

Prof. Dr. G. Lausen  
Lehrstuhl für Datenbanken  
und Informationssysteme  
Albert-Ludwigs-Universität Fr. i. Br.

Seminar Paper

# Emulating XY-Stratified Datalog with Florid2

Michael Meier  
meierm 'AT' informatik 'DOT' uni-freiburg 'DOT' de

*Prof. Dr. G. Lausen*  
*Advisor*

August 7, 2008



# Contents

<b>Abstract</b>	<b>4</b>
<b>1 Introduction</b>	<b>5</b>
1.1 Motivation . . . . .	5
1.2 Notation and definitions . . . . .	7
<b>2 Emulating Well-Founded Evaluation</b>	<b>9</b>
2.1 Introduction by example . . . . .	9
2.2 A note about the implementation of triggers . . . . .	11
2.3 Construction of a Florid-executable version . . . . .	12
2.4 Further examples . . . . .	15
2.5 Evaluation of relative performance . . . . .	17
<b>3 A more general idea</b>	<b>19</b>
3.1 Some Definitions . . . . .	19
3.2 The general picture: An introduction . . . . .	20
3.3 Last example revisited . . . . .	22
3.4 Rho . . . . .	23
3.5 A termination condition . . . . .	25
3.6 An algorithm . . . . .	26
<b>4 Optimization</b>	<b>28</b>
4.1 Introduction . . . . .	28
4.2 A partial optimization . . . . .	28
4.3 What we must not "optimize" . . . . .	30
4.4 An Example . . . . .	30
<b>5 Conclusions and acknowledgments</b>	<b>33</b>
<b>References</b>	<b>33</b>

## Abstract

*We investigate how the F-Logic system Florid2 can emulate the evaluation procedure for XY-stratified Datalog as introduced in [AOT<sup>+</sup>03]. We restrict our attention to a meaningful subset of such programs, which is still equivalent to WHILE. As an example of adapting the general procedure to a special subclass of XY-stratified Datalog, we consider the emulation of the alternating fixed point procedure for general Datalog programs, which can be expressed by such programs.*

# 1 Introduction

## 1.1 Motivation

There have been many extensions to Datalog to treat negation. The most common extensions can be visualized like below, where no two semantics coincide, i.e. the subset relationships given by the graph are strict.

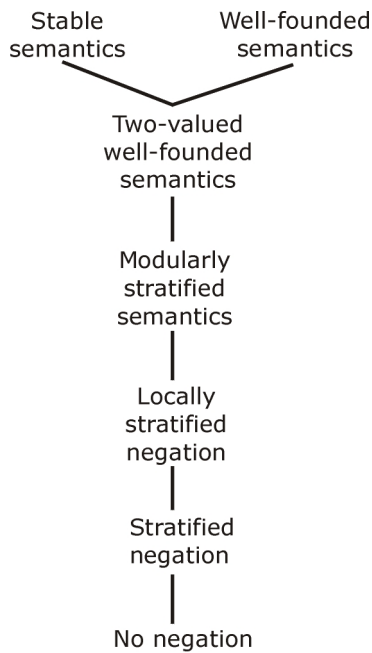


Figure 1: Overview of Datalog semantics for negation, see [Ull].

Locally stratified databases were introduced in [Prz88], extending the notion of stratification by defining a noetherian order relation on the Herbrand base of a Datalog program. Computing a program with respect to that order relation, one can see that local stratification is an intuitive semantics. But how to compute a local stratification of a logic program? The easiest and computationally cheapest way to get one, is to have it explicitly encoded in a program's rules, so that we do not have to look at the fact base of the program. That is what XY-stratified Datalog does! If one considers the distinguished temporal variable in a rule of an XY-stratified program separate from all other variables, one can see this type of programs as locally stratified. Our aim is to show that the F-Logic implementation Florid2 is able to emulate these programs, adding a rewriting step beforehand. Before defining formally what XY-stratification actually means, we give an example.

### Example 1 (Computation of ancestors)

$P$  :  
 $\text{anc}(X, Y) \leftarrow \text{anc}(X, Z), \text{anc}(Z, Y)$

Using seminaive evaluation (see [AHV95]), one could rewrite  $P$  in the following way.

$P'$  :  
 $\text{temp}_{anc}(J, X, Y) \leftarrow \Delta_{anc}(J-1, X, Z), \text{anc}(J-2, Z, Y)$   
 $\text{temp}_{anc}(J, X, Y) \leftarrow \text{anc}(J-1, X, Z), \Delta_{anc}(J-1, Z, Y)$   
 $\Delta_{anc}(J, X, Y) \leftarrow \text{temp}_{anc}(J, X, Y), \text{not } \text{anc}(J-1, X, Y)$   
 $\text{anc}(J, X, Y) \leftarrow \text{anc}(J-1, X, Y)$   
 $\text{anc}(J, X, Y) \leftarrow \Delta_{anc}(J, X, Y)$

The first thing to observe, is that  $P'$  is not a Datalog program any more due to the existence of the function symbol minus. So, what is the intended meaning of  $P'$ ? Here comes local stratification into play. Imagine the value for the variable  $J$  is fixed value. Then, the expressions  $J, J-1, J-2$  can be considered as a succession of states: new information for the state (or stage) with number  $J$  is derived from the previous two stages (and from stage  $J$  itself). So,  $J$  is not considered as a normal variable in a Datalog program, but it takes a special place within the evaluation procedure. Considering the expressions  $J, J-1$  or  $J-2$  as numbers denoting states or stages, we will call them temporal variables. Of course, the values for  $J-1$  and  $J-2$  are uniquely determined by the value for  $J$ .

Looking at the first two rules in  $P'$ , one can say now that the tuples in  $\text{temp}_{anc}$  depend on the previous two temporal stages, which is why we call these first two rules Y-rules. In contrast to that the last rule depends only on the same temporal stage, which is why we call it an X-rule.

$P'$  is not stratified, due to the negative cycle over  $\text{anc}$  and  $\Delta_{anc}$ . Yet, we will break down the computation of  $P'$  to a kind of stratified negation.

The general idea of the computation of  $P'$  is that we add a fixed value for  $J$  to the program and thus compute the tuples for the fixed temporal stage  $J$ . If this done, we increase the value for  $J$  by one and compute the next temporal stage. This procedure is repeated until, in a yet to define sense, the program terminates.

Now, we address the question how to compute a fixed temporal stage. The easiest way to get a feeling for that is to do a little program rewriting. Label all head predicates in  $P'$  with the prefix 'new'. The body predicates with the same temporal argument as the head are also labeled with the prefix 'new'. Predicates with temporal argument  $J-1$  are labeled with the prefix *old*. Predicates with temporal argument  $J-2$  are labeled with

the prefix *old\_old*. Finally, all temporal variables are dropped. The resulting program is called the bistrate version  $P_{bis}$  of  $P$ . It looks as follows:

$P'_{bis}$  :

```

new_tempanc(X,Y) ← old_Δanc(X,Z), old_old_anc(Z,Y)
new_tempanc(X,Y) ← old_anc(X,Z), old_Δanc(Z,Y)
new_Δanc(X,Y) ← new_tempanc(X,Y), not old_anc(X,Y)
new_anc(X,Y) ← old_anc(X,Y)
new_anc(X,Y) ← new_Δanc(X,Y)

```

Note that the computation of the bistrate version of a program is just a different way of saying that we want to fix the value for the temporal variable  $J$ . By adding '*old*' one time as a prefix to some relation symbols, we express that the tuples in that relation belong to the previous temporal stage, i.e. the temporal stage  $J - 1$ . Having computed the bistrate version one can easily see that  $P'_{bis}$  is stratified with the strata:

$$S_0 = \{parent, old\_Δ_{anc}, old\_anc, old\_old\_anc\}$$

$$S_1 = \{new\_temp_{anc}, new\_Δ_{anc}, new\_anc\}$$

For  $P'$ , the fixpoint procedure to compute its stable model becomes quite easy, it reduces to an iterated computation of  $P_{bis}$ , see [AOT<sup>+</sup>03]. For example, `new_tempanc` is derived from `old_Δanc`. After the computation of  $P'_{bis}$  has finished, the relation `old_old_anc` is reinitialized with the contents from `old_anc` before and `old_anc` is reinitialized with the values from `new_tempanc`, which were derived in the step before and so  $P_{bis}$  is computed again and again. So, what we have done so far, is that have reduced the semantics of  $P'$  to a kind of repeated computation of a stratified program, which gives us hope that we can rewrite  $P'$  in such a manner that we can execute it with Florid's inflationary fixpoint operator for F-Logic programs.

