Proof Procedures for Branching-Time Logic Programs[*]

M. Gergatsoulis[1], **P. Rondogiannis**[2], **T. Panayiotopoulos**[3]

[1] Inst. of Informatics & Telecom., N.C.S.R. 'Demokritos',
153 10 A. Paraskevi Attikis, Greece
e_mail: manolis@iit.nrcps.ariadne-t.gr

[2] Dept. of Computer Science, University of Ioannina,
P.O. BOX 1186, 45110 Ioannina, Greece,
e_mail: prondo@zeus.cs.uoi.gr

[3] Dept. of Informatics, University of Piraeus
80 Karaoli & Dimitriou Str., 18534 Piraeus, Greece
e-mail : themisp@unipi.gr

Abstract

Traditional implementation techniques for temporal logic programming languages are based on the notion of *canonical* temporal atoms/clauses. Although such an approach is satisfactory for proving goals that refer to specific moments in time, it usually leads to non-terminating computations when considering *open-ended goals*.

In this paper, we propose a new generalized proof procedure for implementing branching time logic programming languages. The particular strength of the new proof procedure, called *CSLD-resolution*, is that it can handle in a more general way open-ended queries, without the need of enumerating all their canonical instances.

Keywords: Temporal Logic Programming, Branching Time, Proof Procedures.

1 Introduction

Temporal logic programming languages [OM94, Org91] are recognized as natural and expressive formalisms for describing *dynamic* systems, and have been widely used in many application areas such as program specification and verification [LO97], in modelling temporal databases [Org96] as well as in knowledge representation [LO96] and temporal reasoning [Vil94].

However, most temporal languages [Wad88, OM94, Hry93, OWD93, Bau93, Brz91, GRP96] are based on linear flow of time, a fact that makes them unsuitable for certain types of applications. For example, as M. Ben-Ari, A. Pnueli and Z. Manna have pointed out in [BAPM83], branching time logics are necessary in order to express certain properties of non-deterministic programs.

In [RGP97b, RGP97a], a new temporal logic programming language called *Cactus* is presented, which is based on a tree-like notion of time; that is, every moment in time may

[*]This work has been funded by the Greek General Secretariat of Research and Technology under the project "TimeLogic" of ΠΕΝΕΔ'95, contract no 1134.

This paper appears in Proc. of the 10th International Symposium on Languages for Intensional Programming (ISLIP'97), Bill Wadge, editor, papes 12-26, University of Victoria, Victoria, BC Canada, May 15-17, 1997.

have more than one next moments. Cactus supports two main operators: the temporal operator `first` refers to the beginning of time (or alternatively to the root of the tree). The temporal operator $\texttt{next}_i$ refers to the i-th child of the current moment (or alternatively, the i-th branch of the current node in the tree). Notice that we actually have a family $\{\texttt{next}_i \mid i \in \mathcal{N}\}$ of `next` operators, each one of them representing the different next moments that immediately follow the present one.

As an example, consider the following Cactus program, which maps sequences of '`a`' and '`b`' on a binary time tree as shown in figure 1.

```
first sequence([]).
next0 sequence([a|R]) <-  sequence(R).
next1 sequence([b|R]) <-  sequence(R).
```

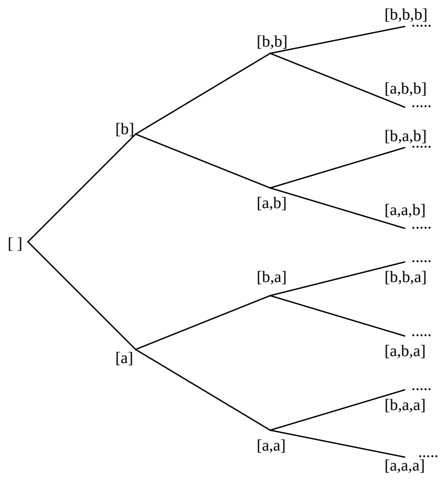

Figure 1: A mapping of sequences of '`a`' and '`b`'s on a binary time tree

Traditional implementation techniques for temporal logic programming languages are based on the notion of *canonical* temporal atoms/clauses [OW93]. Following this approach, we introduce a resolution-based proof procedure for Cactus programs, called BSLD-resolution. BSLD-resolution requires both program and goal clauses to be in the so called *canonical form*, i.e. all atoms in the clause should have a (possibly empty) sequence of `next` operators applied to them, and the operator `first` should be in the front of this sequence. As it will be clear from the semantics of Cactus, canonical clauses are instances in time of program clauses. BSLD-resolution follows the ideas previously introduced in the context of the linear time logic programming language Chronolog and its proof system, TiSLD-resolution [OW93].

Although this approach is satisfactory for proving goals that refer to specific moments in time, it usually leads to non-terminating computations when considering *open-ended goals*. Motivated from this remark, we also developed a new resolution-based proof procedure for *Cactus* programs called *CSLD-resolution* (**C**actus **SLD**-resolution). The particular strength of the new proof procedure, called *CSLD-resolution*, is that it can handle in a more general way open-ended queries, without the need of enumerating all their canonical instances.

CSLD-resolution can directly apply to Chronolog programs [Org91]. Moreover, it can be easily extended to apply also to multi-dimensional logic programs [OD94]. TiSLD-resolution [OW93] can be seen as a special case of CSLD-resolution.

2 The syntax of Cactus programs

The syntax of Cactus programs is an extension of the syntax of Prolog programs [Llo87].

A *temporal atom* is an atomic formula with a number (possibly 0) of applications of temporal operators. The sequence of temporal operators applied to an atom is called the *temporal reference* of that atom. A *temporal clause* is a formula of the form:

$$H \leftarrow B_1, ..., B_m$$

where $H, B_1, ..., B_m$ are temporal atoms and $m \geq 0$. If $m = 0$, the clause is said to be a *unit temporal clause*. A *Cactus program* is a finite set of *temporal clauses*. A *goal clause* in Cactus is a formula of the form $\leftarrow A_1, ..., A_n$ where A_i, $i = 1, ..., n$ are temporal atoms.

