Extending the FOLON Environment for Automatically Deriving Totally Correct Prolog Procedures from Logic Descriptions[1]

Baudouin Le Charlier and Sabina Rossi
University of Namur, 21 rue Grandgagnage, B-5000 Namur (Belgium)
{ble,sro}@info.fundp.ac.be

Abstract

The FOLON environment [2, 8] is based on Deville's methodology for logic program development [6]. In this context, we propose an algorithm which proves the total correctness of a Prolog procedure with respect to a formal specification. Another algorithm for automatically deriving such procedures is also presented.

Introduction

The FOLON environment (see [2, 4, 8]) was designed with the goal of supporting the automatable aspects of Deville's methodology for logic program development presented in [6]. In this context, we aim at specifying a complete tool in order to automatically derive (totally) correct Prolog procedures from a formal specification (consisting of a pure logic description of the relation, type information and a set of behavioural assumptions).

First, we present an analyzer which combines various static analysis techniques in order to verify the correctness criteria for a Prolog procedure. This is a refinement and an extension of the analyzer presented in [3] which only performs mode and type verification. The main novelties of our analyzer are the following:

1. abstract sequences (see [1, 10]) are used instead of abstract substitutions: this allows us to infer information about the number of solutions and termination;

2. completeness and termination analyses are performed (they were not considered in [3]): we adapt the framework proposed by De Schreye, Verschaetse and Bruynooghe in [5] for proving termination in the case of recursive calls.

A synthesizer, based on the analyzer, is also proposed. It finds correct permutations (of literals and clauses) of a procedure and returns one according to some minimality criteria.

The paper is organized as follows. In section 1, the context of the FOLON environment is illustrated. Section 2 contains some basic definitions. In section 3, the analyzer is described. An example illustrates the main operations. In section 4, the synthesizer is presented. Section 5 concludes the paper.

1 The Context

According to Deville's methodology for logic program development, the FOLON environment is based on three main development steps:

1. elaboration of a specification,

2. construction of a correct logic description,

3. derivation of a correct and efficient Prolog procedure.

[1] Partly supported by the EEC Human Capital Mobility individual grant: "Logic Program Synthesis and Transformation" CHRX-CT93-0414.

Let us illustrate this context with the construction of the procedure `efface/3` which removes the first occurrence of `X` from the list `L` containing it and returns the list `LEff`.

A specification for `efface/3` is given in figure 1. Indeed, we extend the general specification form proposed by Deville in [6] with extra information which is useful for proving termination of the derived procedures. First, the name and the formal parameters of the procedure are specified. The type, the relation and the size relation components express properties on the formal parameters which are intended to be satisfied after any successful execution. The semi-linear norm (see [5]) $\|\cdot\|_\lambda$ associates to each term `t` a natural number $\|\mathtt{t}\|_\lambda$ by: $\|[\mathtt{t}_1|\mathtt{t}_2]\|_\lambda = 1 + \|\mathtt{t}_2\|_\lambda$ and $\|\mathtt{t}\|_\lambda = 0$ if `t` is not of the form $[\mathtt{t}_1|\mathtt{t}_2]$.

procedure `efface(X,L,LEff)`
Type: `X : term; L, LEff : Lists;`
Relation: `X` is an element of `L` and `LEff` is the list `L` without the first occurence of `X` in `L`
Semi-linear norm: $\|\cdot\|_\lambda$
Size relation: `L = LEff + 1`
Application conditions:

in(ground, ground, any) :: **out**(ground, ground, ground)	$<0,1>$	`L`
in(ground, any, ground) :: **out**(ground, ground, ground)	$<0,*>$	`LEff`

Figure 1: Specification for `efface/3`

The size relation is a system of linear equations on the formal parameters of the procedure expressing a relation between the corresponding norms. In the example, after any successful execution, the norm of `L` is required to be equal to the norm of `LEff` plus `1`. The application conditions consist of three components: directionality, multiplicity and size expression. Each directionality specifies the allowed modes of the parameters before the execution (**in** part) and the corresponding modes after a successful execution (**out** part). A multiplicity is a pair $<\mathtt{Min},\mathtt{Max}>$ with $\mathtt{Min}\in\mathbf{N}\cup\{\infty\}$ and $\mathtt{Max}\in\mathbf{N}\cup\{\infty,*\}$. For any call satisfying the **in** part of the corresponding directionality and producing the sequence `S` of answer substitutions, the following is expected to hold: $\mathtt{Min}\leq|\mathtt{S}|$ and if $\mathtt{Max}=*$ then $|\mathtt{S}|$ is finite, otherwise $|\mathtt{S}|\leq\mathtt{Max}$. The size expression is a positive linear expression[2] on the formal parameters of the procedure. It associates to each possible call, respecting the corresponding **in** part, a weight from a well-founded set which will be not affected by any further instantion of the parameters. Such a weight is obtained by replacing the formal parameters, in the size expression, with the norm of the corresponding actual ones. In order to prove termination of a recursive call we need to prove that its weight is smaller than the weight for the initial one. In the example, the weight for the calls of `efface/3` respecting the first (resp. the last) **in** part is given by the norm of the actual parameter corresponding to `L` (resp. `LEff`). Since this parameter is required to be ground, such a weight is guaranteed to be invariant for any further instantiation.

