Intensional Programming II

Based on the Papers at ISLIP '99

Editors

Manolis Gergatsoulis
Institute of Informatics & Telecommunications
NCSR "Demokritos" Athens, Greece

Panos Rondogiannis
Department of Computer Science
University of Ioannina
Ioannina, Greece

World Scientific
Singapore • New Jersey • London • Hong Kong

EXTENSIONS OF THE BRANCHING-TIME LOGIC PROGRAMMING LANGUAGE *CACTUS*

MANOLIS GERGATSOULIS
Institute of Informatics and Telecommunications,
National Centre for Scientific Research (NCSR) "Demokritos",
153 10 Aghia Paraskevi Attikis, GREECE
E-mail: manolis@iit.demokritos.gr

Cactus has been proposed as a temporal logic programming language based on the branching notion of time. Cactus supports two main operators: the temporal operator `first` which refers to the first moment in time and the temporal operator $\mathtt{next}_i$ which refers to the i-th child of the current moment. Actually by $\mathtt{next}_i$ we denote a family $\{\mathtt{next}_i \mid i \in \mathcal{N}\}$ of `next` operators, each one referring to a different next moment that immediately follows the present one. In this paper we propose the extension of *Cactus* with new temporal operators. More specifically, we investigate the use of two variants of the operator `next` namely the operators $\bigcirc_F$ and $\bigcirc_G$ referring to "some next moment" and "all next moments" respectively. We also investigate the use of branching variants of the well known temporal operators $\Box$ (always) and $\Diamond$ (sometime).

1 Introduction

Temporal logic programming languages[1,2] provide a powerful means for the description and implementation of *dynamic* systems. Most temporal logic programming languages[1,3,4,5,6,7] are based on linear time temporal logics. Recently some research work has been done in the direction of developing temporal logic programming languages based on the branching notion of time. In[8,9,10] the temporal logic programming language **Cactus** has been presented. Cactus is based on a tree-like notion of time; that is, every moment in time may have more than one immediate next moments.

Cactus supports two main temporal operators: the operator `first` which refers to the beginning of time (or alternatively to the root of the tree) and the operator $\mathtt{next}_i$ which 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 $\{\mathtt{next}_i \mid i \in \mathcal{N}\}$ of `next` operators, each one of them referring to a different next moments that immediately follows the present one.

As an example consider the non-deterministic finite automaton shown in figure 1 which accepts the regular language $L = (01 \cup 010)^*$. The behaviour

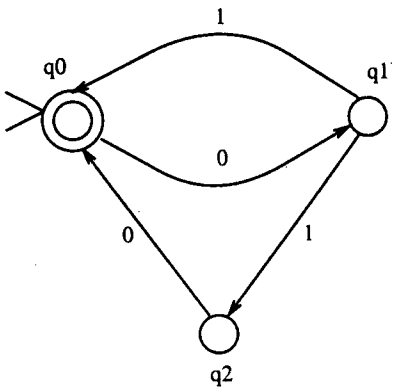

Figure 1. A non-deterministic finite automaton

of this automaton can be described by the following Cactus program:

$$\begin{aligned}
&\texttt{first state(q0).}\\
&\texttt{next}_0\ \texttt{state(q1)} \leftarrow \texttt{state(q0).}\\
&\texttt{next}_1\ \texttt{state(q0)} \leftarrow \texttt{state(q1).}\\
&\texttt{next}_1\ \texttt{state(q2)} \leftarrow \texttt{state(q1).}\\
&\texttt{next}_0\ \texttt{state(q0)} \leftarrow \texttt{state(q2).}
\end{aligned}$$

Notice that q0 is both the initial and the final state. Posing the goal clause:

$$\leftarrow \texttt{first next}_0\ \texttt{next}_1\ \texttt{next}_0\ \texttt{state(q0).}$$

will return the answer **yes** which indicates that the string 010 belongs to the language L.

In[10], it is argued that Cactus is appropriate for describing non-deterministic computations or more generally computations that involve the manipulation of trees. Moreover, as it is shown in[11], a fragment of Cactus can be used as a target language for the transformation of a subclass of Datalog programs, namely the chain Datalog ones. The Cactus programs obtained by applying this transformation are simpler in structure and it is believed (as similar techniques have been successfully used in functional programming[12]) that they can be implemented efficiently.

In this paper, we propose the extension of Cactus with more expressive temporal operators. More specifically, we investigate the extension of Cactus with two variants of the operator **next** namely the operators $\bigcirc_F$ and $\bigcirc_G$ referring to "some next moment" and "all next moments" respectively. We

also investigate the use of branching variants of the well known temporal operators $\Box$ (always) and $\Diamond$ (sometime). In particular, we use the operators $\Diamond_F$ and $\boxdot_G$ whose intuitive meaning is "some moment in some future of the first moment in time" and "all moments in all futures of the first moment in time" respectively.