3 Branching time logic

Branching time logic programming (BTLP), is based on a relatively simple *branching time logic*. In branching time logic, time has an initial moment and flows towards the future in a tree-like way. The set of moments in time can be modelled by the set $List(\mathcal{N})$ of lists of natural numbers $\mathcal{N}$. Thus, each node may have a countably infinite number of branches (`next` operators). The empty list $[\,]$ corresponds to the beginning of time and the list $[i|t]$ (that is, the list with head i, where $i \in \mathcal{N}$, and tail t) corresponds to the i-th child of the moment identified by the list t. Branching time logic uses the temporal operators `first` and $\texttt{next}_i$, $i \in \mathcal{N}$. The operator `first` is used to express the first moment in time, while $\texttt{next}_i$ refers to the i-th child of the current moment in time. The syntax of branching time logic extends the syntax of first-order logic with two formation rules: if A is a formula then so are `first` A and $\texttt{next}_i$ A.

We write $BTL(\mathcal{N})$ for branching time logic with countably infinite number of branches starting from each node (i.e. countably infinite `next` operators). In practice, we are especially interested in branching time logics with finite number $n \in \mathcal{N}$ of `next` operators. The set of moments in time in these logics can be modelled by the set $List(n)$ of lists of numbers in the set $S = \{i | i \in \mathcal{N} \;\; and \; 0 \leq i \leq n-1\}$. We write $BTL(n)$ for the branching time logic with n `next` operators. For example, $BTL(2)$ is the branching time logic with two `next` operators, i.e. from each node in the time tree start two branches.

We also write Cactus($\mathcal{N}$) or Cactus(n) for the Cactus language based on $BTL(\mathcal{N})$ or $BTL(n)$ respectively.

3.1 Semantics of $BTL(\mathcal{N})$ formulas

The semantics of temporal formulas of $BTL(\mathcal{N})$ are given using the notion of *branching temporal interpretation*. Branching temporal interpretations extend the temporal interpretations of the linear time logic of Chronolog [Org91].

Definition 3.1. A *branching temporal interpretation* or simply a *temporal interpretation* I of the temporal logic $BTL(\mathcal{N})$ comprises a non-empty set D, called the domain of the interpretation, over which the variables range, together with an element of D for each variable; for each n-ary function symbol, an element of $[D^n \rightarrow D]$; and for each n-ary predicate symbol, an element of $[List(\mathcal{N}) \rightarrow 2^{D^n}]$.

In the following definition, the satisfaction relation $\models$ is defined in terms of temporal interpretations. $\models_{I,t} A$ denotes that a formula A is true at a moment t in some temporal interpretation I.

Definition 3.2. The semantics of the elements of the temporal logic $BTL(\mathcal{N})$ are given inductively as follows:

1. If $\mathtt{f}(e_0, \ldots, e_{n-1})$ is a term, then $I(\mathtt{f}(e_0, \ldots, e_{n-1})) = I(\mathtt{f})(I(e_0), \ldots, I(e_{n-1}))$.
2. For any n-ary predicate symbol $\mathtt{p}$ and terms $e_0, \ldots, e_{n-1}$,
   $\models_{I,t} \mathtt{p}(e_0, \ldots, e_{n-1})$ *iff* $\langle I(e_0), \ldots, I(e_{n-1})\rangle \in I(\mathtt{p})(t)$
3. $\models_{I,t} \neg A$ *iff it is not the case that* $\models_{I,t} A$
4. $\models_{I,t} A \wedge B$ *iff* $\models_{I,t} A$ *and* $\models_{I,t} B$
5. $\models_{I,t} A \vee B$ *iff* $\models_{I,t} A$ *or* $\models_{I,t} B$
6. $\models_{I,t} (\forall x)A$ *iff* $\models_{I[d/x],t} A$ *for all* $d \in D$ *where the interpretation* $I[d/x]$ *is the same as* I *except that the variable* x *is assigned the value* d.
7. $\models_{I,t} \mathtt{first}\ A$ *iff* $\models_{I,[\,]} A$
8. $\models_{I,t} \mathtt{next}_i\ A$ *iff* $\models_{I,[i|t]} A$

If a formula A is true in a temporal interpretation I at all moments in time, it is said to be true in I (we write $\models_I A$) and I is called a *model* of A.

Notice that, the above semantics defined for $BTL(\mathcal{N})$, obviously apply to $BTL(n)$.

3.2 Axioms and Rules of Inference

The following axioms, many of which are similar to those adopted for the case of linear time logics [Org91], refer to some important properties of the temporal operators. In the following, the symbol ∇ stands for any of `first` and $\mathtt{next}_i$.

Temporal operator cancellation rules: The intuition behind these rules is that the operator `first` cancels the effect of any other "outer" operator. Formally:

$$\nabla(\mathtt{first}\ A) \leftrightarrow (\mathtt{first}\ A)$$

Temporal operator distribution rules: These rules express the fact that the branching time operators of $BTL(\mathcal{N})$ distribute over the classical operators $\neg$, $\wedge$ and $\vee$. Formally:

$$\begin{array}{l} \nabla(\neg A) \leftrightarrow \neg(\nabla A) \\ \nabla(A \wedge B) \leftrightarrow (\nabla A) \wedge (\nabla B) \\ \nabla(A \vee B) \leftrightarrow (\nabla A) \vee (\nabla B) \end{array}$$

From the temporal operator distribution rules we see that if we apply a temporal operator to a whole program clause, the operator can be pushed inside until we reach atomic formulas. This is why we did not consider applications of temporal operators to whole program clauses.

Rigidness of variables: The following rule states that a temporal operator ∇ can "pass inside" $\forall$:

$$\nabla(\forall X)(A) \leftrightarrow (\forall X)(\nabla A)$$

The above rule holds because variables represent data-values composed of function symbols and constants which are independent of time (i.e. they are *rigid*).

