A Theoretical Foundation for Programming Languages Aggregation

Stefan Ciobaca, Dorel Lucanu, Vlad Rusu, Grigore Rosu

A Theoretical Foundation for Programming Languages Aggregation *

Ştefan Ciobâcă[1], Dorel Lucanu[1], Vlad Rusu[2], and Grigore Roşu[1,3]

[1] "Alexandru Ioan Cuza" University, Romania
[2] Inria Lille, France
[3] University of Illinois at Urbana-Champaign, USA

Abstract. Programming languages should be formally specified in order to reason about programs written in them. We show that, given two formally specified programming languages, it is possible to construct the formal semantics of an aggregated language, in which programs consist of pairs of programs from the initial languages. The construction is based on algebraic techniques and it can be used to reduce relational properties (such as equivalence of programs) to reachability properties (in the aggregated language).

1 Introduction

In this paper we are concerned with the problem of language aggregation: given two programming languages (in some formalism), construct a new language in which programs consist of pairs of programs from the original languages. Furthermore, a program (P, Q) in the aggregated language should behave as if the programs P and Q (in the initial languages) would run interleaved or in parallel.

The main motivation behind the construction of the aggregated language is to be able to reduce reasoning about relational properties of programs (such as the equivalence of two programs P and Q) to reasoning about a single program (the aggregated program (P, Q)). We have shown [6] for example that partial equivalence of programs reduces to partial correctness in an aggregated language. In general, aggregation is important because there are fewer results and tools for relational properties (e.g. equivalence of programs) than single program properties (e.g. partial correctness). All of our constructions are effective and therefore aggregation can be implemented as a module in a language framework such as K [15].

The main difficulty in aggregating two languages is making sure that there is a link between the datatypes being shared by the two languages. For example, if both languages have variables of type natural numbers, it is important that the naturals be interpreted consistently in the aggregated language in order to be able to express properties such as the equality of a variable in the first

* This paper is supported by the Sectorial Operational Programme Human Resource Development (SOP HRD), financed from the European Social Fund and by the Romanian Government under the contract number POSDRU/159/1.5/S/137750.

language with a variable in the second language. We use known results like pushouts of first-order signatures and amalgamation of first-order models in order to formalize the sharing of information between the two languages. In order to show that the language resulting from the construction is indeed the language we want (i.e. that is has the desirable properties), we have to show several non-trivial results about aggregated configurations (Lemmas 1 and 3) and about the aggregated semantics (Theorems 3, 4 and 5) that were not known before.

In this paper, we show that if the two languages are formalized by their matching logic semantics [14], then the matching logic semantics of the aggregated language can also be constructed. The main advantage of a matching logic semantics is that it allows to faithfully express several operational semantics ([18]) and that a Hoare-like proof system can be obtained for free directly from the semantics [13, 8]. Therefore, our method allows one to reason about relational properties (such as equivalence) of programs written in two potentially different programming languages "for free", starting from the matching logic semantics of the languages.

2 Topmost Matching Logic

Matching logic was introduced by Roşu et al. ([14, 11]) for specifying programming languages and reasoning about programs. In this section, we recall *topmost matching logic*, a subset of the full matching logic theory described in [12]. For simplicity, we use "matching logic" instead of "topmost matching logic" in this paper.

2.1 Signature

A *matching logic signature* (Cfg, S, Σ, Π) extends a many-sorted first order signature (S, Σ, Π) (where S is the set of sorts, Σ is the many-sorted set of function symbols and Π is the many-sorted set of predicate symbols) with a sort $Cfg \in S$ of configurations. By Var we denote the (sorted) set of variables. By $T_s(Var)$ we denote the set of (well-sorted) terms of sort s built from function symbols in Σ and variables in Var. Matching logic signatures are used to define the abstract syntax of programming languages.

Example 1. The signatures $(\texttt{CfgI}, S_I, \Sigma_I, \Pi_I)$ and $(\texttt{CfgF}, S_F, \Sigma_F, \Pi_F)$ in Figure 1 model the syntax of an imperative and, respectively, of a functional programming language, with sorts $S_I = \{\texttt{Int}, \texttt{Id}, \texttt{Exp}, \texttt{ExpI}, \texttt{Stmt}, \texttt{Code}, \texttt{CfgI}\}$ in IMP and sorts $S_F = \{\texttt{Id}, \texttt{Int}, \texttt{Exp}, \texttt{ExpF}, \texttt{Val}, \texttt{CfgF}\}$ in FUN, and function symbols

