# The Intensional Implementation Technique for Chain Datalog Programs*

**P. Rondogiannis**[1] and **M. Gergatsoulis**[2]

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

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

**Abstract**

The notion of *branching time* has been shown to be a promising means of implementing first and higher-order functional languages. More specifically, functional programs are transformed into zero-order branching-time programs which can then be executed in a tagged demand-driven way. Although this approach has been widely used in Lucid implementations, it has not been shown to apply to logic programming languages as well. In this paper we propose a transformation algorithm from a subclass of logic programs to branching time logic programs, making in this way the first step towards an intensional implementation technique for logic programming languages.

## 1 Introduction

The main technique that has been used in implementing functions in intensional languages such as Lucid and GLU, is based on the notion of *branching time*. More specifically, the functional program is transformed into a zero-order branching time program [Yag84, Wad91, Ron94, RW97], which can then be easily executed using tagged, demand-driven evaluation (also called *eduction* [FW87, DW90]). Of course, the technique need not be restricted to intensional functional languages. It can also be applied on more mainstream functional languages, giving a promising alternative to the reduction-based implementations [Jon87].

It is therefore natural to ask whether a similar intensional-logic based implementation technique exists for logic programming languages. The question was first examined by Rolston and Faustini (see for example [RF93]), who considered the transformation of Prolog programs into intensional ones that could be executed in a dataflow way. However, their target language is not a branching time one. Our work aims at exactly this point: to examine whether logic programs can be transformed into simpler in structure branching-time logic programming ones. In this paper we present work in progress towards this goal. More specifically, we define a transformation algorithm from a class of logic programs (the *chain Datalog* ones) to the class of unary branching-time logic programs. In this way we set the basis for a new dataflow approach for implementing logic programming languages (and of course temporal logic programming languages such as Chronolog [Wad88, Org91, OW92]).

*This work has been partially supported by the Greek General Secretariat of Research and Technology under the project "Logic Programming Systems and Environments for Developing Logic Programs" of ΠΕΝΕΔ'95, contract no 952.

## 2 Preliminaries

In the languages we adopt, we assume the existence of *constants* (denoted by **a, b, c**), *variables* (denoted by **X, Y, Z**) and predicates (**p, q, r**). A *term* is either a variable or a constant. An *atom* is a formula of the form $\mathbf{p}(\mathbf{e}_0,\ldots,\mathbf{e}_{n-1})$ where $\mathbf{e}_0,\ldots,\mathbf{e}_{n-1}$ are terms.

In the following, we assume familiarity with the basic notions of logic programming. Datalog is the subset of logic programming that does not use function symbols. We are particularly interested in the class of *chain Datalog* programs, which is defined below:

**Definition 2.1.** [DG95] A *chain rule* is a clause of the form

$$\mathbf{p}(\mathbf{X},\mathbf{Z}) \leftarrow \mathbf{q}_1(\mathbf{X},\mathbf{Y}_1), \mathbf{q}_2(\mathbf{Y}_1,\mathbf{Y}_2), \ldots, \mathbf{q}_{k+1}(\mathbf{Y}_k,\mathbf{Z}).$$

where $k \geq 0$, and **X, Z** and each $\mathbf{Y}_i$ are distinct variables. Here $\mathbf{q}(\mathbf{X},\mathbf{Z})$ is the *head* and $\mathbf{q}_1(\mathbf{X},\mathbf{Y}_1), \mathbf{q}_2(\mathbf{Y}_1,\mathbf{Y}_2), \ldots, \mathbf{p}_{k+1}(\mathbf{Y}_k,\mathbf{Z})$ is the *body*. The body becomes $\mathbf{q}_1(\mathbf{X},\mathbf{Z})$, when $k = 0$. A *chain Datalog program* is a finite set of rules. Programs are denoted by **P**. A goal is of the form $\leftarrow \mathbf{q}(\mathbf{a},\mathbf{X})$, where **a** is a constant, **X** is a variable and **q** is a predicate.

**Definition 2.2.** A *simple chain Datalog program* is one in which all rules have at most two atoms in their body.

Notice that each chain rule contains no constants and has at least one atom in its body. Moreover, notice that we assume that the first argument of a goal atom is always ground. The necessity of this assumption will become clear in later sections.

The first argument of a predicate will often be called its *input* argument, while the second one its *output* argument. It is customary to distinguish two classes of predicates in a given (chain) Datalog program:

- The IDB predicates, which are the ones that appear in rule heads and possibly in rule bodies.
- The EDB predicates, which can appear in rule bodies only.

The semantics of (chain) Datalog programs can be defined in accordance to the semantics of classical logic programming. The notions of *minimum model* and *immediate consequence operator* $T_{\mathbf{P}}$, transfer directly.

## 3 Branching Datalog

Braching Datalog programs are Cactus programs [RGP97] without function symbols. The syntax of Branching Datalog programs is an extension of the syntax of Datalog programs. More specifically, the temporal operators **first** and $\mathbf{call}_i$, $i \in \mathcal{N}$, are added to the syntax of Datalog. The declarative reading of these temporal operators will be discussed shortly.