Temporal operator introduction rules: The following rule states that if A is a theorem of $BTL(\mathcal{N})$ then ∇A is also a theorem of $BTL(\mathcal{N})$.

$$\textit{if } \vdash A \textit{ then } \vdash \nabla A$$

The validity of the above axioms is easily proved using the semantics of $BTL(\mathcal{N})$.
As a final remark, we should note that in general $\texttt{next}_i\ \texttt{next}_j\ A$ and $\texttt{next}_j\ \texttt{next}_i\ A$ are not equivalent.

4 Declarative Semantics

In this section, we present briefly the main results concerning the declarative semantics of Cactus programs. As we will see, the usual minimal model and fixpoint semantics that apply to logic programs, can be extended to apply to Cactus programs. However, more elaborated presentation of the declarative semantics is outside the scope of this paper and is reported in a forthcoming one [RGP97c].

4.1 Minimal Herbrand Model Semantics

The declarative semantics of Cactus programs are defined in terms of the minimal temporal Herbrand models. In order to introduce *temporal Herbrand models* we need the notion of canonical atoms and clauses:

Definition 4.1. A temporal reference (sequence of temporal operators) of Cactus(n) is said to be a *canonical temporal reference* if it is of the form $\texttt{first}\ \texttt{next}_{i_1} \cdots \texttt{next}_{i_k}$, for some $k \geq 0$. A *canonical temporal atom* is an atom whose temporal reference is canonical. A *canonical temporal clause* is a temporal clause whose temporal atoms are canonical.

Every temporal clause can be transformed into a (possibly infinite) set of canonical temporal clauses. This can be done by applying $\texttt{first}\ \texttt{next}_{i_1} \cdots \texttt{next}_{i_k}$ to the clause and then using the axioms of branching time logic, presented in section 3.2, to distribute the temporal reference so as to be applied to each individual temporal atom of the clause; finally any superfluous operator is eliminated by applying the cancellation rules. Intuitively, a canonical temporal clause is an instance in time of the corresponding temporal clause.

The notion of canonical atom/clause is very important since the value of a given clause in a temporal interpretation of $BTL(n)$, can be expressed in terms of the values of its canonical instances, as the following lemma shows:

Lemma 4.1 *Let A be a clause and I a temporal interpretation of $BTL(n)$. Then $\models_I A$ if and only if $\models_I A_t$ for all canonical instances A_t of A.*

As in the theory of classical logic programming [Llo87], the set U_P generated by function and constant symbols that appear in P, called *Herbrand universe*, is used to define *temporal Herbrand interpretations*. Temporal Herbrand interpretations can be regarded as subsets of the *temporal Herbrand Base* THB_P of P, consisting of all *ground canonical temporal atoms* whose predicate symbols appear in P and whose arguments are terms in the Herbrand universe U_P of P. A *temporal Herbrand model* is a temporal Herbrand interpretation, which is a model of the program.

In analogy with the theory of logic programming, the *model intersection property* holds for temporal Herbrand models. The intersection of all temporal Herbrand models denoted by MM_P, is a temporal Herbrand model, called the *least temporal Herbrand model*. The following theorem says that the least temporal Herbrand model consists of all ground canonical temporal atoms which are logical consequences of P.

Theorem 4.1 *Let P be a Cactus program. Then $MM_P = \{A \in THB_P | P \models A\}$.*

4.2 Fixpoint Semantics

A fixpoint characterization of the semantics of Cactus(n) programs is provided using a closure operator that maps temporal Herbrand interpretations to temporal Herbrand interpretations:

Definition 4.2. Let P be a Cactus(n) program and THB_P the temporal Hebrand base of P. The operator $T_P : 2^{THB_P} \rightarrow 2^{THB_P}$ is defined as follows:

$T_P(I) = \{A \mid A \leftarrow B_1, ..., B_n$ is a canonical ground instance of a program clause in P and $\{B_1, ..., B_n\} \subseteq I$ $\}$

It can be proved [RGP97c] that THB_P is a complete lattice under the partial order of set inclusion ($\subseteq$). Moreover, T_P is continuous and hence monotonic over the complete lattice $(THB_P, \subseteq)$, and therefore T_P has a least fixpoint. The least fixpoint of T_P provides a characterization of the minimal Herbrand model of a Cactus(n) program, as it is shown in the following theorem.

Theorem 4.2 *Let P be a Cactus(n) program. Then $MM_P = lfp(T_P) = T_P \uparrow \omega$.*

5 Proof procedures for branching time logic programs

5.1 BSLD-resolution

Traditional implementation techniques for temporal logic programming languages are based on the notion of canonical temporal atoms/clauses. In [OW93], the authors propose a resolution-type proof procedure for the linear-time logic programming language Chronolog, called TiSLD-resolution, which requires program and goal clauses to be in canonical form. The same idea is behind MSLD-resolution, the proof procedure for multi-dimensional logic programs [OD94].

In this section we propose a resolution-type proof procedure for Cactus(n) programs, called *BSLD-resolution* (**B**ranching-time **SLD**-resolution). BSLD-resolution is an extension of TiSLD-resolution, for branching time logic programs, and requires program and goal clauses to be in canonical form.

