Compiling (for Validating) Explicit Specifications into Recursive Specifications in Linear Stratified Theories

Francisco José Galán and José Miguel Cañete

Dpto. Lenguages y Sistemas Informaticos. ETSI Informática
Av. Reina Mercedes s/n 41012 Sevilla, Spain
{galanm,canete}@lsi.us.es
Voice +34 954552773, Fax +34 954557139

Abstract. To write "correct" formal specifications from informal requirements constitutes a difficult problem. In any case, we are not able to prove formally that our formalizations (i.e. specifications) captures correctly requirements or intentions and then testing activities are needed for validating them. Automatic testing is only possible if specifications adopt particular forms. We propose linear stratified theories as a particular class of recursive theories to formalize requirements by means of executable specifications. A stratified theory determines a language to write explicit (non executable) specifications. A pattern-based transformational method is proposed to derive executable versions in a incremental way. Correctness and termination are key questions in our proposal.
Key words: expressiveness, specification, transformation, validation, executable specification.

1 Motivation

We agree with some authors [2,4] in considering that a formal specifications is not a goal by itself. A specification is an intermediate (and precise) representation between the program (formality) and its purpose (informality). Swartout and Balzer [11] establish an inevitable intertwining between the formal specification elaboration process and the programming process. This kind of reasonings leads Flener [4] to say that "*the problem with formal specifications is that there is no way to construct them so that we have a formal proof that they capture our intentions*". So an informal proof is needed somewhere. Therefore, writing a formal specification shifts the obligation of performing an informal verification between specifications and requirements.

A formal specification can have different purposes (validation, prototyping, synthesis, transformation, contracts). Programming is costly, then we think that a reasonable approach is to write all specifications of subprograms before writing the code. As the same manner programmers gain confidence in their programs by executing them, our intention is also to gain confidence in our specifications

by executing them. The class of models behind specifications and their forms determine our ability to transform them into executable descriptions. In the literature, a huge variety of transformation and synthesis mechanisms have been developed [3,4] to obtain programs and executable descriptions from specifications. From a synthesis point of view, the specification is supposed to be *the correct formalization of our requirements* and then a program must be found to compute it. Some authors claims that, for practical purposes, a compromise must be reached between expressivity of specification language and ease of mechanization [8] however, to be effective in practice, the behavior of many transformation and synthesis mechanisms must be conducted by humans [8,12]. For our validation purposes, the executability requirement is a *precondition*. Therefore, we must revisit the compromise between expressivity of specification language and ease of mechanization in order to ensure (a priori) the existence of executable descriptions for specifications. Following [1,6], we adopt contextual theories as underlying languages to write specifications. We will call them linear stratified theories and they present the following form:

Example 1 (Linear Stratified Theory).

Theory $\mathcal{S}$
Requirements: "Natural numbers and sequences of natural numbers."
Formalization:
Σ_0: Nat generated by $0, s$ $\quad$ $Seq(Nat)$ generated by $[], [|]$
$0 : () \rightarrow Nat$ $\quad$ $[] : () \rightarrow Seq(Nat)$
$s : (Nat) \rightarrow Nat$ $\quad$ $[|] : (Nat, Seq(Nat)) \rightarrow Seq(Nat)$
Layer 1:
Requirements for *eq*: "Identity between natural numbers."
Formalization for *eq*:
Σ_1: $eq : (Nat, Nat)$
$Ax_{1,1} : eq(0,0) \Leftrightarrow true$ $\quad$ $Ax_{1,3} : eq(0, s(y)) \Leftrightarrow false$
$Ax_{1,2} : eq(s(x), 0) \Leftrightarrow false$ $\quad$ $Ax_{1,4} : eq(s(x), s(y)) \Leftrightarrow eq(x, y)$
Layer 2:
Requirements for *nocc*: "Number of occurrences in a sequence."
Formalization for *nocc*:
Σ_2: $nocc : (Nat, Seq(Nat), Nat)$
$Ax_{2,1} : nocc(e, [], z) \Leftrightarrow eq(z, 0)$
$Ax_{2,2} : nocc(e, [x|y], 0) \Leftrightarrow \neg eq(x, e) \wedge nocc(e, y, 0)$
$Ax_{2,3} : nocc(e, [x|y], s(z)) \Leftrightarrow (eq(x, e) \wedge nocc(e, y, z)) \vee$
$(\neg eq(x, e) \wedge nocc(e, y, s(z)))$

They are organized by means of a sequence of layers. Each layer contains a set of relation symbols which are described by recursive definitions. In essence, this kind of formalizations represents pieces of software a specifier has validated by execution on a set of data. We consider that a specifier is interested in validating its requirements by means of ground data tests of the form, for example, $\{nocc(0, [], s(z)), nocc(0, [], 0), ...\}$. It is easy to see that an unfolding-based execution mechanism is *sufficient* to execute this kind of tests from theories such as

$\mathcal{S}$. Therefore, specifications in a stratified theory can be seen as "quasi-programs" because the truth value for any ground literal can be established by means of a finite number of unfolding steps.

However, the situation is different for the following class of specifications:

Example 2 (Explicit Specification).

Considering $\mathcal{S}$.
Requirements for *perm*: "Permutation between finite sequences."
Formalization for *perm*: Σ_{perm} : $perm$: $(Seq(Nat), Seq(Nat))$
$Spec_{perm}$: $perm(l, s) \Leftrightarrow \forall a, n(nocc(a, l, n) \Leftrightarrow nocc(a, s, n))$

Although this specification is constructed from "quasi-programs" the result is not a "quasi-program". The existence of infinite quantification for variables in the right-hand side of its definition (i.e. a and n) implies that we are not able to prove any ground literal of *perm* by means of a finite number of unfolding steps (we need some kind of inductive reasoning). In this way, we say that *perm* specification is an explicit or descriptive specification [6,7] (in opposition to recursive or operational specifications). Hence, our objective is to establish a method for transforming explicit specifications into recursive specifications. Once a recursive specification has been constructed for an explicit specification, the specifier can validate it by executing it on a set of data.

After studying some transformation and synthesis paradigms (constructive, transformational and inductive) [3,4], we are interested in transformational mechanisms [9,10]; a sequence of meaning-preserving transformation rules (e.g. unfolding, folding, universal instantiation, abstraction, predicate definition, etc.) is applied to a specification until another specification is obtained. The objective of applying transformations is to filter out a new version of the specification where recursion may be introduced by a folding step. However, among many others problems, deciding about when and how to define *automatically* a new predicate (i.e. recursive predicate) represents a difficult activity [5].

Our work is explained in the following manner. In section 2 we define formally linear stratified theories. Section 3 defines the set of rules for transforming explicit specifications in the language of a stratified theory. In section 4, we show that a transformation process is an incremental process guided by the information in the stratified theories. Correctness and termination are key questions in our proposal. Finally, we establish conclusions.

2 Linear Stratified Theories

A linear stratified theory is a theory organized by means of a sequence of layers. Each layer contains a set of relation symbols which are described by linear recursive specifications. Every definition is constructed using relation symbols located at the same layer or at the precedent layer (see Example 1). A linear stratified theory defines a language to specify new relations by means of explicit

(non-recursive) specifications. We consider these new relations as goals to be transformed or compiled (see Example 2).

The following definitions are needed to formalize the concept of linear stratified theory.

