# Executing Assertions via Synthesized Logic Programs

F. J. Galán and J. M. Cañete

Dept. of Languages and Computer Systems. Faculty of Computer Science of Seville
Av. Reina Mercedes s/n 41012 Sevilla, Spain.
phone: (34) 95 455 27 73, fax: (34) 95 455 71 39
e-mail: galanm@lsi.us.es, canete@lsi.us.es

**Abstract.** Programming with assertions constitutes an effective tool to detect and correct programming errors. The ability of executing formal specifications is essential in order to test automatically an implementation against its assertions. However, formal assertions may describe recursive models which are difficult to identify so current assertion checkers limit, in a considerable way, the expressivity of the assertion language. In this paper, we are interested in showing how transformational synthesis can help to execute "expressive" assertions $r$ of the form $\forall \bar{x}(r(\bar{x}) \Leftrightarrow Q\bar{y}R(\bar{x},\bar{y}))$ where $Q$ is either an existential or universal quantifier and $R$ a quantifier free formula in the language of a formal theory $\mathcal{C}$ we call assertion context. This sort of theories is interesting because it presents a balance between expressiveness for writing assertions and existence of effective methods for compiling and executing them.

**Key words:** assertion, automatic testing, correctness, definite logic program, formal specification, synthesis, transformation.

## 1 Introduction

Experience has shown that writing assertions while programming is an effective way to detect and correct programming errors. As an added benefit, assertions serve to document programs, enhancing maintainability. Programming languages such as Eiffel [16], SPARK [2] and recent extensions to the Java programming language, such as iContract [12], JML [15] and Jass [6], allow to write assertions into the program code in the form of pre-post conditions and invariants. However, and due to mechanization problems, current assertion checkers avoid the use of expressive assertions. In this paper, we write assertions by assuming a set of free types [18] and a set of (partial) function *abs* (abstraction functions) to relate concrete data values in the implementation to abstract data values in the specification of a free type. We also assume that every free type is reachable by its implementation. To motivate the problem, we follow the following examples: Example 1 shows a specification of a free type, Example 2 shows an assertion to test if $n$ is the number of occurrences of an element $e$ in a (finite) sequence $l$ and

Example 3 shows a Java procedure annotated with a clause **assertion** to refer to assertion in Example 2 where free types $\mathcal{N}at$ and $\mathcal{S}eq$ have been implemented by Java types int and List respectively. A mapping between assertion variables and program variables is established by means of a clause **where**. By default, program variables supply values to assertion variables at locations where assertions are written, however, such a rule can be broken by using expressions such as old(elem) and old(list) to refer to initial (relative to the starting point of the procedure) values of elem and list respectively.

*Example 1 (Free type $\mathcal{N}at$. Specification).*

**Free Type** $\mathcal{N}at$
Data:
$0 : \rightarrow \mathcal{N}at \qquad s : \mathcal{N}at \rightarrow \mathcal{N}at$
**Assertion**
Identity: $id_{\mathcal{N}at}(x\colon \mathcal{N}at, y\colon \mathcal{N}at)$
Identity Specification:
$id_{\mathcal{N}at}(0,0) \Leftrightarrow true \qquad \forall y(id_{\mathcal{N}at}(0, s(y)) \Leftrightarrow false)$
$\forall x(id_{\mathcal{N}at}(s(x), 0) \Leftrightarrow false) \quad \forall x, y(id_{\mathcal{N}at}(s(x), s(y)) \Leftrightarrow id_{\mathcal{N}at}(x, y))$

*Example 2 (Assertion nocc).*

**Assertion**
Signature: $nocc(e\colon \mathcal{N}at, l\colon \mathcal{S}eq, n\colon \mathcal{N}at)$
Specification:
$\forall e, n(nocc(e, [\,], n) \Leftrightarrow id_{\mathcal{N}at}(n, 0))$
$\forall e, x, y(nocc(e, [x|y], 0) \Leftrightarrow \neg id_{\mathcal{N}at}(x, e) \wedge nocc(e, y, 0))$
$\forall e, x, y, z(nocc(e, [x|y], s(z)) \Leftrightarrow (id_{\mathcal{N}at}(x, e) \wedge nocc(e, y, z)) \vee$
$(\neg id_{\mathcal{N}at}(x, e) \wedge nocc(e, y, s(z))))$

*Example 3 (Java procedure annotated with a post-condition).*

```
int numberOfOccurrences(int elem, List list) {
    int result = 0; int aux;
    while (list.hasNext( )){
        aux = (int)list.next( );
        if (aux == elem) result++;
     }
    // assertion  nocc(e: Nat, l: Seq, n: Nat)
    // where e is abs(old(elem)), l is abs(old(list)), n is abs(result)
    return result;
}
```

It is important to note that assertions such as *nocc* are close to functional or logic programs. Due to this similarity, current assertion checkers are able to execute this kind of assertions. Example 4 shows a logic program which is able to execute ground atoms for *nocc*.

*Example 4 (Logic program for assertion nocc).*

```
idNat(0,0).
idNat(s(X),s(Y)):- idNat(X,Y).

nocc(E,[ ],N):- idNat(N,0).
nocc(E,[X|Y],0):- not(idNat(X,E)),nocc(E,Y,0).
nocc(E,[X|Y],s(Z)):- idNat(X,E),nocc(E,Y,Z).
nocc(E,[X|Y],s(Z)):- not(idNat(X,E)),nocc(E,Y,s(Z)).
```

Example 5 shows an assertion to test the *subset* relation. We will refer to this kind of assertions as *expressive assertions* due to the presence of infinite quantification in its definition (i.e. $\forall e(\ldots)$).

*Example 5 (Expressive assertion subset).*

**Assertion**
Signature : $subset(l\colon \mathcal{S}eq, s\colon \mathcal{S}eq)$
Specification :
$\forall l, s(subset(l, s) \Leftrightarrow \forall e(member(e, l) \Rightarrow member(e, s)))$

Inductive reasoning is needed to execute ground atoms for *subset*. For instance, Table 1 has been constructed by applying structural induction on $e$ in $subset([0, s(s(0))], [s(0), 0]) \Leftrightarrow \forall e(member(e, [0, s(s(0))]) \Rightarrow member(e, [s(0), 0]))$. After induction, we are able to know that $subset([0, s(s(0))], [s(0), 0])$ is $false$ because induction case (3) is equal to $false$.

| Index | Induction case | $subset([s(s(0))], [s(0)])$ |
|---|---|---|
| (1) | $e = 0$ | $true$ |
| (2) | $e = s(0)$ | $true$ |
| (3) | $e = s(s(0))$ | $false$ |
| (4) | $e = s(s(s(k)))$ | $true$ |

**Table 1.** Example of structural induction.

Current assertion checkers [16], [2], [6], [12], [15] do not implement this kind of inductive reasoning so they avoid the use of expressive assertions. This fact limits the expressivity of the assertion language and, therefore, the effectiveness of testing activities.

Our objective can be summarized in the following question:

*Is it possible to extend current assertion checkers to execute expressive assertions?* To answer this question we need to characterize a "theory of expressive assertions". For us, an expressive assertion $r$ is a relation which is defined by axioms of the form $\forall \bar{x}(r(\bar{x}) \Leftrightarrow Q\bar{y}R(\bar{x}, \bar{y}))$ where $Q$ is either an existential or universal quantifier and $R$ a quantifier free formula in the language of a formal theory $\mathcal{C}$ we call assertion contexts. Hence, to answer the question affirmatively, assertion checkers must be able to decide if *any ground atom* $r(\bar{t})$ *is a logical consequence of* $\mathcal{C}$.

*How can we do it?* Synthesis methods constitute an important aid to overcome this problem. Our intention is:

- If $Q = \exists$ then to synthesize a *totally correct* (definite) logic program $r_1$ of the form $\forall \bar{x}, \bar{y}(r_1(\bar{x}, \bar{y}) \Leftarrow P(\bar{x}, \bar{y}))$ from specification $\forall \bar{x}, \bar{y}(r_1(\bar{x}, \bar{y}) \Leftrightarrow R(\bar{x}, \bar{y}))$.
- If $Q = \forall$ then to synthesize a *totally correct* (definite) logic program $r_1^{neg}$ of the form $\forall \bar{x}, \bar{y}(r_1^{neg}(\bar{x}, \bar{y}) \Leftarrow P^{neg}(\bar{x}, \bar{y}))$ from specification $\forall \bar{x}, \bar{y}(r_1^{neg}(\bar{x}, \bar{y}) \Leftrightarrow \neg R(\bar{x}, \bar{y}))$.

As we will show, a mere inspection of logic programs $r_1$ and $r_1^{neg}$ will allow to decide about the existence of executions (i.e. finite derivations) for $\exists \bar{y}(r_1(\bar{t}, \bar{y}))$ and $\exists \bar{y}(r_1^{neg}(\bar{t}, \bar{y}))$ respectively.

*How can assertion checkers decide about logical consequences from executions of logic programs $r_1$ and $r_1^{neg}$?*

- If $Q = \exists$ then the execution of $\exists \bar{y}(r_1(\bar{t}, \bar{y}))$ will compute a set of substitutions $\{\theta_j\}$ (e.g. Table 1, rows (1), (2) and (4)). Resolution-based systems are refutation systems, thus:
  If $\{\theta_j\} = \emptyset$ then, by total correctness of $r_1$, $\mathcal{C} \models \neg r(\bar{t})$ else $\mathcal{C} \models r(\bar{t})$.
- If $Q = \forall$ then the execution of $\exists \bar{y}(r_1^{neg}(\bar{t}, \bar{y}))$ will compute a set of substitutions $\{\theta_j^{neg}\}$ (e.g. Table 1, row (3)), thus:
  If $\{\theta_j^{neg}\} = \emptyset$ then, by total correctness of $r_1^{neg}$, $\mathcal{C} \models r(\bar{t})$ else $\mathcal{C} \models \neg r(\bar{t})$.

To define our compilation method, we have studied some of the most relevant synthesis paradigms (constructive, transformational and inductive) [5, 7, 8]. In particular, we are interested in transformational mechanisms [4, 17, 13]. Transformation rules are easy to understand and simple to implement but as P. Flener says in [7]: *"A transformation usually involves a sequence of unfolding steps, then some rewriting, and finally a folding step. The eureka about when and how to define a new predicate is difficult to find automatically. It is also hard when to stop unfolding. There is a need for loop-detection techniques to avoid infinite synthesis through symmetric transformations"*. In order to overcome these problems, we develop transformations within assertion contexts [10]. These theories are interesting because they allow to (1) *structure* the search space for new predicates, (2) define a particular notion of similarity between formulas *for deciding when to introduce new predicates*. From this notion, a particular folding rule is defined to answer *how to introduce new predicates without human intervention* and (3) define a compilation method where *no symmetric transformations are possible*.

Our work is explained in the following manner. Section 2 introduces the notion of assertion context as a formal frame for interpreting and writing expressive assertions. Section 3 defines a transformation-based compilation method for expressive assertions. Finally, in section 4, we conclude.

## 2 Assertion Contexts

An expressive assertion is a relation whose definition includes axioms of the form $\forall \bar{x}(r(\bar{x}) \Leftrightarrow Q\bar{y}R(\bar{x}, \bar{y}))$ where $R(\bar{x}, \bar{y})$ is a formula written in the language of a

theory $\mathcal{C}$ containing free types and assertions we call *assertion context*. For each recursive assertion in $\mathcal{C}$, the specifier must supply the recursive parameters. As we will show, this information will be needed for ensuring the existence of executions for expressive assertions in an automatic way. Example 6 shows assertion context $\mathcal{S}$. Expressive assertion *subset* in Example 5 has been written in the language of $\mathcal{S}$.