Definition 5.1. Let P be a program in Cactus(n) and G be a canonical temporal goal. A *BSLD-derivation* of $P \cup \{G\}$ consists of a (possibly infinite) sequence of canonical temporal goals $G_0 = G, G_1, ..., G_n, ...$ a sequence C_1, ..., C_n, ... of canonical instances of program clauses (called the *input clauses*), and a sequence θ_1,...., θ_n.... of most general unifiers such that, for all i the goal G_{i+1} is obtained from the goal:
$G_i = \ \leftarrow A_1, ..., A_{m-1}, A_m, A_{m+1}, ..., A_p$
as follows:

1. A_m is a canonical temporal atom in G_i (called the *selected* atom)
2. $H \leftarrow B_1, ..., B_r$ is the input clause C_{i+1} in P (standardized apart from G_i),
3. $\theta_{i+1} = mgu(A_m, H)$
4. G_{i+1} is the goal: $G_{i+1} = \ \leftarrow (A_1, ..., A_{m-1}, B_1, ..., B_r, A_{m+1}, ..., A_p)\theta_{i+1}$

Definition 5.2. A *BSLD-refutation* of $P \cup \{G\}$ is a finite BSLD-derivation of $P \cup \{G\}$ which has the empty goal clause $\Box$ as the last clause of the derivation.

Definition 5.3. Let P be a program in Cactus(n). The *success set* of P is the set of all canonical temporal atoms A in THB_P such that $P \cup \{\leftarrow A\}$ has a BSLD-refutation.

Definition 5.4. Let P be a program in Cactus(n) and G be a canonical temporal goal. A *computed answer* for $P \cup \{G\}$ is the substitution obtained by restricting the composition $\theta_1\theta_2....\theta_n$ to the variables of G, where $\theta_1, \theta_2, ..., \theta_n$, is the sequence of the most general unifiers used in a BSLD-refutation of $P \cup \{G\}$.

It easy to prove that BSLD-resolution is both sound and complete.

Example 5.1. Consider the following program in Cactus(2):

```
(1)    first num(0).
(2)    next₀ num(s(X))  ←  num(X).
(3)    next₁ num(X)  ←  next₀ num(Y), num(Z), sum(Z,Y,X).
(4)    sum(0,Y,Y).
(5)    sum(s(X),Y,s(Z))  ←  sum(X,Y,Z).
```

The above program assigns integer values to the nodes of the time tree through the predicate '`num`' as follows: The value assigned to the root of the tree is 0. The value assigned to the left child of a node, is the value of the node plus 1. Finally, the value assigned to the right child of a node, is the sum of the value assigned to the node itself and the value assigned to its left child. A BSLD-refutation of the canonical temporal goal:

```
← first next₁ next₀ num(N)
```

is given below (the selected temporal atom in every step is the underlined one):

$\leftarrow$ $\underline{\texttt{first next}_1\ \texttt{next}_0\ \texttt{num(N)}}$
$\theta_1 = \{$ `N / s(N1)` $\}$ using clause (2)
$\leftarrow$ $\underline{\texttt{first next}_1\ \texttt{num(N1)}}$
$\theta_2 = \{$ `N1 / X` $\}$ using clause (3)
$\leftarrow$ `first` $\texttt{next}_0$ `num(Y),` $\underline{\texttt{first num(Z)}}$`, first sum(Z,Y,X).`
$\theta_3 = \{$ `Z / 0` $\}$ using clause (1)
$\leftarrow$ $\underline{\texttt{first next}_0\ \texttt{num(Y)}}$`, first sum(0,Y,X).`
$\theta_4 = \{$ `Y / s(Y1)` $\}$ using clause (2)
$\leftarrow$ $\underline{\texttt{first num(Y1)}}$`, first sum(0,s(Y1),X).`
$\theta_5 = \{$ `Y1 / 0` $\}$ using clause (1)
$\leftarrow$ $\underline{\texttt{first sum(0,s(0),X)}}$.
$\theta_6 = \{$ `X / s(0)` $\}$ using clause (4)
$\square$

The value of the variable N in the goal, is given by the composition of the substitutions $\theta_1, ..., \theta_6$ obtained in the above process. This value is: `N = s(s(0))`.

5.2 Open-ended goal clauses

When one or more temporal atoms included in a goal clause are not canonical, we say that the goal clause is *open-ended*. Following the approach of M. Orgun and W. Wadge [OW93, OD94], we can see an open-ended goal clause G as a representation of the infinite set of all canonical goal clauses corresponding to G. In this way, open-ended goal clauses are used to imitate non-terminating computations. An implementation strategy for executing an open-ended goal clause is by enumerating and evaluating (one by one) using BSLD-resolution, the set of all canonical instances of the goal clause (e.g. by traversing in a breadth-first way the time-tree). It is obvious that the evaluation of an open-ended goal in this way, is in fact a generate-and-test procedure. Although this treatment of open-ended goal clauses is satisfactory for a wide range of applications, there are cases in which a more general proof procedure which can operate directly on open-ended goal and program clauses, is required. For example, if our program in Cactus(2), consists of exactly the unit program clause '$\texttt{next}_0$ `p(a)`', then the open-ended goal clause '$\leftarrow$ `p(X)`', will initiate a non-terminating computation in which all canonical instances of this goal: '$\leftarrow$ `first p(X)`', '$\leftarrow$ `first` $\texttt{next}_0$ `p(X)`', '$\leftarrow$ `first` $\texttt{next}_1$ `p(X)`', '$\leftarrow$ `first` $\texttt{next}_0$ $\texttt{next}_1$ `p(X)`' and so on, will be generated and tested. In this way, we will never obtain the 'obvious' correct answer which says that '$\texttt{next}_0$ `p(a)`' is a logical consequence of the program. Moreover, generating and evaluating all canonical instances of an open-ended goal clause, often requires the evaluation of a specific (sub)goal more than once.

5.3 CSLD-resolution

In this section, we attempt to drop the requirement that program and goal clauses should be in canonical form, by defining a new proof procedure, called *CSLD-resolution*.

In the following, we often require that temporal references are in their *normal form*. We say that a temporal reference T is in normal form, if either the operator `first` does not appear in T or there is only one occurrence of `first` in T, which is the first operator of T. Every temporal reference T can be transformed in its normal form $normal(T)$ by eliminating all operators appearing before the final occurrence of `first`. Because of the operator cancellation rules, it is obvious that for every formula A we have: $T\ A \leftrightarrow normal(T)\ A$.