Definition 1 (Patterns). *We say that $t_{\mathcal{P}}$ is a* term pattern *for a term t if and only if $t_{\mathcal{P}}$ is obtained by replacing each variable occurrence in t by the symbol _. Let $t_{1\mathcal{P}}$ and $t_{2\mathcal{P}}$ be two term patterns (of the same sort), we say $t_{1\mathcal{P}} > t_{2\mathcal{P}}$ if and only if either $t_{1\mathcal{P}} = \_$ and $t_{2\mathcal{P}} = f(y_{1\mathcal{P}}, ..., y_{n\mathcal{P}})$ or $t_{1\mathcal{P}} = f(x_{1\mathcal{P}}, ..., x_{n\mathcal{P}})$ and $t_{2\mathcal{P}} = f(y_{1\mathcal{P}}, ..., y_{n\mathcal{P}})$ and there exists $s \subseteq \{1..n\}$ with $s \neq \emptyset$ such that $x_{i\mathcal{P}} > y_{i\mathcal{P}}$ for all $i \in s$ and $x_{j\mathcal{P}} = y_{j\mathcal{P}}$ for all $j \in \{1..n\} - s$.*

We say that $a_{\mathcal{P}}$ is an atom pattern *for an atom a if and only if $a_{\mathcal{P}}$ is obtained by replacing every term occurrence in a by its respective term pattern. Let $r(x_{1\mathcal{P}}, ..., x_{n\mathcal{P}})$ and $r(y_{1\mathcal{P}}, ..., y_{n\mathcal{P}})$ be two atom patterns, we say that $r(x_{1\mathcal{P}}, ..., x_{n\mathcal{P}}) > r(y_{1\mathcal{P}}, ..., y_{n\mathcal{P}})$ if and only if there exists $s \subseteq \{1..n\}$ with $s \neq \emptyset$ such that $x_{i\mathcal{P}} > y_{i\mathcal{P}}$ for all $i \in s$ and $x_{j\mathcal{P}} = y_{j\mathcal{P}}$ for all $j \in \{1..n\} - s$. We say that $l_{\mathcal{P}}$ is a* literal pattern *for an atom a if and only if $l_{\mathcal{P}} = a_{\mathcal{P}}$ or $l_{\mathcal{P}} = \neg a_{\mathcal{P}}$.*

We say that $F_{\mathcal{P}}$ is a formula pattern *for a formula F if and only if it is obtained by replacing every term occurrence in F by its respective term pattern.*

Definition 2 (Covering). *For every relation symbol r in $\mathcal{T}$, we construct an* atom covering *$\mathcal{A}(r)$ by calculating the set of all atom patterns induced from its axioms. The covering includes patterns for atoms in the rhs of its axioms ("upper patterns"), patterns for atoms in the lhs of its axioms ("bottom patterns") and patterns between upper patterns and bottom patterns ("intermediate patterns").*

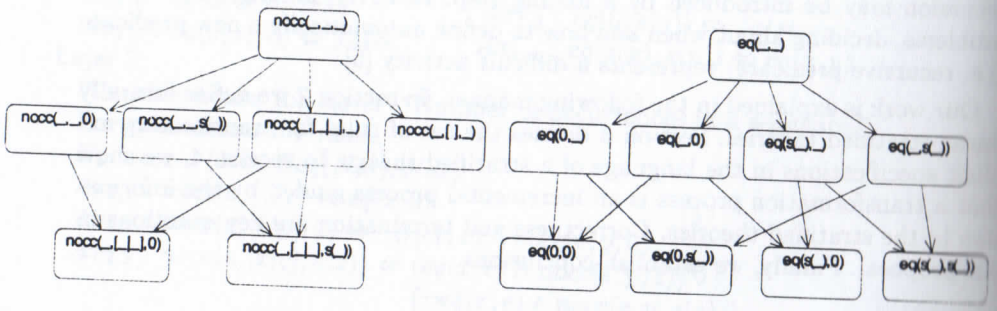

Fig. 1. $\mathcal{A}(nocc)$ and $\mathcal{A}(eq)$ in $\mathcal{S}$ (Example 1).

We display atom coverings by means of layered directed graphs. Links in the graph represent instances of the $>$ relation previously defined for atom patterns. In Fig. 1, we show the graphical representation of $\mathcal{A}(nocc)$ and $\mathcal{A}(eq)$ in $\mathcal{S}$ (Example 1). We say that an atom covering $\mathcal{A}(r)$ is *complete* if and only if every

bottom node presents instantiations in the same arguments. For example, in Fig. 1, $\mathcal{A}(eq)$ is complete because every bottom node instantiates the arguments 1 and 2. However $\mathcal{A}(nocc)$ is incomplete because the bottom node $nocc(\_,[],\_)$ instantiates only argument 2 and the rest of bottom nodes in $\mathcal{A}(nocc)$ instantiate arguments 2 and 3. A completion procedure for incomplete atom coverings can be done by selecting bottom nodes responsible of incompleteness and deriving new nodes, containing the respective instantiations and links. In the following, for every incomplete atom covering we consider implicitly its completion. In Fig. 2 we display the completion of $\mathcal{A}(nocc)$ signaling new nodes and new links.

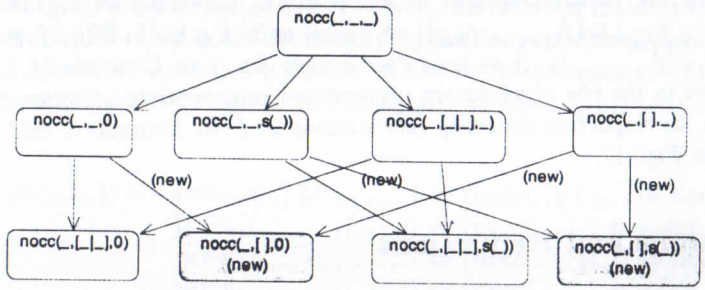

Fig. 2. Completion of $\mathcal{A}(nocc)$.

Definition 3 (Linear Stratified Theory). *A* k-stratified theory *$\mathcal{T}$ is a first-order theory of the form:*

$$\mathcal{T} = \langle \Sigma_0, Layer_1, ..., Layer_k \rangle \text{ with } Layer_i = \langle Req_i, \Sigma_i, Ax_i \rangle. \quad (1)$$

where Σ_0 represents the functional kernel *of $\mathcal{T}$ (sorts and function symbols) and $Layer_1, ..., Layer_k$ is the sequence of k layers of $\mathcal{T}$. Each $Layer_i$ is composed of the* layer requirements *Req_i containing requirements or intentions for relations,* layer signature *Σ_i containing declarations for relation symbols and the* layer axioms *Ax_i containing axioms for relation symbols in Σ_i (See example 1). Σ_i and Ax_i represent the formalization of Req_i. Each axiom in $\mathcal{T}$ is a closed formula of the form $\forall \bar{x}(r(\bar{x}) \Leftrightarrow R(\bar{x}))$ where $r(\bar{x})$ is called the* left-hand side (lhs) *of the axiom and $R(\bar{x})$ is called the* right-hand side (rhs) *of the axiom. The symbol r is called the* defined symbol *and $R(\bar{x})$ is called the* defining formula. *A specification for r in $\mathcal{T}$, denoted by $Spec_r$, is the set of (all) axioms in $\mathcal{T}$ with the same defined symbol r. A relation symbol is of level i in $\mathcal{T}$ if and only if it has been declared in layer i.*

The following restrictions must be considered for linear stratified theories:

1. *The level of the relation symbol of every positive atom occurring in $R(\bar{x})$ is either less than or equal to the level of r. The level of the relation symbol of every negative atom occurring in $R(\bar{x})$ is less than the level of r.*
2. *Every specification $Spec_r$ is total (all ground instances of r must be defined in $Spec_r$).*
3. *Non-ambiguity: the left-hand sides of any pair of axioms in $\mathcal{T}$ are not unifiable.*
4. *Linearity: the rhs of every axiom in $Spec_r$ in a layer i must be either of the form true or false (see axioms $Ax_{1,1}$, $Ax_{1,2}$ and $Ax_{1,3}$ in Example 1) or of the form $\bigwedge_{j=1..m} e_j$ or $(\bigwedge_{j=1..m} e_j) \wedge r$ where the level of the relation symbol of every literal occurring in e_j is $i-1$ (see axioms $Ax_{2,1}$ and $Ax_{2,2}$ respectively in Example 1) or of the form $\bigvee_{i=1..2^m}(\bigwedge_{j=1..m} e_{i,j}) \wedge r$ where the level of the relation symbol of every literal occurring in $e_{i,j}$ is $i-1$, $\models ((\bigwedge_{j=1..m} e_{f,j}) \wedge (\bigwedge_{j=1..m} e_{g,j})) \Leftrightarrow false$ with $f,g \in \{1..2^m\}$, $f \neq g$ and $\models (\bigvee_{i=1..2^m}(\bigwedge_{j=1..m} e_{i,j})) \Leftrightarrow true$ (see axiom $Ax_{2,3}$ in Example 1).*
5. *Every atom in the rhs of an axiom presents an intermediate or upper (atom) pattern in its respective covering (see axioms in $\mathcal{S}$ in Example 1 and atoms patterns in Fig. 1).*

3 Expansions and Reductions

The compilation process for an explicit specification (goal in the following) (see example 2) is done by means of a sequence of transformation steps. Each of these steps is composed by an expansion substep followed by a reduction substep. The *expansion substeps* is intended to unfold a goal in a guided manner producing new subgoals (i.e. to generate (unfold) formulas in the search space) and the *reduction substep* is intended to replace subgoals by recursive calls (i.e. to select (fold) formulas from the search space). Our intention is to organize the search space in order to answer the following questions:

1. *Can the search space be finite?*
2. *Can the search space be managed automatically?*

In order to answer these questions we have the following intuitions: *(a) the search space is determined by the language of the stratified theory. It is potentially infinite. We conjecture that the information supplied by goals are needed to narrow it in an effective manner and (b) the search space is potentially infinite but if we consider formula patterns for a goal then this space is finite.* In order to precise these intuitions we establish the following definitions:

Definition 4 (Goal). *We formalize a* goal *as the tuple $G = \langle \mathcal{T}, Req_r, \Sigma_r, Spec_r \rangle$ where $\mathcal{T}$ is a k-linear stratified theory, Req_r, is the* goal requirements *for r, Σ_r is the* goal signature *of a new relation r (i.e. not defined in $\mathcal{T}$) and $Spec_r$ is the* goal specification *of r. $Spec_r$ must be total and must contain axioms of the form $\forall \bar{x}(r(\bar{x}) \Leftrightarrow \forall \bar{z}(R(\bar{x},\bar{z})))$ where R is a quantifier-free formula in the language of $\mathcal{T}$ (see example 2).*

Definition 5 (Search Space). *For every goal $G = \langle \mathcal{T}, Req_r, \Sigma_r, Spec_r \rangle$ in a k-linear stratified theory $\mathcal{T}$, we define a search space $\Omega(rhs(Spec_r))$ by considering all possible variations by replacing atoms in $rhs(Spec_r)$ by upper and intermediate literal patterns of level $k, k-1, \ldots, 1$ and propositions true and false. Each variation represents an element of the search space. Every subscript in patterns in $\Omega(rhs(Spec_{perm}))$ encodes a kind of element. For example, $perm_1$ depends on the sequence of symbols $(nocc, nocc)$, $perm_2$ depends on the sequence of symbols $(nocc, eq)$, $perm_3$ depends on the sequence of symbols $(nocc, \neg eq)$, and so on.*

Example 3 (Search Space). Let G be the goal in Example 2. The cardinality of $\Omega(rhs(Spec_{perm}))$ is $VR(22,2) = 462$ (VR means variations with repetition, 22 is the number of literal patterns in $\mathcal{S}$ defined on *nocc* and *eq* symbols and 2 is the number of literals in $rhs(Spec_{perm})$). However, this does not means necessarily to select such amount of subgoals. It means only that $\Omega(rhs(Spec_{perm}))$ represents the search space for potential subgoals in the transformation of G. For space limitations and legibility reasons we display $\Omega(rhs(Spec_{perm}))$ partially:

$$
\begin{aligned}
\Omega(rhs(Spec_{perm})) = \{\ & perm_1(\_,[\vdash\vdash],\_,\_,\_,\_) \Leftrightarrow (nocc(\_,[\vdash\vdash],\_) \Leftrightarrow nocc(\_,\_,\_)),\\
& perm_1(\_,\_,\_,\_,[\vdash\vdash],\_) \Leftrightarrow (nocc(\_,\_,\_) \Leftrightarrow nocc(\_,[\vdash\vdash],\_)),\\
& perm_1(\_,\_,s(\_),\_,\_,\_) \Leftrightarrow (nocc(\_,\_,s(\_)) \Leftrightarrow nocc(\_,\_,\_)),\\
& perm_1(\_,\_,\_,\_,\_,s(\_)) \Leftrightarrow (nocc(\_,\_,\_) \Leftrightarrow nocc(\_,\_,s(\_))),\\
& \ldots\\
& perm_2(\_,[\vdash\vdash],\_,\_,\_) \Leftrightarrow (nocc(\_,[\vdash\vdash],\_) \Leftrightarrow eq(\_,\_)),\\
& perm_2(\_,\_,s(\_),\_,\_) \Leftrightarrow (nocc(\_,\_,s(\_)) \Leftrightarrow eq(\_,\_)),\\
& perm_3(\_,[\vdash\vdash],\_,\_,\_) \Leftrightarrow (nocc(\_,[\vdash\vdash],\_) \Leftrightarrow \neg eq(\_,\_)),\\
& perm_3(\_,\_,0,\_,\_) \Leftrightarrow (nocc(\_,\_,0) \Leftrightarrow \neg eq(\_,\_)),\\
& \ldots\\
& perm_5(\_,[\vdash\vdash],\_) \Leftrightarrow (nocc(\_,[\vdash\vdash],\_) \Leftrightarrow true),\\
& perm_5(\_,\_,s(\_)) \Leftrightarrow (nocc(\_,\_,s(\_)) \Leftrightarrow true),\\
& perm_6(\_,[\vdash\vdash],\_) \Leftrightarrow (nocc(\_,[\vdash\vdash],\_) \Leftrightarrow false),\\
& perm_6(\_,\_,\_) \Leftrightarrow (nocc(\_,\_,\_) \Leftrightarrow false),\\
& \ldots \}
\end{aligned}
$$

At this point, a mechanism to compare formulas is needed because we will need to decide if a formula is an instance of a pattern in $\Omega(rhs(Spec_{perm}))$. Our intentions are to supply "operational" definitions in order to justify the mechanization of our proposal.