A *temporal reference* is a sequence (possibly empty) of the above temporal operators. A *canonical temporal reference* is a temporal reference of the form **first** $\mathbf{call}_{i_1} \cdots \mathbf{call}_{i_n}$, where $i_1, \ldots, i_n \in \mathcal{N}$ and $n \geq 0$. An *open temporal reference* is a temporal reference of the form $\mathbf{call}_{i_1} \cdots \mathbf{call}_{i_n}$, where $i_1, \ldots, i_n \in \mathcal{N}$ and $n \geq 0$.

A *temporal atom* is an atom preceeded by either a canonical or an open temporal reference. A *temporal clause* is a formula of the form:

$$\mathbf{H} \leftarrow \mathbf{B}_1, \ldots, \mathbf{B}_m.$$

where $\mathbf{H}, \mathbf{B}_1, \ldots, \mathbf{B}_m$ are temporal atoms and $m \geq 0$. If $m = 0$, the clause is said to be a *unit temporal clause*. A *Branching Datalog program* is a finite set of *temporal clauses*. A *goal clause* in Branching Datalog is a formula of the form $\leftarrow \mathbf{A}_1, \ldots, \mathbf{A}_n$ where $\mathbf{A}_i$, $i = 1, \ldots, n$ are temporal atoms.

Branching Datalog is based on a relatively simple *branching time logic* (BTL). 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 (**call** 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 $t$. BTL uses the temporal operators **first** and $\mathbf{call}_i$, $i \in \mathcal{N}$. The operator **first** is used to express the first moment in time, while $\mathbf{call}_i$ 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 $\mathbf{A}$ is a formula then so are **first** $\mathbf{A}$ and $\mathbf{call}_i$ $\mathbf{A}$.

The semantics of temporal formulas of $BTL$ are given using the notion of *branching temporal interpretation* [RGP97]. 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 $BTL$ comprises a non-empty set $D$, called the domain of the interpretation, together with an element of $D$ for each variable; for each constant, an element of $D$; 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. $\models_{I,t} \mathbf{A}$ denotes that a formula $\mathbf{A}$ is true at a moment $t$ in some temporal interpretation $I$.

**Definition 3.2**. The semantics of the elements of the temporal logic $BTL$ are given inductively as follows:

1. For any $n$-ary predicate symbol $\mathbf{p}$ and terms $\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}$,
   $\models_{I,t} \mathbf{p}(\mathbf{e}_0, \ldots, \mathbf{e}_{n-1})$ *iff* $\langle I(\mathbf{e}_0), \ldots, I(\mathbf{e}_{n-1})\rangle \in I(\mathbf{p})(t)$
2. $\models_{I,t} \neg\mathbf{A}$ *iff it is not the case that* $\models_{I,t} \mathbf{A}$
3. $\models_{I,t} \mathbf{A} \wedge \mathbf{B}$ *iff* $\models_{I,t} \mathbf{A}$ *and* $\models_{I,t} \mathbf{B}$
4. $\models_{I,t} \mathbf{A} \vee \mathbf{B}$ *iff* $\models_{I,t} \mathbf{A}$ *or* $\models_{I,t} \mathbf{B}$
5. $\models_{I,t} (\forall\mathbf{x})\mathbf{A}$ *iff* $\models_{I[d/\mathbf{x}],t} \mathbf{A}$ *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** $\mathbf{A}$ *iff* $\models_{I,[\,]} \mathbf{A}$
7. $\models_{I,t} \mathbf{call}_i\ \mathbf{A}$ *iff* $\models_{I,[i|t]} \mathbf{A}$

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

## 3.1 Semantics of Branching Datalog

The semantics of Branching Datalog are defined in terms of *temporal Herbrand interpretations*. A notion that is crucial in the discussion that follows, is that of *canonical instance of a clause*, which is formalized below.

**Definition 3.3**. A *canonical temporal atom* is a temporal atom whose temporal reference is canonical. An *open temporal atom* is a temporal atom whose temporal reference is open. A *canonical temporal clause* is a temporal clause whose temporal atoms are canonical.

It can be shown that every Branching Datalog program can be transformed into a (possibly infinite) set of canonical temporal clauses, which has the same set of temporal models as the initial program (see Lemma 3.1 below). Therefore, the transformation prererves the set of canonical atoms that are logical consequences of the program. The construction of this set of canonical temporal clauses is formalized by the following definitions:

**Definition 3.4**. A *canonical temporal instance* of a temporal clause $\mathbf{C}$ is a canonical temporal clause $\mathbf{C}'$ which can be obtained by applying the same canonical temporal reference to all open atoms of $\mathbf{C}$.

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 3.1** *Let* $\mathbf{C}$ *be a clause and* $I$ *a temporal interpretation of* $BTL$. $\models_I \mathbf{C}$ *if and only if* $\models_I \mathbf{C}_t$ *for all canonical instances* $\mathbf{C}_t$ *of* $\mathbf{C}$.

As in Datalog, the set $U_{\mathbf{P}}$ generated by constant symbols that appear in $\mathbf{P}$, called *Herbrand universe*, is used to define *temporal Herbrand interpretations*. Temporal Herbrand interpretations can be regarded as subsets of the *temporal Herbrand Base* $TB_{\mathbf{P}}$ of $\mathbf{P}$, consisting of all *ground canonical temporal atoms* whose predicate symbols appear in $\mathbf{P}$ and whose arguments are terms in the Herbrand universe $U_{\mathbf{P}}$ of $\mathbf{P}$. 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 $M(\mathbf{P})$, 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 $\mathbf{P}$.