A correct logic description for `efface/3` is depicted in figure 2.

$$\mathtt{efface(X,L,LEff)} \iff \mathtt{L=[H|T]} \wedge (\mathtt{H=X}\wedge\mathtt{LEff=T}\wedge\mathtt{list(T)} \vee \mathtt{H}\neq\mathtt{X}\wedge\mathtt{efface(X,T,TEff)}\wedge\mathtt{LEff=[H|TEff]})$$

Figure 2: Logic description for `efface/3`

It can be syntactically translated into the Prolog procedure $\mathtt{LP}_1$`(efface/3)` in figure 3[3].

```
efface(X,L,LEff) ← L=[H|T], H=X, LEff=T, list(T)
efface(X,L,LEff) ← L=[H|T], not(H=X), efface(X,T,TEff), LEff=[H|TEff]
```

Figure 3: $\mathtt{LP}_1$`(efface/3)`

[2] It corresponds, in a sense, to the natural level mapping used by De Schreye *et al.* in [5].
[3] This translation step can be easily automated using a well-defined set of rules as described by Deville in [6].

To be correct the procedure has to respect the following criteria: during any execution (based on the SLDNF-resolution) of `efface/3` called with arguments respecting at least one **in** part of its specification and producing the sequence `S` of answer substitutions,

1. the computation rule is safe, i.e., when selected, the negative literals are ground;
2. any subcall is called with arguments respecting at least one **in** part of its specification;
3. the arguments of the procedure after the execution respect the types, the size relation and the **out** part of each directionality whose **in** part is satisfied by the initial call;
4. `S` respects the multiplicity of each directionality whose **in** part is satisfied by the initial call;
5. completeness: every computed answer substitution in the SLDNF-tree belongs to `S` (i.e. it must eventually be reached according to Prolog search rule);
6. termination: if `S` is finite then the execution terminates.

Information about the length of `S` is given by the specification (multiplicity). When `S` is finite, points 5 and 6 are satisfied if the execution of each clause of the procedure terminates. Termination of a clause is achieved when each literal in its body which is not a recursive call terminates and the weights for the recursive calls are smaller than the weight for the initial call. When `S` can be infinite then only completeness has to be verified. A sufficient criterion for completeness is the following: at most one literal in a clause of the procedure produces an infinite sequence of answer substitutions and either this literal is in the last clause or the following clauses are finitely failed (i.e. they terminate without producing any result).

Let us check the correctness of the procedure $\mathtt{LP}_1(\mathtt{efface/3})$. Consider the second clause for the calls of $\mathtt{efface(X,L,LEff)}$ respecting the first directionality $\mathbf{in}(\mathtt{ground},\mathtt{ground},\mathtt{any})$, i.e. `X` and `L` are ground terms and `LEff` is any term. After the execution of the first literal, `H` and `T` are ground terms and `L` is a ground list. So, when selected, the negative literal is ground, i.e. the computation rule is safe. In order to analyze the next literal, which is a procedure call, its specification is used instead of its code. $\mathtt{efface(X,T,TEff)}$ is called with `X` and `T` being ground terms and `TEff` being a variable, i.e. it satisfies the first directionality. Looking at the specification we can state that, after the execution, `X` is a ground term and `T` and `TEff` are ground lists. The last built-in binds `LEff` to a ground list. According to the approach proposed by De Schreye *et al.* in [5] for proving termination, at each execution point, we infer information about the size relations between the parameters and then we check if such information implies that the weight for the recursive call is smaller than the weight for the initial one. After the execution of the first literal, we can state that the norm of `L` is equal to the norm of `T` plus 1, i.e. $\mathtt{L} = \mathtt{T} + 1$. This still holds after the execution of the second literal. Therefore, when the recursive call is reached, we can infer that its weight, the norm of `T`, is smaller than the weight of the initial call, the norm of `L`.
Consider now the calls of $\mathtt{efface(X,L,LEff)}$ satisfying the second directionality $\mathbf{in}(\mathtt{ground},\mathtt{any},\mathtt{ground})$, i.e. `X` and `LEff` are ground terms and `L` is any term. With the same kind of static analysis, we find a problem when the negative literal is called since `H` is any term, i.e. the computation rule is not guaranteed to be safe. This problem can be solved by permuting literals in the clause. The procedure $\mathtt{LP}_2(\mathtt{efface/3})$ in figure 4 can be proved to be correct for any call respecting some of the specified directionalities.