We show how we can extend the syntax of Cactus by allowing the use of $\bigcirc_G$, and $\boxdot_G$ in the clause heads and the use of $\bigcirc_F$, and $\Diamond_F$ in the clause bodies, while retaining the SLD-resolution style of the proof procedure.

The rest of the paper is organized as follows: The temporal logic on which the extended Cactus is based is presented in section 2. The syntax of the extended Cactus programs is presented in section 3. The declarative semantics of the extended Cactus programs is given in section 4. In section 5, a SLD-resolution like roof procedure for extended Cactus programs is outlined. Finally, section 6 gives the concluding remarks.

2 The branching-time temporal logic

2.1 Temporal operators

In[8,9,10] it is presented a simple *branching-time temporal logic* (BTL) on which the language Cactus is based. In BTL, time varies over a tree-like structure. The set of moments in time is modeled by the set $List(\mathcal{N})$ of lists of natural numbers $\mathcal{N}$. Each node may have a countably infinite number of branches. The empty list [ ] corresponds to the beginning of time and the list $[i|t]$ (that is, the list with head $i \in \mathcal{N}$, and tail t) corresponds to the i-th child of the moment identified by the list t. In BTL there are two temporal operators namely the operators **first** and $\mathbf{next}_i$, $i \in \mathcal{N}$. The operator **first** refers to the first moment in time, while the operator $\mathbf{next}_i$ refers to the i-th child of the current moment in time.

Branching-time logics with richer sets of temporal operators than those of BTL have been proposed in the literature[13,14]. In this section, we consider the extension of BTL with the temporal operators $\bigcirc_G$, $\bigcirc_F$, $\Diamond_F$ and $\Box_G$. The intuitive meaning of these temporal operators is as follows:

$\bigcirc_G A$: holds at t iff the formula A is true at every immediate successor of t.

$\bigcirc_F A$: holds at t iff there is an immediate successor of t at which A is true.

$\Diamond_F A$: holds at t iff there is some node in the subtree rooted from t at which A is true.

$\Box_G A$: holds at t iff A is true at all nodes of the subtree rooted at t.

The syntax of the formulae of the extended BTL extends the syntax of first order logic with four new formation rules: If A is a formula, so are **first** A, **next**$_i$ A, $\bigcirc_F A$, $\bigcirc_G A$, $\Diamond_F A$, and $\Box_G A$.

In the following we will also use the operators $\Diamond_F$ and $\boxdot_G$, as shorthands for the sequences of operators "**first** $\Diamond_F$" and "**first** $\Box_G$" respectively.

Notice that the above operators are simiral to the operators of the branching-time logic proposed by Ben-Ari, Pnueli and Manna in[13]. In particular, the operators $\Box_G$, $\bigcirc_G$, and $\Diamond_F$ are denoted in[13] as $\forall G$, $\forall X$, and $\exists F$ respectively. In[13] two more operators are considered namely the operators $\forall F$ and $\exists G$ (in our notation $\Box_F$ and $\Diamond_G$ respectively). The intuitive meaning of these operators is:

$\Diamond_G A$: holds at t iff for all paths departing from t there is a time point (node) in which A is true.

$\Box_F A$: holds at t iff there is a path departing from t such that A is true at all time points (nodes) of this path.

We do not include these operators in our extended BTL.

2.2 Semantics of the formulas of the extended BTL

The semantics of the formulas of the extended BTL are given using the notion of *branching temporal interpretation.*

Definition 2.1. A *branching temporal interpretation* (or simply *temporal interpretation*) I of the extended BTL 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 constant; for each n-ary function symbol, an element of $[D^n \to D]$; and for each n-ary predicate symbol, an element of $[List(\mathcal{N}) \to 2^{D^n}]$.

In the following definition, the satisfaction relation $\models$ is defined in terms of temporal interpretations. $\models_{I,t} A$ denotes that the formula A is true at the moment t in the temporal interpretation I. Finally, by $s \circ t$ we denote the list obtained by concatenating the lists s and t.

Definition 2.2. The semantics of the elements of the extended BTL 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 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. $\models_{I,t} A \vee B$ *iff* $\models_{I,t} A$ *or* $\models_{I,t} B$

5. $\models_{I,t} (\forall \mathbf{x})A$ *iff* $\models_{I[d/\mathbf{x}],t} A$ *for all* $d \in D$ *where the interpretation* $I[d/\mathbf{x}]$ *is the same as* I *except that the variable* $\mathbf{x}$ *is assigned the value* d.