Definition 6 (Parsing Formulas as Trees). *We call $tree(F)/tree(F_{\mathcal{P}})$ to the* binary tree representation *of a formula/formula pattern $F/F_{\mathcal{P}}$ where leaf nodes contain literals/literal patterns and internal (i.e. non-leaf) nodes contain binary logical connectives. We say that a formula F is* similar wrt connectives *to a formula E if and only if $tree(E)$, excluding leaf nodes, is a subtree of $tree(F)$. For example, in Fig. 3, F is similar wrt connectives to E but E is not similar wrt connectives to F.*

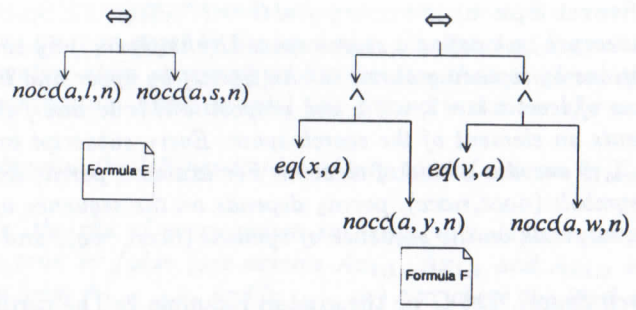

Fig. 3. F is similar to E

We say that a node in a tree is preterminal if and only if it has at least a leaf node as child node. Similarity wrt connectives induces a mapping f from preterminal nodes in $tree(E)$ to subtrees in $tree(F)$. For example, in Fig. 3 the preterminal node in $tree(E)$ containing the symbol $\Leftrightarrow$ is mapped to (the subtree) $tree(F)$.

Definition 7 (Similar Formula). *We say that a formula F is* similar *to a formula E if and only if F is similar wrt connectives to E and there exists a substitution θ such that*

1. *For each preterminal node $n \in tree(E)$ with two literals, e_{left} and e_{right}, as child nodes, there exist two literals l_{left}, on the left of the root of $f(n)$, and l_{right}, on the right of the root of $f(n)$ with $e_{left}\theta = l_{left}$ and $e_{right}\theta = l_{right}$.*
2. *For each preterminal node $n \in tree(E)$ with one literal, e_{left}/e_{right} as child node, there exists a literal l_{left}/l_{right}, on the left/right of the root of $f(n)$ with $(e_{left}/e_{right})\theta = l_{left}/l_{right}$.*

Example 4 (Similar Formula). In Fig. 3, F is similar to E because F is similar wrt connectives to E with $f(\Leftrightarrow) = tree(F)$ and there exists a substitution $\theta = \{(l,y),(s,w)\}$ such that for the preterminal node in E with $e_{left} = nocc(a,l,n)$ and $e_{right} = nocc(a,s,n)$ there exist two literals in $tree(F)$, $l_{left} = nocc(a,y,n)$ and $l_{right} = nocc(a,w,n)$, with $e_{left}\theta = l_{left}$ and $e_{right}\theta = l_{right}$.

Definition 8 (Similar Pattern). *We say that a formula pattern $F_{\mathcal{P}}$ is* similar *to a formula pattern $E_{\mathcal{P}}$ if and only if $F_{\mathcal{P}}$ is similar wrt connectives to $E_{\mathcal{P}}$ and*

1. *For each preterminal node $n \in tree(E)$ with two literal patterns, $e_{left_{\mathcal{P}}}$ and $e_{right_{\mathcal{P}}}$, as child nodes, there exist two literal patterns $l_{left_{\mathcal{P}}}$, on the left of the root of $f(n)$, and $l_{right_{\mathcal{P}}}$, on the right of the root of $f(n)$, where $e_{left_{\mathcal{P}}} = l_{left_{\mathcal{P}}}$ and $e_{right_{\mathcal{P}}} = l_{right_{\mathcal{P}}}$.*
2. *For each preterminal node $n \in tree(E)$ with one literal, $e_{left_{\mathcal{P}}}/e_{right_{\mathcal{P}}}$ as child node, there exists a literal $l_{left_{\mathcal{P}}}/l_{right_{\mathcal{P}}}$, on the left/right of the root of $f(n)$, where $e_{left_{\mathcal{P}}}/e_{right_{\mathcal{P}}} = l_{left_{\mathcal{P}}}/l_{right_{\mathcal{P}}}$.*

Once we have introduced the form of representing formulas, we define transformation rules to construct executable versions for explicit specifications. A transformation step is composed of a expansion step followed a reduction step. An expansion step is intended to decomposed the formula in order to find subformulas and a reduction step is intended to collapse such subformulas into "recursive calls". It is important to note that stratification represents a guide we consider in expansions and reduction activities. In order to realize expansion rule, we introduce instantiation and unfolding rules. In order to realize reduction rule, we introduce evaluation, simplification and folding rules.

Definition 9 (Variable Substituitions). *Let x be a variable of type (i.e. sort) s and let $\theta_1 = (x, f_1), ..., \theta_n = (x, f_n)$ be a set of substitutions for x. We say that $\{\theta_1, ..., \theta_n\}$ is a complete instantiation for x if and only if $\{f_1, ..., f_n\}$ generates s. For example, let $\theta_1 = (n, 0)$ and $\theta_2 = (n, s(z))$ be two substitutions for a variable n of type Nat then we say that $\{\theta_1, \theta_2\}$ is a complete instantiation for n because the set $\{0, s\}$ generates Nat in $\mathcal{S}$ (see Example 1).*

Definition 10 (Instantiation). *An* i-instantiation *$inst(F,i)$ of a formula F in a linear stratified theory $\mathcal{T}$ is the set of formulas $\{F\theta_1, ..., F\theta_k\}$ where $\{\theta_1, ..., \theta_k\}$ includes all possible complete instantiations for variables in F occurring in atoms of level i such that every atom pattern of level i occurring in $F\theta_{j_{\mathcal{P}}}$ with $j \in \{1..k\}$ must be included in its respective atom covering.*

Example 5 (i-Instantiation). Let $F = nocc(e, l, s(z)) \Leftrightarrow nocc(e, [\,], z)$ be a formula in the language of $\mathcal{S}$ (Example 1). Then

$$inst(F,2) = \{nocc(e,[\,],s(z)) \Leftrightarrow nocc(e,[\,],z),\\ nocc(e,[x\,|\,y],s(z)) \Leftrightarrow nocc(e,[\,],z)\}$$

Justification: $\theta_1 = (l,[\,]), \theta_2 = (l,[x\,|\,y]), \{\theta_1,\theta_2\} = \{(l,[\,]),(l,[x\,|\,y])\}$

$$\begin{array}{ll} nocc(e,[\,],s(z))_{\mathcal{P}} & = nocc(\_,[\,],s(\_)) \in \mathcal{A}(nocc)\\ nocc(e,[\,],z)_{\mathcal{P}} & = nocc(\_,[\,],\_) \in \mathcal{A}(nocc)\\ nocc(e,[x\,|\,y],s(z))_{\mathcal{P}} & = nocc(\_,[\_\,|\,\_],s(\_)) \in \mathcal{A}(nocc) \end{array}$$

Definition 11 (Unfolding Step). *Let a be an atom in F with $a_{\mathcal{P}}$ as a bottom pattern in its respective atom covering, let Ax be an axiom in $\mathcal{T}$ and let θ be a variable substitution with $lhs(Ax\theta) = a$. We say that $unf(F,a,Ax)$ is the* unfolding step *of a in F wrt Ax if and only if a is replaced in F by $rhs(Ax\theta)$.*

Example 6 (Unfolding Step). Let $nocc(e,[\,],z) \Leftrightarrow eq(z,0)$ be the axiom $Ax_{2,1}$ in $\mathcal{S}$. Let $F = nocc(a,[\,],s(k)) \Leftrightarrow eq(k,0)$ be a formula in the language of $\mathcal{S}$ (Example 1).

$$unf(F, nocc(a,[\,],s(k)), Ax_{2,1}) = eq(s(k),0) \Leftrightarrow eq(k,0)$$

Justification: $nocc(a,[],s(k))_{\mathcal{P}} = nocc(\_,[],s(\_))$ is a bottom element in (the completion of) $\mathcal{A}(nocc)$ and $Ax_{2,1}\theta = nocc(a,[],s(k)) \Leftrightarrow eq(s(k),0)$ with $\theta = \{(e,a),(z,s(k))\}$.