$$\begin{array}{lcl} \mathtt{efface(X,L,LEff)} & \leftarrow & \mathtt{L} = [\mathtt{H}|\mathtt{T}],\ \mathtt{H} = \mathtt{X},\ \mathtt{LEff} = \mathtt{T},\ \mathtt{list(T)} \\ \mathtt{efface(X,L,LEff)} & \leftarrow & \mathtt{L} = [\mathtt{H}|\mathtt{T}],\ \mathtt{LEff} = [\mathtt{H}|\mathtt{TEff}],\ \mathtt{efface(X,T,TEff)},\ \mathtt{not(H = X)} \end{array}$$

Figure 4: $\mathtt{LP}_2(\mathtt{efface/3})$

2 Basic Definitions

In this section we briefly describe the abstract domains used by the analyzer. First, we introduce the notion of mode-type which encompasses mode and type information.

A *mode-type* represents some set of terms with the same form and/or structure given by the concretization function γ. For instance, $\mathtt{var}, \mathtt{ground}, \mathtt{ngv}, \mathtt{any}, \mathtt{list}, \mathtt{groundlist}, \mathtt{anylist}$ represent the sets: $\gamma(\mathtt{var}) = \{\mathtt{t} : \mathtt{t}$ is a variable term$\}$, $\gamma(\mathtt{ground}) = \{\mathtt{t} : \mathtt{t}$ is a ground term$\}$, $\gamma(\mathtt{ngv}) = \{\mathtt{t} : \mathtt{t}$ is neither a ground nor a variable term$\}$, $\gamma(\mathtt{any}) = \gamma(\mathtt{var}) \cup \gamma(\mathtt{ground}) \cup \gamma(\mathtt{ngv})$, $\gamma(\mathtt{list}) = \{\mathtt{t} : \mathtt{t}$ is a list $\}$, $\gamma(\mathtt{groundlist}) = \{\mathtt{t} : \mathtt{t}$ is a ground list $\}$ and $\gamma(\mathtt{anylist}) = \{\mathtt{t} : \mathtt{t}$ has at least one instance which is a ground list$\}$.

Definition 2.1 [mode-type assignment]
A *mode-type assignment* α is a finite set $\{\mathtt{X_1}/\mathtt{mt_1}, \ldots, \mathtt{X_n}/\mathtt{mt_n}\}$ where $\mathtt{n} \geq 0$, $\mathtt{X_1}, \ldots, \mathtt{X_n}$ are distinct variables and $\mathtt{mt_1}, \ldots, \mathtt{mt_n}$ are mode-types.

An abstract substitution gives information about modes, types, nosharing and size relations. In order to represent information about modes and types, it binds the variables of its domain to special terms called *mode-typed* terms defined as follows.

Definition 2.2 [mode-typed term]
A *mode-typed term* is either an *indexed mode-type* $< \mathtt{mt}, \mathtt{i} >$ where $\mathtt{mt}$ is a mode-type and $\mathtt{i}$ is an index, or an expression of the form $\mathtt{f}(\mathtt{mtt_1}, \ldots, \mathtt{mtt_n})$ where $\mathtt{n} \geq 0$, $\mathtt{f}$ is an n-ary functor and $\mathtt{mtt_1}, \ldots, \mathtt{mtt_n}$ are mode-typed terms.
A mode-typed term $\mathtt{mtt}$ denotes the set of terms $\gamma(\mathtt{mtt}) = \{\mathtt{t} : \mathtt{t}$ is a term obtained from $\mathtt{mtt}$ by replacing each indexed mode-type $< \mathtt{mt}, \mathtt{i} >$ with a term belonging to $\gamma(\mathtt{mt})\}$.

Note that indexed mode-types with the same mode-type but distinct indices may receive distinct terms or the same term. Only indexed mode-types with the same mode-type and the same index are forced to be instantiated to the same value.

Abstract substitutions express nosharing constraints in the form of pairs of indexed mode-types. The size relations are represented by means of a (possibly empty) system of linear equations[4].

Definition 2.3 [abstract substitution]
An *abstract substitution* β is either $\bot$ or a term $(\{\mathtt{X_1}/\mathtt{mtt_1}, \ldots, \mathtt{X_n}/\mathtt{mtt_n}\}, \mathtt{nosh}, \mathtt{E})$ where

1. $\mathtt{X_1}, \ldots, \mathtt{X_n}$ $(\mathtt{n} \geq 0)$ are distinct variables and form the domain of the abstract substitution;
2. $\mathtt{mtt_1}, \ldots, \mathtt{mtt_n}$ are mode-typed terms;
3. $\mathtt{nosh}$ is a set of pairs $(\mathtt{imt_1}, \mathtt{imt_2})$ where $\mathtt{imt_1}$ and $\mathtt{imt_2}$ are distinct indexed mode-types used in $[\mathtt{mtt_1}, \ldots, \mathtt{mtt_n}]$;
4. $\mathtt{E}$ is a (possibly empty) system of linear equations only using variables in $\mathtt{X_1}, \ldots, \mathtt{X_n}$.