**Theorem 3.1** *Let* $\mathbf{P}$ *be a Branching Datalog program. Then*

$$M(\mathbf{P}) = \{\mathbf{A} \in TB_{\mathbf{P}} \mid \mathbf{P} \models \mathbf{A}\}.$$

A fixpoint characterization of the semantics of Branching Datalog programs is provided using a closure operator that maps temporal Herbrand interpretations to temporal Herbrand interpretations:

**Definition 3.5**. Let $\mathbf{P}$ be a Branching Datalog program and $TB_{\mathbf{P}}$ the temporal Herbrand base of $\mathbf{P}$. The operator $T_{\mathbf{P}} : 2^{TB_{\mathbf{P}}} \to 2^{TB_{\mathbf{P}}}$ is defined as follows:

$T_{\mathbf{P}}(I) = \{\mathbf{A} \mid \mathbf{A} \leftarrow \mathbf{B}_1, \ldots, \mathbf{B}_n$ is a canonical ground instance of a program clause in $\mathbf{P}$ and $\{\mathbf{B}_1, \ldots, \mathbf{B}_n\} \subseteq I\ \}$

It can be proved that $TB_{\mathbf{P}}$ is a complete lattice under the partial order of set inclusion ($\subseteq$). Moreover, $T_{\mathbf{P}}$ is continuous and hence monotonic over the complete lattice $(TB_{\mathbf{P}}, \subseteq)$, and therefore $T_{\mathbf{P}}$ has a least fixpoint. The least fixpoint of $T_{\mathbf{P}}$ provides a characterization of the minimal Herbrand model of a Branching Datalog program, as it is shown in the following theorem.

**Theorem 3.2** *Let* $\mathbf{P}$ *be a Branching Datalog program. Then*

$$M(\mathbf{P}) = lfp(T_{\mathbf{P}}) = T_{\mathbf{P}} \uparrow \omega.$$

In the following sections we will also use the notation $M(\mathbf{P}, \mathbf{p})$ to denote the set of atoms in $M(\mathbf{P})$ whose predicate symbol is $\mathbf{p}$.

# 4 The Transformation Algorithm

In the following we present an algorithm which transforms every simple chain datalog program $\mathbf{P}$ into an equivalent intensional program $\mathbf{P}^*$ which has the following properties:

1. All predicates in $\mathbf{P}^*$ are unary.

2. There is at most one atom in the body of each clause in $\mathbf{P}^*$.

The transformation algorithm is as follows:

For each predicate $\mathbf{p}$ we introduce two unary predicates $\mathbf{p}_0$ and $\mathbf{p}_1$, where $\mathbf{p}_0$ corresponds to the first argument of $\mathbf{p}$ while $\mathbf{p}_1$ corresponds to the second argument of $\mathbf{p}$. At various points in the transformation we use intensional operators of the form $\mathbf{call}_i$. Each new such operator that we introduce is assumed to have a different index $i$ than all previous operators used.

1. Each unit clause (fact) in $\mathbf{P}$ of the form:

$$\mathbf{p}(\mathbf{e}_0, \mathbf{e}_1).$$

   is transformed into a clause in $\mathbf{P}^*$ of the form:

$$\mathbf{p}_1(\mathbf{e}_1) \leftarrow \mathbf{p}_0(\mathbf{e}_0).$$

2. Each clause in $\mathbf{P}$ of the form:

$$\mathbf{p}(\mathbf{X}, \mathbf{Y}) \leftarrow \mathbf{q}(\mathbf{X}, \mathbf{Y}).$$

   is transformed into two clauses in $\mathbf{P}^*$ of the form:

$$\begin{array}{l} \mathbf{p}_1(\mathbf{Y}) \leftarrow \mathbf{call}_i\ \mathbf{q}_1(\mathbf{Y}) \\ \mathbf{call}_i\ \mathbf{q}_0(\mathbf{X}) \leftarrow \mathbf{p}_0(\mathbf{X}). \end{array}$$

3. Each non unit clause in $\mathbf{P}$ of the form:

$$\mathbf{p}(\mathbf{X}, \mathbf{Y}) \leftarrow \mathbf{q}(\mathbf{X}, \mathbf{Z}), \mathbf{r}(\mathbf{Z}, \mathbf{Y}).$$

   is transformed into the set of clauses:

$$\begin{array}{l} \mathbf{p}_1(\mathbf{Y}) \leftarrow \mathbf{call}_i\ \mathbf{r}_1(\mathbf{Y}). \\ \mathbf{call}_i\ \mathbf{r}_0(\mathbf{Z}) \leftarrow \mathbf{call}_j\ \mathbf{q}_1(\mathbf{Z}). \\ \mathbf{call}_j\ \mathbf{q}_0(\mathbf{X}) \leftarrow \mathbf{p}_0(\mathbf{X}). \end{array}$$