XY-stratified programs can be seen as a way to make the state-based computation of a problem explicit. Moreover, they can be considered locally stratified (not only with the help of the temporal argument but also with the number of strata of the bistrate version), though we will not discuss this in detail here.

□

## 1.2 Notation and definitions

Here, we introduce some notation we will use throughout this paper.

### Definition 2 (XY-stratified program)

A Datalog program  $P$  is called XY-stratified if it only consists of X- and Y-rules and its bistrate version (usually called  $P_{bis}$ ) is stratified (see [AOT<sup>+</sup>03]). Throughout this paper,

we use the convention that the temporal variable in the head of a rule is always  $J$  (and not  $J + 1, \dots$ ).

We extend the notion of X- and Y-rules and therefore have to adapt the definition for the bistate version. A rule in  $P$  is called an X-rule if and only if all temporal variables in the rule body are (syntactically) the same as in the rule head, which allows EDB-predicates in an X-rule. A rule is called a Y-rule if and only if

- there exists a temporal variable/expression of the form  $J - k$  in the rule body ( $k \in \mathbb{N} \setminus \{0\}$ ),
- there is a temporal variable/expression in the rule body which is decreased in comparison to the temporal argument/expression in the rule head, then it must be decreased by a constant (and not by division,...),
- all temporal arguments/expressions in the rule body are smaller than or equal to the temporal argument/expression in the head.

Arithmetic comparisons which do not contain the temporal argument in the rule body of an X- or Y-rule are also allowed. The bistate version  $P_{bis}$  is defined as follows (see before). Label all head predicates in  $P$  with the prefix 'new'. The body predicates with the same temporal argument as the head are also labeled with the prefix 'new'. All other predicates are labeled  $k$  times with the prefix 'old' if the temporal variable 'J-k' appears in the position of their temporal argument. Finally, all temporal variables are dropped.  $\square$

This definition does not change the fact that such programs can be considered locally stratified with the same explanation as given in [AOT<sup>+</sup>03].

**Definition 3 (depth(P) and height(P))**

For any program  $P$ ,  $depth(P) \in \mathbb{N}$  is defined as the maximal natural number  $k$  with the property that it appears in a temporal variable " $J - k$ " in  $P$ . If such a number does not exist,  $depth(P)$  is defined as 1, where it is assumed that the string of characters " $J - 0$ " does not appear in the first position of a predicate in  $P$ .

$height(P) \in \mathbb{N}$  is the maximal natural number with the property that it appears in the first position in a fact of the input data of an IDB-predicate in  $P$ . If such a number does not exist,  $height(P)$  is defined as 0.

$\square$

The usefulness of  $depth(P)$  and  $height(P)$  will become clear over time.

## 2 Emulating Well-Founded Evaluation

This section shall serve as an example that, when restricting our attention to subclasses of XY-stratified Datalog, we can achieve much more efficient results, but due to some "good" semantical properties the results do not carry over completely to the general case. A whole class of XY-stratified programs can be gained from rewriting unstratifiable Datalog programs in an XY-stratified manner. We could also have rewritten seminaive evaluation of Datalog programs in such a manner, but alternating fixpoint computation seemed a little more interesting to us.

Basically the results in this section are an XY-stratified paraphrase of the results in [MU99] and [MLL97] with the advantage that computation here is less inefficient due to restricting evaluation to only one temporal stage.

### 2.1 Introduction by example

Consider the following well-known example.

**Example 4 (Win-Move, see [AHV95])**

```
P :
move(1,2) move(2,1) move(2,3) move(3,4) move(4,5) move(5,6)
win(5)
win(X) ← move(X,Y), not win(Y)
```

This can be rewritten to:

```
PXY :
move(1,2) move(2,1) move(2,3) move(3,4) move(4,5) move(5,6)
win(5)
win(J,X) ← move(X,Y), not win(J-1,Y)
```

This is a syntactical paraphrase of Gelfond-Lifschitz-Transformation. All negated subgoals must refer to the previous temporal stage and all positive subgoals to the stage considered at the moment. *J* is not safe in the rule body: it does not appear in a positive literal there. Yet, this does not matter because in our Florid-executable version of this program, we will enforce the safeness of the temporal variable.

If you compute  $P_{XY_{bis}}$ , you will easily see that it is stratified because negation refers, as already said, to a lower temporal stage. Moreover, all rules in  $P_{XY_{bis}}$  can be put in a single stratum. Consider the following Florid-executable version  $P_{XY_{florid}}$  of  $P_{XY}$ :

```
move(1,2). move(2,1). move(2,3). move(3,4). move(4,5). move(5,6).
win(5).
```

```

f1: temporal(1,1):state.

?- sys.strat.doIt.
f2: win(J,X) :- move(X,Y), not win(J-1,Y), 1=max{M[J]; temporal(J,M):state},
           J=max{M; temporal(M,L):state}.
f3: win(J,X) :- win(X), 1=max{M[J]; temporal(J,M):state},
           J=max{M; temporal(M,L):state}.

f4: temporal(J+1,1):state :- temporal(J,2):state, integer(J/2), win(J,X),
           not win(J-2,X), J=max{M; temporal(M,L):state}.
f5: temporal(J+1,1):state :- temporal(J,2):state, not integer(J/2),
           2=max{M[J]; temporal(J,M):state},
           J=max{M; temporal(M,L):state}.
f6: temporal(J,2):state :- 1=max{M[J]; temporal(J,M):state},
           J=max{M; temporal(M,L):state}.

?- sys.strat.doIt.
f7: win_true(X) :- win(J,X), J=max{M; temporal(M,L):state}.
f8: win_undef(X) :- win(J-1,X), not win(J,X), J=max{M; temporal(M,L):state}.
□

```

### Explanation:

Fact f1 initializes a two-dimensional counter. Its first argument represents a temporal value (always initialized with 1) and its second argument represents the number of a step (always initialized with 1). Facts are always separated from the rest of the program by

```

?- sys.strat.doIt.

```

f2 is the rule to be computed. f3 just copies given facts of the IDB-relation win to each temporal stage. Note that the goals

$$1=\max\{M[J]; \text{temporal}(J,M):\text{state}\}, J=\max\{M; \text{temporal}(M,L):\text{state}\}$$

are always added to the rules to be computed. They mean that the rule can only produce tuples for the most current temporal stage and it is only enabled to fire in step one.

When we are in an even temporal stage, rule f4 checks if an alternating fixpoint has been reached. This is done by a usual test for subset relationship of sets (possible due to well-founded evaluation). By f5 a new even temporal stage is always created. F6 represents the changing from step one to step two (second argument in the predicate 'temporal') for the temporal stage considered at the moment. In step one all rules are executed that belong to stratum one in the bistate version of  $P_{XY}$ . Step two serves to handle the control flow of the rewritten program (testing subset relationships). Note that when we are in step one, we immediately go to step two, which means that we assume that the rules for one step can be computed with a single application of Florid's

$T_P$ -operator, which is not the case for all programs. Instead of steps, we will, also in this context, speak about strata. They naturally correspond to the stratum of a rule in the bistrate version of the original program.

F7 to f8 compute the tuples which are true or undefined. Everything in the final stage and the stage before is true, but because we know, that the final stage is even and everything that is in the final stage is also in the stage before, it holds that f7 suffices. If the truth value of a tuple oscillates, it becomes undefined in the final outcome. Due to the properties of alternating fixpoint computation a rule like

`win_undef(X) :- win(J-1,X), not win(J,X), J=max{M; temporal(M,L):state}.`

is not needed. False tuples cannot be explicitly computed because Florid does not work with the Herbrand Base of a program but with safe variables.