An abstract substitution β represents the set, $\gamma(\beta)$, of all concrete substitutions $\{\mathtt{X_1}/\mathtt{t_1}, \ldots, \mathtt{X_n}/\mathtt{t_n}\}$ such that each $\mathtt{t_i}$ is a term obtained from $\mathtt{mtt_i}$ by replacing all indexed mode-types $< \mathtt{mt}, \mathtt{i} >$ with a term belonging to $\gamma(\mathtt{mt})$ such that if $(< \mathtt{mt_1}, \mathtt{i_1} >, < \mathtt{mt_2}, \mathtt{i_2} >) \in \mathtt{nosh}$ then the term substituted to $< \mathtt{mt_1}, \mathtt{i_1} >$ has no common variables with the one substituted to $< \mathtt{mt_2}, \mathtt{i_2} >$ and $(\|\mathtt{t_1}\|, \ldots, \|\mathtt{t_n}\|)$ is a solution of $\mathtt{E}$.

Example 2.1 Let $\mathtt{X}$ be a groundlist and $\mathtt{Y}, \mathtt{Z}$ be both anylist whose heads have no common variables and whose tails are $\mathtt{X}$ itself. This statement can be expressed by the abstract substitution

[4] For more precise details about the use of systems of linear equations for automatically computing size relations the reader is referred to [5].

$$(\{\mathtt{X}/ < \mathtt{groundlist}, 1 >, \mathtt{Y}/[< \mathtt{any}, 1 > | < \mathtt{groundlist}, 1 >], \mathtt{Z}/[< \mathtt{any}, 2 > | < \mathtt{groundlist}, 1 >\},$$
$$\{(< \mathtt{any}, 1 >, < \mathtt{any}, 2 >)\}, \{\mathtt{Y} = \mathtt{X} + 1, \mathtt{Y} = \mathtt{Z}\}).$$

At each execution point, the analyzer computes (so-called) abstract sequences giving information about variables in the form of an abstract substitution, and also information about the number of solutions and the termination of the execution.

Definition 2.4 [abstract sequence]
An *abstract sequence* $\mathcal{B}$ is a pair $(\beta, < \mathtt{Min}, \mathtt{Max} >)$ where β is an abstract substitution, $\mathtt{Min} \in \mathbf{N} \cup \{\infty\}$ and $\mathtt{Max} \in \mathbf{N} \cup \{\infty, *\}$.
An abstract sequence $\mathcal{B}$ represents the set, $\gamma(\mathcal{B})$, of sequences $\mathtt{S}$ of concrete substitutions such that for all $\sigma \in \mathtt{S}$, $\sigma \in \gamma(\beta)$, $\mathtt{Min} \leq |\mathtt{S}|$ and if $\mathtt{Max} = *$ then $|\mathtt{S}|$ is finite, otherwise $|\mathtt{S}| \leq \mathtt{Max}$.

A behaviour for a procedure is a formalization of its specification (excluding the relation part which is formalized by the logic description).

Definition 2.5 [behaviour]
A *behaviour* for a procedure $\mathtt{p/n}$ is a 4-tuple of the form $(\mathtt{p}, [\mathtt{Y}_1, \ldots, \mathtt{Y}_n], \mathtt{E}, \mathtt{Prepost})$ where

1. $\mathtt{Y}_1, \ldots, \mathtt{Y}_n$ are distinct variables representing the formal parameters of the procedure $\mathtt{p/n}$;
2. $\mathtt{E}$ is a system of linear equations only using variables in $\mathtt{Y}_1, \ldots, \mathtt{Y}_n$;
3. $\mathtt{Prepost}$ is a set of 4-tuples of the form $(\alpha_{\mathtt{in}}, \alpha_{\mathtt{out}}, < \mathtt{Min}, \mathtt{Max} >, \mathtt{sizexp})$ such that $\alpha_{\mathtt{in}}$ and $\alpha_{\mathtt{out}}$ are mode-type assignments, $\mathtt{Min} \in \mathbf{N} \cup \{\infty\}$, $\mathtt{Max} \in \mathbf{N} \cup \{\infty, *\}$ and $\mathtt{sizexp}$ is a positive linear expression on the parameters $\mathtt{Y}_1, \ldots, \mathtt{Y}_n$.