6. $\models_{I,t}$ **first** A *iff* $\models_{I,[\,]} A$

7. $\models_{I,t}$ $\mathbf{next}_i$ A *iff* $\models_{I,[i|t]} A$

8. $\models_{I,t} \bigcirc_G A$ *iff for all* $i \in \mathcal{N}, \models_{I,[i|t]} A$

9. $\models_{I,t} \bigcirc_F A$ *iff for some* $i \in \mathcal{N}, \models_{I,[i|t]} A$

10. $\models_{I,t} \Diamond_F A$ *iff for some* $s \in List(\mathcal{N}), \models_{I,sot} A$

11. $\models_{I,t} \Box_G A$ *iff for all* $s \in List(\mathcal{N}), \models_{I,sot} A$

Since $\Diamond_F$ and $\boxdot_G$ are shorthands of the sequences "**first** $\Diamond_F$" and "**first** $\Box_G$" respectively, it is easy to see that:

1. $\models_{I,t} \Diamond_F A$ *iff for some* $s \in List(\mathcal{N}), \models_{I,s} A$

2. $\models_{I,t} \boxdot_G A$ *iff for all* $s \in List(\mathcal{N}), \models_{I,s} A$

The semantics of formulas involving the symbols $\leftarrow$, $\rightarrow$, $\leftrightarrow$, $\wedge$, and $\exists$ are defined in the usual way with respect to the semantics of $\vee$, $\neg$ and $\forall$.

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

2.3 Axioms

In this section we present some axioms of the extended BTL which will be used in this paper. In the following, the symbol ∇ stands for either of **first**, $\mathbf{next}_i$, $\bigcirc_G$, $\bigcirc_F$, $\Box_G$, and $\Diamond_F$. We do not present axioms for the operators $\Diamond_F$ and $\boxdot_G$ since these axioms immediately follow from the axioms of the operators composing the operators $\Diamond_F$ and $\boxdot_G$.

Temporal operator cancellation rules:

$$\nabla(\mathbf{first}\ A) \leftrightarrow (\mathbf{first}\ A) \tag{1}$$

Temporal operator distribution rules:

$$\texttt{first}\ (A \wedge B) \leftrightarrow (\texttt{first}A) \wedge (\texttt{first}\ B) \tag{2}$$
$$\texttt{next}_i\ (A \wedge B) \leftrightarrow (\texttt{next}_i A) \wedge (\texttt{next}_i\ B) \tag{3}$$
$$\nabla(A \wedge \texttt{first}\ B) \leftrightarrow\ \nabla A \wedge \texttt{first}\ B \tag{4}$$
$$\texttt{first}\ (\neg A) \leftrightarrow\ \neg(\texttt{first}\ A) \tag{5}$$
$$\texttt{next}_i\ (\neg A) \leftrightarrow\ \neg(\texttt{next}_i\ A) \tag{6}$$
$$\bigcirc_F\ (\neg A) \leftrightarrow\ \neg\ (\bigcirc_G\ A) \tag{7}$$
$$\Diamond_F(\neg A) \leftrightarrow\ \neg\ (\Box_G\ A) \tag{8}$$

Notice that although the following formulas are valid:

$$\Diamond_F(A \wedge B) \rightarrow \Diamond_F A \wedge \Diamond_F B \tag{9}$$
$$\bigcirc_F(A \wedge B) \rightarrow \bigcirc_F A \wedge \bigcirc_F B \tag{10}$$

the inverse implications are not valid.

Other useful axioms The following formulas are valid:

$$\Box_G\ A \rightarrow\ \Box_G\ \bigcirc_G\ A \tag{11}$$
$$\Diamond_F\ \bigcirc_F\ A \rightarrow \Diamond_F A \tag{12}$$

Notice also that although the extended BTL formula:

$$\bigcirc_F \Diamond_F A\ \leftrightarrow \Diamond_F\ \bigcirc_F\ A \tag{13}$$

is a tautology, this is not true for the formula

$$\texttt{next}_i\ \Diamond_F\ A\ \leftrightarrow \Diamond_F\ \texttt{next}_i\ A \tag{14}$$

Rigidness of variables: The following tautologies state that the temporal operators **first**, $\texttt{next}_i$ and $\Box_G$ can "pass inside" $\forall$:

$$\texttt{first}\ (\forall X)(A) \leftrightarrow (\forall X)(\texttt{first}\ A) \tag{15}$$
$$\texttt{next}_i\ (\forall X)(A) \leftrightarrow (\forall X)(\texttt{next}_i\ A) \tag{16}$$
$$\Box_G(\forall X)(A) \leftrightarrow (\forall X)(\Box_G A) \tag{17}$$

However, the operator $\Diamond_F$ cannot pass inside $\forall$ since although the implication

$$\Diamond_F(\forall X)(A) \rightarrow (\forall X)(\Diamond_F A) \tag{18}$$

is valid, the inverse implication is not.

The validity of formulas 15, 16 and 17 expresses the fact that variables represent data-values which are independent of time (i.e. they are *rigid*).