Moreover, by $T_1\ T_2$ we denote the temporal reference obtained by putting the temporal reference T_2 after the temporal reference T_1. We say that $T_1\ T_2$ is the *composition* of the temporal references T_1 and T_2.

Example 5.2. The temporal reference $\texttt{first next}_1$ is the normal form of the temporal reference $\texttt{next}_2\ \texttt{first next}_0\ \texttt{first next}_1$.

Notice that the composition of two temporal references which are in normal form is not necessarily in normal form. Nevertheless, we can easily prove that: $normal(T_1\ T_2) = normal(normal(T_1)\ T_2) = normal(T_1\ normal(T_2)) = normal(normal(T_1)\ normal(T_2))$.

Definition 5.5. Let A_1 and A_2 be two temporal atoms, such that $A_1 = R_1\ A_1'$ and $A_2 = R_2\ A_2'$ where A_1' and A_2' are classical atoms and R_1, R_2 are (possibly empty) temporal references in normal form. Let $\theta^t = (T, \theta, S)$, where T, S are temporal references in normal form and θ is a substitution[1]. Then θ^t is said to be a *temporal unifier* of A_1 and A_2 iff $T\ R_1\ A_1'\theta = S\ R_2\ A_2'\theta$ and both $T\ R_1$ and $S\ R_2$ are also in normal form.

Definition 5.6. A temporal unifier $\theta^t = (T, \theta, S)$ of two temporal atoms A_1 and A_2, is said to be a *most general temporal unifier* of A_1 and A_2 (we write $\theta^t = mgu^t(A_1, A_2)$) iff for any unifier $\sigma^t = (T', \sigma, S')$ of A_1 and A_2 there is a temporal substitution $\xi^t = (T'', \xi, S'')$ such that $\sigma = \theta\xi$, $T' = T''\ T$, and $S' = S''\ S$.

Example 5.3. Let '$\texttt{next}_0\ \texttt{p(f(X),7)}$' and '$\texttt{next}_2\ \texttt{next}_1\ \texttt{next}_0\ \texttt{p(f(Y),W)}$' be two temporal atoms. Then $\sigma^t = (\texttt{next}_4\ \texttt{next}_2\ \texttt{next}_1, \{\texttt{X/6}, \texttt{Y/6}, \texttt{W/7}\}, \texttt{next}_4)$ is a temporal unifier of these atoms, while[2] $\theta^t = (\texttt{next}_2\ \texttt{next}_1, \{\texttt{X/Y}, \texttt{W/7}\}, \epsilon)$ is a most general temporal unifier of the above atoms.

It is easy to prove the following properties of temporal unifiers:

1. (T, θ, S) is a (most general) temporal unifier of A_1 and A_2 iff (S, θ, T) is a (most general) temporal unifier of A_2 and A_1.

2. If $\theta^t = (T, \theta, S)$ is a most general temporal unifier, then at least one of the temporal references T and S, is empty.

3. Suppose that (T, θ, S) is a temporal unifier of two temporal atoms A_1 and A_2 and $(T', \theta', S') = mgu^t(A_1, A_2)$. Then there is a common prefix F of T and S such that, $F\ T' = T$ and $F\ S' = S$. This common prefix is the maximal one.

Definition 5.7. Let P be a Cactus(n) program and G a temporal goal. A *CSLD-derivation* of $P \cup \{G\}$ consists of a (possibly infinite sequence) of temporal goals $G_0 = G, G_1, ..., G_n, ...$, a sequence C_1, ..., C_n, ... of program clauses (called *input clauses*), and a sequence (T_1, θ_1, S_1), ..., (T_n, θ_n, S_n), ... of most general temporal unifiers such that, for all i, the goal G_{i+1} is obtained from the goal:

$G_i = \ \leftarrow A_1, ..., A_{m-1}, A_m, A_{m+1}, ..., A_p$

as follows:

[1] The triple (T, θ, S) is called a *temporal substitution*

[2] By ϵ, we denote the empty temporal reference.

1. A_m is a temporal atom in G_i (called the *selected* atom)
2. $H \leftarrow B_1, ..., B_r$ is the input clause C_{i+1} (standardized apart from G_i),
3. $\theta^t_{i+1} = mgu^t(A_m, H) = (T_{i+1}, \theta_{i+1}, S_{i+1})$
4. G_{i+1} is the goal[3]:

   $G_{i+1} = \leftarrow (T_{i+1}\ A_1, ..., T_{i+1}\ A_{m-1}, S_{i+1}\ B_1, ..., S_{i+1}\ B_r, T_{i+1}\ A_{m+1}, ..., T_{i+1}\ A_p)\theta_{i+1}$

Definition 5.8. A *CSLD-refutation* of $P \cup \{G\}$ is a finite CSLD-derivation of $P \cup \{G\}$ which has the empty goal clause $\Box$ as the last clause of the derivation.

Definition 5.9. Let P be a program in Cactus(n) and G a temporal goal. A *computed answer* for $P \cup \{G\}$ is a pair $< \theta, T >$ of a substitution θ and a temporal reference T such that θ is the substitution obtained by restricting the composition $\theta_1....\theta_n$ to the variables of G, and $T = normal(T_n\ T_{n-1}\ ....\ T_1)$, where $(T_1, \theta_1, S_1), ..., (T_n, \theta_n, S_n)$ is the sequence of the most general temporal unifiers used in a CSLD-refutation of $P \cup \{G\}$.

Definition 5.10. Let P be a program in Cactus(n) and $G\ = \leftarrow A_1, ..., A_n$ a temporal goal. A pair $< \theta, T >$, of a substitution θ, and a temporal reference T, is said to be a *correct answer* for $P \cup \{G\}$ if $\forall (T\ (A_1 \wedge .... \wedge A_n)\theta)$ is a logical consequence of P.

Notice that, if $T\ (A_1 \wedge .... \wedge A_n)\theta$ is not canonical, then all its canonical instances are logical consequences of P.