4. The goal clause:

$$\leftarrow \mathbf{p}(\mathbf{a}, \mathbf{Y})$$

   is transformed into the clauses:

$$\begin{array}{l} \leftarrow \mathbf{first}\ \mathbf{p}_1(\mathbf{Y}). \\ \mathbf{first}\ \mathbf{p}_0(\mathbf{a}). \end{array}$$

**Example 4.1.** Let $P = \{I_1, I_2, I_3\} \cup \{E_1, E_2, E_3\}$ be a chain Datalog program where:

$$\begin{array}{lll} (I_1) & & \leftarrow\ \mathtt{p}(\mathtt{a}, \mathtt{Y}). \\ (I_2) & \mathtt{p}(\mathtt{X}, \mathtt{Z}) & \leftarrow\ \mathtt{e}(\mathtt{X}, \mathtt{Z}). \\ (I_3) & \mathtt{p}(\mathtt{X}, \mathtt{Z}) & \leftarrow\ \mathtt{p}(\mathtt{X}, \mathtt{Y}), \mathtt{e}(\mathtt{Y}, \mathtt{Z}). \\ (E_1) & \mathtt{e}(\mathtt{a}, \mathtt{b}). & \\ (E_2) & \mathtt{e}(\mathtt{b}, \mathtt{c}). & \\ (E_3) & \mathtt{e}(\mathtt{c}, \mathtt{d}). & \end{array}$$

The corresponding intensional program $P^I$ obtained by applying the transformation algorithm as follows:

Transforming clause $I_1$ we get:

$$\begin{array}{l} \leftarrow\ \mathtt{first}\ \mathtt{p}_1(\mathtt{Y}). \\ \mathtt{first}\ \mathtt{p}_0(\mathtt{a}). \end{array}$$

Transforming $I_2$ we get:

```
p1(Z)  ←  call1 e1(Z).
call1 e0(X)  ←  p0(X).
```

Transforming $I_3$ we get:

```
p1(Z)  ←  call3 e1(Z).
call3 e0(Y)  ←  call2 p1(Y).
call2 p0(X)  ←  p0(X).
```

Finally, transforming the clauses $E_1 - E_3$ (corresponding to the EDB atoms) we get:

```
e1(b)  ←  e0(a).
e1(c)  ←  e0(b).
e1(d)  ←  e0(c).
```

# 5 Correctness Proof

Chain Datalog programs can be shown to be equivalent to simple chain Datalog programs, as the following proposition demonstrates.

**Proposition 5.1** *Every chain Datalog program* $\mathbf{P}$ *can be transformed into a simple Datalog program* $\mathbf{P}_s$ *such that for every predicate symbol* $\mathbf{p}$*, it holds* $M(\mathbf{P},\mathbf{p}) = M(\mathbf{P_s},\mathbf{p})$.

**Proof:** Consider a *chain rule* in $\mathbf{P}$ of the form

$$\mathbf{p}(\mathbf{X},\mathbf{Z}) \leftarrow \mathbf{q}_1(\mathbf{X},\mathbf{Y}_1), \mathbf{q}_2(\mathbf{Y}_1,\mathbf{Y}_2), \ldots, \mathbf{q}_{k+1}(\mathbf{Y}_k,\mathbf{Z}). \tag{1}$$

where $k \geq 2$. The rule (1) can be replaced by the two following rules (in which $\mathbf{r}$ is a new predicate name that we introduce):

$$\mathbf{p}(\mathbf{X},\mathbf{Z}) \leftarrow \mathbf{q}_1(\mathbf{X},\mathbf{Y}_1), \mathbf{r}(\mathbf{Y}_1,\mathbf{Z}). \tag{2}$$

$$\mathbf{r}(\mathbf{X},\mathbf{Z}) \leftarrow \mathbf{q}_2(\mathbf{Y}_1,\mathbf{Y}_2), \ldots, \mathbf{q}_{k+1}(\mathbf{Y}_k,\mathbf{Z}). \tag{3}$$

Now, clause (2) has two atoms in its body, while clause (3) has $k$ (one less than clause (1) initially had). We can apply the same process on clause (3), and continuing in this way we end-up with a simple chain Datalog program $\mathbf{P}_s$.

It is easy to see that $M(\mathbf{P},\mathbf{p}) = M(\mathbf{P_s},\mathbf{p})$ as the new clauses we introduce can be considered as *Eureka* definitions while the replacement of $\mathbf{q}_2(\mathbf{Y}_1,\mathbf{Y}_2), \ldots, \mathbf{q}_{k+1}(\mathbf{Y}_k,\mathbf{Z})$ by $\mathbf{r}(\mathbf{Y}_1,\mathbf{Z})$ is a folding step [TS84, Ger94, GK94]. Now the derired result is an immediate consequence of the correctness of the fold/unfold transformation system. ■

Let $\mathbf{P}$ be a program and $\mathbf{P}^*$ the translated intensional one. We show the following lemma:

**Lemma 5.1** *Let* $\mathbf{p}$ *be a predicate defined in* $\mathbf{P}$ *and let* $R$ *be a canonical temporal reference. If* $R\ \mathbf{p}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow \omega$ *and* $\mathbf{p}(\mathbf{a},\mathbf{b}) \in T_{\mathbf{P}} \uparrow \omega$ *then* $R\ \mathbf{p}_1(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow \omega$.