We should also note that the formulas $\texttt{next}_i\ \texttt{next}_j\ A$ and $\texttt{next}_j\ \texttt{next}_i\ A$ are not equivalent in general when $i \neq j$.

3 Syntax of extended Cactus programs

Programs in extended Cactus extend classical Horn clause programs by allowing the temporal operators **first**, $\texttt{next}_i$, $\bigcirc_G$, and $\boxdot_G$ to be used in the heads of the clauses and the temporal operators **first**, $\texttt{next}_i$, $\bigcirc_F$, and $\Diamond_F$ to be used in the bodies of the clauses. The syntax of the extended Cactus programs is given formally by the following definitions:

Definition 3.1. A *goal* is defined as follows:
- A classical atom A is an *open goal.*
- If G is an open goal then $\bigcirc_F\ G$ and $\texttt{next}_i\ G$ are also *open goals.*
- If G is an open goal then $\texttt{first}\ G$, and $\Diamond_F\ G$, are *fixed goals.*
- An open goal or a fixed goal is a *goal.*
- If G_1 and G_2 are goals then $G_1 \wedge G_2$ is a *goal.* $G_1 \wedge G_2$ is a *fixed goal* if both G_1 and G_2 are fixed goals otherwise $G_1 \wedge G_2$ is an *open goal.*

Definition 3.2. A *head* is defined as follows:
- An atom A is an *open head.*
- If H is an open head then $\bigcirc_G\ H$ and $\texttt{next}_i\ H$ are also *open heads.*
- If H is an open head then $\texttt{first}\ H$ and $\boxdot_G\ H$ are *fixed heads.*
- A *head* is either an open head or a fixed head.

Definition 3.3. An *extended Cactus clause* is a formula of the form $H \leftarrow G$ where H is a head and G is a (possibly empty) goal. An *extended Cactus program* is a set of extended Cactus clauses.

Example 3.1. The following set of clauses is an extended Cactus program which defines the relations "**parent**", "**sibling**", "**uncle**" and "**grandparent**".

(1) $\texttt{parent(X,Y)} \leftarrow \texttt{node(X)}, \bigcirc_F\ \texttt{node(Y)}.$
(2) $\texttt{sibling(X,Y)} \leftarrow \bigcirc_F\ \texttt{node(X)}, \bigcirc_F\ \texttt{node(Y)}.$
(3) $\texttt{uncle(X,Z)} \leftarrow \texttt{sibling(X,Y)}, \bigcirc_F\ \texttt{parent(Y,Z)}.$
(4) $\texttt{grandparent(X,Y)} \leftarrow \texttt{node(X)}, \bigcirc_F\ \bigcirc_F\ \texttt{node(Y)}.$

(5) $\texttt{first node(john)}.$
(6) $\texttt{first next}_0\ \texttt{node(nick)}.$
(7) $\texttt{first next}_1\ \texttt{node(steve)}.$
(8) $\texttt{first next}_0\ \texttt{next}_0\ \texttt{node(edward)}.$
(9) $\texttt{first next}_0\ \texttt{next}_1\ \texttt{node(peter)}.$
(10) $\texttt{first next}_1\ \texttt{next}_0\ \texttt{node(bill)}.$
(11) $\texttt{first next}_1\ \texttt{next}_1\ \texttt{node(mike)}.$

□

Example 3.2. Consider te following clauses redefining the relations "parent", "sibling" and "grandparent" of example 3.1:

(1′) $\texttt{parent(X, Y)} \leftarrow \Diamond_F(\texttt{node(X)}, \bigcirc_F\ \texttt{node(Y)})$.
(2′) $\texttt{sibling(X, Y)} \leftarrow \Diamond_F(\bigcirc_F\ \texttt{node(X)}, \bigcirc_F\ \texttt{node(Y)})$.
(4′) $\texttt{grandparent(X, Y)} \leftarrow \Diamond_F(\texttt{node(X)}, \bigcirc_F\ \bigcirc_F\ \texttt{node(Y)})$.

The difference between these definitions and the corresponding definitions of example 3.1 is that in example 3.1 an instance of one of these relations may be true in a specific time point and false in some other time points of the time tree. On the other hand, the relations "parent", "sibling" and "grandparent" defined by the clauses 1′, 2′, 4′ are "time-independent" in the sense that if an instance of one of these relations is true in a specific time point then it is true in all time points of the time tree. □

Definition 3.4. A head H is said to be in *normal form* if there are no occurrences of the operators first and $\boxdot_G$ in H in the scope of any other operator. A goal is in *normal form* if there are no occurrences of the operators first or $\Diamond_F$ in the scope of any other operator in the goal. A clause is in normal form if its head and its body are in normal form.