$$\begin{aligned}
\Sigma_0 &= \{_\texttt{+}_, _\texttt{-}_, _\texttt{*}_, _\texttt{/}_, _\texttt{<}_, _\texttt{==}_\} \cup \\
&\quad \{_\texttt{+}__{Int}, _\texttt{-}__{Int}, _\texttt{*}__{Int}, _\texttt{/}__{Int}, _\texttt{<}__{Int}, _\texttt{<=}__{Int}, _\texttt{==}__{Int}\} \\
\Sigma_I &= \Sigma_0 \cup \{_\texttt{:=}_, \texttt{skip}, _\texttt{;}_, \texttt{if_then_else_}, \texttt{while_do_}, \langle _\texttt{,}_ \rangle\} \\
\Sigma_F &= \Sigma_0 \cup \{__, \texttt{letrec__=_in_}, \texttt{if_then_else_}, \mu_._, \lambda_._, \langle _ \rangle\}.
\end{aligned}$$

```
  Exp::= Id | Int | ExpI + ExpI | ExpI - ExpI | ExpI * ExpI | ExpI / ExpI
           | ExpI < ExpI | ExpI <= ExpI | ExpI == ExpI
ExpI ::= Exp                              ExpF ::= Exp
Stmt ::=  Id := ExpI                             | letrec Id Id = ExpF in ExpF
       | skip | Stmt ; Stmt                      | if ExpF then ExpF else ExpF
       | if ExpI then Stmt else Stmt             | μ Id . ExpF
       | while ExpI do Stmt                      | ExpF ExpF
Code ::=  ExpI | Stmt                     Val  ::=  Int | λ Id . ExpF
CfgI ::= ⟨Code, Map{Id, Int}⟩             CfgF ::= ⟨ExpF⟩
```

Fig. 1. $(\texttt{CfgI}, S_I, \Sigma_I, \Pi_I)$ and $(\texttt{CfgF}, S_F, \Sigma_F, \Pi_F)$, the signatures of IMP and FUN, detailed in Example 1. Only the function symbols are detailed in the figure; the predicates consist of the arithmetic comparison operators: $\Pi_I = \Pi_F = \{=_{Int}, <_{Int}, \leq_{Int}\}$. The difference between the operators $\_\texttt{+}\_$, $\_\texttt{*}\_$, etc. and their correspondants $\_ + \_{Int}$, $\_ * \_{Int}$, etc. is that the former are the syntactic language constructs for addition, etc., while the latter are the actual function symbols denoting integer addition, etc.

The functions above are written in Maude-like notation [4], the underscore ($\_$) denoting the position of an argument. Althought not written explicitly above, the signatures also include the one-argument injections needed to inject sorts like `Int` and `Id` into `ExpI`.

2.2 Syntax

Given a matching logic signature (Cfg, S, Σ, Π), the set of *matching logic formulae* is given by the following grammar:

$$\varphi ::= P(t_1, \ldots, t_n), \neg\varphi, \varphi \wedge \varphi, \exists x.\varphi, \pi,$$

where P ranges over Π, $t_1, \ldots, t_n$ are terms of the appropriate sort for the predicate P, $x \in Var$ is a variable and $\pi \in \mathcal{T}_{Cfg}(Var)$ is a term of sort Cfg.

Matching logic formulae include the classical constructs in first-order logic (predicates $P(t_1, \ldots, t_n)$, negation, conjunction and existential quantifier) and also a new construct π, called *basic pattern*, which allows to use terms of sort Cfg as atomic formulae. We also assume that the other first-order connectives (disjunction - $\vee$, implication - $\rightarrow$, universal quantifier - $\forall$) are available and interpreted as usual as syntactic sugar over the existing connectives.

Example 2. The expression

$$\langle \texttt{while (}E\texttt{)}\ S,\ \texttt{x} \mapsto a\ \texttt{y} \mapsto b \rangle \wedge a <_{Int} b$$

is a $(\texttt{CfgI}, S_I, \Sigma_I, \Pi_I)$-matching logic formula, where $(\texttt{CfgI}, S_I, \Sigma_I, \Pi_I)$ is that given in Figure 1.

We distinguish two particular types of matching logic formulae:

Definition 1. *A matching logic formula is* patternless *if it conforms to the following grammar:*

$$\varphi_{pless} ::= P(t_1, \ldots, t_n), \neg\varphi_{pless}, \varphi_{pless} \wedge \varphi_{pless}, \exists x.\varphi_{pless}.$$

Patternless matching logic formulae simply do not contain basic patterns.

Therefore they can be identified with FOL formulae. The second particular type of formulae we consider are *pure* formulae:

Definition 2. *A matching logic formula is* pure *if it conforms to the following grammar:*

$$\varphi_{pure} ::= \pi, \varphi_{pure} \wedge \varphi_{pless}, \exists x.\varphi_{pure}.$$

Pure formulae contain at least one basic pattern and no basic pattern appears under negation.

2.3 Semantics

We denote by $\mathcal{T}$ a first-order model for the many-sorted first-order signature (S, Σ, Π) that assigns sets to sorts, functions to function symbols and predicates to predicate symbols. By $\mathcal{T}_o$ we denote the interpretation of the object o in the model $\mathcal{T}$. Well-sorted valuations are denoted by $\rho : Var \to \mathcal{T}$. Elements of $\mathcal{T}_{Cfg}$ (the set interpreting the sort of configurations) are denoted by the greek letter $\gamma \in \mathcal{T}_{Cfg}$ and are called *configurations*.

Matching logic formulae are interpreted in the presence of a (first-order) model $\mathcal{T}$, a (well-sorted) valuation ρ and an element $\gamma \in \mathcal{T}_{Cfg}$.