**Proof:** We show the above by induction on the approximations of $T_{\mathbf{P}} \uparrow \omega$.

<u>Induction Basis:</u>
To establish the induction basis, we need to show that if $R\ \mathbf{p}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow \omega$ and $\mathbf{p}(\mathbf{a},\mathbf{b}) \in T_{\mathbf{P}} \uparrow 0$ then $R\ \mathbf{p}_1(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow \omega$.

But $\mathbf{p}(\mathbf{a},\mathbf{b}) \in T_{\mathbf{P}} \uparrow 0$ means that in $\mathbf{P}$ there exists a fact $\mathbf{p}(\mathbf{a},\mathbf{b})$ (or a fact $\mathbf{p}(\mathbf{a},\mathbf{Y})$, or a fact $\mathbf{p}(\mathbf{X},\mathbf{b})$, or $\mathbf{p}(\mathbf{X},\mathbf{Y})$). We consider the case $\mathbf{p}(\mathbf{a},\mathbf{b})$ (the other cases can be examined in a similar way). According to the transformation algorithm, in $\mathbf{P}^*$ there exists the rule $\mathbf{p}_1(\mathbf{b}) \leftarrow \mathbf{p}_0(\mathbf{a})$. Using this and the fact that $R\,\mathbf{p}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow \omega$ we conclude that $R\,\mathbf{p}_1(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow \omega$.

*Induction Hypothesis:*
We assume that if $R\,\mathbf{p}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow \omega$ and $\mathbf{p}(\mathbf{a}, \mathbf{b}) \in T_{\mathbf{P}} \uparrow k$ then $R\,\mathbf{p}_1(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow \omega$. Notice that the induction hypothesis holds for any $\mathbf{p}$ in $\mathbf{P}$ and any temporal reference $R$.

*Induction Step:*
We show that if $R\,\mathbf{p}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow \omega$ and $\mathbf{p}(\mathbf{a}, \mathbf{b}) \in T_{\mathbf{P}} \uparrow (k+1)$ then $R\,\mathbf{p}_1(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow \omega$.
*Case 1:* Assume that **p(a,b)** has been added in $T_{\mathbf{P}} \uparrow (k+1)$ using a rule of the form:

$$\mathbf{p}(\mathbf{X}, \mathbf{Y}) \leftarrow \mathbf{q}(\mathbf{X}, \mathbf{Z}), \mathbf{r}(\mathbf{Z}, \mathbf{Y}) \quad (0)$$

But then, there exists a constant $\mathbf{c}$ such that $\mathbf{q}(\mathbf{a}, \mathbf{c}) \in T_{\mathbf{P}} \uparrow k$ and $\mathbf{r}(\mathbf{c}, \mathbf{b}) \in T_{\mathbf{P}} \uparrow k$.

Consider now the transformation of the above clause (0) in program $\mathbf{P}^*$. The new clauses obtained are:

$$\begin{array}{ll} \mathbf{p}_1(\mathbf{Y}) \leftarrow \mathbf{call}_i\ \mathbf{r}_1(\mathbf{Y}). & (1) \\ \mathbf{call}_i\ \mathbf{r}_0(\mathbf{Z}) \leftarrow \mathbf{call}_j\ \mathbf{q}_1(\mathbf{Z}). & (2) \\ \mathbf{call}_j\ \mathbf{q}_0(\mathbf{X}) \leftarrow \mathbf{p}_0(\mathbf{X}). & (3) \end{array}$$

Using the assumption that $R\,\mathbf{p}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow \omega$ together with clause (3) above, we get that $R\,\mathbf{call}_j\ \mathbf{q}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow \omega$. Given this, we can now apply the induction hypothesis on $\mathbf{q}$ which gives:

Since $R\,\mathbf{call}_j\ \mathbf{q}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow \omega$ and $\mathbf{q}(\mathbf{a}, \mathbf{c}) \in T_{\mathbf{P}} \uparrow k$ then $R\,\mathbf{call}_j\ \mathbf{q}_1(\mathbf{c}) \in T_{\mathbf{P}^*} \uparrow \omega$.

Using now the fact that $R\,\mathbf{call}_j\ \mathbf{q}_1(\mathbf{c}) \in T_{\mathbf{P}^*} \uparrow \omega$ together with clause (2) we get $R\,\mathbf{call}_i\ \mathbf{r}_0(\mathbf{c}) \in T_{\mathbf{P}^*} \uparrow \omega$. Given this, we can now apply the induction hypothesis on $\mathbf{r}$ which gives:

Since $R\,\mathbf{call}_i\ \mathbf{r}_0(\mathbf{c}) \in T_{\mathbf{P}^*} \uparrow \omega$ and $\mathbf{r}(\mathbf{c}, \mathbf{b}) \in T_{\mathbf{P}} \uparrow k$ then $R\,\mathbf{call}_i\ \mathbf{r}_1(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow \omega$.