*Example 6 (Assertion context $\mathcal{S}$).*

**Assertion Context** $\mathcal{S}$
**Global**.
Import Free Types $\mathcal{N}at, \mathcal{S}eq$
**Layer 1**.
**Assertion**
Signature: $nocc(e\colon \mathcal{N}at, l\colon \mathcal{S}eq, n\colon \mathcal{N}at)$, recursive parameters: $l$
Specification:
$\forall e, n(nocc(e, [\,], n) \Leftrightarrow id_{\mathcal{N}at}(n, 0))$
$\forall e, x, y(nocc(e, [x|y], 0) \Leftrightarrow \neg id_{\mathcal{N}at}(x, e) \wedge nocc(e, y, 0))$
$\forall e, x, y, z(nocc(e, [x|y], s(z)) \Leftrightarrow (id_{\mathcal{N}at}(x, e) \wedge nocc(e, y, z)) \vee$
$(\neg id_{\mathcal{N}at}(x, e) \wedge nocc(e, y, s(z))))$
**Assertion**
Signature: $member(e\colon \mathcal{N}at, l\colon \mathcal{S}eq)$, recursive parameters: $l$
Specification:
$\forall e(member(e, [\,]) \Leftrightarrow false)$
$\forall e, x, y(member(e, [x|y]) \Leftrightarrow (id_{\mathcal{N}at}(x, e) \vee member(e, y)))$

The following preliminary definitions are needed to formalize the notion of assertion context.

**Definition 1 (Patterns).** *We say that $t_{\mathcal{P}}$ is a term pattern for a term $t$ if $t_{\mathcal{P}}$ is obtained by replacing each variable occurrence in $t$ by the symbol $\_$. We say that $t_{1\mathcal{P}} > t_{2\mathcal{P}}$ if either $t_{1\mathcal{P}} = \_$ and $t_{2\mathcal{P}} = f(t_{2,1\mathcal{P}}, ..., t_{2,n\mathcal{P}})$ or $t_{1\mathcal{P}} = f(t_{1,1\mathcal{P}}, ..., t_{1,n\mathcal{P}})$ and $t_{2\mathcal{P}} = f(t_{2,1\mathcal{P}}, ..., t_{2,n\mathcal{P}})$ and there exists a non-empty subset $s \subseteq \{1..n\}$ such that $t_{1,i\mathcal{P}} > t_{2,i\mathcal{P}}$ for every $i \in s$ and $t_{1,j\mathcal{P}} = t_{2,j\mathcal{P}}$ for every $j \in (\{1..n\} - s)$. For instance, $s(x)_{\mathcal{P}} = s(\_)$ and $s(\_) > s(0)$.*

*We say that $r(\bar{t})_{\mathcal{P}}$ is an atom pattern for an atom $r(\bar{t})$ if $r(\bar{t})_{\mathcal{P}}$ is obtained by replacing every term occurrence in $r(\bar{t})$ by its respective term pattern. Let $r(t_{1,1\mathcal{P}}, ..., t_{1,n\mathcal{P}})$ and $r(t_{2,1\mathcal{P}}, ..., t_{2,n\mathcal{P}})$ be two atom patterns, we say that $r(t_{1,1\mathcal{P}}$,- $..., t_{1,n\mathcal{P}}) > r(t_{2,1\mathcal{P}}, ..., t_{2,n\mathcal{P}})$ if there exists a non-empty set $s \subseteq \{1..n\}$ such that $t_{1,i\mathcal{P}} > t_{2,i\mathcal{P}}$ for every $i \in s$ and $t_{1,j\mathcal{P}} = t_{2,j\mathcal{P}}$ for every $j \in (\{1..n\} - s)$. For instance, $nocc(a, [x|y], n)_{\mathcal{P}} = nocc(\_, [\_\,|\_], \_)$ and $nocc(\_, \_, \_) > nocc(\_, [\_\,|\_], \_)$.*

*We say that $l_{\mathcal{P}}$ is a literal pattern for $r(\bar{t})$ if either $l_{\mathcal{P}} = r(\bar{t})_{\mathcal{P}}$ or $l_{\mathcal{P}} = \neg r(\bar{t})_{\mathcal{P}}$. We say that $F_{\mathcal{P}}$ is a formula pattern for a quantifier-free formula $F$ if and only if it is obtained by replacing every atom in $F$ by its respective literal pattern.*

Every axiom in an assertion context is a universally closed formula of the form $\forall(r(\bar{x}) \Leftrightarrow R(\bar{z}))$ where $r(\bar{x})$ is called the left-hand side (lhs) of the axiom and $R(\bar{z})$ is a quantifier-free formula composed of literals and binary logical connectives we call right-hand side (rhs) of the axiom.

**Definition 2 (Atom Covering).** *An atom covering for an assertion $r$, $\mathcal{A}(r)$, is the set of atom patterns defined on symbol $r$ which is induced from its axioms. $\mathcal{A}(r)$ includes patterns from the right-hand sides of its axioms (called "upper patterns"), patterns from the left-hand sides of its axioms (called "lower patterns") and "intermediate patterns" which are induced from upper and lower patterns by means of relation $>$.*

We display atom coverings by means of directed graphs[1] where atom patterns are nodes and directed links are instances of relation $>$. For instance, Fig. 1 shows $\mathcal{A}(nocc)$ in $\mathcal{S}$. We say that an atom covering $\mathcal{A}(r)$ is *complete* if and only if every lower pattern instantiates the same set of parameter positions. As we will show, complete coverings are needed for constructing formula instantiations. For instance, $\mathcal{A}(member)$ is complete (i. e. every lower pattern instantiates parameter $l$) and $\mathcal{A}(nocc)$ is incomplete. (i.e. lower pattern in first axiom instantiates parameter $l$ and the rest of lower patterns instantiate parameters $l$ and $n$). A completion procedure for incomplete coverings is done by deriving new nodes from lower patterns responsible of incompleteness. In Fig. 1 we show the completion of $\mathcal{A}(nocc)$ by highlighting new nodes and links. In the following, for every incomplete covering we consider implicitly its completion.

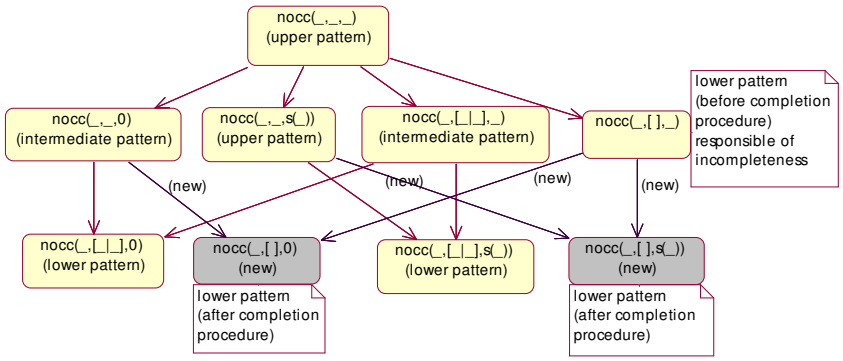


**Fig. 1.** Completion of $\mathcal{A}(nocc)$.

**Definition 3 (Assertion Context).** *An assertion context $\mathcal{C}$ is a first-order theory $\mathcal{C} = \langle Global, Layer_1, ..., Layer_k \rangle$ where Global section is intended to model data structures by means of free types and $Layer_1, ..., Layer_k$ sections are intended to model assertions. Propositions $true$ and $false$ complete the language of $\mathcal{C}$.*

[1] These can be either connected or disconnected graphs.

Every relation symbol $r$ defined in $Layer_i$ is a symbol of *level* $i$. Every identity symbol in the *Global* section is a symbol of level 0. To unify, propositions $true$ and $false$ are symbols of level 0. The level of a formula is equal to the greatest level induced from its relation symbols.

**Definition 4 (Free Type).** *A free type* $\mathcal{T} = \langle Data, \langle \Sigma_{id_\mathcal{T}}, Spec_{id_\mathcal{T}} \rangle \rangle$ *where Data is a set of constants and functions to model data in a recursive manner and* $\langle \Sigma_{id_\mathcal{T}}, Spec_{id_\mathcal{T}} \rangle$ *is a particular assertion which defines an identity relation for data* $\mathcal{T}$*.*

**Definition 5 (Assertion).** *Every assertion* $\langle \Sigma_r, Spec_r \rangle$ *is defined by signature* $\Sigma_r$ *and specification* $Spec_r$ *where* $\Sigma_r$ *includes information about which parameters are recursive in* $r$ *and* $Spec_r$ *is a set of axioms of the form* $\forall(r(\bar{x}) \Leftrightarrow R(\bar{z}))$ *for defining* $r$ *in* $\mathcal{C}$ *with the following restrictions:*

- *Restriction 1 (Layers): If* $r$ *is a symbol of level* $i > 0$ *then* $Spec_r$ *includes at least an atom of level* $i-1$*. Every positive atom occurring in* $R(\bar{z})$ *is of level* $i$ *(only recursive atoms, if they exists),* $i-1$ *or* $0$*. Every negative atom occurring in* $R(\bar{z})$ *is of level* $i-1$ *or* $0$*.*
- *Restriction 2 (Totality): Every ground instance of* $r$ *must be defined in* $Spec_r$*.*
- *Restriction 3 (No ambiguity): The left-hand sides of any pair of axioms in* $Spec_r$ *are not unifiable.*
- *Restriction 4 (No internal variables):* $\bar{z} \subseteq \bar{x}$ *in* $\forall(r(\bar{x}) \Leftrightarrow R(\bar{z}))$*.*
- *Restriction 5 (Well-formed): Every atom occurring in* $R(\bar{z})$ *induces either intermediate or upper atom pattern in its respective covering.*
- *Restriction 6 (Well-founded): If* $r$ *is recursive then* $Spec_r$ *must be well-founded wrt recursive parameters.*

The following definitions are needed to formalize the consistency of assertion contexts.

**Definition 6 (Unfolding Step).** *Let* $r(\bar{y})$ *be an atom in a universally closed formula* $\forall(F)$ *and* $Ax = \forall(r(\bar{x}) \Leftrightarrow R(\bar{z}))$ *an axiom in* $\mathcal{C}$ *with* $r(\bar{x})\theta = r(\bar{y})$*. We say that* $unf(\forall(F), r(\bar{y}), Ax)$ *is the unfolding step of* $r(\bar{y})$ *in* $\forall(F)$ *wrt* $Ax$ *if and only if* $r(\bar{y})$ *is replaced in* $\forall(F)$ *by* $R(\bar{z})\theta$*.*

**Definition 7 (Simplification Rules).** *In order to simplify formulas in presence of propositions* $true$ *and* $false$*, we consider the following set of rewrite rules where* $H$ *is a formula.*

| | | |
|---|---|---|
| (1) $\neg true \rightarrow false$ | (2) $\neg false \rightarrow true$ | (3) $true \vee H \rightarrow true$ |
| (4) $false \vee H \rightarrow H$ | (5) $true \wedge H \rightarrow H$ | (6) $false \wedge H \rightarrow false$ |
| (7) $false \Rightarrow H \rightarrow true$ | (8) $true \Rightarrow H \rightarrow H$ | (9) $false \Leftrightarrow H \rightarrow \neg H$ |
| (10) $H \Rightarrow true \rightarrow true$ | (11) $H \Rightarrow false \rightarrow \neg H$ | (12) $true \Leftrightarrow H \rightarrow H$ |
| (13) $\forall(true) \rightarrow true$ | (14) $\forall(false) \rightarrow false$ | |

**Definition 8 (Execution).** *We call execution of a ground atom* $r(\bar{t})$ *in* $\mathcal{C}$ *to every terminating derivation for* $r(\bar{t})$ *which is constructed by means of unfolding steps wrt axioms in* $\mathcal{C}$ *and simplifications (Def. 7). We will write* $\mathcal{C} \vdash r(\bar{t})$ *if the execution of* $r(\bar{t})$ *in* $\mathcal{C}$ *ends in* $true$ *and* $\mathcal{C} \vdash \neg r(\bar{t})$ *if ends in* $false$*.*

**Theorem 1 (Ground Decidability).** *For every ground atom $r(\bar{t})$ in $\mathcal{C}$ either $\mathcal{C} \vdash r(\bar{t})$ or $\mathcal{C} \vdash \neg r(\bar{t})$.* (A proof of this theorem can be found in Appendix).

From Theorem 1, we formalize the semantics of assertion contexts. Our proposal is borrowed from previous results in the field of deductive synthesis [3], [13], [14], [1], [11].