Example 2.2 A behaviour for $\mathtt{efface/3}$ formalizing the specification depicted in figure 1 has the form

$$(\mathtt{efface}, [\mathtt{Y}_1, \mathtt{Y}_2, \mathtt{Y}_3], \mathtt{E} = \{\mathtt{Y}_2 = \mathtt{Y}_3 + 1\}, \mathtt{Prepost} = \{\mathtt{prepost}_1, \mathtt{prepost}_2\})$$

with

$$\mathtt{prepost}_1 = (\alpha_1^{\mathtt{in}}, \alpha_1^{\mathtt{out}}, < \mathtt{0}, \mathtt{1} >, \mathtt{Y}_2)$$
$$\mathtt{prepost}_2 = (\alpha_2^{\mathtt{in}}, \alpha_2^{\mathtt{out}}, < \mathtt{0}, * >, \mathtt{Y}_3)$$

wherein $\alpha_1^{\mathtt{in}} = \{\mathtt{Y}_1/\mathtt{ground}, \mathtt{Y}_2/\mathtt{ground}, \mathtt{Y}_3/\mathtt{any}\}$, $\alpha_2^{\mathtt{in}} = \{\mathtt{Y}_1/\mathtt{ground}, \mathtt{Y}_2/\mathtt{any}, \mathtt{Y}_3/\mathtt{ground}\}$ and $\alpha_1^{\mathtt{out}} = \alpha_2^{\mathtt{out}}$ $= \{\mathtt{Y}_1/\mathtt{ground}, \mathtt{Y}_2/\mathtt{groundlist}, \mathtt{Y}_3/\mathtt{groundlist}\}$.

3 The Analyzer

We first describe a clause analyzer which verifies the correctness of a clause. It receives as inputs a Prolog clause $\mathtt{CL}$ of the form $\mathtt{p}(\mathtt{X}_1, \ldots, \mathtt{X}_n) \leftarrow \mathtt{L}_1, \ldots, \mathtt{L}_f$ where $\mathtt{X}_1, \ldots, \mathtt{X}_n$ are distinct variables, a behaviour for each subprocedure (including $\mathtt{p/n}$) in $\mathtt{CL}$, an element $\mathtt{prepost}_{\mathtt{in}} = (\alpha_{\mathtt{in}}, \alpha_{\mathtt{out}}, < \mathtt{Min}, \mathtt{Max} >, \mathtt{sizexp})$ from the behaviour of $\mathtt{p/n}$ and an abstract substitution $\beta_{\mathtt{in}}$ on $\mathtt{X}_1, \ldots, \mathtt{X}_n$ respecting $\alpha_{\mathtt{in}}$ (i.e. if $\mathtt{mtt}_i$ is assigned to $\mathtt{X}_i$ by $\beta_{\mathtt{in}}$ and $\mathtt{mt}_i$ is assigned to $\mathtt{Y}_i$ by $\alpha_{\mathtt{in}}$ then $\gamma(\mathtt{mtt}_i) \subseteq \gamma(\mathtt{mt}_i)$ $(1 \leq i \leq n)$). It checks the following:

1. any subcall in the body of $\mathtt{CL}$ is called with arguments respecting at least one $\alpha_j^{\mathtt{in}}$ in its behaviour;
2. the arguments of $\mathtt{p}$ after the execution of the clause $\mathtt{CL}$ respect the size relation and $\alpha_{\mathtt{out}}$;
3. if $\mathtt{Max}$ is ∞ then at least one subcall in the body of $\mathtt{CL}$ can produce infinite solutions, otherwise, if $\mathtt{Max}$ is either a natural number or $*$ then the execution of $\mathtt{CL}$ terminates.

If all these properties are satisfied then a pair $< \mathtt{Min_{CL}}, \mathtt{Max_{CL}} >$ expressing information about the length of the sequence of answer substitutions and the termination of the execution of CL is returned. In particular, a set $\mathcal{B}_0, \mathcal{B}_1, \ldots, \mathcal{B}_\mathtt{f}$ of abstract sequences on the variables in CL and an abstract sequence $\mathcal{B}_\mathtt{out}$ on $\mathtt{X_1}, \ldots, \mathtt{X_n}$ are computed. The following main operations are required to analyze a clause.

Initial operation: it extends $\beta_\mathtt{in}$ to an abstract sequence $\mathcal{B}_0$ on all variables in CL.

Derivation operation: it computes $\mathcal{B}_\mathtt{i}$ from $\mathcal{B}_{\mathtt{i}-1}$ and $\mathtt{L_i}$ $(1 \leq \mathtt{i} \leq \mathtt{f})$.

Reduction operation: it computes $\mathcal{B}_\mathtt{out}$ from $\mathcal{B}_\mathtt{f}$ by restricting it to the variables $\mathtt{X_1}, \ldots, \mathtt{X_n}$.

Exit operation: it verifies whether $\mathcal{B}_\mathtt{out}$ respects the size relation and $\alpha_\mathtt{out}$.