Using now the fact that $R\,\mathbf{call}_i\ \mathbf{r}_1(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow \omega$ together with clause (1), we get the desired result which is that $R\,\mathbf{p}_1(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow \omega$.
*Case 2:* Assume that **p(a,b)** has been added in $T_{\mathbf{P}} \uparrow (k+1)$ using a rule of the form:

$$\mathbf{p}(\mathbf{X}, \mathbf{Y}) \leftarrow \mathbf{q}(\mathbf{X}, \mathbf{Y}) \quad (0)$$

This implies that $\mathbf{q}(\mathbf{a}, \mathbf{b}) \in T_{\mathbf{P}} \uparrow k$. Consider now the transformation of the above clause (0) in program $\mathbf{P}^*$. The new clauses obtained are:

$$\begin{array}{ll} \mathbf{p}_1(\mathbf{Y}) \leftarrow \mathbf{call}_i\ \mathbf{q}_1(\mathbf{Y}). & (1) \\ \mathbf{call}_i\ \mathbf{q}_0(\mathbf{X}) \leftarrow \mathbf{p}_0(\mathbf{X}). & (2). \end{array}$$

Using the assumption that $R\,\mathbf{p}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow \omega$ together with clause (2) above, we get that $R\,\mathbf{call}_i\ \mathbf{q}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow \omega$. Given this, we can now apply the induction hypothesis on $\mathbf{q}$ which gives:

Since $R\,\mathbf{call}_i\ \mathbf{q}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow \omega$ and $\mathbf{q}(\mathbf{a}, \mathbf{b}) \in T_{\mathbf{P}} \uparrow k$ then $R\,\mathbf{call}_i\ \mathbf{q}_1(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow \omega$.

But this together with clause (1) above gives $R\,\mathbf{p}_1(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow \omega$, which is the desired result. ∎

**Lemma 5.2** *Let* $\mathbf{P}$ *be a simple chain Datalog program and* $?\mathbf{p}(\mathbf{a}, \mathbf{X})$ *be a goal clause. Let* $\mathbf{P}^*$ *be the intensional program obtained by applying the tranformation algorithm to* $\mathbf{P} \cup \{\leftarrow \mathbf{p}(\mathbf{a}, \mathbf{X})\}$. *If* $\mathbf{p}(\mathbf{a}, \mathbf{b}) \in T_{\mathbf{P}} \uparrow \omega$ *then* $\mathbf{first}\ \mathbf{p}_1(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow \omega$.

**Proof:** Since by transforming the goal clause, the fact $\mathbf{first}\ \mathbf{p}_0(\mathbf{a})$ is added to $\mathbf{P}^*$, this lemma is a special case of lemma 5.1. ∎

We now show the following lemma which is the "inverse" of lemma 5.1:

**Lemma 5.3** *Let* $\mathbf{p}$ *be a predicate defined in* $\mathbf{P}$ *and let* $R$ *be a canonical temporal reference. If* $R\,\mathbf{p}_1(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow \omega$ *then there exists a constant* $\mathbf{a}$ *such that* $\mathbf{p}(\mathbf{a},\mathbf{b}) \in T_{\mathbf{P}} \uparrow \omega$ *and* $R\,\mathbf{p}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow \omega$.

**Proof:** We show the above by induction on the approximations of $T_{\mathbf{P}^*} \uparrow \omega$.

*Induction Basis:*
To establish the induction basis, we need to show that If $R\,\mathbf{p}_1(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow 0$ then there exists a constant $\mathbf{a}$ such that $\mathbf{p}(\mathbf{a},\mathbf{b}) \in T_{\mathbf{P}} \uparrow \omega$ and $R\,\mathbf{p}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow 0$.

But $R\,\mathbf{p}_1(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow 0$ is false because in $T_{\mathbf{P}^*} \uparrow 0$ there belong only temporal atoms regarding input predicates. Therefore, the basis case holds vacously.

*Induction Hypothesis:*
If $R\,\mathbf{p}_1(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow k$ then there exists a constant $\mathbf{a}$ such that $\mathbf{p}(\mathbf{a},\mathbf{b}) \in T_{\mathbf{P}} \uparrow \omega$ and $R\,\mathbf{p}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow k$.

*Induction Step:*
We show that if $R\,\mathbf{p}_1(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow (k+1)$ then there exists $\mathbf{a}$ such that $\mathbf{p}(\mathbf{a},\mathbf{b}) \in T_{\mathbf{P}} \uparrow \omega$ and $R\,\mathbf{p}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow (k+1)$.
*Case 1:* Assume now that there exists in $\mathbf{P}$ a rule of the form:

$$\mathbf{p}(\mathbf{X},\mathbf{Y}) \leftarrow \mathbf{q}(\mathbf{X},\mathbf{Z}), \mathbf{r}(\mathbf{Z},\mathbf{Y}). \quad (0)$$

Consider now the transformation of the above clause (0) in program $\mathbf{P}^*$. The new clauses obtained are:

$$\begin{array}{ll} \mathbf{p}_1(\mathbf{Y}) \leftarrow \mathbf{call}_i\ \mathbf{r}_1(\mathbf{Y}). & (1) \\ \mathbf{call}_i\ \mathbf{r}_0(\mathbf{Z}) \leftarrow \mathbf{call}_j\ \mathbf{q}_1(\mathbf{Z}). & (2) \\ \mathbf{call}_j\ \mathbf{q}_0(\mathbf{X}) \leftarrow \mathbf{p}_0(\mathbf{X}). & (3) \end{array}$$

Assume also that $R\,\mathbf{p}_1(\mathbf{b})$ has been introduced in $T_{\mathbf{P}^*} \uparrow (k+1)$ by clause (1) above. Then, this means that $R\,\mathbf{call}_i\ \mathbf{r}_1(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow k$. By the induction hypothesis, we get that there exists a constant $\mathbf{c}$ such that $\mathbf{r}(\mathbf{c},\mathbf{b}) \in T_{\mathbf{P}} \uparrow \omega$ and $R\,\mathbf{call}_i\ \mathbf{r}_0(\mathbf{c}) \in T_{\mathbf{P}^*} \uparrow k$.