**Definition 9 (Consistency).** *A model for $\mathcal{C}$ is defined in the following terms:*

$$\mathcal{C} \models r(\bar{t}) \;\; \textit{iff} \;\; \mathcal{C} \vdash r(\bar{t}) \;\; \textit{and} \;\; \mathcal{C} \models \neg r(\bar{t}) \;\; \textit{iff} \;\; \mathcal{C} \vdash \neg r(\bar{t})$$

*for every ground atom $r(\bar{t})$ in $\mathcal{C}$.*

Once we have formalized the notion of assertion context, we formalize the notion of expressive assertion. Example 5 shows an expressive assertion in $\mathcal{S}$.

**Definition 10 (Expressive Assertion).** *We say that $\langle \Sigma_r, Spec_r \rangle$ is an expressive assertion in $\mathcal{C}$ if and only if $r$ is a new symbol not defined in $\mathcal{C}$ and $Spec_r$ is total, non-ambiguous and, at least, one of its axioms is of the form $\forall \bar{x}(r(\bar{x}) \Leftrightarrow Q\,\bar{y} R(\bar{x}, \bar{y}))$ where $Q$ is either an existential or universal quantifier and $R(\bar{x}, \bar{y})$ is a well-formed and quantifier-free formula in the language of $\mathcal{C}$.*

## 3 Compilation Method

The compilation process for an expressive assertion $r$ can be seen as a sequential activity where, at each step, an axiom of the form $\forall \bar{x}(r(\bar{x}) \Leftrightarrow Q\bar{y}R(\bar{x}, \bar{y}))$ in $Spec_r$ is compiled. If $Q = \exists$ then we synthesize a new *totally correct* recursive assertion $r_1$ from $\forall \bar{x}, \bar{y}(r_1(\bar{x}, \bar{y}) \Leftrightarrow R(\bar{x}, \bar{y}))$ and if $Q = \forall$ then we synthesize a new *totally correct* recursive assertion $r_1^{neg}$ from $\forall \bar{x}, \bar{y}(r_1^{neg}(\bar{x}, \bar{y}) \Leftrightarrow \neg R(\bar{x}, \bar{y}))$. From these recursive assertions, logic programs are derived by a mere translation method. Example 7 shows the starting point for compiling the unique axiom of *subset* in Example 5.

*Example 7 (Compiling subset. Starting point[2]).*

**Assertion**
Signature : $subset_1^{neg}(a\colon \mathcal{N}at, l\colon \mathcal{S}eq, b\colon \mathcal{N}at, s\colon \mathcal{S}eq)$
Specification :
$\forall e, l, s(subset_1^{neg}(e, l, e, s) \Leftrightarrow (member(e, l) \wedge \neg member(e, s)))$

The compilation of an axiom is done by a finite sequence of meaning-preserving transformation steps. Each transformation step is composed of an expansion phase followed by a reduction phase. Expansion phase is intended to decompose the original formula into a set of formulas and reduction phase is intended to replace sub-formulas by new predicates. As we will show, the set of new predicates ("recursive predicates") is computable.

[2] To normalize the form of axioms (Def. 12), the compilation of $subset_1^{neg}$ starts from $\forall(subset_1^{neg}(e, l, e, s) \Leftrightarrow (member(e, l) \wedge \neg member(e, s)))$ which is an equivalent formula to $\forall(subset_1^{neg}(e, l, e, s) \Leftrightarrow \neg(member(e, l) \Rightarrow member(e, s)))$.

### 3.1 Expansion Phase

Expansion phase decomposes a formula $F$ into a set of formulas by means of instantiations and unfolding steps. Our intention is to decomposed a formula in a guided manner by using a particular rule, we call $i$-instantiation: If $F$ is a formula of level $i$ then only atoms of level $i$ are selected to be instantiated. To implement $i$-instantiations, we will use atom coverings. If $r(\bar{y})$ is a selected atom to be instantiated in $F$ and $r(\bar{y})_{\mathcal{P}}$ dominates a subtree in $\mathcal{A}(r)$ then lower patterns in such a subtree induces a set of substitutions for variables in $r(\bar{y})$. Such sets of substitutions will be the basis to construct $i$-instantiations. In Fig. 2 we show two examples.

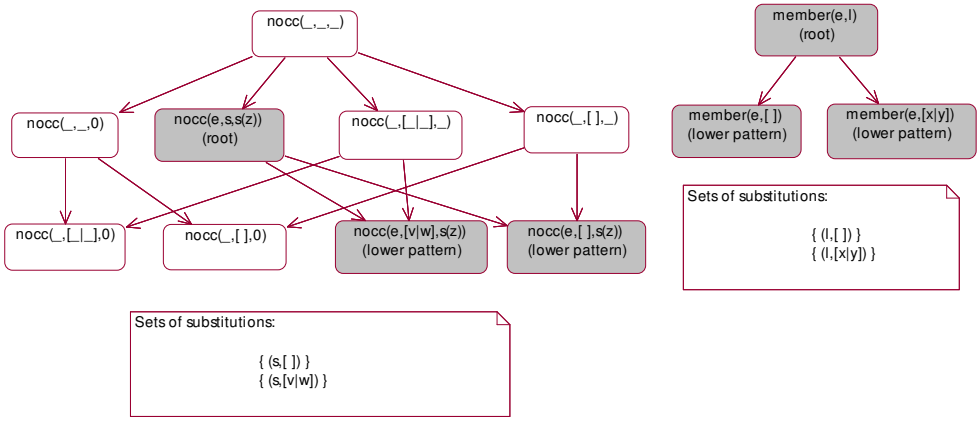


**Fig. 2.** Substitutions for $nocc(a, s, s(z))$ and $member(e, l)$ in their respective coverings.

In the following definitions, we consider that $F$ is a formula of the form $\forall(r_i(\bar{x}) \Leftrightarrow R(\bar{x}))$ with $R(\bar{x})$ a quantifier-free formula written in (the language of an assertion context) $\mathcal{C}$ and $r_i$ a symbol not defined in $\mathcal{C}$.

**Definition 11 ($i$-Instantiation).** *We say that* $inst(F, i, r(\bar{y})) = \{\forall(r_i(\bar{x})\theta_1 \Leftrightarrow R(\bar{x})\theta_1), ..., \forall(r_i(\bar{x})\theta_j \Leftrightarrow R(\bar{x})\theta_j)\}$ *is the $i$-instantiation of $r(\bar{y})$ in $F$ if and only if*

1. *$r(\bar{y})$ is an atom in $R(\bar{x})$ of level $i$ whose pattern $r(\bar{y})_{\mathcal{P}}$ dominates a subtree in $\mathcal{A}(r)$ with lower patterns $\{r(\bar{y}_1)_{\mathcal{P}}, ..., r(\bar{y}_j)_{\mathcal{P}}\}$.*
2. *$\{\theta_1, ..., \theta_j\}$ is the set of substitutions such that $(r(\bar{y})\theta_k)_{\mathcal{P}} = r(\bar{y}_k)_{\mathcal{P}}$ with $k = 1..j$.*
3. *Every atom in $R(\bar{x})\theta_k$ has a pattern which is included in its respective covering.*

*Example 8 ($inst(F, 1, member(e, l))$ for $F$ equal to the axiom of $subset_1^{neg}$ in Example 7).*

$\forall\,(subset_1^{neg}(e,[\,],e,s) \Leftrightarrow (member(e,[\,]) \wedge \neg member(e,s))) \qquad \theta_1 = \{(l,[\,])\}$
$\forall\,(subset_1^{neg}(e,[x|y],e,s) \Leftrightarrow (member(e,[x|y]) \wedge \neg member(e,s))) \; \theta_2 = \{(l,[x|y])\}$

**Definition 12 (Normalization Rules).** *To avoid negations in front of formulas, we normalize them by using the following set of rewrite rules where $G$ and $H$ are formulas.*

(1) $\neg false \rightarrow true$, (2) $\neg true \rightarrow false$
(3) $\neg\neg G \rightarrow G$, (4) $\neg(G \Rightarrow H) \rightarrow (G \wedge \neg H)$
(5) $\neg(G \wedge H) \rightarrow (\neg G \vee \neg H)$ (6) $\neg(G \vee H) \rightarrow (\neg G \wedge \neg H)$
(7) $\neg(G \Leftrightarrow H) \rightarrow (\neg(G \Rightarrow H) \vee \neg(H \Rightarrow G))$

**Definition 13 ($i$-Expansion).** *We say that $exp(F,i)$ is the $i$-expansion of $F$ if and only if every formula in $exp(F,i)$ is constructed by applying all the $i$-instantiations (at least 1) to $F$, then all the unfolding steps (at least 1) to each resulting formula. After unfolding steps, it can appear negative sub-formulas (i.e. presence of negation in front of unfolded sub-formulas). To avoid negations in front of such sub-formulas, we normalize them.*

*Example 9 ($exp(F,1)$ for $F$ equal to the axiom of $subset_1^{neg}$ in Example 7).*

(1) $\forall\,(subset_1^{neg}(e,[\,],e,[\,]) \Leftrightarrow (false \wedge true))$
(2) $\forall\,(subset_1^{neg}(e,[\,],e,[v|w]) \Leftrightarrow (false \wedge (\neg id_{\mathcal{N}at}(v,e) \wedge \neg member(e,w))))$
(3) $\forall\,(subset_1^{neg}(e,[x|y],e,[\,]) \Leftrightarrow ((id_{\mathcal{N}at}(x,e) \vee member(e,y)) \wedge true))$
(4) $\forall\,(subset_1^{neg}(e,[x|y],e,[v|w]) \Leftrightarrow ((id_{\mathcal{N}at}(x,e) \vee member(e,y)) \wedge (\neg id_{\mathcal{N}at}(v,e) \wedge \neg member(e,w))))$

Once a formula has been "decomposed into a set of simple formulas" (expansion), we are interested in *finding recursive compositions from such formulas.* This can be done by identifying sub-formulas and replacing them by new predicates (reduction). Our intention is to anticipate and organize the search space of sub-formulas and new predicates in order to manage reductions automatically.

To precise our intentions, we say that $\mathcal{D}(r)$ is the transitive closure of relation symbols used in the definition of $r$. For instance, $\mathcal{D}(member) = \{member, id_{\mathcal{N}at}\}$. From this definition, $\mathcal{L}(r)$ is the set of all the *literal patterns for a relation $r$* which is constructed from intermediate and upper atom patterns in $\mathcal{A}(r_j)$ for every $r_j$ in $\mathcal{D}(r)$. For instance,

$$\begin{aligned}\mathcal{L}(member) = \{\ & member(\_,\_),\ \neg member(\_,\_),\ id_{\mathcal{N}at}(\_,\_),\ id_{\mathcal{N}at}(s(\_),\_),\\ & id_{\mathcal{N}at}(\_,s(\_)),\ id_{\mathcal{N}at}(0,\_),\ id_{\mathcal{N}at}(\_,0),\ \neg id_{\mathcal{N}at}(\_,\_),\\ & \neg id_{\mathcal{N}at}(s(\_),\_),\ \neg id_{\mathcal{N}at}(\_,s(\_)),\ \neg id_{\mathcal{N}at}(0,\_),\ \neg id_{\mathcal{N}at}(\_,0)\ \}\end{aligned}$$

We say that $\mathcal{F}(R)$ is the set of all the *formula patterns for a quantifier-free formula $R$* if it is constructed by replacing each literal defined on $r_j$ in $R$ by elements in $\{true, false\} \cup \mathcal{L}(r_j)$.

**Definition 14 (Search Space).** *We say that $\Omega(r)$ is the search space for an expressive assertion $\langle \Sigma_r, Spec_r \rangle$ if and only if*

$$\Omega(r) = \bigcup_{Ax \in Spec_r} \mathcal{F}(rhs(Ax))$$

Every element $E_{\mathcal{P}}$ in $\Omega(r)$ encodes a sort of formulas. Such a codification depends on the sequence of relation symbols in $E_{\mathcal{P}}$. Our method considers that every formula pattern $E_{\mathcal{P}}$ is equivalent to a fresh atom pattern whose relation symbol, say $r_{E_{\mathcal{P}}}$, represents such a codification. In order to establish a precise codification, a bijection is proposed between term patterns in $E_{\mathcal{P}}$ and parameter positions in $r_{E_{\mathcal{P}}}$.