Example 5.4. Consider the program defining the predicate '`num`' in example 5.1, and suppose that we want to evaluate the following goal clause:

$\leftarrow$ `num(N)`

A CSLD-refutation which corresponds to the goal clause $\leftarrow$ `first next`$_1$ `next`$_0$ `num(N)` evaluated in example 5.1 using BSLD-resolution, is as follows:

$\leftarrow \underline{\texttt{num(N)}}$
$T_1 = \texttt{next}_0$
$\theta_1 = \{\ \texttt{N / s(N1)}\}$ using clause (2)
$S_1 = \epsilon$
$\leftarrow \underline{\texttt{num(N1)}}$
$T_2 = \texttt{next}_1$
$\theta_2 = \{\ \texttt{N1 / X}\}$ using clause (3)
$S_2 = \epsilon$
$\leftarrow \texttt{next}_0\ \texttt{num(Y),}\ \underline{\texttt{num(Z)}}\texttt{, sum(Z,Y,X).}$
$T_3 = \texttt{first}$
$\theta_3 = \{\ \texttt{Z / 0}\}$ using clause (1)
$S_3 = \epsilon$

[3]We suppose that the temporal reference of each atom is transformed into normal form after the application of the temporal unifier.

$\leftarrow$ $\underline{\texttt{first next}_0\texttt{ num(Y)}}$, `first sum(0,Y,X).`
$T_4 = \epsilon$
$\theta_4 = \{$ `Y / s(Y1)` $\}$ using clause (2)
$S_4 =$ `first`
$\leftarrow$ $\underline{\texttt{first num(Y1)}}$, `first sum(0,s(Y1),X).`
$T_5 = \epsilon$
$\theta_5 = \{$ `Y1 / 0` $\}$ using clause (1)
$S_5 = \epsilon$
$\leftarrow$ $\underline{\texttt{first sum(0,s(0),X)}}$.
$T_6 = \epsilon$
$\theta_6 = \{$ `X / s(0)` $\}$ using clause (4)
$S_6 =$ `first`
$\square$

Thus the computed answer is: `<{ N / s(s(0))}, first` $\texttt{next}_1$ $\texttt{next}_0$`>` where `first` $\texttt{next}_1$ $\texttt{next}_0$ = T_6 T_5 T_4 T_3 T_2 T_1, and { `X / s(s(0))`} $= \theta_1\theta_2\theta_3\theta_4\theta_5\theta_6$.

5.4 Soundness of CSLD-resolution

CSLD-resolution is a sound proof procedure:

Lemma 5.1 *Let P be a program in Cactus(n) and G a temporal goal. Then every computed answer for $P \cup \{G\}$ is a correct answer for $P \cup \{G\}$.*

Corollary 5.1 *If A is a canonical ground temporal atom and $P \cup \{\leftarrow A\}$ has a CSLD-refutation then $A \in MM_P$.*

Nevertheless, the completeness of CSLD-resolution, as formulated above, depends on the choice of the computation rule as it is shown in the following example:

Example 5.5. Consider the following program P in Cactus(2):

(1)	`first p` $\leftarrow$ `r.`
(2)	`first q` $\leftarrow$ `s.`
(3)	`first r.`
(4)	`first` $\texttt{next}_0$ `s.`

It is easy to see that '`first p` $\wedge$ `first q`' is a logical consequence of P. Nevertheless, there are problems in finding a refutation for the goal clause:

$\leftarrow$ `first p, first q.`

Consider for example the derivation:
$\leftarrow$ $\underline{\texttt{first p}}$, `first q.`
$\{\epsilon, \{\}, \epsilon\}$ using clause (1)
$\leftarrow$ `r,` $\underline{\texttt{first q}}$
$\{\epsilon, \{\}, \epsilon\}$ using clause (2)
$\leftarrow$ $\underline{\texttt{r}}$, `s`
$\{\texttt{first}, \{\}, \epsilon\}$ using clause (3)
$\leftarrow$ $\underline{\texttt{first s}}$
`FAIL.`

It is easy to see that, if we select '`r`' instead of '`first q`' in the second step, the derivation will finally succeed. Thus the proof of this goal depends on the computation rule.

The problem in the above failed derivation is due to the fact that although the proof of the subgoal '`r`' should be independent to the proof of the subgoal '`s`', the fact that they co-exist in the goal '$\leftarrow$ $\underline{\texttt{r}}$`, s`' imposes that they must be true at the same moment in time. The problem arises when it happens to mix in the same subgoal, bodies of clauses whose heads are in canonical form but their bodies contain open-ended atoms.

5.5 Generalized CSLD-resolution

We can define a (family of) computation rule(s) under which CSLD-resolution as formulated in section 5.3, is complete. Nevertheless, we are interested in a proof procedure whose completeness does not depend on the choice of the computation rule. For this reason, we redefine CSLD-resolution by generalizing the notion of goal.

A *multi-goal* (or *m-goal* for short) is an expression of the form:

$\leftarrow \{A_{1,1}, A_{1,2}, ..., A_{1,k_1}\}, \{A_{2,1}, A_{2,2}, ..., A_{2,k_2}\}, ..., \{A_{m,1}, A_{m,2}, ..., A_{m,k_m}\}$

where $A_{i,j}$ are temporal atoms. Each set $\{A_{l,1}, A_{l,2}, ..., A_{l,k_l}\}$, of atoms is said to be a *temporal group*. Intuitively, each group in a m-goal represents an ordinary goal clause. The difference between a m-goal and an ordinary goal is that in the later the different groups are timely independent (i.e. they may be true at different time moments). The exact meaning of an m-goal will become clear from the definition of correct answer that follows.