Notice now that the only way that $R\,\mathbf{call}_i\ \mathbf{r}_0(\mathbf{c}) \in T_{\mathbf{P}^*} \uparrow k$ can have been obtained is by using clause (3) above (all other clauses defining predicate $\mathbf{r}_0$, have a different index in the **call** operator. Therefore, using clause (2) above, we then get that $R\,\mathbf{call}_j\ \mathbf{q}_1(\mathbf{c}) \in T_{\mathbf{P}^*} \uparrow (k-1)$ which means that $R\,\mathbf{call}_j\ \mathbf{q}_1(\mathbf{c}) \in T_{\mathbf{P}^*} \uparrow k$. Using the induction hypothesis, we get that there exists a constant $\mathbf{a}$ such that $\mathbf{q}(\mathbf{a},\mathbf{c}) \in T_{\mathbf{P}} \uparrow \omega$ and $R\,\mathbf{call}_j\ \mathbf{q}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow k$. But then, using clause (3) above as before we get $R\,\mathbf{p}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow (k-1)$, which implies that $R\,\mathbf{p}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow k$. Moreover, since $\mathbf{q}(\mathbf{a},\mathbf{c}) \in T_{\mathbf{P}} \uparrow \omega$ and $\mathbf{r}(\mathbf{c},\mathbf{b}) \in T_{\mathbf{P}} \uparrow \omega$ from (0) we also get $\mathbf{p}(\mathbf{a},\mathbf{b}) \in T_{\mathbf{P}} \uparrow \omega$. Using these results we derive the desired lemma.
*Case 2:* Assume that in $\mathbf{P}$ there exists a rule of the form:

$$\mathbf{p}(\mathbf{X},\mathbf{Y}) \leftarrow \mathbf{q}(\mathbf{X},\mathbf{Y}) \quad (0)$$

Consider now the transformation of the above clause (0) in program $\mathbf{P}^*$. The new clauses obtained are:

$$\begin{array}{ll} \mathbf{p}_1(\mathbf{Y}) \leftarrow \mathbf{call}_i\ \mathbf{q}_1(\mathbf{Y}). & (1) \\ \mathbf{call}_i\ \mathbf{q}_0(\mathbf{X}) \leftarrow \mathbf{p}_0(\mathbf{X}). & (2). \end{array}$$

Assume also that $R\,\mathbf{p}_1(\mathbf{b})$ has been introduced in $T_{\mathbf{P}^*} \uparrow (k+1)$ by clause (1) above. Then, this means that $R\,\mathbf{call}_i\ \mathbf{q}_1(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow k$. By the induction hypothesis, we get that there exists a constant $\mathbf{a}$ such that $R\,\mathbf{call}_i\ \mathbf{q}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow k$ and $\mathbf{q}(\mathbf{a},\mathbf{b}) \in T_{\mathbf{P}} \uparrow \omega$. Using clause (0), we get that $\mathbf{p}(\mathbf{a},\mathbf{b}) \in T_{\mathbf{P}} \uparrow \omega$.

Using clause (2) above together with the fact that $R\,\mathbf{call}_i\ \mathbf{q}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow k$, we get $R\,\mathbf{p}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow (k-1)$, which implies that $R\,\mathbf{p}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow k$.
*Case 3:* Assume that in $\mathbf{P}$ there exists a fact of the form:

$$\mathbf{p}(\mathbf{a},\mathbf{b}). \quad (0)$$

Consider now the transformation of the above clause (0) in program $\mathbf{P}^*$. The new clause obtained is:

$$\mathbf{p}_1(\mathbf{b}) \leftarrow \mathbf{p}_0(\mathbf{a}). \quad (1)$$

Assume now that $R\, \mathbf{p}_1(\mathbf{b})$ has been introduced in $T_{\mathbf{P}^*} \uparrow (k+1)$ by clause (1) above. This means that $R\, \mathbf{p}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow k$ and therefore $R\, \mathbf{p}_0(\mathbf{a}) \in T_{\mathbf{P}^*} \uparrow (k+1)$. Moreover, $\mathbf{p}(\mathbf{a}, \mathbf{b}) \in T_{\mathbf{P}} \uparrow \omega$, because $\mathbf{p}(\mathbf{a},\mathbf{b})$ is a fact in $\mathbf{P}$.