A goal like

`1=max{M[J]; temporal(J,M):state}, J=max{M; temporal(M,L):state}`

has been added to control that the actual rules to be computed are only fired when the computation is in stratum one of the most current temporal stage at the moment. From now on, we will need such strings very often, so:

### Definition 5

For readability, we make the definitions:

$\psi(J) := J=\max\{M; \text{temporal}(M,L):\text{state}\}$   
 $\varphi(i,J) := i=\max\{M[J]; \text{temporal}(J,M):\text{state}\}, \psi(J)$   
 $\square$

When writing a Florid-executable version  $P_{florid}$  of  $P$  we often will say that some rules are kept in a separate stratum even though there is no explicit stratification by

`?- sys.strat.doIt.`

We mean by that that the added strings  $\varphi(i,J)$  and  $\varphi(j,J)$  contain different values for  $i$  and  $j$ . Intuitively, the meaning of  $\varphi(i,J)$  is that a rule can only fire for the most current stage and only in step  $i$ . Because we identify a rule from  $P$  with its rewritten rule in  $P_{bis}$  (and vice versa), we use the word stratum also here and drop the word step in this context (as already said above).

## 2.2 A note about the implementation of triggers

In the next sections the trigger semantics in Florid is of great importance. In the F-Logic standard triggers are fired non-deterministically. Yet, in Florid the firing of a trigger is delayed as much as possible. That means that a trigger is only fired when no other

rule is applicable. Thus, the following program indeed computes the complement of the transitive closure.

**Example 6 (Complement of transitive closure)**

```

dom(X) :- E(X,Y).
dom(Y) :- E(X,Y).
T(X,Y) :- E(X,Y).
T(X,Y) :- T(X,Z), T(Z,Y).
CT(X,Y) :- not T(X,Y), temporal:state[fixpoint *- > blubb], dom(X), dom(Y).
state[fixpoint *- > blubb].
temporal:state.
□

```

The underlying problem in the context of XY-stratification is that the computation of one stratum in our program may need more than one application of Florid’s  $T_P$ -operator. We do not go to the following stratum until an inheritance trigger is fired, which is only done when the computation of the stratum considered at the moment is guaranteed to have finished. We will mention that again, when we use it later. This idea was first used in [MU99].

**2.3 Construction of a Florid-executable version**

Here, we will outline an algorithm that computes a Florid-executable program that emulates alternating fixpoint computation for Datalog programs with safe variables. First, we translate our Datalog program to an XY-stratified program, then we rewrite it to a version which is executable by Florid.

**Algorithm 7 ( $P_{XY}$ )**

Let  $P$  denote an arbitrary Datalog program with safe variables. Construct an XY-stratified program  $P_{XY}$  as follows:

- $P_{XY}$  contains all the facts that  $P$  contains,
- if  $P$  contains facts for an IDB-relation  $rel$  with arity  $l$  add the following rule

$$rel(J, X_1, \dots, X_l) \leftarrow rel(X_1, \dots, X_l)$$

where  $X_1, \dots, X_l$  are variables.

- if all  $edb_a$  are EDB-relations (in  $P$ ), all  $idb_b$  are IDB-relations (in  $P$ ),  $arithmetic$  is a conjunction of arithmetic expressions and

$$rel_i(A_1, \dots, A_{n_i}) \leftarrow [not] edb_{i_1}(\dots), \dots, [not] edb_{i_k}(\dots), arithmetic, idb_{j_1}(B_{j_1 1}, \dots, B_{j_1 l_{j_1}}), \dots, idb_{j_m}(B_{j_m 1}, \dots, B_{j_m l_{j_m}}),$$

$$\text{not idb}_{i_1}(B_{i_1 1}, \dots, B_{i_1 l_{i_1}}), \dots, \text{not idb}_{i_m}(B_{i_m 1}, \dots, B_{i_m l_{i_m}})$$

is a rule in  $P$ , then add the following rule to  $P_{XY}$ :

$$\begin{aligned} \text{rel}_i(J, A_1, \dots, A_{n_i}) \leftarrow & \text{[not] edb}_{i_1}(\dots), \dots, \text{[not] edb}_{i_k}(\dots), \text{arithmetic}, \\ & \text{idb}_{j_1}(J, B_{j_1 1}, \dots, B_{j_1 l_{j_1}}), \dots, \text{idb}_{j_m}(J, B_{j_m 1}, \dots, B_{j_m l_{j_m}}) \\ & \text{not idb}_{i_1}(J-1, B_{i_1 1}, \dots, B_{i_1 l_{i_1}}), \dots, \\ & \text{not idb}_{i_m}(J-1, B_{i_m 1}, \dots, B_{i_m l_{i_m}}) \end{aligned}$$

We assume that  $J$  is a newly introduced variable (if this is not so, add a renaming substitution step beforehand). We also assume that predicate names are not overloaded in the original program (see step 2). Obviously,  $\text{depth}(P) = 1$ .

□

$P_{XY}$  is a representation of  $P$  in an XY-stratified manner.  $P_{XY}$  indeed is XY-stratified, because stratification for  $P_{XY_{bis}}$  is trivial. All *old*- and all EDB-relations are put in stratum zero and all *new*-relations (see [AOT<sup>+</sup>03]) can be put in stratum one. This follows directly from Gelfond-Lifschitz-Transformation: negated IDB-subgoals have to refer to the previous temporal stage and therefore the dependency arc between a predicate from a rule head that computes its tuples from a subgoal with the same temporal argument is positive. So there is only one stratum which contains *new*-rules, which is a very good-natured property.

### Algorithm 8 ( $P_{XY_{florid}}$ )

Let  $P$  denote an arbitrary Datalog program with safe variables. Note that three dots in a rule represent always the same variables, but because names of variable do not matter here, we omit them. This convention is only used here and nowhere else. We now construct a program  $P_{XY_{florid}}$  that emulates  $P$  as follows:

- (i) Construct  $P_{XY}$ .
- (ii) Add all EDB-facts from  $P$  to  $P_{XY_{florid}}$ .
- (iii) Add the following lines to the program:
 

```
temporal(1,1):state.
?- sys.strat.doIt.
```
- (iv) If  $\text{head}(J, \dots) \leftarrow \text{body}$  is a rule in  $P_{XY}$ , then add the following rules to  $P_{XY_{florid}}$ :
 

```
head(J, ...) :- body,  $\varphi(1, J)$ .
temporal(J+1,1):state :- temporal(J,2):state, integer(J/2), head(J, ...),
                           not head(J-2, ...),  $\psi(J)$ .
```

If there are facts in  $P$  which belong to *head* add the rule:

`head(J, ...) :- head(...),  $\varphi(1,J)$ .`

(v) Add

`temporal(J+1,1):state :- temporal(J,2):state, not integer(J/2),  $\varphi(2,J)$ .  
temporal(J,2):state :- temporal(J,1):state[fixpoint  $\rightarrow$  blubb],  $\varphi(1,J)$ .  
state[fixpoint  $\ast \rightarrow$  blubb].  
?- sys.strat.doIt.`

(vi) and for any IDB-predicate *head* add

`head_true(...) :- head(J,...),  $\psi(J)$ .  
head_undef(...) :- head(J-1,...), not head(J,...),  $\psi(J)$ .`

(vii) That is it!

Let  $k$  denote the number of rules in  $P$ .  $P_{XY_{florid}}$  contains at most  $5k + 3$  rules and one new fact.

□

### Explanation:

Note that variables followed by a subtraction in the first position of a predicate (e.g.  $J-2$ ) are not allowed in Florid as used above, this has to be corrected in an implementation. But for readability we will use this notation here.

We now see why the computation of this version in comparison to the version proposed in [MU99] is faster. We restrict the computation to the highest temporal stage. In [MU99] computation of lower temporal stages is repeated over and over.

Step (iii) adds a helper fact to the database. A counter is initialized for temporal stages. Step (iv) adds the actual rules to be computed and a rule to control termination. Computation stops if everything in an even stage  $J > 0$  can also be found in stage  $J-2$ , which is a sufficient condition due to the properties of wellfounded evaluation (we know here that termination always runs into a two-cycle [see [AHV95]]). In other words, we know exactly where to look for the sets to be tested for equality, a property we will lack of in a general XY-stratified program. Note that at the end of temporal stage  $J = 2$  computation may already terminate. This is the case if nothing was computed in this step. Yet, results delivered here would be correct because there would not be created any more true tuples for an IDB-relation in an alternating fixpoint computation, either. Step (v) adds rules for the control flow. The first rule always creates the next even stage when computation is in an odd stage at the moment. When the computation of stratum one has ended the inheritance trigger in (v) is fired (and not before). Then,  $temporal(J, 2) : state$  is created. The idea behind this construction is that stratum one may need more than one application of Florid's  $T_P$ -operator to be fully computed. So, we have to check, when this is the case in order not to go the next stratum, too early. A computationally cheap way of doing this is to check if the inheritance trigger has already

fired, which, we know, only happens, when the computation of stratum one is guaranteed to have finished before.

