D}} \subseteq [\{\mathbf{f}_1, \cdots, \mathbf{f}_i\}]_{\mathcal{D}} \cup CM(\mathcal{P}_i)$; and

(b) $[\mathbf{f}^{<t_i}]_{\mathcal{D}} \subseteq [\{\mathbf{f}_1, \cdots, \mathbf{f}_{i-1}\}]_{\mathcal{D}}$.

Proof. (a) is a corollary of lemma 7. (b) follows lemmata 7 and 5. □

5 Conclusion and Discussion

We have presented a bottom-up operational semantics for temporally stratified Starlog programs. Its correctness and completeness with respect to its model semantics are given. For simplicity, we have assumed that every atom is timestamped. The operational semantics can be easily modified to cope with untimestamped literals by applying only predicate symbol stratification to untimestamped literals.

The operational semantics strictly generalises the previous work on bottom-up execution of CLP programs. There has been little effort on bottom-up execution of CLP programs and bottom-up execution of CLP programs has only been proposed for constraint deductive databases [25, 23]. These proposals do not deal with negation. Upon generating a fact, our operational semantics propagates it through rules in the current program resulting in a new program. This removes the need for maintaining a list of generated facts and the need for garbage collecting useless facts in the list.

Bottom-up execution has been proposed for general logic programs. Among others, Fages [13], Teusink [43], Kemp etc. [24], and Sacca and Zaniolo [40] proposed fixpoint operators for computing stable and well-founded models of general logic programs. A major problem with applying these operator to Starlog programs is that none of these operators ensure the time-orderedness of generated facts which is essential in temporal applications such as simulation. Furthermore, a fact in a stable model of a general logic program can be generated by operators in [13, 43, 40] only after the model has been fully constructed. This is because a fact added to a model under construction may have to be withdrawn from the model later in order to resolve an inconsistency in the model. A second major problem is that these operators do not deal with constraints. Though our model semantics is based on the stable semantics of ground general logic programs, our operational semantics deals with Starlog programs directly instead of the corresponding ground general logic program. This is necessary because the ground general logic program corresponding to a Starlog program is usually infinite. This is in contrast with deductive databases for which the operators in [40] and [24] are formulated where the ground general logic program is finite.

TLP languages such as Templog [1], Tokio [2], Temporal Prolog [14], Tempura [32], Chronology [47] use top-down operational semantics that extend SLD resolution. Brzoska shows that Templog programs can be translated into CLP programs [7], and Orgun et. al suggest that programs in other TLP languages can also be translated into CLP programs [34]. Thus, our operational semantics offers a bottom-up execution mechanism for these TLP languages.

Xiao et. al propose a bottom-up algorithm for executing Starlog programs without rigorous proof of its correctness and completeness [49]. Xiao's algorithm also generates facts in time order and uses causality to deal with negation. However, its correctness and completeness with respect to a model semantics are not addressed. Our operational semantics works on constraint programs while Xiao's doesn't.

The operational semantics of Starlog presented in this paper is abstract. We have so far not considered the issue of termination for a number of reasons. Termination is undecidable, and there is no operational semantics which will terminate on all programs. Also, we observe that techniques for improving the termination of bottom-up evaluation of logic programs can be easily incorporated into our operational semantics without affecting its correctness and completeness.

The following is an example that illustrates the need for subsumption tests [25, 23, 42, 29]. Consider the following program.

```
p(X)@T :- T=0, X=a.
p(X)@T :- T>=0, p(X)@T.
```

It is obvious that the meaning of this program is the singleton set consisting of the fact

```
p(X)@T :- T=0, X=a.
```

It is easy to verify that each iteration of the operational semantics will generate the above fact and leave the program unchanged. That is, the program does not terminate. Simple checking for duplicate solutions will solve this case but in general more powerful subsumption tests are necessary [29]. This raises complex issues of the tradeoff between computational efficiency and the set of programs on which the operational semantics will terminate.