This concludes the proof of the particular case and of the lemma. ∎

**Lemma 5.4** *Let* $\mathbf{P}$ *be a simple chain Datalog program and* $?\mathbf{p}(\mathbf{a}, \mathbf{X})$ *be a goal clause. Let* $\mathbf{P}^*$ *be the intensional program obtained by applying the tranformation algorithm to* $\mathbf{P} \cup \{\leftarrow \mathbf{p}(\mathbf{a}, \mathbf{X})\}$. *If* **first** $\mathbf{p_1}(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow \omega$ *then* $\mathbf{p}(\mathbf{a}, \mathbf{b}) \in T_{\mathbf{P}} \uparrow \omega$.

**Proof:** From lemma 5.3 we have that there is a constant $\mathbf{c}$ such that $\mathbf{p}(\mathbf{c}, \mathbf{b}) \in \mathrm{T}_{\mathbf{P}} \uparrow \omega$ and **first** $\mathbf{p}_0(\mathbf{c}) \in \mathrm{T}_{\mathbf{P}^*} \uparrow \omega$. But as the only instance of **first** $\mathbf{p}_0(\mathbf{X})$ in $\mathrm{T}_{\mathbf{P}^*} \uparrow \omega$ is **first** $\mathbf{p}_0(\mathbf{a})$ then $\mathbf{c} = \mathbf{a}$. ∎

**Theorem 5.1** *Let* $\mathbf{P}$ *be a simple chain Datalog program and* $?\mathbf{p}(\mathbf{a}, \mathbf{X})$ *be a goal clause. Let* $\mathbf{P}^*$ *be the intensional program obtained by applying the tranformation algorithm* $\mathbf{P} \cup \{\leftarrow \mathbf{p}(\mathbf{a}, \mathbf{X})\}$. *Then* **first** $\mathbf{p_1}(\mathbf{b}) \in T_{\mathbf{P}^*} \uparrow \omega$ *iff* $\mathbf{p}(\mathbf{a}, \mathbf{b}) \in T_{\mathbf{P}} \uparrow \omega$.

**Proof:** It is an immediate consequence of lemmas 5.2 and 5.4. ∎

# 6 Conclusions

In this paper, we have developed a transformation algorithm from chain Datalog programs to Branching Datalog ones. The programs obtained by this transformation have the following interesting properties:

- All predicates are unary
- Every rule has at most one atom in its body

Apart from its theoretical interest, the transformation algorithm can be viewed as an implementation technique for chain Datalog programs. In fact, the programs that result from the transformation can be easily executed using appropriate SLD-like proof procedures that have been developed for temporal logic programming languages [GRP97, RGP97]). Such an implementation may use techniques borrowed from the dataflow area of research (such as *tagging*, *warehousing*, etc.). It remains to be seen whether such an approach can compete with the usual implementation strategies adopted in the case of Datalog programs.

# References


[DG95] G. Dong and S. Ginsburg. On decompositions of chain datalog programs into $\mathcal{P}$ (left-)linear 1-rule components. *The Journal of Logic Programming*, 23(3):203–236, 1995.

[DW90] W. Du and W. W. Wadge. The Eductive Implementation of a Three-dimensional Spreadsheet. *Software-Practice and Experience*, 20(11):1097–1114, November 1990.

[FW87] A. Faustini and W. Wadge. An Eductive Interpreter for the Language pLucid. In *Proceedings of the SIGPLAN 87 Conference on Interpreters and Interpretive Techniques (SIGPLAN Notices 22(7))*, pages 86–91, 1987.

[Ger94] M. Gergatsoulis. *Logic program transformations: Transformation rules and application strategies.* PhD thesis, Dept. of Computer Science, University of Athens, 1994. (In Greek).

[GK94] M. Gergatsoulis and M. Katzouraki. Unfold/fold transformations for definite clause programs. In M. Hermenegildo and J. Penjam, editors, *Programming Language Implementation and Logic Programming (PLILP'94), Proceedings*, Lecture Notes in Computer Science (LNCS) 844, pages 340–354. Springer-Verlag, 1994.

[GRP97] 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.

[Jon87] S. L. Peyton Jones. *The Implementation of Functional Programming Languages.* Prentice-Hall, 1987.

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

[OW92] M. A. Orgun and W. W. Wadge. Towards a unified theory of intensional logic programming. *The Journal of Logic Programming*, 13(4):413–440, 1992.

[RF93] D. Rolston and T. Faustini. An Overview of an Eductive Evaluation Mechanism for Logic Programs. In *Proceedings of the Sixth International Symposium on Lucid and Intensional Programming*, pages 136–157, 1993.

[RGP97] 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.

[Ron94] P. Rondogiannis. *Higher-order functional languages and intensional logic.* PhD thesis, Dept. of Computer Science, University of Victoria, Canada, December 1994.

[RW97] P. Rondogiannis and W. W. Wadge. First-order functional languages and intensional logic. *Journal of Functional Programming*, 7(1):73–101, 1997.

[TS84] H. Tamaki and T. Sato. Unfold/fold transformations of logic programs. In Sten-Åke Tarnlund, editor, *Proc. of the Second International Conference on Logic Programming*, pages 127–138, 1984.

[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.

[Wad91] W. W. Wadge. Higher-Order Lucid. In *Proceedings of the Fourth International Symposium on Lucid and Intensional Programming*, 1991.

[Yag84] A. Yaghi. *The intensional implementation technique for functional languages.* PhD thesis, Dept. of Computer Science, University of Warwick, Coventry, UK, 1984.