Definition 3. *The satisfaction relation $\models$ for matching logic is defined as follows:*

1. $\mathcal{T}, \gamma, \rho \models P(t_1, \ldots, t_n)$ *if* $(\rho(t_1), \ldots, \rho(t_n)) \in \mathcal{T}_P$;
2. $\mathcal{T}, \gamma, \rho \models \neg\varphi$ *if* $\mathcal{T}, \gamma, \rho \not\models \varphi$;
3. $\mathcal{T}, \gamma, \rho \models \varphi_1 \wedge \varphi_2$ *if* $\mathcal{T}, \gamma, \rho \models \varphi_1$ *and* $\mathcal{T}, \gamma, \rho \models \varphi_2$;
4. $\mathcal{T}, \gamma, \rho \models \exists x.\varphi$, *where x is a variable of sort s, if there exists an element $u \in \mathcal{T}_s$ such that* $\mathcal{T}, \gamma, \rho[x \mapsto u] \models \varphi$;
5. $\mathcal{T}, \gamma, \rho \models \pi$ *for a basic pattern* $\pi \in \mathcal{T}_{Cfg}(Var)$ *if* $\rho(\pi) = \gamma$.

The first four cases are as in first-order logic and the last case (for basic patterns) is new. The semantics of basic patterns is given by *matching* the element γ in the presence of which matching logic formulae are evaluated. This is where the name of *matching logic* comes from. When two basic patterns are connected by a logical and ($\wedge$) in the formula, the element γ has to match both basic patterns, hence $\wedge$ plays the role of intersection. Using basic patterns under implications, logical or and existential/universal quantifiers makes it possible to express several interesting properties, but this is outside the scope of the current article.

Example 3. Let $\mathcal{T}_I$ denote the model for Σ_I that interprets `Int` as the set of integers, the function and the predicate symbols over `Int` with the usual functions and predicates, respectively, and the function symbols corresponding to the BNF productions as term constructors. If

$$\gamma = \langle \texttt{while (x<y) x := x+1;}, \texttt{x} \mapsto 2\ \texttt{y} \mapsto 5 \rangle$$

$\langle x\ ,\ \sigma\rangle \Rightarrow \langle eval(\sigma, x)\ ,\ \sigma\rangle$
$\langle i_1\ \texttt{op}\ i_2\ ,\ \sigma\rangle \Rightarrow \langle i_1\ \texttt{op}_{Int}\ i_2\ ,\ \sigma\rangle$
$\langle X\ \texttt{:=}\ I\ ,\ \sigma\rangle \Rightarrow \langle \texttt{skip}\ ,\ \sigma[I/X]\rangle$
$\langle \texttt{skip;}S\ ,\ \sigma\rangle \Rightarrow \langle S\ ,\ \sigma\rangle$
$\langle \texttt{if I then}\ S_1\ \texttt{else}\ S_2\ ,\ \sigma\rangle \wedge I \neq 0 \Rightarrow \langle S_1\ ,\ \sigma\rangle$
$\langle \texttt{if 0 then}\ S_1\ \texttt{else}\ S_2\ ,\ \sigma\rangle \Rightarrow \langle S_2\ ,\ \sigma\rangle$
$\langle \texttt{while}\ E\ \texttt{do}\ S\ ,\ \sigma\rangle \Rightarrow \langle \texttt{if}\ E\ \texttt{then}\ S;\ \texttt{while}\ E\ \texttt{do}\ S\ \texttt{else skip}\ ,\ \sigma\rangle$
$\langle C[code]\ ,\ \sigma\rangle \Rightarrow \langle C[code']\ ,\ \sigma'\rangle \quad \text{if } \langle code\ ,\ \sigma\rangle \Rightarrow \langle code'\ ,\ \sigma'\rangle$
where $C\ \texttt{::=}\ \_ \mid C\ \texttt{op}\ E \mid i\ \texttt{op}\ C \mid \texttt{if}\ C\ \texttt{then}\ S_1\ \texttt{else}\ S_2 \mid v\ \texttt{:=}\ C \mid C;\ S$

Fig. 2. Specifying the semantics of IMP as a set $\mathcal{A}_I$ of reachability rules (schemata). *op* ranges over the binary function symbols and op_{Int} is their denotation in $\mathcal{T}_I$.

and $\rho(E) = \texttt{x<y}$, $\rho(S) = \texttt{x := x+1;}$, $\rho(a) = 2$, $\rho(b) = 5$ then

$$\mathcal{T}_I, \gamma, \rho \models \langle \texttt{while (E) } S, \texttt{x} \mapsto a\ \texttt{y} \mapsto b\rangle \wedge a <_{Int} b.$$

The Σ_F-model $\mathcal{T}_F$ is defined in a similar way. If

$$\gamma' = \langle \texttt{letrec f x = if (x<1) then 1 else x*f(x-1) in f(5)}\rangle$$

and $\rho'(F) = \texttt{f}$, $\rho'(X) = \texttt{x}$ and

$$\rho'(E_1) = \texttt{if (x<1) then 1 else x*f(x-1)},\ \rho'(E_2) = \texttt{f(5)},$$

then

$$\mathcal{T}_F, \gamma', \rho' \models \langle \texttt{letrec}\ F\ X\ \texttt{=}\ E_1\ \texttt{in}\ E_2\rangle \wedge true.$$

3 Reachability Logic

While matching logic allows to reason about individual configurations (of a program), reachability logic builds on matching logic to allow to reason and define the dynamic behaviour of programs.

3.1 Syntax

Reachability logic formulae are constructed in the presence of a matching logic signature (Cfg, S, Σ, Π) as pairs of matching logic formulae:

Definition 4. *A reachability logic formula (or equivalently, a reachability rule) $\varphi \Rightarrow \varphi'$ is a pair of matching logic formulae.*

The intuition behind reachability formulae is that a configuration matching φ advances into a configuration matching φ'.