In step (vi) it remains to put the tuples which are true or undefined into some relations that do not carry a temporal argument, which is mostly done for user convenience and transparency of computation.

## 2.4 Further examples

We give two more examples here. Explanation of the code is similar as in the example in section 2.1 and is therefore omitted.

### Example 9 (The Barber)

*P*:

```
person(barber)
person(finrod_felagund)
shaves(barber,X) ← person(X), not shaves(X,X)
```

*P<sub>XY</sub>*:

```
person(barber)
person(finrod_felagund)
shaves(J,barber,X) ← person(X), not shaves(J-1,X,X)
```

*P<sub>XYflorid</sub>* :

```
person(barber)
person(finrod_felagund)
temporal(1,1):state.
?- sys.strat.doIt.
```

```
shaves(J,barber,X) :- person(X), not shaves(J-1,X,X),  $\varphi(1,J)$ .
```

```
temporal(J+1,1):state :- temporal(J,2):state, integer(J/2), shaves(J,X,Y),
                        not shaves(J-2,X,Y),  $\psi(J)$ .
```

```
temporal(J+1,1):state :- temporal(J,2):state, not integer(J/2),  $\varphi(2,J)$ .
```

```
temporal(J,2):state :- temporal(J,1):state[fixpoint -> blubb],  $\varphi(1,J)$ .
```

```
state[fixpoint *- > blubb].
```

```
?- sys.strat.doIt.
```

```
shaves_true(X,Y) :- shaves(J,X,Y),  $\psi(J)$ .
```

```
shaves_undef(X,Y) :- shaves(J-1,X,Y), not shaves(J,X,Y),  $\psi(J)$ .
```

□

### Example 10 (Anything)

*P*:  
succ(0,1).  
succ(1,2).  
succ(2,3).  
s(0).  
s(1).  
s(3).  
rel(X)  $\leftarrow$  s(X)  
s(X)  $\leftarrow$  not rel(X), succ(X,Y)

*P<sub>XY</sub>*:  
rel(J,X)  $\leftarrow$  s(J,X)  
s(J,X)  $\leftarrow$  not rel(J-1,X), succ(X,Y)

*P<sub>XYflorid</sub>*:  
succ(0,1).  
succ(1,2).  
succ(2,3).  
s(0).  
s(1).  
s(3).  
temporal(1,1):state.  
?- sys.strat.doIt.

rel(J,X) :- s(J,X),  $\varphi(1,J)$ .  
s(J,X) :- not rel(J-1,X), succ(X,Y),  $\varphi(1,J)$ .  
s(J,X) :- s(X),  $\varphi(1,J)$ .  
temporal(J+1,1):state :- temporal(J,2):state, integer(J/2), rel(J,X),  
not rel(J-2,X),  $\psi(J)$ .  
temporal(J+1,1):state :- temporal(J,2):state, integer(J/2), s(J,X), not s(J-2,X),  
 $\psi(J)$ .  
temporal(J+1,1):state :- temporal(J,2):state, not integer(J/2),  $\varphi(2,J)$ .  
temporal(J,2):state :- temporal(J,1):state[fixpoint -> blubb],  $\varphi(1,J)$ .  
state[fixpoint \*- > blubb].  
?- sys.strat.doIt.

rel\_true(X) :- rel(J,X),  $\psi(J)$ .  
rel\_undef(X) :- rel(J-1,X), not rel(J,X),  $\psi(J)$ .  
s\_true(X) :- s(J,X),  $\psi(J)$ .  
s\_undef(X) :- s(J-1,X), not s(J,X),  $\psi(J)$ .

□

## 2.5 Evaluation of relative performance

How much do we lose, using this kind of alternating fixpoint computation? How large is our penalty, when computing the transitive closure of our borders relation from Mondial first with Florid's normal inflationary fixpoint operator and then with the rewritten version as introduced before. Obviously, well-founded evaluation would not be necessary here, because we do not need negation for computing the transitive closure, but in this way, we can easily compare how (in-)efficient this way of program rewriting really is. We will already use the optimizations from chapter four, so we recommend reading the chapters three and four before going on here.

### Example 11 (Transitive Closure - normal)

```
(... facts from Mondial ...)
?- sys.strat.doIt.
reach(X,Y) :- borders(X,Y,A).
?- sys.strat.doIt.
reach(X,Y) :- borders(X,Z,A), reach(Z,Y).
?- sys.eval.
□
```

### Example 12 (Transitive Closure - well-founded)

```
(... facts from Mondial ...)
temporal(1,1):state.
?- sys.strat.doIt.
reach(J,X,Y) :- borders(X,Y,A),  $\varphi(1,J)$ .
reach(J,X,Y) :- borders(X,Z,A), reach(J,Z,Y),  $\varphi(1,J)$ .
temporal(J+1,1):state :- temporal(J,2):state, not integer(J/2),  $\varphi(2,J)$ .
temporal(J,2):state :-  $\varphi(1,J)$ , temporal(J,1):state[fixpoint -> blubb].
state[fixpoint *-> blubb].
// end of the actual computation
?- sys.strat.doIt.
// copying: mark relevant facts as true or undefined
reach_true(X,Y) :- reach(J,X,Y),  $\psi(J)$ .
reach_undef(X,Y) :- reach(J-1,X,Y), not reach(J,X,Y),  $\psi(J)$ .
?- sys.eval.
□
```

<i>Evaluation</i>		
	<i>well-founded</i>	<i>normal</i>
Loading all data:	8 sec	8 sec
Evaluate facts:	2,66 sec	2,67 sec
Actual computation:	33,02 sec	5,28 sec
Copying (true/undefined):	217,25 sec	-
Total time spent in evaluation:	252,93 sec	7,95 sec
Normal computation was 31,8 times better!		

We wonder why the copying of the facts to their true/undefined relations after the alternating fixpoint has been reached takes so much time. But, even if this was not so, we would still be six times worse than normal computation.

### 3 A more general idea

In the previous section we have had a look at XY-stratified programs which do not terminate but are guaranteed to run into a two-cycle due to their special structure. In general, it is undecidable if a given XY-stratified program terminates on every input (see [LHL95]). Even worse, cycles can have exponential length. One thing we can try is to look for further refined conditions a program has to fulfill, so that we can infer from its syntactical information that it terminates. Here, we will add a cycle detection, which is able to stop a computation, to an XY-program, so that the user has the possibility to decide if the results are useful for him. Naturally, this cycle detection will be inefficient.

#### 3.1 Some Definitions

Let  $P$  be an arbitrary XY-stratified program obeying (\*). Let  $A_P$  denote the set of all non-temporal constants (i.e. constants that do not appear in the place of a temporal argument) in  $P$ . Let  $P_{temporal}$  denote the set of predicates of  $P$  with a temporal argument. For  $h \in P_{temporal}$  let  $arity(h)$  denote the arity of  $h$ .  $m \in \mathbb{N}$

**Definition 13** ( $\mathcal{H}_m(P)$  and  $f_P$ )

$\mathcal{H}_m(P) := \{ h(m, a_1, \dots, a_n) : h \in P_{temporal}, n + 1 = arity(h), a_i \in A_P, i \in [n] \}$   
 We define the function  $f_P : \bigcup_{i=0}^{\infty} \mathcal{H}_i(P) \longrightarrow \mathcal{H}_0(P)$  by  $f_P(h(m, a_1, \dots, a_n)) = h(0, a_1, \dots, a_n)$ .

□

That means that  $\mathcal{H}_m(P)$  stands for the subset of the Herbrand base of  $P$  for a fixed temporal argument  $m$ . It holds that  $\mathcal{H}_m(P)$  is finite, because obviously  $A_P$  and  $P_{temporal}$  are finite. By applying  $f_P$ , we want to express that we do not care about the temporal argument.