Let us illustrate the derivation operation with an example[5]. Consider the second clause of $\mathtt{LP_2(efface/3)}$,

$$\mathtt{CL} \;=\; \mathtt{efface(X,L,LEff)} \leftarrow \mathtt{L = [H|T], LEff = [H|TEff], efface(X,T,TEff), not(H = X)}.$$

Let $\mathtt{prepost_{in}} = \mathtt{prepost_1}$ in the behaviour for efface/3 and $\mathcal{B}_2$ be the abstract sequence $(\beta_2, < 0, 1 >)$ holding after the execution of the second literal where

$\beta_2 = (\{\mathtt{X/} < \mathtt{ground}, 1 >, \mathtt{L/}[< \mathtt{ground}, 2 > | < \mathtt{groundlist}, 1 >], \mathtt{H/} < \mathtt{ground}, 2 >, \mathtt{T/} < \mathtt{groundlist}, 1 >,$
$\mathtt{LEff/}[< \mathtt{ground}, 2 > | < \mathtt{var}, 1 >], \mathtt{TEff/} < \mathtt{var}, 1 >\}, \mathtt{nosh_2} = \{\,\}, \mathtt{E_2} = \{\mathtt{L = T + 1}, \mathtt{LEff = TEff + 1}\})$.

$\mathcal{B}_3$ is obtained from $\mathcal{B}_2$ and the literal efface(X, T, TEff). Eight main steps are distinguished.

1. *mode-projection*: it returns a mode-type assignment α associating to each $\mathtt{Y_j}$ $(1 \leq \mathtt{j} \leq 3)$ a mode $\mathtt{mod_j}$ such that if $\mathtt{mtt_j}$ is the mode-typed term assigned to the j-th argument of efface(X, T, TEff) by β_2 then $\gamma(\mathtt{mtt_j}) \subseteq \gamma(\mathtt{mod_j})$; in the example, $\alpha = \{\mathtt{Y_1/ground}, \mathtt{Y_2/ground}, \mathtt{Y_3/var}\}$.

2. *selection*: it selects the maximal set S of $\mathtt{prepost_i}$ in the behaviour for efface/3 such that α "respects" $\alpha_\mathtt{i}^\mathtt{in}$ in $\mathtt{prepost_i}$. If $\mathtt{S} = \emptyset$ then the clause analyzer fails since it is not guaranteed that the literal is called with arguments respecting at least one **in** part of its directionalities. In our example, $\mathtt{S} = \{\mathtt{prepost_1}\}$.

3. *conjunction*: it returns an abstract sequence $\mathcal{B}_\mathtt{post}$ on the variables $\mathtt{Y_1}, \mathtt{Y_2}, \mathtt{Y_3}$ which approximates the sequence of substitutions obtained by executing $\mathtt{efface(Y_1, Y_2, Y_3)}$ called with arguments respecting at least one precondition in S. In the example, $\mathcal{B}_\mathtt{post}$ is $(\beta_\mathtt{post}, < 0, 1 >)$ where

$\beta_\mathtt{post} = (\{\mathtt{Y_1/} < \mathtt{ground}, 1 >, \mathtt{Y_2/} < \mathtt{groundlist}, 1 >, \mathtt{Y_3/} < \mathtt{groundlist}, 2 >\}, \{\,\}, \{\mathtt{Y_2 = Y_3 + 1}\})$.

4. *extraction*: it returns an abstract sequence $\mathcal{B}_\mathtt{ext}$ on the variables in efface(X, T, TEff) whose values are determined by the values of $\mathtt{Y_1}, \mathtt{Y_2}, \mathtt{Y_3}$ in $\mathcal{B}_\mathtt{post}$. In our example, $\mathcal{B}_\mathtt{ext}$ is $(\beta_\mathtt{ext}, < 0, 1 >)$ where

$\beta_\mathtt{ext} = (\{\mathtt{X/} < \mathtt{ground}, 1 >, \mathtt{T/} < \mathtt{groundlist}, 1 >, \mathtt{TEff/} < \mathtt{groundlist}, 2 >\}, \{\,\}, \{\mathtt{T = TEff + 1}\})$.

5. *extension*: it returns $\mathcal{B}_\mathtt{cl}$ obtained by extending $\mathcal{B}_\mathtt{ext}$ to all the other variables in CL, stating that the new variables are not instantiated and used nowhere else. In the example, $\mathcal{B}_\mathtt{cl}$ is $(\beta_\mathtt{cl}, < 0, 1 >)$ where

$\beta_\mathtt{cl} = (\{\mathtt{X/} < \mathtt{ground}, 1 > \; \mathtt{L/} < \mathtt{var}, 2 >, \mathtt{H/} < \mathtt{var}, 3 >, \mathtt{T/} < \mathtt{groundlist}, 1 >, \mathtt{TEff/} < \mathtt{groundlist}, 2 >,$
$\mathtt{LEff/} < \mathtt{var}, 4 >\}, \{(< \mathtt{var}, 2 >, < \mathtt{var}, 3 >), (< \mathtt{var}, 2 >, < \mathtt{var}, 4 >), (< \mathtt{var}, 3 >, < \mathtt{var}, 4 >)\}, \{\mathtt{T = TEff + 1}\})$.