We say that $\Omega_{ext}(r)$ is an *extended search space* for an expressive assertion $\langle \Sigma_r, Spec_r \rangle$ if and only if $\Omega_{ext}(r)$ is constructed from $\Omega(r)$ by including an element of the form $r_{E_{\mathcal{P}}} \Leftrightarrow E_{\mathcal{P}}$ for each element $E_{\mathcal{P}} \in \Omega(r)$. An extended search space represents a repository of new predicates and sub-formulas to be considered at reduction time.

Experimentally, it is important to note that no complete extended search spaces are needed when compiling expressive assertions. For instance, from a theoretical point of view, $|\Omega_{ext}(subset_1^{neg})| = 196$ but only 8 of these patterns have been needed when compiling $subset_1^{neg}$. Table 2 shows these patterns where positions for recursive parameters have been highlighted. A practical result is proposed in [9] where we show that search spaces can be constructed on demand using tabulation techniques.

| |
|---|
| (1) $subset_1^{neg}(\__1,\__2,\__3,\__4) \Leftrightarrow (member(\__1,\__2) \wedge \neg member(\__3,\__4))$ |
| (2) $subset_2^{neg}(\__1,\__2) \Leftrightarrow (false \wedge \neg member(\__1,\__2))$ |
| (3) $subset_3^{neg}(\__1,\__2) \Leftrightarrow (member(\__1,\__2) \wedge false)$ |
| (4) $subset_4^{neg}(\__1,\__2) \Leftrightarrow (true \wedge \neg member(\__1,\__2))$ |
| (5) $subset_5^{neg} \Leftrightarrow (false \wedge true)$ |
| (6) $subset_6^{neg} \Leftrightarrow (false \wedge false)$ |
| (7) $subset_7^{neg} \Leftrightarrow (true \wedge false)$ |
| (8) $subset_8^{neg} \Leftrightarrow (true \wedge true)$ |

**Table 2.** $\Omega_{ext}(subset_1^{neg})$ (partial).

In order to automate reductions, we propose a method to decide when a formula is similar to an element in a search space. We supply "operational" definitions to justify the mechanization of our proposal.

By $tree(R_{\mathcal{P}})$ we denote the *tree representation* of a formula pattern $R_{\mathcal{P}}$ where each leaf node contains a literal pattern and each internal node contains a binary logical connective. We say that a node in $tree(R_{\mathcal{P}})$ is *preterminal* if it has, at least, one leaf node.

We say that $R_{\mathcal{P}}$ is *similar wrt connectives* to $E_{\mathcal{P}}$ if and only if every binary logical connective in $tree(E_{\mathcal{P}})$ is located at the same place in $tree(R_{\mathcal{P}})$. In Fig. 3, we show that $R_{\mathcal{P}}$ is similar wrt connectives to $E_{\mathcal{P}}$ (but $E_{\mathcal{P}}$ is not similar wrt connectives to $R_{\mathcal{P}}$). Similarity wrt connectives induces a mapping $f$ from preterminal nodes in $tree(E_{\mathcal{P}})$ to subtrees in $tree(R_{\mathcal{P}})$ (for instance, $f$ in Fig. 3).

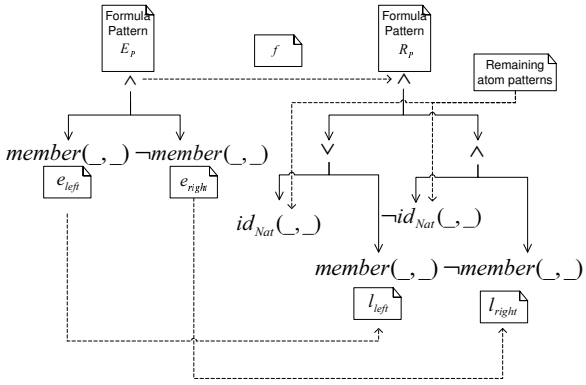


**Fig. 3.** $R_{\mathcal{P}}$ is similar to $E_{\mathcal{P}}$.

**Definition 15 (Similar Pattern).** *We say that a formula pattern $R_{\mathcal{P}}$ is similar to a formula pattern $E_{\mathcal{P}}$ if and only if*

1. *$R_{\mathcal{P}}$ is similar wrt connectives to $E_{\mathcal{P}}$ via mapping $f$,*
2. *(a) For each preterminal node $n \in tree(E_{\mathcal{P}})$ with two leaf nodes, $e_{left_{\mathcal{P}}}$ and $e_{right_{\mathcal{P}}}$, there exist two leaf nodes, $l_{left_{\mathcal{P}}}$ in the left subtree of $f(n)$ and $l_{right_{\mathcal{P}}}$ in the right subtree of $f(n)$, where $e_{left_{\mathcal{P}}} = l_{left_{\mathcal{P}}}$ and $e_{right_{\mathcal{P}}} = l_{right_{\mathcal{P}}}$ and*
   *(b) For each preterminal node $n \in tree(E_{\mathcal{P}})$ with one leaf node, $e_{left_{\mathcal{P}}}/e_{right_{\mathcal{P}}}$, there exists a leaf node $l_{left_{\mathcal{P}}}/l_{right_{\mathcal{P}}}$ in the left/right subtree of $f(n)$, where $e_{left_{\mathcal{P}}}/e_{right_{\mathcal{P}}} = l_{left_{\mathcal{P}}}/l_{right_{\mathcal{P}}}$.*

Fig. 3 shows an example of similarity.

**Theorem 2 (Expansion Preserves Correctness).** *Let $\{\forall(r_i(\bar{x}_1) \Leftrightarrow R^{exp}(\bar{x}_1)),$-$...,\forall(r_i(\bar{x}_j) \Leftrightarrow R^{exp}(\bar{x}_j))\}$ be the set of formulas in the $i$-expansion of a formula $\forall(r_i(\bar{x}) \Leftrightarrow R(\bar{x}))$. For every ground atom $r_i(\bar{x})\phi$ there exists a ground atom $r_i(\bar{x}_k)\delta$, with $k \in \{1..j\}$, such that*

$$\mathcal{C} \models R(\bar{x})\phi \Leftrightarrow R^{exp}(\bar{x}_k)\delta$$

(A proof of this theorem can be found in Appendix).

The following result ensures that every expansion can be reduced by new predicates in a (extended) search space.

**Theorem 3 (Expansion is an Internal Operation in $\Omega_{ext}$).** *Let $F$ be a formula in $\mathcal{C}$ of the form $\forall(r_i(\bar{x}) \Leftrightarrow R(\bar{x}))$. If $rhs(F)_{\mathcal{P}}$ is similar to the rhs of some pattern in $\Omega_{ext}(r_i)$ then for every $F_k \in exp(F, i)$, $rhs(F_k)_{\mathcal{P}}$ is similar to the rhs of some pattern in $\Omega_{ext}(r_i)$.* (A proof of this theorem can be found in Appendix).

### 3.2 Reduction Phase

Reduction phase is intended to replace sub-formulas by new predicates. To identify and replace sub-formulas by equivalent atoms are two key activities in a transformation step. When a formula is similar to an element in a search space, it is rewritten (rewriting step), preserving its semantics, in order to facilitate an automatic replacement of sub-formulas by new predicates (folding step). In the following definitions, we consider that $F$ is a formula of the form $\forall(r_i(\bar{x}) \Leftrightarrow R(\bar{x}))$ with $R(\bar{x})$ a quantifier-free formula written in (the language of an assertion context) $\mathcal{C}$ and $r_i$ a symbol not defined in $\mathcal{C}$.

**Definition 16 (Rewriting Step).** *Let $\{a_1, ..., a_p, ..., a_n\}$ be the set of atoms in $R(\bar{x})$. By $R(a_1, ..., a_p, ..., a_n)$ we denote an alternative representation of $R(\bar{x})$. Let $P_{\mathcal{P}}$ be a pattern in $\Omega_{ext}(r_i)$ such that $R(\bar{x})_{\mathcal{P}}$ is similar to $E_{\mathcal{P}} = rhs(P_{\mathcal{P}})$ with $f$ as the induced mapping for deciding about similarity wrt connectives and $A = \{a_1, a_2, ..., a_p\}$ as the set of atoms in $R(a_1, ..., a_p, ..., a_n)$ which have not been used for deciding about similarity (i.e. remaining atoms). We say $rew(F, P_{\mathcal{P}})$ is the rewriting step of $F$ wrt $P_{\mathcal{P}}$ if and only if*

1. *We consider the set of all the evaluations for atoms $a_1, a_2, ..., a_p$ in $R(\bar{x})$ in the following schematic manner:*

$$
\begin{array}{l}
rew(F, P_{\mathcal{P}}) = \forall(r_i(\bar{x}) \Leftrightarrow \\
\quad (R(true, true, ..., true, a_{p+1}, ..., a_n) \quad \wedge a_1 \quad \wedge a_2 \quad \wedge ... \wedge a_p) \quad \vee \\
\quad (R(false, true, ..., true, a_{p+1}, ..., a_n) \quad \wedge \neg a_1 \wedge a_2 \quad \wedge ... \wedge a_p) \quad \vee \\
\quad (R(true, false, ..., true, a_{p+1}, ..., a_n) \quad \wedge a_1 \quad \wedge \neg a_2 \wedge ... \wedge a_p) \quad \vee \\
\quad ... \quad \vee \\
\quad (R(false, false..., false, a_{p+1}, ..., a_n) \wedge \neg a_1 \wedge \neg a_2 \wedge ... \wedge \neg a_p))
\end{array}
$$

   *where each $R(c_1, c_2,..., c_p, a_{p+1}, ..., a_n)$ represents the replacement in $R(\bar{x})$ of the set of atoms $\{a_1, a_2, ..., a_p\}$ by the combination $\{c_1, c_2, ..., c_p\}$ of propositions $true$ and $false$.*
2. *We simplify each $R(c_1, c_2,..., c_p, a_{p+1}, ..., a_n)$ in the following form:*
   (a) *For each preterminal node $n \in tree(E_{\mathcal{P}})$ with two leaf nodes, we simplify (Def. 7) sub-formulas in $R(c_1, c_2,..., c_p, a_{p+1}, ..., a_n)$ which correspond to left and right subtrees of $f(n)$ in $R(c_1, c_2,..., c_p, a_{p+1}, ..., a_n)_{\mathcal{P}}$.*
   (b) *For each preterminal node $n \in tree(E_{\mathcal{P}})$ with one left/right leaf node, we simplify (Def. 7) the sub-formula in $R(c_1, c_2,..., c_p, a_{p+1}, ..., a_n)$ which corresponds to left/right subtrees of $f(n)$ in $R(c_1, c_2,..., c_p, a_{p+1}, ..., a_n)_{\mathcal{P}}$.*

   *This selective simplification is intended to preserve similarity wrt connectives between $R(c_1, c_2, ..., c_p, a_{p+1}, ..., a_n)_{\mathcal{P}}$ and $E_{\mathcal{P}}$.*

*Example 10 (Rewriting Step).* Let $F$ be the formula (4) in Example 9, $P_{\mathcal{P}}$ the pattern (1) in $\Omega_{ext}(subset_1^{neg})$ and $E_{\mathcal{P}} = rhs(P_{\mathcal{P}})$. In Fig. 3 we can verify that $A = \{a_1 = id_{\mathcal{N}at}(x, e), a_2 = id_{\mathcal{N}at}(v, e)\}$ is the set of atoms which has not been used for deciding about similarity (i.e. remaining atoms). After rewriting step, 1 we obtain:

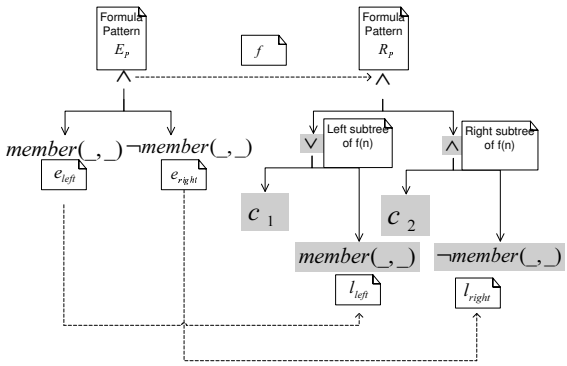


**Fig. 4.** Rewriting step. Sub-formulas to be simpified.

$$
\begin{array}{l}
rew(F, P_{\mathcal{P}}) = \forall(subset_1^{neg}(e,[x|y],e,[v|w]) \Leftrightarrow \\
\quad (true \vee member(e,y)) \wedge (false \wedge \neg member(e,w)) \quad \wedge id_{\mathcal{N}at}(x,e) \quad \wedge id_{\mathcal{N}at}(v,e) \quad \vee \\
\quad (false \vee member(e,y)) \wedge (false \wedge \neg member(e,w)) \wedge \neg id_{\mathcal{N}at}(x,e) \wedge id_{\mathcal{N}at}(v,e) \quad \vee \\
\quad ((true \vee member(e,y)) \wedge (true \wedge \neg member(e,w)) \quad \wedge id_{\mathcal{N}at}(x,e) \quad \wedge \neg id_{\mathcal{N}at}(v,e) \vee \\
\quad (false \vee member(e,y)) \wedge (true \wedge \neg member(e,w)) \quad \wedge \neg id_{\mathcal{N}at}(x,e) \wedge \neg id_{\mathcal{N}at}(v,e)
\end{array}
$$

For preterminal node $\wedge$ in $tree(E_{\mathcal{P}})$, we simplify sub-formulas in $R(c_1, c_2, ..., c_p, a_{p+1}, ..., a_n)$ which correspond to left and right subtrees of $f(\wedge)$ in $R(c_1, c_2, ..., c_p, a_{p+1}, ..., a_n)_{\mathcal{P}}$. In Fig. 4 we have highlighted such subtrees. After rewriting step 2, we obtain:

$$
\begin{array}{l}
rew(\ F, P_{\mathcal{P}}) = \\
\quad \forall(subset_1^{neg}(e,[x|y],e,[v|w]) \Leftrightarrow \\
\qquad true \wedge false \quad \wedge id_{\mathcal{N}at}(x,e) \quad \wedge id_{\mathcal{N}at}(v,e) \quad \vee \\
\qquad member(e,y) \wedge false \quad \wedge \neg id_{\mathcal{N}at}(x,e) \wedge id_{\mathcal{N}at}(v,e) \quad \vee \\
\qquad true \wedge \neg member(e,w) \quad \wedge id_{\mathcal{N}at}(x,e) \quad \wedge \neg id_{\mathcal{N}at}(v,e) \vee \\
\qquad member(e,y) \wedge \neg member(e,w) \wedge \neg id_{\mathcal{N}at}(x,e) \wedge \neg id_{\mathcal{N}at}(v,e)
\end{array}
$$

To apply automatic folding to formulas, we need to instantiate patterns in extended search spaces. We say that a quantifier-free formula $pi(P_{\mathcal{P}}, R_i)$ is the *pattern instantiation* of $P_{\mathcal{P}} \in \Omega_{ext}(r_i)$ wrt $R_i$ if and only if $R_{i\mathcal{P}} = rhs(P_{\mathcal{P}})$ and $pi(P_{\mathcal{P}}, R_i)$ is obtained from $P_{\mathcal{P}}$ by replacing every term pattern in $P_{\mathcal{P}}$ by its respective term in $R_i$. The marks of recursive parameters in $P_{\mathcal{P}}$ are propagated to $pi(P_{\mathcal{P}}, R_i)$.

*Example 11 (Pattern instantiation in $\Omega_{ext}(subset_1^{neg})$).*

$$
\begin{array}{ll}
P_{\mathcal{P}} & = subset_1^{neg}(\__1, \__2, \__3, \__4) \Leftrightarrow (member(\__1, \__2) \wedge \neg member(\__3, \__4)) \\
R_i & = member(e,y) \wedge \neg member(e,w) \\
pi(P_{\mathcal{P}}, R_i) & = subset_1^{neg}(e, \mathbf{y}, e, \mathbf{w}) \Leftrightarrow (member(e, \mathbf{y}) \wedge \neg member(e, \mathbf{w}))
\end{array}
$$

**Definition 17 (Folding Step).** *Let $F$ be a formula in $\mathcal{C}$ of the form $\forall(r_i(\bar{x}) \Leftrightarrow R(\bar{x}))$ and $R_i$ a sub-formula in $R(\bar{x})$ with $R_{i\mathcal{P}} = rhs(P_{\mathcal{P}})$ and $P_{\mathcal{P}} \in \Omega_{ext}(r_i)$. We say that $fold(F, P_{\mathcal{P}})$ is the folding step of $F$ wrt $P_{\mathcal{P}}$ if and only if it is obtained by replacing $R_i$ by $lhs(pi(P_{\mathcal{P}}, R_i))$ in $R(\bar{x})$.*

Although search spaces are finite, to identify sub-formulas to be folded constitutes a highly non-deterministic task. In order to guide the automatic identification of sub-formulas we introduce the notion of encapsulation and explain how rewriting and folding rules contribute to automate reductions.

We say that a formula/formula pattern $R/R_{\mathcal{P}}$ is *completely encapsulated* in $Layer_i$ in $\mathcal{C}$ if and only if every atom/atom pattern in $R/R_{\mathcal{P}}$ is defined on a relation symbol of level $i$. We say that a formula/formula pattern $R/R_{\mathcal{P}}$ is *partially encapsulated* in $Layer_i$ if and only if some of its atom/atom patterns is defined on relation symbol of level $i$ and the rest is defined on relation symbols of lower level.

**Definition 18 ($i$-Reduction).** *Let $F_k \in exp(F, i)$ be a formula of level $i$. The $i$-reduction of $F_k$ wrt $\Omega_{ext}(r_i)$, $red(F_k, i, \Omega_{ext}(r_i))$, is implemented in the following steps:*

1. *(Searching). To search for patterns $P_{\mathcal{P}} \in \Omega_{ext}(r_i)$ with $rhs(P_{\mathcal{P}})$ as a completely encapsulated pattern of level $i$. Literal patterns in $rhs(F_{k\mathcal{P}})$ can be used to accelerate this search. If this search fails then to continue by searching for partially encapsulated patterns of level $i$. If this search fails then to continue in a similar way by searching for patterns of level $i-1$ and so on.*
2. *(Rewriting Step, 1). Let $rhs(F_{k\mathcal{P}})$ be similar to $rhs(P_{\mathcal{P}})$. We fix in $rhs(F_{k\mathcal{P}})$ those atom patterns which are responsible of similarity. The remaining atoms $A$ in $rhs(F_k)$ are selected to be evaluated.*
3. *(Rewriting Step, 2). After evaluating wrt $A$, we simplify by preserving the structure of logical connectives in $P_{\mathcal{P}}$.*
4. *(Folding Step) At this point, we identify sub-formula $R_i$ to be folded (i.e. $R_{i\mathcal{P}} = rhs(P_{\mathcal{P}})$). In addition, we are able to construct a new predicate (i.e. $lhs(pi(P_{\mathcal{P}}, R_i))$) and to replace $R_i$ in $rew(F_k, rhs(P_{\mathcal{P}}))$ by this new predicate automatically.*

*Example 12 (i-Reduction).* Let $F_k$ be the formula (4) in Example 9.

(1) We search for patterns $P_{\mathcal{P}} \in \Omega_{ext}(subset_1^{neg})$ such that $rhs(P_{\mathcal{P}})$ is a completely encapsulated pattern of level 1:

$$P_{\mathcal{P}} = subset_1^{neg}(\__1, \__2, \__3, \__4) \Leftrightarrow (member(\__1, \__2) \wedge \neg member(\__3, \__4))$$

(2) If $rhs(F_{k\mathcal{P}})$ is similar to (the rhs of) several patterns then a non-deterministic choice must be done. In our example, the choice is deterministic (i.e. $P_{\mathcal{P}}$ is the unique candidate). We fix in $rhs(F_{k\mathcal{P}})$ those atom patterns which are responsible of similarity.

$$rhs(F_{k\mathcal{P}}) = (id_{\mathcal{N}at}(\_,\_) \vee \underline{member(\_,\_)}) \wedge (\neg id_{\mathcal{N}at}(\_,\_) \wedge \underline{\neg member(\_,\_)})$$

The set of remaining atoms $A = \{a_1 = id_{\mathcal{N}at}(x,e), a_2 = id_{\mathcal{N}at}(v,e)\}$ is then selected to be evaluated.

(3) After evaluating and simplifying:

$$
\begin{array}{l}
rew(\ F_k, P_{\mathcal{P}}) = \\
\quad \forall(subset_1^{neg}(e,[x|y],e,[v|w]) \Leftrightarrow \\
\qquad true \wedge false \qquad\qquad\qquad \wedge id_{\mathcal{N}at}(x,e) \quad \wedge id_{\mathcal{N}at}(v,e) \quad \vee \\
\qquad member(e,y) \wedge false \qquad\qquad \wedge \neg id_{\mathcal{N}at}(x,e) \wedge id_{\mathcal{N}at}(v,e)) \quad \vee \\
\qquad true \wedge \neg member(e,w) \qquad\quad \wedge id_{\mathcal{N}at}(x,e) \quad \wedge \neg id_{\mathcal{N}at}(v,e) \vee \\
\qquad member(e,y) \wedge \neg member(e,w) \wedge \neg id_{\mathcal{N}at}(x,e) \wedge \neg id_{\mathcal{N}at}(v,e)
\end{array}
$$

(4) At this point, it is easy to identify $R_1$ as a sub-formula in $rew(F_k, P_{\mathcal{P}})$ whose pattern is equal to the $rhs(P_{\mathcal{P}})$.

$$
\begin{array}{l}
rew(\ F_k, P_{\mathcal{P}}) = \\
\quad \forall(subset_1^{neg}(e,[x|y],e,[v|w]) \Leftrightarrow \\
\qquad true \wedge false \qquad\qquad\qquad \wedge id_{\mathcal{N}at}(x,e) \quad \wedge id_{\mathcal{N}at}(v,e) \quad \vee \\
\qquad member(e,y) \wedge false \qquad\qquad \wedge \neg id_{\mathcal{N}at}(x,e) \wedge id_{\mathcal{N}at}(v,e)) \quad \vee \\
\qquad true \wedge \neg member(e,w) \qquad\quad \wedge id_{\mathcal{N}at}(x,e) \quad \wedge \neg id_{\mathcal{N}at}(v,e) \vee \\
\qquad \underbrace{member(e,y) \wedge \neg member(e,w)}_{R_1} \wedge \neg id_{\mathcal{N}at}(x,e) \wedge \neg id_{\mathcal{N}at}(v,e)
\end{array}
$$

A new predicate is obtained by pattern instantiation (Example 11):

$$\boxed{lhs(pi(P_{\mathcal{P}}, R_1)) = subset_1^{neg}(e, \mathbf{y}, e, \mathbf{w})}$$

Finally, the replacement of $R_1$ by the new predicate produces the formula:

$$
\begin{array}{l}
fold(\ rew(F_k, P_{\mathcal{P}}), P_{\mathcal{P}}) = \\
\quad \forall(subset_1^{neg}(e,[x|y],e,[v|w]) \Leftrightarrow \\
\qquad true \wedge false \qquad\qquad\qquad \wedge id_{\mathcal{N}at}(x,e) \quad \wedge id_{\mathcal{N}at}(v,e) \quad \vee \\
\qquad member(e,y) \wedge false \qquad\qquad \wedge \neg id_{\mathcal{N}at}(x,e) \wedge id_{\mathcal{N}at}(v,e)) \quad \vee \\
\qquad true \wedge \neg member(e,w) \qquad\quad \wedge id_{\mathcal{N}at}(x,e) \quad \wedge \neg id_{\mathcal{N}at}(v,e) \vee \\
\qquad \underbrace{subset_1^{neg}(e, \mathbf{y}, e, \mathbf{w})}_{lhs(pi(P_{\mathcal{P}}, R_1))} \qquad \wedge \neg id_{\mathcal{N}at}(x,e) \wedge \neg id_{\mathcal{N}at}(v,e)
\end{array}
$$

We say that an $i$-reduction $red(F_k, i, \Omega_{ext}(r_i))$ is *complete* when all the possible folding steps have been applied to $rew(F_k, P_{\mathcal{P}})$.