Definition 5.11. If $< \theta, T_1, T_2, ..., T_p >$ is a tuple of a substitution θ and a sequence of temporal references T_i for $i = 1, ..., p$, and G is an m-goal of the form $\leftarrow GP_1, GP_2, ..., GP_p$, where $GP_1, ..., GP_p$ are groups, then $(T_1\ GP_1, T_2\ GP_2, ..., T_p\ GP_p)\theta$ is called a *temporal m-instance* of G. If it is canonical, then it is called a *canonical temporal m-instance* of G.

Definition 5.12. Let P be a program in Cactus(n) and $G = \leftarrow GP_1, ..., GP_p$ a m-goal. A (p+1)-tuple $< \theta, T_1, ..., T_p >$, where $T_1, ..., T_p$ are temporal references and θ a substitution, is said to be a *correct m-answer* for $P \cup \{G\}$ if $P \models \forall(T_i\ GP_i\theta)$ for $i = 1, ..., p$.

Definition 5.13. Let P be a Cactus(n) program and G an m-goal. A *CSLD-derivation* of $P \cup \{G\}$ consists of a (possibly infinite sequence) of m-goals $G_0 = G, G_1, ..., G_n, ...$, a sequence C_1, ..., C_n, ... of program clauses, a sequence (T_1, θ_1, S_1),...., (T_n, θ_n, S_n), ... of most general temporal unifiers, and a sequence of tuples (called *answer tuples*) $< \theta_1, R^1_1, R^1_2, ... >$, ..., $< \theta_n, R^n_1, R^n_2, ... >$, such that for all i, the m-goal G_{i+1} is obtained from the m-goal:

$G_i = \ \leftarrow GP_1, ..., GP_{m-1}, GP_m, GP_{m+1}, ..., GP_p$.

as follows:

1. A is a temporal atom in a group GP_m of G_i (A is called the *selected* atom, and GP_m is called the *selected group*)

2. $H \leftarrow B_1, ..., B_r$ is the input clause C_{i+1} (standardized apart from G_i),

3. $mgu^t(A, H) = (T_{i+1}, \theta_{i+1}, S_{i+1})$

Then G_{i+1} is $\leftarrow (GP_1, ..., GP_{m-1}, GP'_m, GP_{m+1}, ..., GP_p, GP_{new})\theta$ such that, if H is canonical then $GP'_m = T_{i+1}\ (GP_m - \{A\})$, and $GP_{new} = \{S_{i+1}\ B_1, ..., S_{i+1}\ B_r\}$, otherwise $GP'_m = (T_{i+1}\ (GP_m - \{A\})) \cup \{S_{i+1}\ B_1, ..., S_{i+1}\ B_r\}$ and $GP_{new} = \emptyset$.

The corresponding answer tuple is $< \theta_{i+1}, \epsilon, ..., \epsilon, T_{i+1}, \epsilon, ... >$, in which all temporal references are ϵ, except the reference corresponding to the selected group, which is T_{i+1}.

Definition 5.14. A *CSLD-refutation* of $P \cup \{G\}$ is a finite CSLD-derivation of $P \cup \{G\}$ which has the empty m-goal $\Box$ as the last clause of the derivation.

Definition 5.15. Let P be a program in Cactus(n) and G an m-goal consisting of k groups. Suppose that there is a CSLD-refutation of $P \cup \{G\}$ of length n, and let $< \theta_1, R^1_1, R^1_2, ... >$, $< \theta_2, R^2_1, R^2_2, ... >$, ..., $< \theta_n, R^n_1, R^n_2, ... >$ be the sequence of answer tuples. Then, a *computed m-answer* for $P \cup \{G\}$ is the (k+1)-tuple $< \theta, T_1, ..., T_k >$, where $T_1, ..., T_k$ are temporal references and θ a substitution, such that θ is obtained by restricting the composition $\theta_1 .... \theta_n$ to the variables of G, and $T_i = R^n_i, R^{n-1}_i, ..., R^1_i$ for $1 \leq i \leq k$.

5.6 Soundness and Completeness of CSLD-resolution

The soundness lemma 5.1 also hold for the revised CSLD-resolution, i.e. all computed m-answers are also correct m-answers. Moreover, the revised CSLD-resolution is also complete.

Example 5.6. The derivation of example 5.4 proceeds now as follows:

$\leftarrow$ {$\underline{\texttt{first p}}$, `first  q`}.
$\{\epsilon, \{\}, \epsilon\}$ using clause (1)
$\leftarrow$ {`r`}, {$\underline{\texttt{first q}}$}
$\{\epsilon, \{\}, \epsilon\}$ using clause (2)
$\leftarrow$ {$\underline{\texttt{r}}$},{`s`}
{`first`, $\{\}, \epsilon$} using clause (3)
$\leftarrow$ {$\underline{\texttt{s}}$}
{`first next`$_0$, $\{\}, \epsilon$} using clause (4)
$\Box$.

It is easy to see that for any choice of the computation rule we reach the same result.

The completeness of CSLD-resolution is expressed through the following lemmas and theorems.

Lemma 5.2 *Let P be a program in Cactus(n) and $A \in MM_P$. Then $P \cup \{\leftarrow A\}$ has a CSLD-refutation.*

Theorem 5.1 *Let P be a program in Cactus(n) and A a canonical ground temporal atom. Then, $A \in MM_P$ iff there is a CSLD-refutation for $P \cup \{\leftarrow A\}$.*

In the following (completeness) theorem, we show that every canonical instance of a correct m-answer is also a canonical instance of a computed m-answer.

Theorem 5.2 (Completeness) *Let P be a program in Cactus(n), and G a temporal m-goal of the form $\leftarrow GP_1, GP_2, ..., GP_p$. If there is a correct m-answer $< \theta, T_1, T_2, ... >$ for $P \cup \{G\}$, then for every canonical m-instance G' of $(T_1\ GP_1, T_2\ GP_2, ..., T_p\ GP_p)\theta$, there exists a computed m-answer $< \sigma, R_1, R_2, ... >$ for $P \cup \{G\}$ such that G' is also a canonical m-instance of $(R_1\ GP_1, R_2\ GP_2, ..., R_p\ GP_p)\sigma$.*

The independence from the specific choice of the computation rule, can also be shown.