**Definition 14** ( $output(P)$  and temporal stage)

The sequence of sets of all inferred IDB-tuples by  $P$  is denoted by  $output(P) \in \prod_{i=0}^{\infty} \mathcal{P}(\mathcal{H}_i(P))$ . The components of  $output(P)$  are the unique maximal subsets of all tuples from IDB-relations (after the computation of  $P$ ) with the correct temporal constant as their first argument.

For  $j \in \mathbb{N}$ , the  $j$ -th temporal stage of  $P$  is the projection on the  $j$ -th component of  $output(P)$ .

□

Note that, we also include facts from the input data in  $P$  to  $output(P)$ . With this definition, we partition the set of all objects into disjunct sets whose elements share the same constant as their temporal argument.

**Definition 15 (k-cyclic)**

Let  $(A_i)_{i \in \mathbb{N}} = \text{output}(P)$ .  $k \in \mathbb{N} \setminus \{0\}$ .

$P$  is called  $k$ -cyclic (or just cyclic)  $:\Leftrightarrow \exists m \in \mathbb{N}, \forall l \in \mathbb{N} : f_P(A_{m+l}) = f_P(A_{m+k+l})$ .

□

A cycle detection for  $P$  tries to detect the values for  $m$  and  $k$ . When they are found, the computation is stopped.

**3.2 The general picture: An introduction**

We now restrict our attention to the subclass of XY-stratified programs (which is still equivalent to Statelog) with the property that

(\*) **temporal variables appear in the first position  
of an IDB-predicate and only there.**

This does not guarantee us at all termination, but we will see that in these cases non-termination is in a certain sense cyclic, which we are able to exploit swinging the "complexity hammer". The reason why we have to unpack it, is motivated by the following examples.

**Example 16 (This is what we try to avoid.)**

Let  $\text{parent}$  be an EDB-relation and  $\text{temp}_{anc}$  an IDB-relation. The underlined temporal arguments are forbidden:

$$\Delta_{anc}(J, X, \underline{J}) \leftarrow \text{temp}_{anc}(J-100, X, \underline{J-1}), \text{parent}(\underline{J-1}, \underline{J-10})$$

The reason for this is that such a use of temporal variables is equivalent to arithmetical comparisons with the temporal argument (i.e. that we could delay the firing of a rule).

$$\Delta_{anc}(J, X, Y) \leftarrow \text{temp}_{anc}(J-100, X, Z), \text{parent}(Z, W), J = Y, Z = J-1, W = J-10$$

So, our cycle detection could become active, too early, and stop the computation under the assumption that it has already become cyclic, although this is not yet the case. Looking at the example, this could be the case if all rules for  $\text{temp}_{anc}$  are unsatisfiable and  $\text{temp}_{anc}(0, a, 99)$  and  $\text{parent}(99, 90)$  are all the input data.

Note that we could allow arithmetical comparisons including temporal variables by computing at least as many temporal stages as the highest integer in our input  $+depth(P)$ . The effect of this would be that an arithmetical comparison including a temporal variable could be fulfilled in all further temporal stages or that it would be unsatisfiable in all further stages. So, one would have to let our program run until they had no effect any more. It could be interesting to have rules able to fire only from stage  $J = 10$  on,

but due to limited time, this will not be considered here.

In contrast to arithmetic comparisons two or more temporal variables in a rule's head cannot be allowed, this may yield in a non-cyclic computation:

```
blubb(0,0)
blubb(J,J) ← blubb(J-1,J-1)
□
```

### Example 17 (Let computation live)

We also have notice that  $height(P)$  may not be zero. In this case, we have to assure that the computation does not stop before reaching the temporal stage  $height(P)$ :

```
succ(0,1)
s(0)
s(10,2)
rel(J,X) ← s(J-1,X)
s(J,X) ← succ(X,Y), not rel(J-1,X)
s(J,X) ← s(X)
```

Note that  $height(P) = 10 !$   
□

### Example 18 (Losing pleasant properties)

A non-terminating XY-stratified program obeying (\*) which cannot come from an alternating fixpoint computation (due to the rule for the IDB-predicate *rel*).

```
succ(0,1)
succ(1,2)
succ(2,3)
t(0)
t(1)
t(3)
rel(J,X) ← s(J-1,X)
s(J,X) ← succ(X,Y), not rel(J-1,X)
s(J,X) ← t(X)
```

This program generates the following tuples (and thus is four-cyclic):

<i>Round</i>					
<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>
s(1,0)	s(2,0)	s(3,0)	s(4,0)	s(5,0)	s(6,0)
s(1,1)	s(2,1)	s(3,1)	s(4,1)	s(5,1)	s(6,1)
s(1,2)	s(2,2)			s(5,2)	s(6,2)
s(1,3)	s(2,3)	s(3,3)	s(4,3)	s(5,3)	s(6,3)
	rel(2,0)	rel(3,0)	rel(4,0)	rel(5,0)	rel(6,0)
	rel(2,1)	rel(3,1)	rel(4,1)	rel(5,1)	rel(6,1)
	rel(2,2)	rel(3,2)			rel(6,2)
	rel(2,3)	rel(3,3)	rel(4,3)	rel(5,3)	rel(6,3)

One can also see that a cycle detection trying to stop the computation of this program has to test both subset relationships of a set equality. In general, we do not have the property that e.g. the even subsequence is increasing.

□

In general, the dependency graph for the bistate version of a program is cyclic. So, we see that we know hardly anything of an XY-stratified program obeying (\*).

### 3.3 Last example revisited

We present here a solution for the last example and in the next two subsections our general solution.

#### Example 19

```
succ(0,1). succ(1,2). succ(2,3).
```

```
t(0). t(1). t(3).
```

```
f0: temporal(0,1):state.
```

```
f1: temporal(1,1):state.
```

```
?- sys.strat.doIt.
```

```
f2: rel(J,X) :- s(J-1,X),  $\varphi(1,J)$ .
```

```
f3: s(J,X) :- not rel(J-1,X), succ(X,Y),  $\varphi(1,J)$ .
```

```
f4: s(J,X) :- t(X),  $\varphi(1,J)$ .
```

```
f5: nss_rel(J,J1) :- rel(J1,X), not rel(J,X), temporal(J1,K):state,  $\varphi(2,J)$ .
```

```
f6: nss_rel(J1,J) :- rel(J,X), not rel(J1,X), temporal(J1,K):state,  $\varphi(2,J)$ .
```

```
f7: nss_s(J,J1) :- s(J1,X), not s(J,X), temporal(J1,K):state,  $\varphi(2,J)$ .
```

```
f8: nss_s(J1,J) :- s(J,X), not s(J1,X), temporal(J1,K):state,  $\varphi(2,J)$ .
```

```
f9: cycle(J,JK) :- not nss_rel(JK,J), not nss_rel(J,JK), not nss_s(JK,J),
not nss_s(J,JK), temporal(JK,K):state, JK >= 1,
not J = JK,  $\varphi(3,J)$ .
```

```

f10: stop_eval :- cycle(J,J2),  $\varphi(3,J)$ .

f11: temporal(J+1,1):state :- temporal(J,4):state, not stop_eval,  $\psi(J)$ .
f12: temporal(J,K+1):state :- temporal(J,K):state[fixpoint -> blubb],
      K <= 3,  $\varphi(K,J)$ .
f13: state[fixpoint *-> blubb].
□

```

### Explanation:

Evaluation of the program begins with  $J = 1$ . Though, f0 is needed to guarantee that  $J = 2$  is created (see f9). F2 to f4 represent the rules to be computed, as usual a subgoal  $\varphi(i,J)$  has been added, denoting that these rules belong to stratum  $i$  of the bistrate version. F5 to f8 represent the tests for subset relationships for our IDB-predicates. F9 uses this result to check for set equality for some distinct temporal variables of our IDB-relations and returns the value of the stages where the cycle in the computation begins. Using this method we can detect cycles of arbitrary length. F10 stops evaluation if a cycle has been detected. F11 creates a new temporal stage, if needed. And finally, f12 and f13 are necessary to change the stratum to be computed by checking if the computation of the most current stratum has ended. The explanation, why this works, is similar as in section 2.

Note that f5 to f13 are needed for the control flow. F5 to f11 are kept in separate strata (value  $i$  in  $\varphi(i,J)$ ) because stratum one must be fully computed before testing subset relationships.