(a) Intial Configuration:

Timer:	Current Program:	Generated Facts:

$t = 0$	`even@T :-  T=0.`
	`even@T :-  T=S+1,S>=0,`
	`           not(even@S).`	

(b) Configuration after 1st iteration:

Timer:	Current Program:	Generated Facts:
$t = 0$	`even@T :-  T=S+1,S>=0,S<>0,` `           not(even@S).`	`even@T :-  T=0.`

(c) Configuration after 2nd iteration:

Timer:	Current Program:	Generated Facts:
$t = 2$	`even@T :-  T=S+1,S>=2,S<>2,` `           not(even@S).`	`even@T :- T=0.` `even@T :- T=2.`

Fig. 1. The first three configurations for even number program

References

1. M Abadi and Z. Manna. Temporal logic programming. In *Proceedings of the 1987 Symposium on Logic Programming*, pages 4–16. The IEEE Computer Society Press, 1987.
2. T. Aoyagi, M. Fujita, and T. Moto-oka. Temporal logic programming language Tokio. *Lecture Notes in Computer Science*, 221:139–147, 1986.
3. K.R. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Declarative Databases and Logic Programming*, pages 89–142. Morgan Kaufmann, 1988.
4. H Barringer, M fisher, D Gabbay, G Gough, and R Owens. METATEM: An introduction. *Formal Aspects of Computing*, 7(5):533–549, 1995.
5. F. Benhamou, D. McAllester, and P. Van Hentenryck. CLP(Intervals) revisited. In M. Bruynooghe, editor, *Logic Programming - Proceedings of the 1994 International Symposium*, pages 124–138. The MIT Press, 1994.
6. F. Benhamou and W. Older. Applying Interval Arithmetic to Real, Integer and Boolean Constraints. *Journal of Logic Programming*, 32(1):1–24, 1997.
7. C. Brzoska. Temporal logic programming and its relation to constraint logic programming. In *Proceedings of the 1991 International Symposium on Logic Programming*, pages 661–677. The MIT Press, 1991.
8. J.G. Cleary. Logical arithmetic. *Future Computing Systems*, 2(2):125–149, 1987.
9. J.G. Cleary. Colliding pucks solved in a temporal logic. In *Proceedings of 1990 Distributed Simulation Conference*, January, 1990.
10. A. Colmerauer. An introduction to Prolog III. *Communications of the ACM*, 33:69–90, 1990.

11. D. Diaz and P. Codognet. A minimal extension of the WAM for clp(FD). In Warren [48], pages 774–790.
12. M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *Proceedings Intl. Conf. on Fifth Generation Computer Systems*, pages 693–702. Ohmsha Publishers, 1988.
13. F. Fages. A new fixpoint semantics for general logic programs compared with the well-founded and the stable model semantics. *New Generation Computing*, 9(3&4):425–444, 1991.
14. D.M. Gabbay. Modal and temporal logic programming. In A. Galton, editor, *Temporal Logics and Their Application*, pages 197–237. Academic Press, 1987.
15. A. Van Gelder. Negation as failure using tight derivations for general logic programs. *Journal of Logic Programming*, 6(1):109–133, 1987.
16. A. Van Gelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
17. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In Kowalski and Bowen [26], pages 1070–1080.
18. S. Grumbach and J. Su. Queries with arithmetical constraints. *Journal of Theoretical Computer Science*, 173(1):151–181, 1997.
19. P. Van Hentenryck, Y. Deville, and C.-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57(2-3):291–321, 1992.
20. J. Jaffar, M. J. Maher, P. J. Stuckey, and R. H. C. Yap. Beyond finite domains. *Lecture Notes in Computer Science*, 874:86–94, 1994.
21. J. Jaffar and M.J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19 & 20:503–582, 1994.
22. J. Jaffar, S. Michayov, P. Stuckey, and R. Yap. The CLP($\mathcal{R}$) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, 1992.
23. P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51(1):26–52, 1995.
24. D.B. Kemp, D. Srivastava, and P.J. Stuckey. Bottom-up evaluation and query optimization of well-founded models. *Journal of Theoretical Computer Science*, 146(1&2):145–184, 1995.
25. A. Klug. On conjunctive queries containing inequalities. *Journal of the ACM*, 35(1):146–160, 1988.
26. R. A. Kowalski and K. A. Bowen, editors. *Proceedings of the Fifth International Conference and Symposium on Logic Programming.* The MIT Press, 1988.
27. J. C. Lagarias. The computational complexity of simultaneous Diophantine approximation problems. *SIAM Journal on Computing*, 14(1):196–209, 1985.
28. M. Liu and J.G. Cleary. Deductive databases: Where to now? In *Proceedings of 1990 Far-East Workshop on Deductive Databases*, April, 1990.
29. M. Maher. A logic programming view of CLP. In Warren [48], pages 737–753.
30. A. Mantsivoda. Flang and its implementation. *Lecture Notes in Computer Science*, 714:151–165, 1993.
31. A. Monfroglio. Integer programs for logic constraint satisfaction. *Journal of Theoretical Computer Science*, 97(1):105–130, 1992.
32. B. Moszkowski. *Executing Temporal Logic Programs.* Cambridge University Press, 1986.
33. W. Older and F. Benhamou. Programming in CLP(BNR). In *The Proceedings of the First Workshop on Principles and Practice of Constraint Programming*, pages 239–249, Newport, RI, USA, 1993.