Definition 12 (Expansion). *We say that a set of formulas $exp(F,i)$ is the* i-expansion *of a formula F in a linear stratified theory $\mathcal{T}$ if and only if $exp(F,i)$ is constructed by applying all possible unfolding steps to every formula in $inst(F,i)$.*

Example 7 (i-Expansion). Let $F = nocc(e,l,z) \Leftrightarrow false$ be a formula in $\mathcal{S}$.

$$inst(F,2) = \{F\theta_1, F\theta_2, F\theta_3, F\theta_4\}$$

$$\theta_1 = \{(l,[]),(z,0))\} \quad \theta_2 = \{(l,[]),(z,s(k))\}$$
$$\theta_3 = \{(l,[x\,|\,y]),(z,0)\} \quad \theta_4 = \{(l,[x\,|\,y]),(z,s(k))\}$$

$$F\theta_1 = nocc(e,[],0) \Leftrightarrow false \qquad F\theta_2 = nocc(e,[],s(k)) \Leftrightarrow false$$
$$F\theta_3 = nocc(e,[x\,|\,y],0) \Leftrightarrow false \quad F\theta_4 = nocc(e,[x\,|\,y],s(k)) \Leftrightarrow false$$

$$\theta_5 = \{(z,0)\} \quad \theta_6 = \{(z,s(k))\} \quad \theta_7 = \{\}$$
$$\theta_8 = \{\} \qquad \theta_9 = \{\} \qquad \theta_{10} = \{\}$$

$$F_1 = unf(F\theta_1, nocc(e,[],0), Ax_{2,1}\theta_5) = eq(0,0) \Leftrightarrow false$$
$$F_2 = unf(F\theta_2, nocc(e,[],s(k)), Ax_{2,1}\theta_6) = eq(s(k),0) \Leftrightarrow false$$
$$F_3 = unf(F\theta_3, nocc(e,[x\,|\,y],0), Ax_{2,2}\theta_7) = (\neg eq(x,e) \wedge nocc(e,y,0)) \Leftrightarrow false$$
$$F_4 = unf(F\theta_4, nocc(e,[x\,|\,y],s(k)), Ax_{2,3}\theta_8) = (eq(x,e) \wedge nocc(e,y,z)) \vee (\neg eq(x,e) \wedge nocc(e,y,s(z))) \Leftrightarrow false$$
$$F_5 = unf(F_1, eq(0,0), Ax_{1,1}\theta_9) = true \Leftrightarrow false$$
$$F_6 = unf(F_2, eq(s(k),0), Ax_{1,2}\theta_{10}) = false \Leftrightarrow false$$

$$exp(F,2) = \{F_3, F_4, F_5, F_6\}$$

Property 1. Expansions preserve similarity wrt connectives. Instantiations and unfolding steps have only effect on atoms in a formula.

Proposition 1 (Expansion preserves correctness). *Let F be a formula in the language of a linear stratified theory $\mathcal{T}$, let $inst(F,i) = \{F\theta_1,...,F\theta_k\}$ be the i-instantiation considered in $exp(F,i)$. Then $(\bigvee_{j=1..k} unf(F\theta_j)) \Leftrightarrow F$.*

Proof. *By construction of θ_j in $inst(F,i)$, $(\bigvee_{j=1..k} F\theta_j)$ represents a partition for ground instances of F. Unfolding steps replace atoms by equivalent formulas. Then $(\bigvee_{j=1..k} unf(F\theta_j)) \Leftrightarrow F$.*

Proposition 2 (Expansion is an Internal Operation in $\Omega(F)$). *Let F be a formula of level k in the language of $\mathcal{T}$ with $F_{\mathcal{P}} = rhs(H_{\mathcal{P}})$ and $H_{\mathcal{P}} \in \Omega(F)$. For every $F_i \in exp(F,k)$, $F_{i\mathcal{P}}$ is similar to the rhs of some formula in $\Omega(F)$.* **Proof.** *By property 1, every $F_i \in exp(F,k)$ is similar wrt connectives to F, hence, every $F_{i\mathcal{P}}$ is similar wrt connectives to $F_{\mathcal{P}}$. Suppose by absurdum that there exists $F_{i\mathcal{P}}$ which is not similar to the rhs of some formula in $\Omega(F)$. Then, $F_{i\mathcal{P}}$ must include, at least, an atom pattern not included in its respective covering. However, due to the stratification of $\mathcal{T}$ and to our instantiation restrictions, unfolding steps can not produce this kind of patterns.*

Definition 13 (Evaluation). *Let $F(a_1,a_2,...,a_p,a_{p+1},...,a_n)$ be a formula in the language of $\mathcal{T}$ constructed from atoms $a_1,a_2,...,a_p,a_{p+1},...,a_n$. We say that $F_{eval}(a_1,a_2,...,a_p)$ is the* evaluation *of F wrt the set of atoms $\{a_1,a_2,...,a_p\}$ if and only if we consider the set of all possible evaluations for atoms $a_1,a_2,...,a_p$ in F in the following manner:*

$$\begin{aligned} F_{eval}(a_1,a_2,...,a_p) = \\ &(F(true,true,...,true,a_{p+1},...,a_n) \wedge a_1 \wedge a_2 \wedge ... \wedge a_p) \vee \\ &(F(false,true,...,true,a_{p+1},...,a_n) \wedge \neg a_1 \wedge a_2 \wedge ... \wedge a_p) \vee \\ &(F(true,false,...,true,a_{p+1},...,a_n) \wedge a_1 \wedge \neg a_2 \wedge ... \wedge a_p) \vee \\ &... \vee \\ &(F(false,false...,false,a_{p+1},...,a_n) \wedge \neg a_1 \wedge \neg a_2 \wedge ... \wedge \neg a_p) \end{aligned}$$

Example 8 (Evaluation). Let $a_1 = eq(x,e)$, $a_2 = eq(v,e)$, $a_3 = nocc(e,y,s(z))$, $a_4 = nocc(e,y,z)$, $a_5 = nocc(e,w,s(z))$, $a_6 = nocc(e,w,z)$ be the atoms in F.

$$F(a_1,...,a_6) = (\neg eq(x,e) \wedge nocc(e,y,s(z))) \vee (eq(x,e) \wedge nocc(e,y,z)) \Leftrightarrow (\neg eq(v,e) \wedge nocc(e,w,s(z))) \vee (eq(v,e) \wedge nocc(e,w,z))$$

Then

$$F(true,true,a_3,a_4,a_5,a_6) = (false \wedge nocc(e,y,s(z))) \vee (true \wedge nocc(e,y,z)) \Leftrightarrow (false \wedge nocc(e,w,s(z))) \vee (true \wedge nocc(e,w,z))$$

In a similar way, we calculate $F(false,true,a_3,a_4,a_5,a_6)$, $F(true,false,a_3,a_4,a_5,a_6)$ and $F(false,false,a_3,a_4,a_5,a_6)$.

$$\begin{aligned} F_{eval}(a_1,a_2) = &F(true,true,a_3,a_4,a_5,a_6) \wedge eq(x,e) \wedge eq(v,e) \vee \\ &F(false,true,a_3,a_4,a_5,a_6) \wedge \neg eq(x,e) \wedge eq(v,e) \vee \\ &F(true,false,a_3,a_4,a_5,a_6) \wedge eq(x,e) \wedge \neg eq(v,e) \vee \\ &F(false,false,a_3,a_4,a_5,a_6) \wedge \neg eq(x,e) \wedge \neg eq(v,e) \end{aligned}$$

Property 2. Let $F\theta$ be any ground instance of a formula F. Then $\models F\theta \Leftrightarrow F_{eval}(a_1,...,a_p)\theta$.