## 3.4 Rho

Now we explore the cyclic structure of an XY-stratified program  $P$  obeying (\*). Basically, the lemma is an analogon of Theorem 3.4 in [LHL95].

### Lemma 20 (Rho)

Be  $k = \text{depth}(P)$ . Then, it holds that

- (i)  $\exists m \in \mathbb{N}, \exists T \in \mathbb{N} \setminus \{0\}, \forall l \in \mathbb{N}: f_P(A_{m+l}) = f_P(A_{m+T+l})$ .
- (ii) This can be tested by checking if  $f_P(A_{m-1}) = f_P(A_{m+T-1}), \dots, f_P(A_{m-k}) = f_P(A_{m+T-k})$ . Note that only  $A_0, \dots, A_{m+T}$  must be computed in order to get the property in (i).
- (iii)  $(A_i)_{i \in \mathbb{N}_{\geq m+T+1}}$  can be parametrized and are therefore known implicitly.

### Proof.

- $f_P^+ : \prod_{i=0}^{\infty} \mathcal{P}(\mathcal{H}_i(P)) \longrightarrow \prod_{i=0}^{\infty} \mathcal{P}(\mathcal{H}_0(P))$  with  $f_P^+((A_i)_{i \in \mathbb{N}}) := (f_P(A_i))_{i \in \mathbb{N}}$  is bijective.

- $(B_i)_{i \in \mathbb{N}} := f_P^+(\text{output}(P)) \in \prod_{i=0}^{\infty} \mathcal{P}(\mathcal{H}_0(P))$
- $|\prod_{i=1}^k \mathcal{H}_0(P)| < \infty \Rightarrow \exists (C_1, \dots, C_k) \in \prod_{i=1}^k \mathcal{H}_0(P)$  so that  $(C_1, \dots, C_k)$  appears at least two times in  $(B_i)_{i \in \mathbb{N}}$ . Choose  $m_0 \in \mathbb{N}_{> \max\{k, \text{height}(P)\}}$  with  $B_{m_0-k} = C_1, \dots, B_{m_0-1} = C_k$ .  $m_0$  is chosen greater than  $\text{height}(P)$  to ensure that all input data has been considered for  $\text{output}(P)$  and that all tuples from IDB-relations come from an instantiation of a rule. Choose  $T_0 \in \mathbb{N} \setminus \{0\}$  with  $B_{m_0+T_0-k} = C_1, \dots, B_{m_0+T_0-1} = C_k$ .
- Show  $\forall l \in \mathbb{N} \cup \{-1\} : B_{m_0+T_0+l} = B_{m_0+l}$  by induction over  $l$ .

$l = -1$ : trivial

$l \rightarrow l + 1$ :

Assume  $B_{m_0+T_0-1} = B_{m_0-1}, \dots, B_{m_0+T_0+l} = B_{m_0+l}$  but  $B_{m_0+T_0+l+1} \neq B_{m_0+l+1}$ . Without any loss of generality, assume that  $\exists x \in B_{m_0+T_0+l+1} \setminus B_{m_0+l+1}$  (the other case is similar). And without further loss of generality, assume that  $x$  is the "first" IDB-predicate with this property, i.e. in all previous iteration steps (if existent) needed to compute the temporal stage  $J = m_0 + T_0 + l + 1$  the same tuples were produced as for  $J = m_0 + l + 1$ . Let

$$\text{idb}_i(J, \mathbf{a}_{i_1}, \dots, \mathbf{a}_{i_r}) \leftarrow \begin{array}{l} \text{[not] edb}_{j_1}, \dots, \text{[not] edb}_{j_a}, \text{arithmetics,} \\ \text{idb}_{j_{a+1}}, \dots, \text{idb}_{j_b}, \text{not idb}_{j_{b+1}}, \dots, \text{not idb}_{j_c}. \end{array}$$

be an instantiation of the non-temporal variables of a rule computing the tuple  $x$  in the temporal stage  $J = m_0 + T_0 + l + 1$ , where all  $edb$  are EDB- and all  $idb$  IDB-tuples.

The EDB and the arithmetics parts of the rule body are also satisfiable in stage  $J = m_0 + l + 1$  because no temporal variables can be found here. So, the only critical parts are the part in the rule body containing positive and negative instantiations of IDB predicates.

- (i) Case 1: assume that a negative IDB-predicate cannot be fulfilled. If there are no negative IDB predicates, were done by induction. Otherwise, there must be a negative IDB subgoal which can not be fulfilled for  $J = m_0 + l + 1$ . Together with the induction prerequisite this implies the existence of a tuple  $y \in B_{m_0+l+1} \setminus B_{m_0+T_0+l+1}$  (notice that  $B_{m_0+T_0-k} = B_{m_0-k}, \dots, B_{m_0+T_0+l} = B_{m_0+l}$  and thus  $y$  cannot come from such a temporal stage).  $P_{bis}$  is stratified, so  $y$  must have been generated in a previous computation step before producing  $x$  and cannot be computed any more in future iteration steps needed to compute stage  $J = m_0 + l + 1$ . This yields a contradiction to the assumption!

(ii) Case 2: assume that a positive IDB-predicate cannot be fulfilled. If there are no positive IDB predicates, were done by induction. Otherwise there must be a tuple  $z \in B_{m_0+l+1} \setminus B_{m_0+T_0+l+1}$  which has been produced in an earlier step needed to compute stage  $J = m_0 + l + 1$ . Contradiction to "the minimality" of  $x$ .

- $m := m_0$   
 $T := T_0$
- $\forall l \in \{0, \dots, T - 1\}, \forall n \in \mathbb{N} : B_{m+n+l} = B_{m+l}$ . This follows by induction over  $n$  from the previous steps. Be  $s \in \mathbb{N}_{\geq m}$ .  
 $\Rightarrow B_s = B_{m+((s-m) \bmod T)}$  which means that  $(B_i)_{i \geq m}$  can be parametrized and does not need to be computed explicitly.

□

### 3.5 A termination condition

Be  $k = \text{depth}(P)$ . If  $\text{head}_i$  ( $i \in [n]$ ) are all head predicates in  $P$ , may  $n_{\text{head}_i} + 1$  denote their arity and  $P_{bis}$  has  $j$  strata, it suffices to add the following rules to test the set equality in lemma Rho:

$$\begin{aligned} \text{nss\_head}_i(J, J1) &:- \text{head}_i(J1, X_1, \dots, X_{n_{\text{head}_i}}), \text{not head}_i(J, X_1, \dots, X_{n_{\text{head}_i}}), \\ &\quad \varphi(j+1, J), \text{temporal}(J1, K) : \text{state}. \\ \text{nss\_head}_i(J1, J) &:- \text{head}_i(J, X_1, \dots, X_{n_{\text{head}_i}}), \text{not head}_i(J1, X_1, \dots, X_{n_{\text{head}_i}}), \\ &\quad \varphi(j+1, J), \text{temporal}(J1, K) : \text{state}. \end{aligned}$$

$$\begin{aligned} \text{cycle}(J, JK) &:- \text{temporal}(JK-(k-1), K) : \text{state}, \text{seq}(\text{head}_1, k), \dots, \text{seq}(\text{head}_n, k), \\ &\quad \varphi(j+2, J), JK \geq (2*k-1), JK > (\text{height}(P)+k-1), \text{not } J = JK. \end{aligned}$$

For shortness define:

$$\begin{aligned} \text{seq}(\text{test}, k) &:= \text{not nss\_test}(JK, J), \text{not nss\_test}(J, JK), \dots, \text{not} \\ &\quad \text{nss\_test}(JK-(k-1), J-(k-1)), \text{not nss\_test}(J-(k-1), JK-(k-1)) \end{aligned}$$

By adding the subgoals  $\varphi(j+1, J)$  and  $\varphi(j+2, J)$  we define that the test, if the termination condition is fulfilled, is done at the end of a temporal stage. Let  $\vartheta$  be an instantiation of the variables in the "cycle" rule which fulfills all subgoals of the body. It holds that  $\vartheta(JK) < \vartheta(J)$ . All  $JK + (h - 1)$  are safe because  $JK$  is safe. If there are several rules in  $P$ , the last rule may become very long:  $2k$  negated subgoals are added to it per head predicate in  $P$ . This must be due to the fact that all set equalities must be tested at the same time:

$$\begin{aligned}
f_P(\{head_1(j_1, \dots) \in p_{j_1}(output(P))\}) &= f_P(\{head_1(j_2, \dots) \in p_{j_2}(output(P))\}) \\
&\vdots \\
f_P(\{head_n(j_1, \dots) \in p_{j_1}(output(P))\}) &= f_P(\{head_n(j_2, \dots) \in p_{j_2}(output(P))\})
\end{aligned}$$

where  $\vartheta(JK) = j_2$  and  $\vartheta(J) = j_1$ . The added subgoals for  $head_i$  are of the form

$$\text{not nss\_head}_i(A, B), \text{ not nss\_head}_i(B, A)$$

for some temporal variables  $A$  and  $B$  with  $\vartheta(A) - \vartheta(B) = \vartheta(JK) - \vartheta(J)$ . The rest of the *cycle*-rule stays as it is.