**Theorem 4 (Reduction Preserves Correctness).** *Let $\forall(r_i(\bar{x}) \Leftrightarrow R^{red}(\bar{x}))$ be the $i$-reduction of an expanded formula $\forall(r_i(\bar{x}) \Leftrightarrow R(\bar{x}))$ wrt $\Omega_{ext}(r_i)$. For every ground atom $r_i(\bar{x})\phi$,*

$$\mathcal{C} \models R^{red}(\bar{x})\phi \Leftrightarrow R(\bar{x})\phi$$

(A proof of this theorem can be found in Appendix).

We say that a reduction phase for a formula $F$ is *complete* when a complete $i$-reduction has been applied to each formula in $exp(F, i)$.

### 3.3 Compilation as an Incremental and Terminating Process

The compilation of an axiom is completed by a finite sequence of meaning-preserving transformation steps. Each transformation step is composed of an expansion phase followed by a (complete) reduction phase. Table 3 shows the axiom of $subset_1^{neg}$ after a transformation step.

**Theorem 5 (Forms of Compiled Axioms).** *After a transformation step, every compiled axiom presents one of the following forms:*

1. $\forall(r_i(\bar{x}) \Leftrightarrow r_j(\bar{x}))$ *where* $r_j(\bar{x})_{\mathcal{P}}$ *is equal to the lhs of some element in* $\Omega_{ext}(r_i)$.
2. $\forall(r_i(\bar{x}) \Leftrightarrow \bigvee r_j(\bar{x}) \wedge G_j(\bar{x}))$ *where* $r_j(\bar{x})_{\mathcal{P}}$ *is equal to the lhs of some element in* $\Omega_{ext}(r_i)$ *and* $G_j(\bar{x})$ *a conjunctive formula of literals whose patterns are included in* $\mathcal{L}(r_i)$.

(A proof of this theorem can be found in Appendix).

Each transformation step represents an *increment* in the overall compilation process. Due to Theorem 5, each successive increment compiles either an axiom for $r_j$ from $r_j(\bar{x})$ (e.g. $\forall(subset_3^{neg}(e,y) \Leftrightarrow (member(e,y) \wedge false))$) or an axiom for a new assertion from a literal in $G_j(\bar{x})$ (e.g. $\forall(subset_9^{neg}(x,e) \Leftrightarrow \neg id_{\mathcal{N}at}(x,e))$).

| |
|---|
| (1) $\forall\,(subset_1^{neg}(e,[\,],e,[\,]) \Leftrightarrow subset_5^{neg})$ |
| (2) $\forall\,(subset_1^{neg}(e,[\,],e,[v\|w]) \Leftrightarrow (subset_6^{neg} \wedge id_{\mathcal{N}at}(v,e) \vee$ |
| $subset_2^{neg}(e,\mathbf{w}) \wedge \neg id_{\mathcal{N}at}(v,e)))$ |
| (3) $\forall\,(subset_1^{neg}(e,[x\|y],e,[\,]) \Leftrightarrow (subset_7^{neg} \wedge id_{\mathcal{N}at}(x,e) \vee$ |
| $subset_3^{neg}(e,\mathbf{y}) \wedge \neg id_{\mathcal{N}at}(x,e)))$ |
| (4) $\forall\,(subset_1^{neg}(e,[x\|y],e,[v\|w]) \Leftrightarrow (subset_7^{neg} \wedge id_{\mathcal{N}at}(x,e) \wedge id_{\mathcal{N}at}(v,e) \vee$ |
| $subset_3^{neg}(e,\mathbf{y}) \wedge \neg id_{\mathcal{N}at}(x,e) \wedge id_{\mathcal{N}at}(v,e) \vee$ |
| $subset_4^{neg}(e,\mathbf{w}) \wedge id_{\mathcal{N}at}(x,e) \wedge \neg id_{\mathcal{N}at}(v,e) \vee$ |
| $subset_1^{neg}(e,\mathbf{y},e,\mathbf{w}) \wedge \neg id_{\mathcal{N}at}(x,e) \wedge \neg id_{\mathcal{N}at}(v,e)))$ |

**Table 3.** Compiled axioms after a transformation step for the axiom of $subset_1^{neg}$.

**Theorem 6 (Termination).** *The compilation of an axiom is completed in a finite amount of increments.* (A proof of this theorem can be found in Appendix).

For instance, the compilation of the axiom of $subset_1^{neg}$ (Example 7) has been completed by means of 10 increments ($subset_1^{neg}$,...,$subset_{10}^{neg}$). The complete set of compiled assertions is shown in Appendix-Table 4.

The form of axioms in compiled assertions (i.e. universal closure, mutually-exclusive disjunctions of conjunctions and absence of negated atoms) allow to define a simple translation method to definite logic programs.

**Definition 19 (Translation Method).** *Let* $\langle \Sigma_r, Spec_r \rangle$ *be a compiled assertion. For every axiom* $Ax \in Spec_r$,

1. *If $Ax$ is of the form $\forall(r \Leftrightarrow P)$ where $P$ is a propositional formula (i.e. it is composed of propositions $true$ and $false$ only) then two situations are possible:*
   (a) *If the evaluation of $P$ is equal to $false$ then $Ax$ is translated to an empty clause.*
   (b) *If the evaluation of $P$ is equal to $true$ then $Ax$ is translated to a clause of the form* r.
2. *If $Ax$ is of the form $\forall(r(\bar{x}) \Leftrightarrow \bigvee_k(r_1(\bar{x}) \wedge ... \wedge r_n(\bar{x}))$ then it is translated to a set of $k$ clauses of the form* r($\bar{X}$):- r1($\bar{X}$),...,rn($\bar{X}$).. *Every clause, which includes an atom occurrence $r$ with axiom of the form $\forall(r \Leftrightarrow P)$ and $P$ equal to $false$, is deleted.*

*Example 13 (Translation of assertions in Table 4 (In Appendix)).*

```
subsetneg1(E,[ ],E,[V|W]):-subsetneg2(E,W), subsetneg10(V,E).
subsetneg1(E,[X|Y],E,[ ]):-subsetneg3(E,Y), subsetneg9(X,E).
subsetneg1(E,[X|Y],E,[V|W]):-subsetneg3(E,Y),subsetneg9(X,E),subsetneg10(V,E).
subsetneg1(E,[X|Y],E,[V|W]):-subsetneg4(E,W),subsetneg10(X,E),subsetneg9(V,E).
subsetneg1(E,[X|Y],E,[V|W]):-subsetneg1(E,Y,E,W),subsetneg9(X,E),subsetneg9(V,E).

subsetneg2(E,[X|Y]):-subsetneg2(E,Y),subsetneg9(X,E).

subsetneg3(E,[X|Y]):-subsetneg3(E,Y),subsetneg9(X,E).

subsetneg4(E,[ ]):-subsetneg8.
subsetneg4(E,[V|W]):-subsetneg4(E,W),subsetneg9(V,E).

subsetneg8.

subsetneg9(0,s(Y)).
subsetneg9(s(X),0).
subsetneg9(s(X),s(Y)):-subsetneg9(X,Y).

subsetneg10(0,0).
subsetneg10(s(X),s(Y)):-subsetneg10(X,Y).
```

In order to decide about the existence of executions for expressive assertions, we propose a simple method based on the inspection of compiled programs. For instance, by inspecting logic program code in Example 13, we can verify that expressive assertion *subset* is executable because any tuple of ground terms for parameters 2 and 4 in subsetneg1 (i.e. original parameters of *subset*) is sufficient to cover, with ground terms, every occurrence of recursive parameter in the definition of subsetneg1. To clarify this fact, we have highlighted, in bold letter, occurrences of recursive parameters.

*How can assertion checkers decide about $\mathcal{C} \models r(\bar{t})$ and $\mathcal{C} \models \neg r(\bar{t})$ from executions of logic programs $r_1$ and $r_1^{neg}$?*

1. Suppose we want to prove that the formula $\forall\bar{y}(\neg r_1^{neg}(\bar{t},\bar{y}))$ is a logical consequence of a program $\forall\bar{x},\bar{y}(r_1^{neg}(\bar{x},\bar{y}) \Leftarrow P^{neg}(\bar{x},\bar{y}))$ which has been compiled from $\forall\bar{x},\bar{y}(r_1^{neg}(\bar{x},\bar{y}) \Leftrightarrow \neg R(\bar{x},\bar{y}))$ for an axiom of the form $\forall\bar{x}(r(\bar{x}) \Leftrightarrow \forall\bar{y}R(\bar{x},\bar{y}))$. Resolution theorem provers are refutation systems. That is, the negation of the formula to be proved is added to the axioms of the program and a contradiction is derived. If we negate $\forall\bar{y}(\neg r_1^{neg}(\bar{t},\bar{y}))$, we obtain the goal $\exists\bar{y}(r_1^{neg}(\bar{t},\bar{y}))$.
   (a) If the empty clause is derived (i.e. there is no substitutions for $\bar{y}$) then a contradiction has been obtained assuring that $\forall\bar{y}(\neg r_1^{neg}(\bar{t},\bar{y}))$ is a logical consequence of the program. Then, by total correctness of the compilation method (Theorems 2 and 4), $\forall\bar{y}(\neg\neg R(\bar{t},\bar{y}))$, or equivalently, $\forall\bar{y}(R(\bar{t},\bar{y}))$ is a logical consequence of $\mathcal{C}$. Finally, by equivalence, $r(\bar{t})$ is also a logical consequence of $\mathcal{C}$.
   (b) If some substitution is computed for $\bar{y}$ then, by total correctness of the compilation method, $\exists\bar{y}(\neg R(\bar{t},\bar{y}))$, or equivalently, $\neg\forall\bar{y}(R(\bar{t},\bar{y}))$ is a logical consequence of $\mathcal{C}$. Finally, by equivalence, $\neg r(\bar{t})$ is also a logical consequence of $\mathcal{C}$.
2. Suppose, this time, we want to prove that the formula $\forall\bar{y}(\neg r_1(\bar{t},\bar{y}))$ is a logical consequence of a program $\forall\bar{x},\bar{y}(r_1(\bar{x},\bar{y}) \Leftarrow P(\bar{x},\bar{y}))$ which has been compiled from $\forall\bar{x},\bar{y}(r_1(\bar{x},\bar{y}) \Leftrightarrow R(\bar{x},\bar{y}))$ for an axiom of the form $\forall\bar{x}(r(\bar{x}) \Leftrightarrow \exists\bar{y}R(\bar{x},\bar{y}))$. If we negate $\forall\bar{y}(\neg r_1(\bar{t},\bar{y}))$, we obtain the goal $\exists\bar{y}(r_1(\bar{t},\bar{y}))$.
   (a) If the empty clause is derived (i.e. there is no substitutions for $\bar{y}$) then a contradiction has been obtained assuring that $\forall\bar{y}(\neg r_1(\bar{t},\bar{y}))$ is a logical consequence of the program. Then, by total correctness of the compilation method, $\forall\bar{y}(\neg R(\bar{t},\bar{y}))$, or equivalently, $\neg\exists\bar{y}(R(\bar{t},\bar{y}))$ is a logical consequence of $\mathcal{C}$. Finally, by equivalence, $\neg r(\bar{t})$ is also a logical consequence of $\mathcal{C}$.
   (b) If some substitution is computed for $\bar{y}$ then, by total correctness of the compilation method, $\exists\bar{y}(R(\bar{t},\bar{y}))$ is a logical consequence of $\mathcal{C}$ so, by equivalence, $r(\bar{t})$ is also a logical consequence of $\mathcal{C}$.