In the rest of this paper we suppose that all clauses of extended Cactus programs are formulas in normal form.

4 Declarative Semantics

The declarative semantics of the extended Cactus programs is defined in terms of the minimal temporal Herbrand models. For this we are based on the notion of *canonical temporal context/atom/clause*, initially introduced in the context of the linear time temporal logic programming language Chronolog[7,3].

Definition 4.1. The sequence of the temporal operators which have an atom in their scope is said to be the *temporal context* of that atom. The *length* of a temporal context T (denoted by $length(T)$) is the number of temporal operators in T. A *temporal atom* is a classical atom preceded by its temporal context. A *canonical temporal context* is a temporal context of the form $\texttt{first next}_{i_1} \cdots \texttt{next}_{i_n}$, where $i_1, \ldots, i_n \in \mathcal{N}$ and $n \geq 0$. A *canonical temporal atom* is a temporal atom whose temporal context is canonical. A *canonical temporal clause* is a temporal clause whose temporal atoms are canonical.

Definition 4.2. A *canonical temporal instance of a temporal clause* C is a canonical temporal clause C' obtained as follows:

- Replace each occurrence of $\bigcirc_G$ in the head of C by $\texttt{next}_i$ for some $i \in \mathcal{N}$.
- Replace each occurrence of $\bigcirc_F$ in the body of C by $\texttt{next}_j$ for some $j \in \mathcal{N}$.
- Replace each occurrence of $\Diamond_F$ in the body of C by $\texttt{first next}_{i_1} \ldots \texttt{next}_{i_n}$ where $n \geq 0$ and $i_1, \ldots, i_n \in \mathcal{N}$.
- Replace the occurrence of $\Box_G$ in the head of C (if any) by $\texttt{first next}_{i_1} \ldots \texttt{next}_{i_m}$ where $m \geq 0$ and $i_1, \ldots, i_m \in \mathcal{N}$.
- Apply the same canonical temporal context to the normal form of C.

The *canonical temporal instance of an extended Cactus program* P is the (possibly infinite) extended Cactus program P' consisting of all canonical temporal instances of all clauses in P.

Intuitively, a canonical temporal instance of a temporal clause C is an instance in time of C. Using the definition 4.2 we can obtain the set of all temporal instances of a given program clause.

Example 4.1. Consider the following clause taken from example 3.1:

$$\texttt{grandparent(X, Y)} \leftarrow \texttt{node(X)}, \bigcirc_F \ \bigcirc_F \ \texttt{node(Y)}.$$

The set of the canonical temporal instances corresponding to this clause is:

$$\{\texttt{first next}_{i_1} \cdots \texttt{next}_{i_n} \ \texttt{grandparent(X, Y)} \leftarrow \texttt{first next}_{i_1} \cdots \texttt{next}_{i_n} \ \texttt{node(X)}, \\ \texttt{first next}_{i_1} \cdots \texttt{next}_{i_n} \ \texttt{next}_j \ \texttt{next}_k \ \texttt{node(Y)} \mid \\ i_1, \ldots, i_n \in \mathcal{N}, n \geq 0, j \in \mathcal{N}, k \in \mathcal{N}\}$$

□

The notion of canonical instance of a clause is very important since the truth value of a given clause in a temporal interpretation, can be expressed in terms of the values of its canonical instances, as the following lemma shows:

Lemma 4.1 *Let C be a clause and I a temporal interpretation of extended BTL. $\models_I C$ if and only if $\models_I C_t$ for all canonical instances C_t of C.*

The domain of the temporal Herbrand interpretations of a program P is its *temporal Hebrand universe* U_P, generated by constant and function symbols that appear in P. The *temporal Herbrand base* B_P of P consists of all canonical temporal atoms generated by the predicates of P with terms in U_P used as arguments. A *temporal Herbrand interpretation* is a subset of B_P. A *temporal Herbrand model* of a program P is a temporal Herbrand interpretation which is a model of P.

It can be proved that every extended Cactus program P has a unique *minimal temporal Herbrand model* M_P which consists of all ground canonical temporal atoms which are logical consequences of P.

5 A proof procedure for extended Cactus programs

In this section we outline a resolution-type proof procedure for extended Cactus programs, called *ECSLD-resolution* (*E*xtended *C*actus *SLD-resolution*). In the following by $tc(A)$ we denote the temporal context of an occurrence of an atom A in a temporal clause.

Example 5.1. The temporal contexts of the atoms in the clause:

$$\texttt{first next}_0\ \texttt{A} \leftarrow \bigcirc_F\ \texttt{B}, \Diamond_F(\bigcirc_F\ (\texttt{C}, \texttt{next}_1\ \texttt{D}), \texttt{E}).$$

are:

$tc(\texttt{A}) = \texttt{first next}_0$
$tc(\texttt{B}) = \bigcirc_F$
$tc(\texttt{C}) = \Diamond_F \bigcirc_F$
$tc(\texttt{D}) = \Diamond_F \bigcirc_F \texttt{next}_1$
$tc(\texttt{E}) = \Diamond_F$

□

Definition 5.1. If a temporal context T is fixed and Op is the leftmost operator in T then we say that T *is fixed by* Op.