Definition 14 (Simplification). *In order to simplify specifications, we consider a set of rewrite rules of the form $l \to r$ in presence of negations and $false$ and $true$ propositions.*

(1) $\neg true \to false$, (2) $\neg false \to true$, (3) $true \vee F \to true$
(4) $false \vee F \to F$, (5) $true \wedge F \to F$, (6) $false \wedge F \to false$
(7) $false > F \to true$, (8) $true > F \to F$, (9) $false \Leftrightarrow F \to \neg F$
(10) $F > true \to true$, (11) $F > false \to \neg F$, (12) $true \Leftrightarrow F \to F$
(13) $\neg\neg F \to F$, (14) $\neg(F > G) \to F \wedge \neg G$, (15) $\neg(F \wedge G) \to \neg F \vee \neg G$
(16) $\neg(F \vee G) \to \neg F \wedge \neg G$, (17) $\neg(F \Leftrightarrow G) \to \neg(F > G) \vee \neg(G > F)$

A formula F is transformed into F_{rew} if and only if we apply repeatedly rewrite rules on F.

Example 9 (Simplification). From example 8, $F_{eval}(a_1, a_2)$ is simplified by applying rewrite rules (4), (5) and (6):

$$\begin{aligned} F_{eval}(a_1,a_2)_{rew} = {} & (nocc(e,y,z) \Leftrightarrow nocc(e,w,z)) && \wedge\, eq(x,e) \wedge eq(v,e) \vee \\ & (nocc(e,y,s(z)) \Leftrightarrow nocc(e,w,z)) && \wedge\, \neg eq(x,e) \wedge eq(v,e) \vee \\ & (nocc(e,y,z) \Leftrightarrow nocc(e,w,s(z))) && \wedge\, eq(x,e) \wedge \neg eq(v,e) \vee \\ & (nocc(e,y,s(z)) \Leftrightarrow nocc(e,w,s(z))) && \wedge\, \neg eq(x,e) \wedge \neg eq(v,e) \end{aligned}$$

Definition 15 (Pattern Instantiation). *Let S be a formula with $S_{\mathcal{P}}$ equal to the rhs of some pattern $F_{i\mathcal{P}} \in \Omega(F)$. We say that $I(F_{i\mathcal{P}}, S)$ is the* instantiation *of $F_{i\mathcal{P}}$ wrt S if and only if every symbol _ in $F_{i\mathcal{P}}$ is replaced by the respective variable in S.*

Example 10 (Pattern Instantiantion). Let $S = nocc(e,y,s(z)) \Leftrightarrow nocc(e,w,z)$ be a formula in $\mathcal{S}$ (Example 1) with $S_{\mathcal{P}}$ equal to the rhs of the pattern $F_{i\mathcal{P}} = perm_1(\_,\_,s(\_),\_,\_,\_) \Leftrightarrow (nocc(\_,\_,s(\_)) \Leftrightarrow nocc(\_,\_,\_)) \in \Omega(rhs(Spec_{perm}))$ (Example 3). Then

$$I(F_{i\mathcal{P}}, S) = perm_1(e,w,s(z),e,y,z) \Leftrightarrow (nocc(e,w,s(z)) \Leftrightarrow nocc(e,y,z))$$

Definition 16 (Encapsulation). *We say that a formula/formula pattern $F/F_{\mathcal{P}}$ is* completely encapsulated *in a layer i if and only if every atom/atom pattern in $F/F_{\mathcal{P}}$ is defined on a relation symbol of level i. We say that a formula/formula pattern $F/F_{\mathcal{P}}$ is* partially encapsulated *in a layer i (of a theory) if and only if some of its atoms/atom patterns are defined on relation symbols of level i and the rest of atoms/atom patterns are defined on relation symbols of lower level.*

Definition 17 (Folding Step). *We say that $fold(G, \Omega(F))$ is a* folding step *of a formula G wrt search space $\Omega(F)$ if and only if there exists a set of subformulas $\{S_i\}_{i=1..k}$ in G such that every $S_{i\mathcal{P}}$ is equal to the rhs of some pattern $F_{i\mathcal{P}} \in \Omega(F)$ and $fold(G, \Omega(F))$ is obtained by replacing every S_i with the lhs of $I(F_{i\mathcal{P}}, S_i)$.*

In a (automatic) folding step, to identify subformulas S_i and to replace these subformulas by equivalent atoms are crucial activities. If G is a formula of level k then we search for patterns $F_{i\mathcal{P}} \in \Omega(F)$ such that $rhs(F_{i\mathcal{P}})$ is a completely encapsulated pattern of level k. Literal patterns in $G_{\mathcal{P}}$ can be used to accelerate this search. If $G_{\mathcal{P}}$ is similar to some pattern $F_{i\mathcal{P}}$ then we fix in $G_{\mathcal{P}}$ those atom patterns which are responsible of similarity. The rest of atom patterns in $G_{\mathcal{P}}$ are selected to be evaluated. Once, we have decided the set of atoms in G to be evaluated, we evaluate G wrt to this set of atoms. This evaluation transforms G into G_{eval}. Due to the form of G_{eval}, it is easy to identify subformulas $\{S_i\}_{i=1..k}$ with $S_{i\mathcal{P}} = rhs(F_{i\mathcal{P}})$ and it is also easy to process the replacement in G_{eval} of every S_i by the lhs of $I(F_{i\mathcal{P}}, S_i)$. Once we have finished the search for totally encapsulated patterns of level k, we continue the search with partially encapsulated patterns of level k. In order to complete the search, we continue through the rest of lower layers (i.e. $k-1$,...,1) in a similar way.