6. *instantiation*: it computes $\mathcal{B}_\mathtt{inst}$ on all variables in CL expressing all possible instantiations of $\mathcal{B}_2$ affecting only the indexed mode-types used in $\mathtt{mtt_1}, \mathtt{mtt_2}, \mathtt{mtt_3}$ (see step 1). $\mathcal{B}_\mathtt{inst}$ is $(\beta_\mathtt{inst}, < 0, 1 >)$ where

[5] A detailed technical description of these operations is given in [9].

$\beta_{\texttt{inst}} = (\{\texttt{X}/<\texttt{ground},1>,\texttt{L}/[<\texttt{ground},2>|<\texttt{groundlist},1>],\texttt{H}/<\texttt{ground},2>,\texttt{T}/<\texttt{groundlist},1>,$
$\texttt{LEff}/[<\texttt{ground},2>|<\texttt{any},1>],\texttt{TEff}/<\texttt{any},1>\},\{\},\{\texttt{L}=\texttt{T}+1,\texttt{LEff}=\texttt{TEff}+1\})$

7. *refinement*: it computes $\mathcal{B}_3$ by making the conjunction of $\mathcal{B}_{\texttt{cl}}$ and $\beta_{\texttt{inst}}$. In the example, $\mathcal{B}_3$ is $(\beta_3, <\texttt{Min}_3,\texttt{Max}_3>)$ where $<\texttt{Min}_3,\texttt{Max}_3>=<0,1>$ and

$\beta_3 = (\{\texttt{X}/<\texttt{ground},1>,\texttt{L}/[<\texttt{ground},2>|<\texttt{groundlist},1>],\texttt{H}/<\texttt{ground},2>,\texttt{T}/<\texttt{groundlist},1>,$
$\texttt{LEff}/[<\texttt{ground},2>|<\texttt{groundlist},2>],\texttt{TEff}/<\texttt{groundlist},2>\},\ \{\},\ \{\texttt{L}=\texttt{T}+1,\texttt{LEff}=\texttt{TEff}+1,$
$\texttt{T}=\texttt{TEff}+1\})$

8. *completeness-termination-test*: first, the multiplicity $<\texttt{Min}_3,\texttt{Max}_3>$ is checked to respect the multiplicity $<0,1>$ in $\texttt{prepost}_{\texttt{in}}$. Moreover, since the sequence of answer substitutions is finite and we are in the case of a recursive call, it is checked whether the weight for the recursive call, i.e. the norm of the actual parameter T, is smaller than the weight for the initial call, i.e. the norm of the actual parameter L. This amounts to verify that $\texttt{E}_2 = \{\texttt{L}=\texttt{T}+1,\texttt{LEff}=\texttt{TEff}+1\}$ implies $\texttt{T}<\texttt{L}$[6]. In the example, this clearly holds.

The analyzer uses the clause analyzer as follows. It receives as inputs a Prolog procedure P defining a predicate p/n and a behaviour for each subprocedure in P. Let $\texttt{prepost}_{\texttt{in}} = (\alpha_{\texttt{in}},\alpha_{\texttt{out}},<\texttt{Min},\texttt{Max}>,\texttt{sizexp})$ from the behaviour of p/n and $\beta_{\texttt{in}}$ be an abstract substitution respecting $\alpha_{\texttt{in}}$. It checks the following:

1. for each clause CL of P the clause analyzer does not fail;
2. the sequence of answer substitutions for the whole procedure satisfies $<\texttt{Min},\texttt{Max}>$;
3. if Max is ∞ then at least one clause of P can produce an infinite number of solutions and if this is not the last one then the executions of all the following clauses in P are finitely failed.

When computing the sequence of answers for the whole procedure, input/output patterns are used to determine whether some clauses are mutually exclusive (see [1]).

4 The Synthesizer

The synthesizer receives the same inputs as the analyzer. First, for each clause of P all correct permutations of the literals in its body are computed and one correct permutation is selected according to some minimality criteria (e.g. minimum number of solutions). Then, a correct permutation of the selected clauses is reached (e.g. if only one clause is not proved to terminate then the sequence with such a clause at the end is returned).