Notice that, since all program clauses are supposed to be in normal form, the operator Op in the above definition is one of $\texttt{first}$, $\Diamond_F$ and $\boxdot_G$.

In order to facilitate the definition of the proof procedure we map the context T of a head atom into a context denoted by $b(T)$, called the *corresponding body context* of T, by replacing each occurrence of $\bigcirc_G$ in T by $\bigcirc_F$ and each occurrence of $\boxdot_G$ by $\Diamond_F$. Moreover, by $open(T)$ we denote the temporal context obtained as follows: If T is open then $open(T) = T$, other-

wise $open(T)$ is obtained by removing the leftmost operator of T. Finally, by $open\Diamond_F(T)$ we denote the temporal context obtained as follows: If T is fixed by $\Diamond_F$ then $open\Diamond_F(T)$ is obtained by removing the leftmost operator of T, otherwise $open\Diamond_F(T) = T$.

Definition 5.2. Two temporal body contexts T_1 and T_2 are said to be *non-unifiable* if when we traverse in parallel $open\Diamond_F(T_1)$ and $open\Diamond_F(T_2)$ from right to left, we find a pair of corresponding operators which is either $(\mathbf{next}_i, \mathbf{next}_j)$, with $i \neq j$, or one of the operators is `first` and the other is either $\mathbf{next}_i$, or $\bigcirc_F$. Two temporal body contexts T_1 and T_2 are said to be *unifiable* if they are not non-unifiable.

Definition 5.3. Let T_1 and T_2 be two unifiable temporal body contexts. We say that T_1' is obtained by *instantiating* T_1 *with respect to* T_2 if T_1' is obtained from T_1 by replacing each occurrence of $\bigcirc_F$ in T_1 which correspond to an operator $\mathbf{next}_i$ in T_2, by $\mathbf{next}_i$.

Definition 5.4. Let T_1 and T_2 be two unifiable temporal body contexts, and $T_1' = open(T_1)$ and $T_2' = open(T_2)$. Let m be the minimum of $length(T_1')$ and $length(T_2')$. We obtain a pair of temporal body contexts (R_b, R_h) which we call a *prefix pair* of T_1 and T_2, as follows: We discurd the m rightmost operators of each one of T_1' and T_2' obtaining T_1'' and T_2'' respectively. Then $(R_b, R_h) = (T_2'', T_1'')$.

Notice that at least one of R_b, R_h in the above definition is the empty temporal context ϵ.

Definition 5.5. Let P be a program in extended Cactus and G be a (canonical) goal clause. An *ECSLD-derivation* of $P \cup \{G\}$ consists of a (possibly infinite) sequence of temporal goals $G_0 = G, G_1, \ldots, G_n, \ldots$ a sequence $C_1, \ldots, C_n, \ldots$ of clauses of P (called the *input clauses*), a sequence θ_1, $\ldots$, θ_n $\ldots$ of most general unifiers, and a sequence of prefix pairs (S_1^B, S_1^C), $\ldots$, (S_n^B, S_n^C), $\ldots$ such that for all i, the goal G_{i+1} is obtained from the goal G_i as follows:

1. T_B B is a temporal atom (B is the classical atom and T_B its temporal context) in G_i (called the *selected* atom)
2. T_H $B' \leftarrow BodyC$ is the input clause C_{i+1} (standardized apart from G_i)
3. $\theta_{i+1} = mgu(B, B')$ and (S_{i+1}^B, S_{i+1}^C) is the prefix pair of T_B and $b(T_H)$.
4. The new goal G_{i+1} is obtained as follows:

(a) If T_B is fixed by **first** and T_H is open then the new goal G_{i+1} is $\leftarrow (G',$ **first** S^C_{i+1} $BodyC)\theta_{i+1}$, where G' is obtained from G_i by removing B, and instantiating T_B with respect to $b(T_H)$.

(b) If T_B is fixed by **first** and T_H is fixed then G_{i+1} is $\leftarrow (G', \Diamond_F BodyC)\theta_{i+1}$ where G' is obtained from G_i by removing B and instantiating T_B with respect to $b(T_H)$.

(c) If both T_B and T_H are open then: if S^B_{i+1} is ϵ then G_{i+1} is $\leftarrow G'\theta_{i+1}$ where G' is obtained from G_i by removing B, puting the $BodyC$ in the scope of the prefix S^C_{i+1} of T_B, and instantiating T_B with respect to $b(T_H)$. Otherwise G_{i+1} is $\leftarrow (G',\ BodyC)\theta_{i+1}$, where G' is obtained from G_i by removing B, instantiating T_B with respect to $b(T_H)$ and putting S^B_{i+1} before the leftmost operator of T_B.