Example 11 (Folding Step). Let $rhs(Spec_{perm})$ be a formula of level 2. Let $\Omega(rhs(Spec_{perm}))$ be our search space. Considering the formula F in example 8 as the formula to be folded. We search for patterns $F_{i\mathcal{P}} \in \Omega(rhs(Spec_{perm}))$ such that $rhs(F_{i\mathcal{P}})$ is a completely encapsulated pattern of level 2. We select the following patterns:

$$\begin{aligned} F_1 &= perm_1(\_,\_,\_,\_,\_,\_) && \Leftrightarrow (nocc(\_,\_,\_) \Leftrightarrow nocc(\_,\_,\_)), \\ F_2 &= perm_1(\_,\_,s(\_),\_,\_,\_) && \Leftrightarrow (nocc(\_,\_,s(\_)) \Leftrightarrow nocc(\_,\_,\_))), \\ F_3 &= perm_1(\_,\_,\_,\_,\_,s(\_)) && \Leftrightarrow (nocc(\_,\_,\_) \Leftrightarrow nocc(\_,\_,s(\_))), \\ F_4 &= perm_1(\_,\_,s(\_),\_,\_,s(\_)) && \Leftrightarrow (nocc(\_,\_,s(\_)) \Leftrightarrow nocc(\_,\_,s(\_))) \end{aligned}$$

$$\begin{aligned} F_{\mathcal{P}} &= (\neg eq(\_,\_) \wedge nocc(\_,\_,s(\_))) \vee eq(\_,\_) \wedge nocc(\_,\_,\_)) \\ &\Leftrightarrow \\ &(\neg eq(\_,\_) \wedge nocc(\_,\_,s(\_))) \vee (eq(\_,\_) \wedge nocc(\_,\_,\_)) \end{aligned}$$

Hence, the set of atoms $\{eq(x,e), eq(v,e)\}$ is selected to evaluate F (and then simplified).

$$\begin{aligned} & F_{eval}(eq(x,e), eq(v,e))_{rew} = \\ & (nocc(e,y,z) \Leftrightarrow nocc(e,w,z)) && \wedge\, eq(x,e) \wedge eq(v,e) \vee \\ & (nocc(e,y,s(z)) \Leftrightarrow nocc(e,w,z)) && \wedge\, \neg eq(x,e) \wedge eq(v,e) \vee \\ & (nocc(e,y,z) \Leftrightarrow nocc(e,w,s(z))) && \wedge\, eq(x,e) \wedge \neg eq(v,e) \vee \\ & (nocc(e,y,s(z)) \Leftrightarrow nocc(e,w,s(z))) && \wedge\, \neg eq(x,e) \wedge \neg eq(v,e) \end{aligned}$$

In $F_{eval}(eq(x,e), eq(v,e))_{rew}$, the subformulas

$S_1 = nocc(e,y,z) \Leftrightarrow nocc(e,w,z)$ $S_2 = nocc(e,y,s(z)) \Leftrightarrow nocc(e,w,z)$
$S_3 = nocc(e,y,z) \Leftrightarrow nocc(e,w,s(z))$ $S_4 = nocc(e,y,s(z)) \Leftrightarrow nocc(e,w,s(z))$

are selected to calculate $I(F_{i\mathcal{P}}, S_i)$ with $i = 1..4$. The lhs of selected patterns are

$perm_1(e,y,z,e,w,z)$ $perm_1(e,y,s(z),e,w,z)$
$perm_1(e,y,z,e,w,s(z))$ $perm_1(e,y,s(z),e,w,s(z))$

Once we have calculated lhs of $I(F_{i\mathcal{P}}, S_i)$ with $i = 1..4$ we replace S_i by lhs of $I(F_{i\mathcal{P}}, S_i)$

$$
\begin{aligned}
fold(F_{eval}(eq(x,e), eq(v,e))_{rew}, \Omega(rhs(Spec_{perm}))) = \\
\quad perm_1(e,y,z,e,w,z) \quad \wedge eq(x,e) \wedge eq(v,e) \quad \vee \\
\quad perm_1(e,y,s(z),e,w,z) \quad \wedge \neg eq(x,e) \wedge eq(v,e) \quad \vee \\
\quad perm_1(e,y,z,e,w,s(z)) \quad \wedge eq(x,e) \wedge \neg eq(v,e) \quad \vee \\
\quad perm_1(e,y,s(z),e,w,s(z)) \quad \wedge \neg eq(x,e) \wedge \neg eq(v,e)
\end{aligned}
$$

Definition 18 (Reduction). *We say that $red(\{G_1,...,G_k\}, \Omega(F))$ is the reduction of a set of formulas $\{G_1,...,G_k\}$ wrt the search space $\Omega(F)$ if and only if it is constructed by applying a folding step to every formula in $\{G_1,...,G_k\}$ wrt $\Omega(F)$.*

Example 12 (Reduction). Let $G = nocc(e,l,z) \Leftrightarrow false$ be the formula in example 7.

$$
\begin{aligned}
exp(G,2) = \{\ & (\neg eq(x,e) \wedge nocc(e,y,0)) \Leftrightarrow false\,, \\
& (eq(x,e) \wedge nocc(e,y,z)) \vee (\neg eq(x,e) \wedge nocc(e,y,s(z))) \Leftrightarrow false\,, \\
& true \Leftrightarrow false\,,\ \ false \Leftrightarrow false\ \}
\end{aligned}
$$

Then,

$$
\begin{aligned}
red(exp(G,2),\ \Omega(rhs(Spec_{perm}))) = \{ & (perm_6(e,y,0) \wedge \neg eq(x,e)) \vee eq(x,e)\,, \\
& (perm_6(e,y,z) \wedge eq(x,e)) \vee (perm_6(e,y,s(z)) \wedge \neg eq(x,e))\,, \\
& false\,,\ \ true\ \}
\end{aligned}
$$

Proposition 3 (Reduction preserves correctness). *Let $\{G_1,...,G_k\}$ be a set of formulas and let $red(\{G_1,...,G_k\}, \Omega(F))$ be the reduction of $\{G_1,...,G_k\}$ wrt $\Omega(F)$. Then $\models \forall(G_j \Leftrightarrow G_{j_{red}})$ with $G_{j_{red}}$ as the reduction of G_j.* **Proof.** *Every reduction is composed of evaluation, simplification and folding steps. By property 2, evaluation and simplification steps replace subformulas by equivalent subformulas and folding steps replace subformulas by equivalent atoms.*

As an initial step before applying transformations, we construct an equivalent goal for $G = \langle \mathcal{T}, Req_r, \Sigma_r, Spec_r \rangle$ in the following manner: we select a pattern $F_{\mathcal{P}}$ in the search space $\Omega(rhs(Spec_r))$ with $rhs(F_{\mathcal{P}}) = rhs(Spec_r)_{\mathcal{P}}$ and we instantiate this pattern obtaining $Spec_{nr} = I(F_{\mathcal{P}}, rhs(Spec_r))$. For example, $Spec_{perm_1}$, for G in Example 2, presents the form

$$perm_1(a,l,n,a,s,n) \Leftrightarrow (nocc(a,l,n) \Leftrightarrow nocc(a,s,n)). \qquad (2)$$

It is important to note that $Spec_{perm_1}$ has the same rhs than $Spec_{perm}$, hence the meaning of *perm* and $perm_1$ coincide in $\mathcal{S}$.

Definition 19 (Transformation Step). *We say that a* transformation step *for G is an expansion of the rhs of $Spec_{nr}$ followed by a reduction of the resulting formulas where every variable substitution is propagated to the lhs of $Spec_{nr}$.*