6 Conclusions

Cactus is a temporal logic programming language, which is based on a tree-like notion of time; that is, every moment in time may have more than one next moments. Cactus supports the temporal operator `first` which refers to the beginning of time (or alternatively to the root of the tree) and a set of temporal operators $\texttt{next}_i$ each one referring to a child of the current moment (or alternatively, to a branch of the current node in the tree).

In this paper, we work on resolution-type proof procedures for *Cactus* programs. We first introduce BSLD-resolution, a proof procedure which requires program and goal clauses to be in canonical form. Then, we define a generalized proof procedure, called CSLD-resolution. The particular strength of *CSLD-resolution*, is that it can handle in a more general way open-ended queries, without the need of enumerating all the canonical instances of the goal clause. Moreover, program clauses can be directly used in a refutation, without the need to consider their canonical instances. In this sense, CSLD-resolution generalizes previously introduced proof procedures for linear time logic languages, such as Chronolog and Chronolog($\mathcal{Z}$) [Org91, OWD93] as well as for multidimensional logic programming languages [OD94]. It is easy to see that CSLD-resolution can directly apply to Chronolog programs, which can be seen as programs in Cactus(1). Moreover, CSLD-resolution can be easily extended for the case of multidimensional logic programs.

The generality of CSLD-resolution could lead to efficient implementations for Cactus programs as well as for Chronolog programs. The main source of improvements in efficiency comes from the fact that CSLD-resolution solves open-ended goals directly, i.e. it does not enumerate all canonical instances of an open-ended goal. Thus a CSLD-refutation corresponds to a (possibly infinite) set of BSLD-refutations.

Currently, an experimental WAM-based implementation of Cactus proof system, based on CSLD-resolution, is under development.

References

[BAPM83] M. Ben-Ari, A. Pnueli, and Z. Manna. The Temporal Logic of Branching Time. *Informatica*, pages 207–226, 1983.

[Bau93] M. Baudinet. A simple proof of the completeness of temporal logic programming. In L. Farinas del Cerro and M. Penttonen, editors, *International Logics for Programming*, pages 51–83. Oxford University Press, 1993.

[Brz91] C. Brzoska. Temporal logic programming and its relation to constraint logic programming. In *Proc. of the Logic Programming Symposium*, pages 661–677. MIT Press, 1991.

[GRP96] M. Gergatsoulis, P. Rondogiannis, and T. Panayiotopoulos. Disjunctive Chronolog. In M. Chacravarty, Y. Guo, and T. Ida, editors, *Proceedings of the JICSLP'96 Post-Conference Workshop "Multi-Paradigm Logic Programming"*, pages 129–136, Bonn, 5-6 Sept. 1996.

[Hry93] T. Hrycej. A temporal extension of Prolog. *The Journal of Logic Programming*, 15:113–145, 1993.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.

[LO96] C. Liu and M. Orgun. Knowledge-based simulation with Chronolog. In M. A. Orgun and E. A. Arhcroft, editors, *Intensional Programming I*, pages 281–295. World-Scientific Publishing Company, Singapore, 1996.

[LO97] C. Liu and M. A. Orgun. Specifying timing properties of reactive systems with TCL. In *Proc. of the 10th Australasian Computer Science Conference, Syndney, Australia, February 5-7*, pages 150–157, 1997.

[OD94] M. A. Orgun and W. Du. Multi-dimensional logic programming. *Journal of Computing and Information*, 1(1):1501–1520, 1994. Special Issue: Proc. of the 6th International Conf. on Computing and Information.

[OM94] M. A. Orgun and W. Ma. An overview of temporal and modal logic programming. In *Proc. of the First International Conference on Temporal Logics (ICTL'94)*, pages 445–479. Springer Verlag, 1994. LNCS No 827.

[Org91] M. A. Orgun. *Intensional Logic Programming.* PhD thesis, Dept. of Computer Science, University of Victoria, Canada, December 1991.

[Org96] M. A. Orgun. On temporal deductive databases. *Computational Intelligence*, 12(2):235–259, 1996.

[OW93] M. A. Orgun and W. W. Wadge. Chronolog admits a complete proof procedure. In *Proc. of the Sixth International Symposium on Lucid and Intensional Programming (ISLIP'93)*, pages 120–135, 1993.

[OWD93] M. A. Orgun, W. W. Wadge, and W. Du. Chronolog($\mathcal{Z}$): Linear-time logic programming. In O. Abou-Rabia, C. K. Chang, and W. W. Koczkodaj, editors, *Proc. of the fifth International Conference on Computing and Information*, pages 545–549. IEEE Computer Society Press, 1993.

[RGP97a] P. Rondogiannis, M. Gergatsoulis, and T. Panayiotopoulos. The branching-time logic programming language Cactus and its applications. Technical Report 3-97, Dept. of Computer Science, University of Ioannina, Greece, 1997.

[RGP97b] P. Rondogiannis, M. Gergatsoulis, and T. Panayiotopoulos. *Cactus*: A branching-time logic programming language. In *Proc. of the First International Joint Conference on Qualitative and Quantitative Practical Reasoning; ECSQARU-FAPR'97, Bad Honnef, Germany*, Lecture Notes in Artificial Intelligence LNAI 1244. Springer, June 1997.

[RGP97c] P. Rondogiannis, M. Gergatsoulis, and T. Panayiotopoulos. Theoretical foundations of Branching-Time Logic Programming. 1997. In preparation.

[Vil94] L. Vila. A survey on temporal reasoning in artificial intelligence. *AI Communications*, 7(1):4–28, 1994.

[Wad88] W. W. Wadge. Tense logic programming: A respectable alternative. In *Proc. of the 1988 International Symposium on Lucid and Intensional Programming*, pages 26–32, 1988.