(d) If T_B is fixed by $\Diamond_F$ and T_H is fixed then G_{i+1} is $\leftarrow (G',\ \Diamond_F\ BodyC)\theta_{i+1}$, where G' is obtained from G_i by removing B, instantiating T_B with respect to $b(T_H)$, and replacing the $\Diamond_F$ in T_B by $O_p\ S^B_{i+1}$, where O_p is the leftmost operator of $b(T_H)$ (i.e. **first** or $\Diamond_F$).

(e) If T_B is fixed by $\Diamond_F$ and T_H is open then G_{i+1} is $\leftarrow (G')\theta_{i+1}$, where G' is obtained from G_i by removing B, instantiating T_B with respect to $b(T_H)$, replacing $\Diamond_F$ in T_B by '$\Diamond_F\ S^B_{i+1}$' and putting $BodyC$ in the scope of the prefix $\Diamond_F\ S^C_{i+1}$ of T_B.

(f) If T_B is open and T_H is fixed by $\Diamond_F$ then G_{i+1} is $\leftarrow (G',\ \Diamond_F\ BodyC)\theta_{i+1}$, where G' is obtained from G_i by removing B, instantiating T_B with respect to $b(T_H)$ and putting '$\Diamond_F\ S^B_{i+1}$' before the leftmost operator of T_B.

(g) If T_B is open and T_H is fixed by **first** then G_{i+1} is $\leftarrow (G',\ \Diamond_F\ BodyC)\theta_{i+1}$, where G' is obtained from G_i by removing B, instantiating T_B with respect to $b(T_H)$ and putting '**first** S^B_{i+1}' before the leftmost operator of T_B.

Definition 5.6. Let P be a program in extended Cactus and G be a (canonical) goal clause. An *ECSLD-refutation* of $P \cup \{G\}$ is a finite ECSLD-derivation of $P \cup \{G\}$ which has the empty goal clause $\square$ as the last clause of the derivation.

Definition 5.7. Let P be a program and G be a canonical temporal goal. A *computed answer* for $P \cup \{G\}$ is the substitution obtained by restricting

the composition $\theta_1\theta_2\ldots\theta_n$ to the variables of G, where $\theta_1, \theta_2, \ldots, \theta_n$, is the sequence of the most general unifiers used in a ECSLD-refutation of $P \cup \{G\}$.

Example 5.2. Consider the program in example 3.1. An ECSLD-refutation of the canonical temporal goal:

$\leftarrow$ `first uncle(X,Z)`.

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

$\leftarrow$ `first uncle(X,Z)`.

using clause (3)

$\leftarrow$ `first sibling(X,Y)`, `first` $\bigcirc_F$ `parent(Y,Z)`

using clause (2)

$\leftarrow$ `first` $\bigcirc_F$ `node(X)`, `first` $\bigcirc_F$ `node(Y)`, `first` $\bigcirc_F$ `parent(Y,Z)`

(`Y = nick`) using clause (6)

$\leftarrow$ `first` $\bigcirc_F$ `node(X)`, `first` $\bigcirc_F$ `parent(nick,Z)`

using clause (1)

$\leftarrow$ `first` $\bigcirc_F$ `node(X)`, `first` $\bigcirc_F$ `(node(nick),` $\bigcirc_F$ `node(Z))`

(`Z = peter`) using clause (9)

$\leftarrow$ `first` $\bigcirc_F$ `node(X)`, `first next`$_0$ `node(nick)`

using clause (6)

$\leftarrow$ `first` $\bigcirc_F$ `node(X)`

(`X = steve`) using clause (7)

□

From the above derivation we conclude that `first uncle(steve,peter)` is a logical consequence of the program. □

Example 5.3. Consider the program obtained by replacing the clauses $\{1, 2, 4\}$ in the program of example 3.1 by the clauses $\{1', 2', 4'\}$ of example 3.2. An ECSLD-refutation of the goal:

$\leftarrow$ **first sibling(edward,X)**.

is given below:

$\leftarrow$ **first sibling(edward,X)**.

using clause (2′)

$\leftarrow \Diamond_F$ ($\bigcirc_F$ **node(edward)**, $\bigcirc_F$ **node(X)**).

using clause (8)

$\leftarrow \Diamond_F$ **next**$_0$ $\bigcirc_F$ **node(X))**.

(**X = peter**) using clause (9)

□ □

In the presentation of the proof procedure we have supposed that the top-level goal is canonical. However, the proof procedure can be extended for the case of non-canonical top level goals. In this case, besides the substitutions of the variables of the goal, we may also find a sequence of temporal operators which should be applied to the goal in order to be logical consequence of the program.