Hence, in summary,

1. If $Q = \forall$ then the execution of $\exists\bar{y}(r_1^{neg}(\bar{t},\bar{y}))$ will compute a set of substitutions $\{\theta_j^{neg}\}$ (e.g. Table 1, row (3)).
   If $\{\theta_j^{neg}\} = \emptyset$ then, by total correctness of $r_1^{neg}$, $\mathcal{C} \models r(\bar{t})$ else $\mathcal{C} \models \neg r(\bar{t})$.
2. If $Q = \exists$ then the execution of $\exists\bar{y}(r_1(\bar{t},\bar{y}))$ will compute a set of substitutions $\{\theta_j\}$ (e.g. Table 1, rows (1), (2) and (4)).
   If $\{\theta_j\} = \emptyset$ then, by total correctness of $r_1$, $\mathcal{C} \models \neg r(\bar{t})$ else $\mathcal{C} \models r(\bar{t})$.

## 4 Conclusions and Future Work

In this paper, we have formalized a class of assertions we call expressive assertions in the sense that they describe recursive models which are no directly translatable into programs. Due to this fact, current assertion checkers avoid its use limiting the expressivity of the assertion language in a considerable way. The existence

of mature studies in the field of transformational synthesis has constituted an important aid to solve this problem. Recurrent problems in transformational synthesis have been addressed, for instance, "eureka steps" (i.e. non-automatic steps) about when and how to define recursive predicates. To overcome them, we have restricted our attention to a particular class of first-order theories we call assertion contexts. This sort of theories is interesting because it presents a balance between expressiveness for writing assertions and existence of effective methods for compiling them into definite logic programs. Finally, we have shown that such programs can be used, in testing activities, as a decision criterion for ground atoms of expressive assertions.

From a practical view point, our work can be used as a tool to extend assertion contexts with expressive assertions in a conservative way without losing execution capabilities. For instance, $\mathcal{S} \cup \langle \Sigma_{subset}, Spec_{subset} \rangle$ is a more expressive assertion context than $\mathcal{S}$ where $subset_1^{neg}$ can be used to execute ground atoms for *subset*. We plan to study this issue as a future work.

## References


[1] A. Avellone, M. Ferrari and P. Miglioli. Synthesis of Programs in Abstract Data Types. $8^{th}$ In *(Proceedings of the International Workshop on Logic Program Synthesis and Transformation)*. LNCS 1559, Springer, 1999, pages 81-100.

[2] J. Barnes. High Integrity Ada: The SPARK Approach. *Addison-Wesley*, 1997.

[3] A. Bertoni, G. Mauri and P. Miglioli. On the Power of Model Theory in Specifying Abstract Data Types and in capturing their Recursiveness. *(Fundamenta Informaticae)*, VI(2):27-170, 1983.

[4] R. M. Burstall and J. Darlington. A Transformational System for Developing Recursive Programs. *(Journal of the ACM)* 24(1):44-67, 1977.

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

[6] D. Bartetzko, C. Fischer, M. Möller and H. Wehrheim. Jass-Java with Assertions. Proc. of the *First Workshop on Runtime Verification*. Paris, France. Electronic Notes on Theoretical Computer Science. Elsevier, 1999.

[7] P. Flener. Logic Program Synthesis from Incomplete Information. *Kluwer Academic Publishers*, Massachusetts, 1995.

[8] P. Flener. Achievements and Prospects of Program Synthesis. Invited chapter in: A.C. Kakas and F. Sadri (eds), *Computational Logic: Logic Programming and Beyond; Essays in Honour of Robert A. Kowalski*, pp. 310-346. Lecture Notes in Artificial Intelligence, volume 2407. Springer-Verlag, 2002.

[9] F. J. Galán and J. M. Cañete. Improving Constructive Synthesizers by Tabulation Techniques and Domain Ordering. In David Warren (ed.), *Tabulation and Parsing Deduction*, pages 37-49. Vigo-Spain 2000.

[10] F. J. Galán, V. J. Díaz and J. M. Cañete. Towards a Rigorous and Effective Functional Contract for Components. In *Informatica. An International Journal of Computing and Informatics*. Vol 25, N 4, pages 527-533, November 2001.

[11] F. J. Galán and J. M. Cañete. Compiling (for Validating) Explicit Specifications into Recursive Specificacions in Linear Stratified Theories. In *Proceedings of the Joint Conference on Declarative Programming*. Madrid-Spain, pages 223-240, 2002.

[12] R. Kramer. iContract-The Java Design by Contract Tool. *TOOLS 26: Technology of Object-Oriented Languages and Systems*, IEEE Computer Society Press, 1998.

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

[14] K. K. Lau and M. Ornaghi. Towards an Object-Oriented Methodology for Deductive Synthesis of Logic Programs. In *(Proceedings of LOPSTR'95)*. Springer-Verlag, 1995.

[15] G. Leavens, A. Baker, and C. Ruby. Preliminary Design of JML. *Technical Report 98-06u, Dept. of Computer Science, Iowa State University*, USA, April 2003.

[16] B. Meyer. Eiffel: The Language. *Prentice-Hall*, 1992.

[17] Tamaki, H. and Sato, T. Unfold/Fold Transformation of Logic Programs. *Proceedings of the Second International Conference on Logic Programming*, Uppsala, Sweden, 1984, pp. 127-138.

[18] J. Woodcock, J. Davies. Using Z: Specification, Refinement, and Proof. *Prentice Hall*, International Series in Computer Science, 1996.

## 5 Appendix

*Proof (Theorem 1. Ground Decidability).* Let $r(\bar{t})$ be a ground atom in the language of $\mathcal{C}$.

1. *By (Totality) and (No ambiguity) in assertions in* $\mathcal{C}$. Only one axiom $Ax = \forall(r(\bar{x}) \Leftrightarrow R(\bar{z}))$ in $Spec_r$ can be used to unfold $r(\bar{t})$. Let $r(\bar{t})$ be equal to $r(\bar{x})\theta$
2. *By (No internal variables) in Ax and definition of unfolding step.* $r(\bar{t})$ is equivalent to the ground formula $R(\bar{z})\theta$.
3. *By absurdum.* There is a non-terminating derivation by unfolding steps in $\mathcal{C}$ for some ground atom $r_j(\bar{t})$ in $R(\bar{z})\theta$. However, if $r_j$ is a recursive symbol then the existence of such a infinite derivation contradicts the fact that recursive assertions in $\mathcal{C}$ are well-founded and if $r_j$ is a non-recursive symbol then one unfolding step is sufficient to replace $r_j(\bar{t})$ by a formula of lower level. The number of layers in $\mathcal{C}$ is finite so, this reasoning can not be propagated indefinitely.
4. *By 3.* In a recurrent way, every ground atom in $R(\bar{z})\theta$ is replaced by a propositional formula (composed of propositions $true$ and $false$ only) in a finite amount of unfolding steps. Then, a finite amount of simplifications will be sufficient to end derivations. Therefore, $r(\bar{t})$ has an execution in $\mathcal{C}$ with $\mathcal{C} \vdash \neg r(\bar{t})$ if this ends in false and $\mathcal{C} \vdash r(\bar{t})$ if this ends in true.

*Proof (Theorem 2. Expansion Preserves Correctness).* Let $\{\forall(r_i(\bar{x}_1) \Leftrightarrow R^{exp}(\bar{x}_1)),$-$..., \forall(r_i(\bar{x}_j) \Leftrightarrow R^{exp}(\bar{x}_j))\}$ be the set of formulas in $exp(F, i)$ with $F = \forall(r_i(\bar{x}) \Leftrightarrow R(\bar{x}))$.

1. *By completeness of i-instantiations.* For every ground atom $r_i(\bar{x})\phi$ there exist $k \in \{1..j\}$ and ground substitution $\delta$ such that $r_i(\bar{x}_k)\delta = r_i(\bar{x})\phi$.
2. *By definition of unfolding step and normalization rules.* Unfolding steps replace atoms by equivalent formulas and normalization rules replace formulas by equivalent formulas. Let $R^{exp}(\bar{x}_k)\delta$ be the resulting formula after applying unfolding steps and then normalization rules to $R(\bar{x}_k)\delta$.
3. *By ground decidability in* $\mathcal{C}$. $\mathcal{C} \vdash R(\bar{x}_k)\delta \Leftrightarrow R^{exp}(\bar{x}_k)\delta$.
4. *By consistency of* $\mathcal{C}$. $\mathcal{C} \models R(\bar{x}_k)\delta \Leftrightarrow R^{exp}(\bar{x}_k)\delta$.
5. *By 1.* $\mathcal{C} \models R(\bar{x})\phi \Leftrightarrow R^{exp}(\bar{x}_k)\delta$.

*Proof (Theorem 3. Expansion is an Internal Operation in $\Omega_{ext}$).* Let $exp(F, i) = \{F_1, ..., F_j\}$ be the set of formulas in the $i$-expansion of $F = \forall(r_i(\bar{x}) \Leftrightarrow R(\bar{x}))$.

1. *By definition of i-instantiation and unfolding steps.* These steps have only effect on atoms in a formula.
2. *By definition of normalization rules.* These rules are only applied to unfolded sub-formulas in $exp(F, i)$ so, they have no effect on binary logical connectives in $F$ .
3. *By 1. and 2.* Every $rhs(F_k)_{\mathcal{P}}$ is similar wrt connectives to $rhs(F)_{\mathcal{P}}$ with $k = 1..j$.
4. *By absurdum.* $rhs(F)_{\mathcal{P}}$ is similar to the rhs of some pattern in $\Omega_{ext}(r_i)$ and there is some $F_k \in exp(F, i)$ such that $rhs(F_k)_{\mathcal{P}}$ is not similar to the rhs of any pattern in $\Omega_{ext}(r_i)$. However, the following reasoning exposes a contradiction:
   (a) *By definition of i-instantiation.* Every atom pattern occurring in $rhs(F_k)_{\mathcal{P}}$ must be included in its respective covering.
   (b) *By (Well-formed) assertions in* $\mathcal{C}$. Every atom in the rhs of any axiom induces an intermediate or upper pattern in its respective covering so, only intermediate and upper patterns can occur after unfolding steps.
   (c) *By definition of normalization rules.* These rules can not produce new atom patterns.
5. *By 4. and definition of* $\mathcal{L}(r_i)$. Every literal pattern in $rhs(F_k)_{\mathcal{P}}$ must be included necessarily in $\mathcal{L}(r_i)$.
6. *By 3., 5. and definition of* $\Omega_{ext}(r_i)$ $rhs(F_k)_{\mathcal{P}}$ must be similar to the rhs of some element in $\Omega_{ext}(r_i)$.

**Corollary 1 (Rewriting Step Preserves Correctness).** *Let* $rew(F, P_{\mathcal{P}}) = \forall(r_i(\bar{x}) \Leftrightarrow R^{rew}(\bar{x}))$ *be the rewriting step of* $F = \forall(r_i(\bar{x}) \Leftrightarrow R(\bar{x}))$ *wrt* $P_{\mathcal{P}}$. *For every ground atom* $r_i(\bar{x})\phi$,

$$\mathcal{C} \models R(\bar{x})\phi \Leftrightarrow R^{rew}(\bar{x})\phi$$

*Proof (Corollary 1. Rewriting Step Preserves Correctness).*

1. *By consistency of* $\mathcal{C}$. There exists a combination $\{c_1, ..., c_p\}$ of propositions $true$ and $false$ such that $c_i = false$ if $\mathcal{C} \models \neg a_i\phi$ and $c_i = true$ if $\mathcal{C} \models a_i\phi$, with $i = 1..p$.

2. *By definition of Rewriting Step, 1.* Only one disjunction in $R^{rew}(\bar{x})\phi$ includes the sub-formula $R(c_1, ..., c_p, a_{p+1}, ..., a_n)$.
3. *By definition of Rewriting Step, 2.* Simplifications replace sub-formulas in $R(c_1, ..., c_p, a_{p+1}, ..., a_n)$ by equivalent sub-formulas.
4. *By 1., 2. and 3.* $\mathcal{C} \models R^{rew}(\bar{x})\phi \Leftrightarrow R(c_1, ..., c_p, a_{p+1}, ..., a_n)\phi$.
5. *By 1.* $\mathcal{C} \models R(a_1, ..., a_p, a_{p+1}, ..., a_n)\phi \Leftrightarrow R(c_1, ..., c_p, a_{p+1}, ..., a_n)\phi$.
6. *By 4. and 5.* $\mathcal{C} \models R(a_1, ..., a_p, a_{p+1}, ..., a_n)\phi \Leftrightarrow R^{rew}(\bar{x})\phi$, or equivalently, $\mathcal{C} \models R(\bar{x})\phi \Leftrightarrow R^{rew}(\bar{x})\phi$.