Example 4. The set of reachability logic formulas $\mathcal{A}_I$ given in Figure 2 are specifying the semantics of IMP and the set of reachability logic formulas $\mathcal{A}_F$ given in Figure 3 are specifying the semantics of FUN. The specification

$$\langle C[code]\ ,\ \sigma\rangle \Rightarrow \langle C[code']\ ,\ \sigma'\rangle \quad \text{if } \langle code\ ,\ \sigma\rangle \Rightarrow \langle code'\ ,\ \sigma'\rangle\ ,$$

(resp. $\langle C[c]\rangle \Rightarrow \langle C[c']\rangle \quad \text{if } \langle c\rangle \Rightarrow \langle c'\rangle$) is a rule schemata that defines an infinite set of reachability logic formulas.

$\langle I_1 \texttt{op}\, I_2 \rangle \Rightarrow \langle I_1 \texttt{op}_{Int} I_2 \rangle$
$\langle \texttt{if I then}\ E_1\ \texttt{else}\ E_2 \rangle \wedge i \neq 0 \Rightarrow \langle E_1 \rangle$
$\langle \texttt{if 0 then}\ E_1\ \texttt{else}\ E_2 \rangle \Rightarrow \langle E_2 \rangle$
$\langle \texttt{letrec}\ F\ X = E\ \texttt{in}\ E' \rangle \Rightarrow \langle E'[f \mapsto (\mu F.\lambda X.E)] \rangle$
$\langle (\lambda X.E)\ V \rangle \Rightarrow \langle E[V/X] \rangle$
$\langle \mu X.E \rangle \Rightarrow \langle E[X \mapsto (\mu X.E)] \rangle$
$\langle C[c] \rangle \Rightarrow \langle C[c'] \rangle \quad \text{if } \langle c \rangle \Rightarrow \langle c' \rangle$

where $C\ ::=\ \_ \mid C\ \texttt{op}\ e \mid \texttt{if}\ C\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \mid C\ e \mid v\ C$

Fig. 3. Specifying the semantics of FUN as a set $\mathcal{A}_F$ of reachability rules schemata. *op* ranges over the binary function symbols and op_{Int} is their denotation in $\mathcal{T}_F$

3.2 Semantics

Reachability logic formulae are interpreted in the presence of a first-order model $\mathcal{T}$ and a (well-sorted) valuation $\rho : Var \to \mathcal{T}$ as in the case of matching logic, and also in the presence of a transition relation $\longrightarrow \subseteq \mathcal{T}_{Cfg} \times \mathcal{T}_{Cfg}$ over the set of configurations $\mathcal{T}_{Cfg}$.

Intuitively, a reachability logic formula $\varphi \Rightarrow \varphi'$ holds if any configuration matched by φ reaches (in one step) a configuration matched by the formula φ'. Note that the one-step requirement we work with in this article defines a satisfaction relation that is different from the ones used in the previous presentations [13, 8] of reachability logic. The satisfaction relation defined here is mainly used to specify transition systems.

Definition 5. *Formally, the satisfaction relation $\models$ of reachability logic is defined as follows:*

$$\mathcal{T}, \longrightarrow, \rho \models \varphi \Rightarrow \varphi'$$

if for any $\gamma \in \mathcal{T}_{Cfg}$ such that $\mathcal{T}, \gamma, \rho \models \varphi$ there exists a $\gamma' \in \mathcal{T}_{Cfg}$ such that $\gamma \longrightarrow \gamma'$ and $\mathcal{T}, \gamma', \rho \models \varphi'$.

Note that the free variables of φ and the free variables of φ' are "shared" in the reachability formula, in the sense that both are interpreted by the same valuation.

Example 5. If

$$\langle \texttt{if (1) then x=x+y else x := 0}, \texttt{x} \mapsto 2\ \texttt{y} \mapsto 5 \rangle \Rightarrow \langle \texttt{x=x+y}, \texttt{x} \mapsto 2\ \texttt{y} \mapsto 5 \rangle$$

and $\varphi \Rightarrow \varphi'$ is

$$\langle \texttt{if}\ (I)\ \texttt{then}\ S_1\ \texttt{else}\ S_2, \sigma \rangle \wedge I \neq_{Int} 0 \Rightarrow \langle S_1, \sigma \rangle,$$

then we have $\mathcal{T}_I, \longrightarrow, \rho \models \varphi \Rightarrow \varphi'$, where $\rho(I) = \texttt{1}$, $\rho(S_1) = \texttt{x := x+y}$, $\rho(S_2) = \texttt{x := 0}$, and $\rho(\sigma) = \texttt{x} \mapsto 2\ \texttt{y} \mapsto 5$.

If the valuation ρ is missing, the free variables of reachability rules are interpreted universally:

Definition 6. *Given a first-order model $\mathcal{T}$ and a transition relation $\longrightarrow$, we say that the pair $(\mathcal{T}, \longrightarrow)$ is a model of $\varphi \Rightarrow \varphi'$, written*

$$\mathcal{T}, \longrightarrow \models \varphi \Rightarrow \varphi'$$

if, for all (well-sorted) valuations $\rho : Var \to \mathcal{T}$, we have that $\mathcal{T}, \longrightarrow, \rho \models \varphi \Rightarrow \varphi'$.