Example 13 (Transformation Step). The transformation step for (2) presents the form:

$$
\begin{aligned}
& perm_1(e,[\,],0,e,[\,],0) \quad \Leftrightarrow true \\
& perm_1(e,[\,],s(z),e,[\,],s(z)) \Leftrightarrow true \\
& perm_1(e,[\,],0,e,[v|w],0) \quad \Leftrightarrow (perm_5(e,w,0) \wedge \neg eq(v,e))
\end{aligned}
$$

$$
\begin{aligned}
perm_1(e,[\,],s(z),e,[v|w],s(z)) \Leftrightarrow\ & (perm_6(e,w,s(z)) \wedge \neg eq(v,e)) \vee \\
& (perm_6(e,w,z) \quad \wedge eq(v,e)) \\
perm_1(e,[x|y],0,e,[\,],0) \quad \Leftrightarrow\ & (perm_5(e,y,0) \quad \wedge \neg eq(x,e)) \\
perm_1(e,[x|y],s(z),e,[\,],s(z)) \Leftrightarrow\ & (perm_6(e,y,s(z)) \wedge \neg eq(x,e)) \vee \\
& (perm_6(e,y,z) \quad \wedge eq(x,e))
\end{aligned}
$$

$$
\begin{aligned}
perm_1(e,[x|y],0,e,[v|w],0) \Leftrightarrow\ & (perm_1(e,y,0,e,w,0) \wedge \neg eq(x,e) \wedge \neg eq(v,e)) \vee \\
& (perm_6(e,y,0) \quad \wedge \neg eq(x,e) \wedge eq(v,e)) \quad \vee \\
& (perm_6(e,w,0) \quad \wedge eq(x,e) \quad \wedge \neg eq(v,e)) \vee \\
& \qquad\qquad (eq(x,e) \quad \wedge eq(v,e))
\end{aligned}
$$

$$
\begin{aligned}
perm_1(e,[x|y],s(z),e,[v|w],s(z)) \Leftrightarrow\ & \\
& (perm_1(e,y,s(z),e,w,s(z)) \wedge \neg eq(x,e) \wedge \neg eq(v,e)) \vee \\
& (perm_1(e,y,s(z),e,w,z) \quad \wedge \neg eq(x,e) \wedge eq(v,e)) \quad \vee \\
& (perm_1(e,y,z,e,w,s(z)) \quad \wedge eq(x,e) \quad \wedge \neg eq(v,e)) \vee \\
& (perm_1(e,y,z,e,w,z) \quad \wedge eq(x,e) \quad \wedge eq(v,e))
\end{aligned}
$$

4 Compilation as an Incremental Process

After a transformation step for a goal of level k, we observe that the rhs of every formula is either of the form *true* or *false*, or of the form $nr \wedge (\bigwedge_{j=1..m} e_j)$ or $\bigvee_{i=1..2^m} nr_i \wedge (\bigwedge_{j=1..m} e_{i,j})$ where the level of the relation symbol of every literal occurring in e_j and $e_{i,j}$ is less than k and nr and nr_i are atoms defined on new relation symbols. Some of these atoms represent new subgoals, (e.g. $perm_1(e,y,z,e,w,s(z))$) and some others represent subgoals already solved (transformed), (e.g. $perm_1(e,y,z,e,w,z)$). In addition, the formulas $\bigwedge_{j=1..m} e_j$ and $\bigwedge_{j=1..m} e_{i,j}$ represent also new subgoals. In order to complete the compilation of the original goal, a transformation step for every new subgoal must be done.

Proposition 4. *The compilation of a goal is a finite process.* **Proof.** *After a transformation step, each atom nr or nr_i represents a subgoal whose pattern is included in the search space of the goal. The cardinality of a search space is finite because the number of (upper and intermediate) patterns in atom coverings and the number of atoms in a goal are finite. After a transformation step, we identify new subgoals of kind $\bigwedge_{j=1..m} e_j$ and $\bigwedge_{j=1..m} e_{i,j}$ (subgoals of level less than k). However, every new goal nr or nr_i produces a finite number of subgoals of kind*

$\bigwedge_{j=1..m} e_j$ *or* $\bigwedge_{j=1..m} e_{i,j}$. *Due to the finite number of layers in a stratified theory, every new goal of kind* $\bigwedge_{j=1..m} e_j$ *or* $\bigwedge_{j=1..m} e_{i,j}$ *can not produce an infinite sequence of lower subgoals.*

Example 14. The transformation step of the goal (2) produces 6 new subgoals of kind nr or nr_i:

$$\begin{array}{lll} perm_5(\_,\_,0) & perm_6(\_,\_,s(\_)) & perm_6(\_,\_,\_) \\ perm_1(\_,\_,s(\_),\_,\_,s(\_)) & perm_1(\_,\_,s(\_),\_,\_,\_) & perm_1(\_,\_,\_,\_,\_,s(\_)) \end{array}$$

and 6 new subgoals of kind $\bigwedge_{j=1..m} e_j$ or $\bigwedge_{j=1..m} e_{i,j}$:

$$\begin{array}{lll} \neg eq(\_,\_) & eq(\_,\_) & \neg eq(\_,\_) \wedge \neg eq(\_,\_) \\ \neg eq(\_,\_) \wedge eq(\_,\_) & eq(\_,\_) \wedge \neg eq(\_,\_) & eq(\_,\_) \wedge eq(\_,\_) \end{array}$$

At this moment, we have a first version of a prototype which implements these ideas. The complete compilation of the goal (2) produces 65 definitions. We observe that these results can be improved by incorporating some basis static analysis into our prototype.

5 Conclusions

We consider that a specification is an abstract description of a piece of software preserving its relevant aspects. We conjecture that to validate requirements from final programs is more expansive that to validate requirements from specifications. To clarify this point of view, we explore linear stratified theories as formal language to specify programs with a reasonable expressivity. These formalizations present a nice property: their specifications are executable and can be validated by an unfolding-based execution mechanism. From linear stratified theories, a class of specifications, called explicit specifications, is proposed to describe algorithmic properties of programs in a declarative form. These specifications are not executable and then a validation by execution is not possible. Hence, we consider these specifications as goals to be transformed into recursive specifications. We propose a method to do it incrementally which preserves correctness. Due to stratification properties of stratified theories, compiling a goal represents a finite process. Once a goal is transformed, it can be validated by executing it on a set of data.

References

[1] Basin, D. A.: Logic Frameworks for Logic Programs. Proceedings of the LOPSTR'94 and META'94. Springer-Verlag (1994).

[2] B. Le Charlier, P. Flener. Specifications are Necessarily Informal or: Some more Myths of Formal Methods. Journal of System and Software. Special Issue on Formal Methods Technology Transfer, 40(3) 275-296, (1998).

[3] Deville, Y., Lau K-K.: Logic Program Synthesis. J. Logic Programming **19,20** (1994) 321-350.

[4] Flener, P.: Logic Program Synthesis from Incomplete Information. Kluwer Academic Publishers (1995).

[5] Galán, F.J., Cañete, J. M.: Folding by Similarity. Joint Conference on Declarative Programming. APPIA-GULP-PRODE 2001. Evora (2001).

[6] Lau, K-K., Ornaghi, M.: On Specification Frameworks and Deductive Synthesis of Logic Programs. In Proceedings of LOPSTR'94 and META'94. Springer-Verlag (1994).

[7] Lau, K-K., Ornaghi, M.: Forms of Logic Specifications. In Proceedings of LOPSTR'96. Springer-Verlag (1994).

[8] K.-K. Lau and S.D. Prestwich. Top-down Synthesis of Recursive Logic Procedures from First-order Logic Specifications. In D.H.D. Warren and P. Szeredi, editors, Proceedings of the Seventh International Conference on Logic Programming, pages 667-684, MIT Press, 1990.

[9] Pettorossi, A., Proietti, M.: Transformation of Logic Programs: Foundations and Techniques. J. Logic Programming **19, 20** (1994) 261-320.

[10] Proietti, M., Pettorossi, A.: An Abstract Strategy for Transforming Logic Programs. Fundamenta Informaticae **18** (1993) 267-286

[11] W. R. Swartout and R. Balzer. On the Inevitable Intertwining of Specification and Implementation. Communications of the ACM 25:438-440, 1982.

[12] G. Wiggins. Synthesis and Transformation of Logic Programs in the Whelk Proof Development System. JICSLP'92. Washington DC 1992.