### 3.6 An algorithm

We give here an algorithm that computes a Florid-executable version  $P_{florid}$  from an arbitrary XY-stratified Datalog program obeying (\*) with safe variables.

#### Algorithm 21 ( $P_{florid}$ )

Let  $P$  denote an arbitrary XY-stratified program with safe variables. Let  $k$  be  $depth(P)$ . We now construct a program  $P_{florid}$  that emulates the computation of  $P$  as follows:

- (i) Compute a stratification of  $P_{bis}$ . Remember which predicate in  $P_{bis}$  comes from which predicate in  $P$  and vice versa. Let  $j$  denote the total number of strata in  $P_{bis}$ .
- (ii) Add all the facts from  $P$ .
- (iii) Add the three lines:

```
temporal(0,1):state.
temporal(1,1):state.
?- sys.strat.doIt.
```

- (iv) For any rule  $head \leftarrow body$  in  $P$  add the following rule:

```
head :- body,  $\varphi(i, J)$ 
```

where  $i$  stands for the number of the stratum which contains the corresponding predicate of  $head$  in  $P_{bis}$ .

- (v) For each IDB-predicate  $head$  let  $n_{head} + 1$  denote its arity. Let  $X_1, \dots, X_{n_{head}}$  be variables. Add:

```
nss_head(J, J1) :- head(J1, X1, ..., X $n_{head}$ ), not head(J, X1, ..., X $n_{head}$ ),
                     $\varphi(j + 1, J)$ , temporal(J1, K):state.
nss_head(J1, J) :- head(J, X1, ..., X $n_{head}$ ), not head(J1, X1, ..., X $n_{head}$ ),
                     $\varphi(j + 1, J)$ , temporal(J1, K):state.
```

(vi) Let  $head_1, \dots, head_m$  be all IDB-predicate symbols. Add the rules:

```

cycle(J,JK) :- seq(head1,k), ..., seq(headm,k), temporal(JK-(k-1),K):state,
               φ(j+2,J), JK >= (2*k-1), JK > (height(P)+k-1), not J=JK.

stop_eval :- cycle(J,J2), φ(j+2,J).

temporal(J+1,1):state :- temporal(J,(j+3)):state, not stop_eval, ψ(J).
temporal(J,K+1):state :- temporal(J,K):state[fixpoint - > b],
                          K <= (j+2), φ(K,J).

state[fixpoint *- > b].

```

We want to state again, that it is thoroughly assumed that  $J$  is the distinguished temporal variable in all rules in  $P$ , that all newly introduced predicate symbols do not appear in  $P$  and that there are no predicate symbols overloaded in  $P$ .  $\square$

**Explanation:**

The use of the helper symbol *temporal* is the same as in the chapter before. In step (iv) the actual rules to be computed are added. By adding  $\varphi(i,J)$  as a subgoal to them we restrict them only to be fired when the stratification of  $P_{bis}$  allows them to. Step (v) should strongly hurt the soul of the reader. Although, we know that the computation  $P$  will run into a  $l$ -cycle, we cannot effectively deduce the value for  $l$  beforehand (especially it depends on the input data and not only on the syntactical structure of the program). So, we unfortunately have to check for any value of  $l$ , which is done in (v) by checking both subset relationships and in the first rule in (vi) by bringing the two subset relationships from (v) together. Computation is stopped, if a cycle is detected by inferring the fact *stop\_eval*. The last three rules in (vi) represent the changing of strata and the creation of new temporal stages, by either looking if the computation of the stratum considered at the moment has ended in a stratum and checking for some cycle in the computation of the temporal stages.

We will give an example at the end of the next chapter!

## 4 Optimization

Deep in our hearts we strongly know that the termination condition above is much too heavy and expensive. Here, we try to give an idea to shorten it by exploiting the structure of the program. First, we want to point the problem out. Then, we give a possible partial solution.

### 4.1 Introduction

Consider a well-known example which has been rewritten in an XY-stratified manner:

#### Example 22 (Seminaive Computation of Ancestors)

$P_{ancestors}$  :

- s1:  $\Delta_{anc}(1, X, Y) \leftarrow \text{parent}(X, Y)$
- s2:  $\Delta_{anc}(J, X, Y) \leftarrow \Delta_{anc}(J-1, X, Z), \text{anc}(J-2, Z, Y), \text{not } \text{anc}(J-1, X, Y)$
- s3:  $\Delta_{anc}(J, X, Y) \leftarrow \text{anc}(J-1, X, Z), \Delta_{anc}(J-1, Z, Y), \text{not } \text{anc}(J-1, X, Y)$
- s4:  $\text{anc}(J, X, Y) \leftarrow \text{anc}(J-1, X, Y)$
- s5:  $\text{anc}(J, X, Y) \leftarrow \Delta_{anc}(J, X, Y)$

□

Optimally, it would be sufficient to check if  $\Delta_{anc}$  becomes empty for some temporal stage to terminate the computation of the program. Here, of course, we use a lot of semantical knowledge about  $P_{ancestors}$ , e.g. that it finally becomes 1-cyclic. In general, we will not be able to give an optimal termination condition for every program as the word "general" already implies. What we can achieve for  $P_{ancestors}$ , is that we can perform our check for set equality only for  $\Delta_{anc}$  and leave the *anc*-relation out of consideration, which can be derived from pure syntactical information: except for the copy rule s4 *anc* depends only on predicates which share the same temporal argument as *anc*.

### 4.2 A partial optimization

Given an XY-stratified program  $P$ , let  $\{rule_1, \dots, rule_n\}$  be all the rules in  $P$ . Let  $edb_j$  denote EDB-relations or IDB-relations with a constant as their temporal argument.

1. First, delete all copy rules, i.e. delete all rules which are of the following form:

$$\text{head}(J, X_1, \dots, X_l) \leftarrow \text{head}(J-1, X_1, \dots, X_l)$$

and mark each head predicate of a removed rule that appears in the head of a remaining rule with a star. For these relations there must only be tested one subset relationship to gain set equality:

$$f_P(\{\text{head}(j_1, \dots) \in p_{j_1}(\text{output}(P))\}) \subseteq f_P(\{\text{head}(j_2, \dots) \in p_{j_2}(\text{output}(P))\})$$

because the other inclusion already holds due to the copy rule above, where  $j_1 > j_2$  and  $p_j$  is the projection on the  $j$ -th component.

2. Let  $\{rule_{n_1}, \dots, rule_{n_i}\}$  denote the set of the remaining rules. Delete all X-rules, i.e. all rules in which only  $J$  appears as a temporal variable (and not  $J - 1, J - 2, \dots$ ). Note that constants in the place of a temporal argument are not meant with this. Delete all rules of the form

$$\text{head} \leftarrow [\text{not}] \text{edb}_1, \dots, [\text{not}] \text{edb}_m$$

Such rules produce the same tuples (modulo  $f_P$ ) in every temporal stage and thus are irrelevant for termination of the program.

3. Let  $\{rule_{n_{i_1}}, \dots, rule_{n_{i_j}}\}$  denote the set of the remaining rules. Let  $head$  be a head predicate of a remaining rule. If
  - there is one appearance of  $head$  in the left side of a rule marked with a star (see step 1), then only the subset relationship mentioned above has to be tested to gain set equality in our termination condition.
  - all appearances of  $head$  as a predicate in a head have not been marked with a star, then we have to test the two usual subset relationships, i.e. no optimization at all.