The universal interpretation of the free variables is justified in the following section, but note that it is not unusual to do so: for example, the same happens with first-oder clauses, where the variables are (implicitly) universally quantified.

4 Language Semantics

In this section, we show that reachability formulae can be used to formally define the operational semantics of a programming language. We consider that a matching logic signature (Cfg, S, Σ, Π) is fixed.

Definition 7. *A (Cfg, S, Σ, Π)-programming language is a pair $(\mathcal{T}, \longrightarrow)$ of a first-order model $\mathcal{T}$ of (Cfg, S, Σ, Π) and a transition relation $\longrightarrow \subseteq \mathcal{T}_{Cfg} \times \mathcal{T}_{Cfg}$.*

When (Cfg, S, Σ, Π) is understood from the context, we omit it and write programming language instead of (Cfg, S, Σ, Π)-programming language.

The matching logic signature defines the syntax of the language, the model $\mathcal{T}$ mainly defines program configurations and $\longrightarrow$ defines the (one-step) transition relation between configurations. Let A be a set of reachability formulae (the axioms of the language).

Definition 8. *We say that $(\mathcal{T}, \longrightarrow)$ is a* model *of A, and we write $(\mathcal{T}, \longrightarrow) \models A$ if $(\mathcal{T}, \longrightarrow) \models \varphi \Rightarrow \varphi'$ for every reachability formula $\varphi \Rightarrow \varphi' \in A$.*

Example 6. If $(\mathcal{T}_I, \longrightarrow_I)$ gives the semantics of IMP, then we have $(\mathcal{T}_I, \longrightarrow_I) \models \mathcal{A}_I$, i.e., $(\mathcal{T}_I, \longrightarrow_I)$ is a model of $\mathcal{A}_I$. Similarly, if $(\mathcal{T}_F, \longrightarrow_F)$ gives the semantics of FUN, then we must have $(\mathcal{T}_F, \longrightarrow_F) \models \mathcal{A}_F$, i.e., $(\mathcal{T}_F, \longrightarrow_F)$ is a model of $\mathcal{A}_F$.

Therefore, the set A of reachability rules are considered to be the specification (the formal semantics) of any language $(\mathcal{T}, \longrightarrow)$ that is a model of the rules. In [18] it is shown that any operational semantics (small-step SOS, big-step SOS, reduction contexts, etc) can be faithfully captured by a (possibly infinite) set of reachability rules. Moreover, all such reachability rules from the semantics are *pure*:

Definition 9. *A reachability formula $\varphi \Rightarrow \varphi'$ is* pure *if both φ and φ' are pure.*

From here on, we assume that the formal semantics of any language is given as a (possibly infinite) set of pure reachability formulae.

5 Language Aggregation

We assume two signatures $(Cfg_1, S_1, \Sigma_1, \Pi_1)$ and $(Cfg_2, S_2, \Sigma_2, \Pi_2)$ for two languages $(\mathcal{T}_1, \longrightarrow_1)$ and $(\mathcal{T}_2, \longrightarrow_2)$ specified by the sets A_1 and A_2 of reachability rules.

In this section, we construct the aggregated signature (Cfg, S, Σ, Π) of the aggregated language from the signature of the first language and the signature of the second language. Also, we define the aggregated language $(\mathcal{T}, \longrightarrow)$ itself and show how to constructively give the aggregated axioms A of the new language from the initial languages.

5.1 Signature Aggregation

This subsection is dedicated to showing how to construct the aggregated signature (Cfg, S, Σ, Π) from the individual signature $(Cfg_1, S_1, \Sigma_1, \Pi_1)$ (of the first language) and $(Cfg_2, S_2, \Sigma_2, \Pi_2)$ (of the second language).

The most delicate part is to make sure that the sorts, function and predicate symbols "shared" between the two language are identified in the aggregated configuration, even if their names are not the same in the first and in the second language.

Therefore, we assume that there exists a first-order signature (S_0, Σ_0, Π_0) that the two languages have in common. This means there exist two morphisms $h_1 : (S_0, \Sigma_0, \Pi_0) \to (S_1, \Sigma_1, \Pi_1)$ and $h_2 : (S_0, \Sigma_0, \Pi_0) \to (S_2, \Sigma_2, \Pi_2)$.

Example 7. For the signatures of the two languages, IMP and FUN, we have $S_0 = \{\texttt{Int}, \texttt{Id}, \texttt{Exp}\}$, Σ_0 that was defined on Page 1, and $\Pi_0 = \Pi_1, \Pi_2$. The morphisms h_1 and h_2 are given by the component inclusions.

The following theorem (the pushout theorem) allows us to combine the two signatures into a single signature, while identifying the objects shared between them. This result is not new, see, for example, [9].