*Proof (Theorem 4. Reduction Preserves Correctness).* Let $\forall(r_i(\bar{x}) \Leftrightarrow R^{red}(\bar{x}))$ be the $i$-reduction of an expanded formula $\forall(r_i(\bar{x}) \Leftrightarrow R(\bar{x}))$ wrt $\Omega_{ext}(r_i)$.

1. *By Corollary 1.* For every ground atom $r_i(\bar{x})\phi$, $\mathcal{C} \models R(\bar{x})\phi \Leftrightarrow R^{rew}(\bar{x})\phi$.
2. *By definition of folding step.* Folding steps replace sub-formulas in $R^{rew}(\bar{x})$ by equivalent atoms. Let $R^{red}(\bar{x})$ be the formula obtained from $R^{rew}(\bar{x})$ after folding steps.
3. *By 1. and 2.* For every ground atom $r_i(\bar{x})\phi$, $\mathcal{C} \models R(\bar{x})\phi \Leftrightarrow R^{red}(\bar{x})\phi$.

**Corollary 2.** *Let $P_{\mathcal{P}}$ be a pattern in $\Omega_{ext}(r_i)$ such that $R(\bar{x})_{\mathcal{P}}$ is similar to $E_{\mathcal{P}} = rhs(P_{\mathcal{P}})$ with $f$ as the induced mapping for deciding about similarity wrt connectives and $A = \{a_1, a_2, ..., a_p\}$ as the set of atoms in $R(\bar{x})$ which have not been used for deciding about similarity (i.e. remaining atoms). Every sub-formula $R(c_1, ..., c_p, a_{p+1}, .., a_n)$ in $rew(F, E_{\mathcal{P}})$ is similar to the rhs of some element in $\Omega_{ext}(r_i)$.*

*Proof (Corollary 2).*

1. *By definition of search space $\Omega_{ext}(r_i)$.* Every element in $\Omega_{ext}(r_i)$ presents the same structure of logical connectives so, the rhs of any element in $\Omega_{ext}(r_i)$ is similar wrt connectives to $R(c_1, ..., c_p, a_{p+1}, ..., a_n)$.
2. *By 1. and definition of rewriting step, 2.* After rewriting step 2, $R(c_1, ..., c_p,$- $a_{p+1}, ..., a_n)$ is similar wrt connectives to any element in $\Omega_{ext}(r_i)$.
3. *By definition of similarity.* Every sub-formula to be simplified in $R(c_1, ..., c_p,$- $a_{p+1}, .., a_n)$ is composed of one literal in $\mathcal{L}(r_i)$ and, possibly, $true/false$ propositions and binary logical connectives. Its simplification will produce either the literal it contains or a proposition $true/false$.
4. *By 2., 3. and definition of search space $\Omega_{ext}(r_i)$.* The resulting formula of simplifying $R(c_1, ..., c_p, a_{p+1}, .., a_n)$ will be always included in $\Omega_{ext}(r_i)$.

*Proof (Theorem 5. Forms of Compiled Axioms).*

1. *By Theorem 3.* Every expanded formula is similar to an element in the search space.
2. *By 1. and definition of rewriting step.*
   (a) If the set of remaining atoms is empty in $R(c_1, ..., c_p, a_{p+1}, ..., a_n)$ (i.e. $p = 0$ in $R(c_1, ..., c_p, a_{p+1}, ..., a_n)$) then, *by Corollary 2*, $R(c_1, ..., c_p, a_{p+1}, ..., a_n)_{\mathcal{P}}$ is equal to the rhs of an element in $\Omega_{ext}(r_i)$.

(b) If the set of remaining atoms is not empty (i.e. $p > 0$ in $R(c_1, ..., c_p, a_{p+1}, ..., a_n)$) then, *by Corollary 2*, $R(c_1, ..., c_p, a_{p+1}, ..., a_n)_{\mathcal{P}}$ is similar to the rhs of an element in $\Omega_{ext}(r_i)$ and $G_j(\bar{x})$ is the conjunctive formula which has been obtained after evaluating remaining atoms. *By (Well-formed) assertions in $\mathcal{C}$*. Every atom in the rhs of any axiom induces an intermediate or upper pattern in its respective covering so, only intermediate and upper patterns can occur after unfolding steps so, remaining atoms must present patterns which are included in $\mathcal{L}(r_i)$.

3. *By 2.(a) and definition of folding*. Reduced formula will be of the form $\forall(r_i(\bar{x}) \Leftrightarrow r_j(\bar{x}))$ where $r_j(\bar{x})_{\mathcal{P}}$ is equal to the lhs of an element in $\Omega_{ext}(r_i)$.
4. *By 2.(b) and definition of folding*. Reduced formula will be of the form $\forall(r_i(\bar{x}) \Leftrightarrow \bigvee r_j(\bar{x}) \wedge G_j(\bar{x}))$ where $r_j(\bar{x})_{\mathcal{P}}$ is equal to the lhs of an element in $\Omega_{ext}(r_i)$ and $G_j(\bar{x})$ a conjunctive formula of literals whose patterns are included in $\mathcal{L}(r_i)$.

*Proof (Theorem 6. Termination).*

1. *By Theorem 5*. Each successive increment compiles either an axiom for $r_j$ from $r_j(\bar{x})$ or an axiom for a new assertion from a literal in $G_j(\bar{x})$. No infinite atoms $r_j(\bar{x})$ are possible due to the finite nature of $\Omega_{ext}(r_i)$ and no infinite new literals are possible in $G_j(\bar{x})$ due to the finite nature of atom coverings.

| |
|---|
| Signature: $subset_1^{neg}(e{:}\,\mathcal{N}at, l{:}\,\mathcal{S}eq, e{:}\,\mathcal{N}at, s{:}\,\mathcal{S}eq)$, recursive parameters: $l$ and $s$<br>Specification:<br>(1) $\forall\,(subset_1^{neg}(e, \mathbf{[\,]}, e, \mathbf{[\,]}) \Leftrightarrow subset_5^{neg})$<br>(2) $\forall\,(subset_1^{neg}(e, \mathbf{[\,]}, e, \mathbf{[v|w]}) \Leftrightarrow (subset_6^{neg} \wedge subset_{10}(\mathbf{v}, e) \vee$<br>$subset_2^{neg}(e, \mathbf{w}) \wedge subset_9(\mathbf{v}, e)))$<br>(3) $\forall\,(subset_1^{neg}(e, \mathbf{[x|y]}, e, \mathbf{[\,]}) \Leftrightarrow (subset_7^{neg} \wedge subset_{10}(\mathbf{x}, e) \vee$<br>$subset_3^{neg}(e, \mathbf{y}) \wedge subset_9(\mathbf{x}, e)))$<br>(4) $\forall\,(subset_1^{neg}(e, \mathbf{[x|y]}, e, \mathbf{[v|w]}) \Leftrightarrow (subset_7^{neg} \wedge subset_{10}(\mathbf{x}, e) \wedge subset_{10}(\mathbf{v}, e) \vee$<br>$subset_3^{neg}(e, \mathbf{y}) \wedge subset_9(\mathbf{x}, e) \wedge subset_{10}(\mathbf{v}, e) \vee$<br>$subset_4^{neg}(e, \mathbf{w}) \wedge subset_{10}(\mathbf{x}, e) \wedge subset_9(\mathbf{v}, e) \vee$<br>$subset_1^{neg}(e, \mathbf{y}, e, \mathbf{w}) \wedge subset_9(\mathbf{x}, e) \wedge subset_9(\mathbf{v}, e)))$ |
| Signature: $subset_2^{neg}(e{:}\,\mathcal{N}at, l{:}\,\mathcal{S}eq)$, recursive parameters: $l$<br>Specification:<br>(5) $\forall\,(subset_2^{neg}(e, \mathbf{[\,]}) \Leftrightarrow subset_5^{neg})$<br>(6) $\forall\,(subset_2^{neg}(e, \mathbf{[x|y]}) \Leftrightarrow (subset_6^{neg} \wedge subset_{10}(\mathbf{x}, e) \vee$<br>$subset_2^{neg}(e, \mathbf{y}) \wedge subset_9(\mathbf{x}, e)))$ |
| Signature: $subset_3^{neg}(e{:}\,\mathcal{N}at, l{:}\,\mathcal{S}eq)$, recursive parameters: $l$<br>Specification:<br>(7) $\forall\,(subset_3^{neg}(e, \mathbf{[\,]}) \Leftrightarrow subset_6^{neg})$<br>(8) $\forall\,(subset_3^{neg}(e, \mathbf{[x|y]}) \Leftrightarrow (subset_7^{neg} \wedge subset_{10}(\mathbf{x}, e)$<br>$subset_3^{neg}(e, \mathbf{y}) \wedge subset_9(\mathbf{x}, e)))$ |
| Signature: $subset_4^{neg}(e{:}\,\mathcal{N}at, l{:}\,\mathcal{S}eq)$, recursive parameters: $l$<br>Specification:<br>(9) $\forall\,(subset_4^{neg}(e, \mathbf{[\,]}) \Leftrightarrow subset_8^{neg})$<br>(10) $\forall\,(subset_4^{neg}(e, \mathbf{[x|y]}) \Leftrightarrow (subset_7^{neg} \wedge subset_{10}(\mathbf{x}, e) \vee$<br>$subset_4^{neg}(e, \mathbf{y}) \wedge subset_9(\mathbf{x}, e)))$ |
| Signature: $subset_5^{neg}$, recursive parameters:<br>Specification:<br>(11) $\forall\,(subset^{neg} \Leftrightarrow false \wedge true)$ |
| Signature: $subset_6^{neg}$, recursive parameters:<br>Specification:<br>(12) $\forall\,(subset_6^{neg} \Leftrightarrow false \wedge false)$ |
| Signature: $subset_7^{neg}$, recursive parameters:<br>Specification:<br>(13) $\forall\,(subset_7^{neg} \Leftrightarrow true \wedge false)$ |
| Signature: $subset_8^{neg}$, recursive parameters:<br>Specification:<br>(14) $\forall\,(subset_8^{neg} \Leftrightarrow true \wedge true)$ |
| Signature: $subset_9^{neg}(x{:}\,\mathcal{N}at, y{:}\,\mathcal{N}at)$, recursive parameters: $x$ or $y$<br>Specification:<br>(15) $\forall\,(subset_9^{neg}(\mathbf{0}, \mathbf{0}) \Leftrightarrow false)$<br>(16) $\forall\,(subset_9^{neg}(\mathbf{0}, \mathbf{s(y)}) \Leftrightarrow true)$<br>(17) $\forall\,(subset_9^{neg}(\mathbf{s(x)}, \mathbf{0}) \Leftrightarrow true)$<br>(18) $\forall\,(subset_9^{neg}(\mathbf{s(x)}, \mathbf{s(y)}) \Leftrightarrow subset_9^{neg}(\mathbf{x}, \mathbf{y}))$ |
| Signature: $subset_{10}^{neg}(x{:}\,\mathcal{N}at, y{:}\,\mathcal{N}at)$, recursive parameters: $x$ or $y$<br>Specification:<br>(19) $\forall\,(subset_{10}^{neg}(\mathbf{0}, \mathbf{0}) \Leftrightarrow true)$<br>(20) $\forall\,(subset_{10}^{neg}(\mathbf{0}, \mathbf{s(y)}) \Leftrightarrow false)$<br>(21) $\forall\,(subset_{10}^{neg}(\mathbf{s(x)}, \mathbf{0}) \Leftrightarrow false)$<br>(22) $\forall\,(subset_{10}^{neg}(\mathbf{s(x)}, \mathbf{s(y)}) \Leftrightarrow subset_{10}^{neg}(\mathbf{x}, \mathbf{y}))$ |

**Table 4.** Compilation of the axiom of $subset_1^{neg}$.