No other subset relationships need to be considered for termination, i.e. head predicates in  $P$  which do not appear as head predicates in  $\{rule_{n_{i_1}}, \dots, rule_{n_{i_j}}\}$  are irrelevant for termination. The reason for the removal of a rule with its only temporal variable being  $J$  is as follows: let  $body_1, \dots, body_n$  be all IDB-relations appearing in this rule with the temporal variable  $J$  and  $head$  its head predicate. If

$$\begin{aligned} f_P(\{body_1(j_1, \dots) \in p_{j_1}(output(P))\}) &= f_P(\{body_1(j_2, \dots) \in p_{j_2}(output(P))\}) \\ &\vdots \\ f_P(\{body_n(j_1, \dots) \in p_{j_1}(output(P))\}) &= f_P(\{body_n(j_2, \dots) \in p_{j_2}(output(P))\}) \end{aligned}$$

for  $j_1 > j_2 \geq \max\{height(P), depth(P)\}$ , then it must hold that

$$f_P(\{head(j_1, \dots) \in p_{j_1}(output(P))\}) = f_P(\{head(j_2, \dots) \in p_{j_2}(output(P))\})$$

which can be shown by induction on the number of steps needed to compute the temporal stage  $j_1$ . And if there is some  $i$  for which

$$f_P(\{body_i(j_1, \dots) \in p_{j_1}(output(P))\}) \neq f_P(\{body_i(j_2, \dots) \in p_{j_2}(output(P))\}),$$

then it trivially holds that the computation of  $P$  has not yet become 1-cyclic.

These two implications assure that our weakened test for termination is still correct.

### 4.3 What we must not "optimize"

Here, we show the necessity of the subgoals in  $seq(test, depth(P))$  (where  $test$  is an IDB-predicate). Consider the following example.

#### Example 23 (Necessity of $seq(\dots)$ )

```
anc(0,5)
bla(0,7)
anc(J,X) ← anc(J-5,X)
bla(J,X) ← bla(J-7,X)
```

It follows that

```
seq(anc,7) := not nss_anc(JK,J), ..., not nss_anc(JK-6,J-6)
seq(bla,7) := not nss_bla(JK,J), ..., not nss_bla(JK-6,J-6)
```

One could be tempted to substitute these two strings of characters by

```
seq'(anc,7) := not nss_anc(JK-4,J-4)
seq'(bla,7) := not nss_bla(JK-6,J-6)
```

with the "reason" that it suffices to check the set equalities only for temporal stages which are needed to compute the last temporal stage. The program above serves as a counter example and thus shows the necessity of the subgoals in  $seq(\dots)$ . We do not go into details here, because this would require the computation of the program until the temporal stage  $J = 43$ , which would take too much space here.

□

### 4.4 An Example

We give an example with  $depth(P) = 1$  and  $height(P) = 0$ , where the optimization is already used in  $P_{florid}$ . Basically, the example program computes pairs of nodes from the same level in the graph given below.

#### Example 24 (mainly taken from [AHV95])

```
P: (EDB is not included)
delta(1,X,Y) ← flat(X,Y)
delta(J,X,Y) ← up(X,A), delta(J-1,B,A), down(B,Y), not rsg(J-1,X,Y)
rsg(J,X,Y) ← rsg(J-1,X,Y)
rsg(J,X,Y) ← delta(J,X,Y), not X=Y
close(J,X,Y) ← rsg(J,X,Y), not flat(X,Y)
```

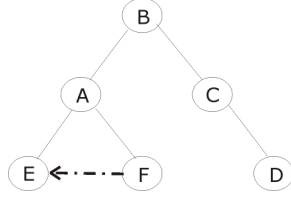


Figure 2: Graphical overview of the EDB

up(a,b)	flat(a,a)	down(a,e)
up(c,b)	flat(b,b)	down(a,f)
up(d,c)	flat(c,c)	down(b,a)
up(e,a)	flat(d,d)	down(b,c)
up(f,a)	flat(e,e)	down(c,d)
	flat(f,e)	
	flat(f,f)	

Table 1: Notational overview of the EDB

*P<sub>florid</sub>*: (EDB is not included)

temporal(0,1):state.

temporal(1,1):state.

?- sys.strat.doIt.

delta(1,X,Y) :- flat(X,Y),  $\varphi(1,J)$ .

delta(J,X,Y) :- up(X,A), delta(J-1,B,A), down(B,Y), not rsg(J-1,X,Y),  $\varphi(1,J)$ .

rsg(J,X,Y) :- rsg(J-1,X,Y),  $\varphi(1,J)$ .

rsg(J,X,Y) :- delta(J,X,Y), not X=Y,  $\varphi(1,J)$ .

close(J,X,Y) :- rsg(J,X,Y), not flat(X,Y),  $\varphi(1,J)$ .

nss\_delta(J,J2) :- delta(J2,X,Y), not delta(J,X,Y), temporal(J2,K):state,  $\varphi(2,J)$ .

nss\_delta(J2,J) :- delta(J,X,Y), not delta(J2,X,Y), temporal(J2,K):state,  $\varphi(2,J)$ .

cycle(J,J2) :- not nss\_delta(J2,J), not nss\_delta(J,J2), temporal(J2,K):state,  
not J = J2,  $\varphi(3,J)$ .

stop\_eval :- cycle(J,J2), temporal(J2,K):state,  $\varphi(3,J)$ .

temporal(J+1,1):state :- temporal(J,4):state, not stop\_eval,  $\psi(J)$ .

temporal(J,K+1):state :- temporal(J,K):state[fixpoint  $\rightarrow$  b], K <= 3,  $\varphi(K,J)$ .

state[fixpoint  $\ast \rightarrow$  b].

□

### Short explanation:

First, *P* was rewritten as described in section 3. Then, optimization was applied:

- leaving out copy rules, *rsg* only depends on X-rules, so no subset relationships have to be tested,
- *close* depends only on the most current temporal stage and EDB, so nothing has to be tested there either and
- finally no optimization could be applied to the rules with head predicate *delta*.

The other changes to the original program  $P$  are explained like in section 3.2 .

## 5 Conclusions and acknowledgments

Florid2 is powerful enough to allow the execution of XY-stratified Datalog programs, which have been rewritten in the presented manner. The depth of a program does not need to be restricted to 1, because earlier temporal stages are present in Florid's object manager anyway, and thus await to be used. This does not change the fact that such programs can be considered locally stratified. The class of XY-stratified programs with  $depth = 1$  are equivalent to WHILE, so the extension of our definition is syntactical sugar, but perhaps sometimes convenient.

The computation of a general XY-stratified program is naturally inefficient because we do not know where to look for the cycle to begin, so we have to check all possibilities. But this implies that, even if we were technically able to delete parts of the object manager, we could not do this, because our cycle could involve parts of the deleted data, which would result in the computation of more temporal stages than actually needed. If the index structures used in the object manager could be optimized to the needs of such programs (e.g. indexing by the temporal stage, ...), computation could be sped up.

We want to thank Ingmar Berger, Matthias Heizmann and Pablo Yáñez Trujillo for their helpful beta reading of this document and mental support!

## References

- [AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*. Addison Wesley, Reading, US, 1995.
- [AOT<sup>+</sup>03] Faiz Arni, KayLiang Ong, Shalom Tsur, Haixun Wang, and Carlo Zaniolo. The deductive database system LDL++. *TPLP*, 3(1):61–94, 2003.
- [LHL95] Bertram Ludäscher, Ulrich Hamann, and Georg Lausen. A logical framework for active rules. In *COMAD*, page 0, 1995.
- [MLL97] W. May, B. Ludaescher, and G. Lausen. Well-founded semantics for deductive object-oriented database languages. *Lecture Notes in Computer Science*, 1341:320–??, 1997.
- [MU99] Wolfgang May and Heinz Uphoff. How to write F-logic programs in FLORID - A tutorial for the database language F-logic, December 15 1999.
- [Prz88] T. Przymusiński. On the declarative semantics of deductive databases and logic programs. In *Foundations of deductive databases and logic programming*, pages 193–216, Los Altos, CA., 1988. Morgan Kaufmann.
- [Ull] Jeffrey D. Ullman. *Lecture Notes in CS345*.

# Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Arbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Freiburg im Breisgau,

\_\_\_\_\_ Datum, Unterschrift