The main operation of the synthesizer is realized by the find-perm function[7] which given a prefix of some permutation of literals in the body of a clause CL which is correct according to the clause analyzer and an abstract sequence $\mathcal{B}$ holding after the execution of such a prefix, returns the set of all correct permutations of literals in the body of CL which begin with the same prefix. The function find-perm uses the operations of the clause analyzer as follows:

- $\texttt{find-perm}(\texttt{prefix},\mathcal{B}) = \{(\texttt{prefix},<\texttt{Min}_{\texttt{out}},\texttt{Max}_{\texttt{out}}>)\}$ if prefix is a complete permutation of CL and $\mathcal{B}_{\texttt{out}} = (\beta_{\texttt{out}},<\texttt{Min}_{\texttt{out}},\texttt{Max}_{\texttt{out}}>)$ is obtained from $\mathcal{B}$ with the *reduction* operation of the clause analyzer and respects both the size relation and $\alpha_{\texttt{out}}$ in $\texttt{prepost}_{\texttt{in}}$;
- $\texttt{find-perm}(\texttt{prefix},\mathcal{B}) = \emptyset$ if prefix is a complete permutation of CL and $\mathcal{B}_{\texttt{out}}$ does not respect either the size relation or $\alpha_{\texttt{out}}$;

[6] This operation can be automated by using linear algebra techniques [7] to prove that the system $\texttt{E}_2 \cup \{\texttt{T} \geq \texttt{L}\}$ is unsolvable.
[7] It is inspired by the *permut* function defined in [4].

- `find-perm(prefix`, $\mathcal{B}$) = $\bigcup$ `find-perm((prefix,L)`, $\mathcal{B}'$) if `prefix` is not a complete permutation of `CL`, `L` is a literal in the body of `CL` but not in `prefix` and $\mathcal{B}'$ is the abstract sequence obtained from `L` and $\mathcal{B}$ with the *derivation operation* of the clause analyzer, i.e. such an operation does not fail and $\mathcal{B}'$ is the computed abstract sequence holding after the execution of `L` called with $\mathcal{B}$ which respects at least one precondition of the behaviour of `L`.

5 Conclusion and Future Work

This paper extends and refines previous tools presented in [2, 3, 4, 8] which are part of the FOLON environment. Future work will consider the actual implementation of the algorithms and their experimental evaluation on practical problems. Lower level transformations such as introduction of control information, e.g. cut, and partial evaluation will also be investigated.

References

[1] C. Braem, B. Le Charlier, S. Modart, P. Van Hentenryck. Cardinality Analysis of Prolog. In *Proc. Int'l Logic Programming Symposium, (ILPS'94), Ithaca, NY.* The MIT Press, Cambridge, Mass., 1994.

[2] P. De Boeck, J. Henrard, B. Le Charlier. FOLON an environment for Declarative construction of Logic programs. In *JCSLP'92 Post-Conference Workshop on Logic Programming Environment*, Washington, U.S.A., 1992.

[3] P. De Boeck and B. Le Charlier. Static Type Analysis of Prolog Procedures for Ensuring Correctness. In P. Deransart and J. Maluszyński, editors, *Proc. Second Int'l Symposium on Programming Language Implementation and Logic Programming, (PLILP'90)*, vol. 456 of *Lecture Notes in Computer Science.* Springer-Verlag, 1990.

[4] P. De Boeck and B. Le Charlier. Mechanical Transformation of Logic Definitions Augmented with Type Information into Prolog Procedures: Some Experiments. In *Proc. Int'l Workshop on Logic Program Synthesis and Transformation, (LOPSTR'93).* Springer Verlag, July 1993.

[5] D. De Schreye, K. Verschaetse, M. Bruynooghe. A Framework for analysing the termination of definite logic programs with respect to call patterns. In H. Tanaka, editor, *FGCS'92*, 1992.

[6] Y. Deville. *Logic Programming: Systematic Program Development.* Addison-Wesley, 1990.

[7] P. E. Gill, W. Murray, M. H. Wright. *Numerical Linear Algebra and Optimization*, vol. 1. Addison-Wesley, 1991.

[8] J. Henrard and B. Le Charlier. FOLON: An Environment for Declarative Construction of Logic Programs (extended abstract). In M. Bruynooghe and M. Wirsing, editors, *Proc. Fourth Int'l Workshop on Programming Language Implementation and Logic Programming (PLILP'92)*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1992.

[9] B. Le Charlier and S Rossi. Automatic Derivation of Totally Correct Prolog Procedures from Logic Descriptions. Technical Report No. RP-95-009, Institut d'Informatique, University of Namur, Belgium, 1995.

[10] B. Le Charlier, S. Rossi, P. Van Hentenryck. An Abstract Interpretation Framework which Accurately Handles Prolog Search-Rule and the Cut. In *Proc. Int'l Logic Programming Symposium, (ILPS'94), Ithaca, NY.* The MIT Press, Cambridge, Mass., 1994.