Theorem 1 (Pushout of Signatures). *Let (S_1, Σ_1, Π_1), (S_2, Σ_2, Π_2) and (S_0, Σ_0, Π_0) be many-sorted FOL signatures, h_1 a morphism from (S_0, Σ_0, Π_0) to (S_1, Σ_1, Π_1) and h_2 a morphism from (S_0, Σ_0, Π_0) to (S_2, Σ_2, Π_2).*

Then the diagram $(S_1, \Sigma_1, \Pi_1) \xleftarrow{h_1} (S_0, \Sigma_0, \Pi_0) \xrightarrow{h_2} (S_2, \Sigma_2, \Pi_2)$ admits a pushout, i.e., there exists a tuple $(h'_1, (S', \Sigma', \Pi'), h'_2)$ with h'_1 a morphism from (S_1, Σ_1, Π_1) to (S', Σ', Π') and h'_2 a morphism from (S_2, Σ_2, Π_2) to (S', Σ', Π') such that:

1. *(commutativity) $h'_1(h_1(x)) = h'_2(h_2(x))$ for any object x from the signature (S_0, Σ_0, Π_0) and*
2. *(minimality) if there exist (S'', Σ'', Π'') and morphisms h''_1 from (S_1, Σ_1, Π_1) to (S'', Σ'', Π'') and h''_2 from (S_2, Σ_2, Π_2) to (S'', Σ'', Π'') with $h''_1(h_1(x)) = h''_2(h_2(x))$ for all $x \in S_0, \cup \Sigma_0 \cup \Pi_0$, then there exists a morphism h from (S', Σ', Π') to (S'', Σ'', Π'').*

$$\begin{array}{ccc} (S_0, \Sigma_0, \Pi_0) & \xrightarrow{h_2} & (S_2, \Sigma_2, \Pi_2) \\ {\scriptstyle h_1}\downarrow & & \downarrow{\scriptstyle h_2'} \\ (S_1, \Sigma_1, \Pi_1) & \xrightarrow{h_1'} & (S', \Sigma', \Pi') \end{array}$$

Fig. 4. Push-out diagram assumed throughout the paper.

Furthermore, the pushout is unique (up to isomorphisms). The push-out is summarised in Figure 4.

The first step to obtain the aggregated signature is to apply the push-out theorem in order to obtain the intermediate signature (S', Σ', Π') and the two morphisms $h_1' : (S_1, \Sigma_1, \Pi_1) \to (S', \Sigma', \Pi')$ and $h_2' : (S_2, \Sigma_2, \Pi_2) \to (S', \Sigma', \Pi')$.

The first-order signature (S', Σ', Π') contains all of the objects from the initial signatures (properly renamed to account for shared objects), but it does not yet have a sort for aggregated configurations. Let $\mathit{Cfg}' = h_1(\mathit{Cfg}_1)$ and $\mathit{Cfg}_2' = h_2(\mathit{Cfg}_2)$ be the names of the sorts of configurations in the new signature.

We therefore choose a fresh sort Cfg for aggregated configurations and we let $S = S' \uplus \{\mathit{Cfg}\}$ be the set of sorts. The signature Σ contains, in addition to the symbols in Σ', a pairing symbol and the respective projections. Formally,

$$\Sigma = \Sigma' \uplus \{\langle \_, \_ \rangle : \mathit{Cfg}_1' \times \mathit{Cfg}_2' \to \mathit{Cfg}, pr_1 : \mathit{Cfg} \to \mathit{Cfg}_1', pr_2 : \mathit{Cfg} \to \mathit{Cfg}_2'\}.$$

The pairing symbol $\langle \_, \_ \rangle$ takes as input two configurations of the initial languages and returns a configuration of the aggregated language. The projection operations pr_1 and pr_2 take an aggregated configuration and deconstruct it into the initial configurations. Finally, we let $\Pi = \Pi'$.

The signature $(\mathit{Cfg}, S, \Sigma, \Pi)$ is the aggregated signature of the language and its construction is summarised in Figure 5.

If $h : (S, \Sigma, \Pi) \to (S', \Sigma', \Pi')$ is a morphism between the two first order signatures, we extend h to terms as expected: if $t \in T_s(\Sigma)$, then $h(t) \in T_{h(s)}(\Sigma')$. We also extend h to transform matching logic formula in the signature $(\mathit{Cfg}, S, \Sigma, \Pi)$ to matching logic formula in the signature $(h(\mathit{Cfg}), S', \Sigma', \Pi')$ as follows:

1. $h(\varphi_1 \wedge \varphi_2) = h(\varphi_1) \wedge h(\varphi_2)$,
2. $h(\exists x.\varphi_1) = \exists x.h(\varphi_1)$,
3. $h(\neg\varphi_1) = \neg h(\varphi_1)$ and
4. $h(P(t_1, \ldots, t_n)) = P(h(t_1), \ldots, h(t_n)))$.

Note that there is no need to have a case for computing $h(\pi)$, since a basic pattern π is nothing but a term. Therefore $h(\pi)$ is already defined. We also extend h to transform reachability formulae over the signature $(\mathit{Cfg}, S, \Sigma, \Pi)$ into reachability formulae over the signature $(h(\mathit{Cfg}), S', \Sigma', \Pi')$:

$$h(\varphi \Rightarrow \varphi') = h(\varphi) \Rightarrow h(\varphi').$$

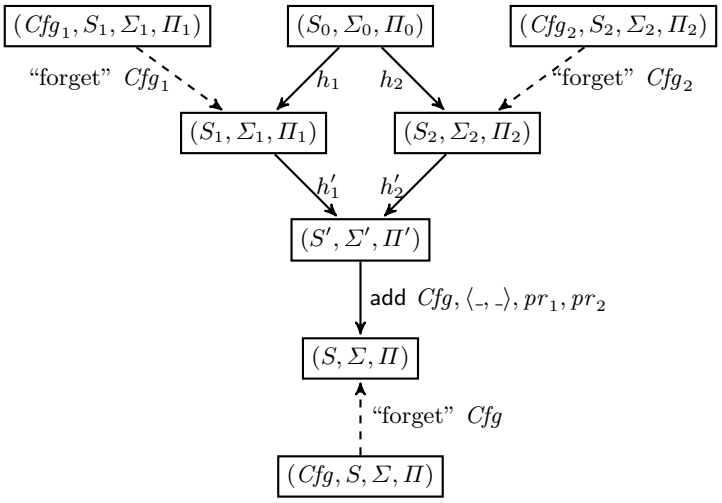

Fig. 5. The aggregation of the two signatures.