34. M. A. Orgun and W. Ma. An overview of temporal and modal logic programming. In D. M. Gabbay and H. J. Ohlbach, editors, *Proceedings of the First International Conference on Temporal Logic*, pages 445–479. Springer-Verlag, 1994.
35. T. C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, Los Altos, CA., 1988.
36. T. C. Przymusinski. Perfect model semantics. In Kowalski and Bowen [26], pages 1081–1096.
37. T. C. Przymusinski. Every logic program has a natural stratification and an iterated fixed point model. In *Proceedings of the Eighth Symposium on Principles of Database Systems*, pages 11–21. ACM SIGACT-SIGMOD, 1989.
38. T. C. Przymusinski. On the declarative and procedural semantics of logic programs. *Journal of Automated Reasoning*, 5(2):167–205, 1989.
39. P. Z. Revesz. A closed-form evaluation for Datalog queries with integer (gap)-order constraints. *Journal of Theoretical Computer Science*, 116(1):117–149, 1993.
40. D. Sacca and C. Zaniolo. Deterministic and non-deterministic stable models. *Journal of Logic and Computation*, 7(5):555–579, 1997.
41. B. Smith, S. Brailsford, P. Hubbard, and H. P. Williams. The Progressive Party Problem: Integer Linear Programming and Constraint Programming Compared. In *Proceedings of the first International Conference on Principles and Practice of Constraint Programming*, Marseilles, September 1995.
42. D. Srivastava. Subsumption and indexing in constraint query languages with linear arithmetic constraints. *Annals of Mathematics and Artificial Intelligence*, 8:315–343, 1993.
43. F. Teusink. A characterization of stable models using a non-monotonic operator. In L.M. Pereira and A. Nerod, editors, *Proceedings of the second International workshop Logic Programming and Non-Monotonic Reasoning*, pages 206–222. The MIT Press, 1993.
44. D. Toman, J. Chomicki, and D. S. Rogers. Datalog with integer periodicity constraints. In M. Bruynooghe, editor, *Logic Programming - Proceedings of the 1994 International Symposium*, pages 189–203. The MIT Press, 1994.
45. J.D. Ullman. *Principles of Database and Knowledge-Base Systems: Volume I.* Computer Science Press, 1988.
46. P. Van Hentenryck, V. Saraswat, and Y. Deville. Design, implementation, and evaluation of the constraint language cc(FD). *Lecture Notes in Computer Science*, 910:293–307, 1995.
47. W.W. Wadge. Tense logic programming: a respectable alternative. In *Proceedings of the 1988 International Symposium on Lucid and Intensional Programming*, pages 26–32, Sidney, B.C., April 7-8 1988.
48. D.S. Warren, editor. *Proceedings of the Tenth International Conference on Logic Programming*. The MIT Press, 1993.
49. Z. Xiao, J.G. Cleary, and B. Unger. Time-ordered forward deduction: A bottom-up evaluation strategy for starlog. Technical Report 98/625/16, Department of Computer Science, University of Calgary, 1998.