6 Discussion

In this paper, we investigate the extension of the branching-time logic programming language Cactus[10] with new, more expressive, temporal operators. In particular, we are interested in extensions of Cactus for which we can define SLD-resolution style refutation proof procedures. We show how we can extend Cactus to allow also the use of the operators $\bigcirc_F$, and $\Diamond_F$ in the bodies of the clauses and the operators $\bigcirc_G$, and $\Box_G$ in the heads of the clauses.

Following the work concerning the linear time logic programming language TEMPLOG[4], in which the linear time temporal operator $\Diamond$ is allowed in the bodies of the clauses, we examined the possibility to allow the unlimited use of the operator $\Diamond_F$ in the clause bodies. However, this seems to be a difficult task for the case of branching-time logic programming. The reason is that while in the linear case the formula:

$$\textbf{next} \Diamond A \leftrightarrow \Diamond \textbf{next} A$$

is valid and thus it is always possible to move $\Diamond$ in front of a temporal context, in banching time logic BTL the formula:

$$\mathbf{next}_i \Diamond_F A \leftrightarrow \Diamond_F \mathbf{next}_i A$$

is not valid and thus $\Diamond_F$ cannot allways be pulled to the front of a temporal context. The existence of $\Diamond_F$ in the context of a selected atom, in places other than the leftmost one, introduces nondeterminism when attempting to unify temporal contexts that contain $\Diamond_F$. Because of this difficulty, we only allow the use of a restricted form of this operator (i.e. the operator $\Diamond_F$ which is equivalent to the operator $\Diamond$ preceded by the operator `first`).

An interesting topic for future work is to investigate the possibility to allow also the use of the operators $\Diamond_G$ and $\Box_F$ in the syntax of the extended Cactus.

Acknowledgements

I would like to thank C. Nomikos for his helpful comments.

References

1. M. A. Orgun and W. Ma. An overview of temporal and modal logic programming. In D. M. Gabbay and H. J. Ohlbach, editors, *Proc. of the First International Conference on Temporal Logics (ICTL'94)*, Lecture Notes in Computer Science (LNCS) 827, pages 445–479. Springer-Verlag, 1994.
2. M. Fisher and R. Owens. An introduction to executable modal and temporal logics. Lecture Notes in Artificial Intelligence (LNAI) 897. Springer-Verlag, February 1995.
3. 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.
4. M. Baudinet. A simple proof of the completeness of temporal logic programming. In L. Farinas del Cerro and M. Penttonen, editors, *Intensional Logics for Programming*, pages 51–83. Oxford University Press, 1993.
5. C. Brzoska. Temporal logic programming with bounded universal modality goals. In D. S. Warren, editor, *Proc. of the Tenth International Conference on Logic Programming*, pages 239–256. MIT Press, 1993.
6. M. Gergatsoulis, P. Rondogiannis, and T. Panayiotopoulos. Disjunctive Chronolog. In M. Chacravarty, Y. Guo, and T. Ida, editors, *Proceed-*

ings of the JICSLP'96 Post-Conference Workshop "Multi-Paradigm Logic Programming", pages 129–136, Bonn, 5-6 Sept. 1996.

7. M. A. Orgun. *Intensional logic programming.* PhD thesis, Dept. of Computer Science, University of Victoria, Canada, December 1991.
8. P. Rondogiannis, M. Gergatsoulis, and T. Panayiotopoulos. *Cactus*: A branching-time logic programming language. In D. Gabbay, R. Kruse, A. Nonnengart, and H. J. Ohlbach, editors, *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, pages 511–524. Springer, June 1997.
9. M. Gergatsoulis, P. Rondogiannis, and T. Panayiotopoulos. Proof procedures for branching-time logic programs. In W. W. Wadge, editor, *Proc. of the Tenth International Symposium on Languages for Intensional Programming (ISLIP'97), May 15-17, Victoria BC, Canada*, pages 12–26, 1997.
10. P. Rondogiannis, M. Gergatsoulis, and T. Panayiotopoulos. Branching-time logic programming: The language Cactus and its applications. *Computer Languages*, 24(3):155–178, October 1998.
11. P. Rondogiannis and M. Gergatsoulis. The intensional implementation technique for chain datalog programs. In *Proc. of the 11th International Symposium on Languages for Intensional Programming (ISLIP'98), May 7-9, Palo Alto, California, USA*, pages 55–64, 1998.
12. P. Rondogiannis and W. W. Wadge. First-order functional languages and intensional logic. *Journal of Functional Programming*, 7(1):73–101, 1997.
13. M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. *Informatica*, pages 207–226, 1983.
14. E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier Science Publishers, B. V., 1990.