5.2 Model Amalgamation

In this subsection, given two models $\mathcal{T}_1$ and $\mathcal{T}_2$ for the matching logic signatures $(Cfg_1, S_1, \Sigma_1, \Pi_1)$ and $(Cfg_2, S_2, \Sigma_2, \Pi_2)$, we show how to construct a model $\mathcal{T}$ for the aggregated signature (Cfg, S, Σ, Π) above.

In order to construct such a model, we need to make sure that the two models $\mathcal{T}_1$ and $\mathcal{T}_2$ agree on the common part of the signature. Formally, we assume that there exists a model $\mathcal{T}_0$ of the signature (S_0, Σ_0, Π_0) such that $\mathcal{T}_1{\restriction_{h_1}} = \mathcal{T}_0 = \mathcal{T}_2{\restriction_{h_2}}$ (i.e. the reduct of $\mathcal{T}_1$ through h_1 is the same as the reduct of $\mathcal{T}_2$ through h_2). Figure 6 below summarizes the construction of $\mathcal{T}$.

We can combine the two languages through the model amalgamation theorem. The theorem is not new (see for example [17]). A proof can also be found in our technical report [5].

Theorem 2 (Amalgamation).
If $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_0$ are models of (S_1, Σ_1, Π_1), (S_2, Σ_2, Π_2) and respectively (S_0, Σ_0, Π_0) such that $\mathcal{T}_1{\restriction_{h_1}} = \mathcal{T}_0 = \mathcal{T}_2{\restriction_{h_2}}$, there exists a unique model $\mathcal{T}'$ of (S', Σ', Π') such that $\mathcal{T}'{\restriction_{h'_2}} = \mathcal{T}_2$ and $\mathcal{T}'{\restriction_{h'_1}} = \mathcal{T}_1$.

In order to obtain the model $\mathcal{T}$ for the aggregated signature (Cfg, S, Σ, Π), we need to augment $\mathcal{T}'$ to interpret the *Cfg* sort, the pairing symbol $\langle \_, \_ \rangle$ and the projection symbols pr_1 and pr_2. Formally, we define $\mathcal{T}$ as follows:

1. $\mathcal{T}_{Cfg} = \mathcal{T}'_{Cfg'_1} \times \mathcal{T}'_{Cfg'_2}$,
2. $\mathcal{T}_{\langle \_, \_ \rangle}(\gamma_1, \gamma_2) = (\gamma_1, \gamma_2)$ for any $\gamma_1 \in \mathcal{T}'_{Cfg'_1}$ and any $\gamma_2 \in \mathcal{T}'_{Cfg'_2}$,
3. $\mathcal{T}_{pr_1}((\gamma_1, \gamma_2)) = \gamma_1$ for any $(\gamma_1, \gamma_2) \in \mathcal{T}_{Cfg} = \mathcal{T}'_{Cfg'_1} \times \mathcal{T}'_{Cfg'_2}$,
4. $\mathcal{T}_{pr_2}((\gamma_1, \gamma_2)) = \gamma_2$ for any $(\gamma_1, \gamma_2) \in \mathcal{T}_{Cfg} = \mathcal{T}'_{Cfg'_1} \times \mathcal{T}'_{Cfg'_2}$, and

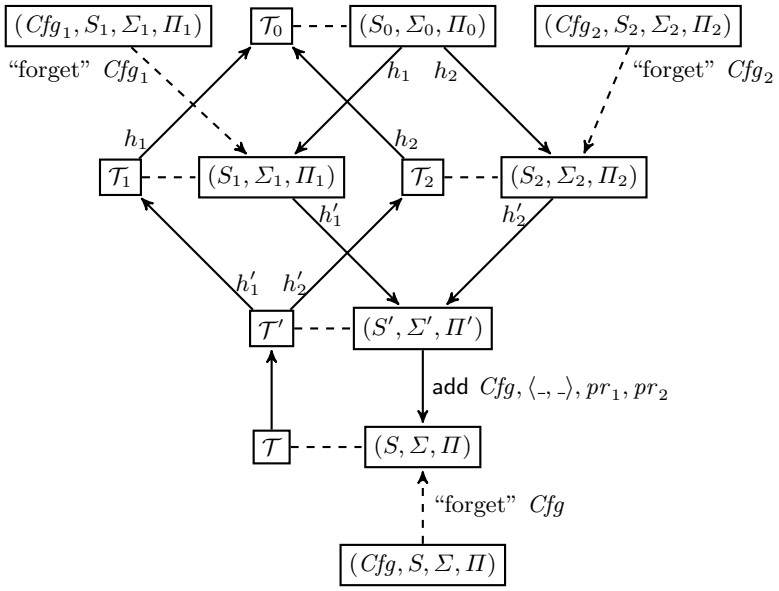

Fig. 6. Construction of amalgamated model

5. $\mathcal{T}_o = \mathcal{T}'_o$ for any other object $o \in S \cup \Sigma \cup \Pi$.

The aggregated signature and model have the following properties:

Lemma 1. *For any basic patterns $\pi_1 \in T_{Cfg_1}(Var)$ and $\pi_2 \in T_{Cfg_2}(Var)$, for any aggregated configuration $(\gamma_1, \gamma_2) \in \mathcal{T}_{Cfg}$, for any (well-sorted) valuation ρ, we have that*

$$\mathcal{T}, (\gamma_1, \gamma_2), \rho \models \langle \pi_1, \pi_2 \rangle \text{ iff } \mathcal{T}_1, \gamma_1, \rho\restriction_{h'_1(S_1)} \models \pi_1 \text{ and } \mathcal{T}_2, \gamma_2, \rho\restriction_{h'_2(S_2)} \models \pi_2.$$

Proof. We show the direct implication first. Assume $\mathcal{T}, (\gamma_1, \gamma_2), \rho \models \langle \pi_1, \pi_2 \rangle$. By definition, we have that $\rho(\langle \pi_1, \pi_2 \rangle) = (\gamma_1, \gamma_2)$. By the interpretation of $\langle \_, \_ \rangle$ in the model $\mathcal{T}$, we have that $\rho\restriction_{h'_1(S_1)}(\pi_1) = \gamma_1$ and $\rho\restriction_{h'_2(S_2)}(\pi_2) = \gamma_2$. But this is equivalent to $\mathcal{T}_1, \gamma_1, \rho\restriction_{h'_1(S_1)} \models \pi_1$ and $\mathcal{T}_2, \gamma_2, \rho\restriction_{h'_2(S_2)} \models \pi_2$, which is what we had to show.

For the reverse direction, we assume that $\mathcal{T}_1, \gamma_1, \rho\restriction_{h'_1(S_1)} \models \pi_1$ and that $\mathcal{T}_2, \gamma_2, \rho\restriction_{h'_2(S_2)} \models \pi_2$. Therefore $\rho\restriction_{h'_1(S_1)}(\pi_1) = \pi_1$ and $\rho\restriction_{h'_2(S_2)}(\pi_2) = \pi_2$, which implies, by the interpretation of $\langle \_, \_ \rangle$ in $\mathcal{T}$, that $\rho(\langle \pi_1, \pi_2 \rangle) = (\gamma_1, \gamma_2)$. But this is equivalent to $\mathcal{T}, (\gamma_1, \gamma_2), \rho \models \langle \pi_1, \pi_2 \rangle$, which is what we had to prove.

5.3 Language Aggregation

Having shown how to construct the matching logic signature (Cfg, S, Σ, Π) and model $\mathcal{T}$ for the aggregated language, we now show how the transition relation

$\longrightarrow$ for the aggregated language is defined and how to construct the axioms A of the aggregated language from the axioms A_1 and A_2 of the initial languages.

We identify three types of language aggregations, depending on how the $\longrightarrow$ transition relation is defined from $\longrightarrow_1$ and $\longrightarrow_2$. Each of the three constructions could be useful in various contexts:

1. $\longrightarrow_1 \otimes_a \longrightarrow_2$ is the **a**synchronous interleaving product of the two transition relations, i.e.
$$(\gamma_1, \gamma_2) \longrightarrow (\gamma_1', \gamma_2') \text{ if}$$
$$\gamma_1 = \gamma_1' \text{ and } \gamma_2 \longrightarrow_2 \gamma_2' \text{ or } \gamma_1 \longrightarrow_1 \gamma_1' \text{ and } \gamma_2 = \gamma_2',$$
2. $\longrightarrow_1 \otimes_p \longrightarrow_2$ is the **p**arallel product of the two transition relations, i.e.
$$(\gamma_1, \gamma_2) \longrightarrow (\gamma_1', \gamma_2') \text{ if}$$
$$\gamma_1 \longrightarrow_1 \gamma_1' \text{ and } \gamma_2 \longrightarrow_2 \gamma_2',$$
3. finally, $\longrightarrow_1 \otimes \longrightarrow_2 = (\longrightarrow_1 \otimes_a \longrightarrow_2) \cup (\longrightarrow_1 \otimes_p \longrightarrow_2)$ is the (general) product of $\longrightarrow_1$ and $\longrightarrow_2$.

The asynchronous product with interleaving semantics means that in one step of the aggregated language, either the left-hand side takes a step (in the first language) or the right-hand side takes a step (in the second language). The parallel product forces both sides to take steps simultaneously. The (general) product requires at least one side to take a step and it allows (but not requires) the other side to do the same.

5.4 Constructing the Axioms for the Three Products

We next show how to construct a set of axioms A for the aggregated language from the set of axioms A_1 and A_2 of the initial languages, depending on which of the three constructions is chosen for the aggregated transition relation. The main result is that, for each of the three constructions, the transition relation is a model of the aggregated axioms. This means that we can construct the formal semantics of the aggregated language directly from the formal semantics of the initial languages.

Let A_1 be a set of pure reachability rules over the signature $(Cfg_1, S_1, \Sigma_1, \Pi_1)$ that capture the semantics of the first language (the axioms of the first language): $\mathcal{T}_1, \longrightarrow_1 \models A_1$. Let A_2 be a set of pure reachability rules over the signature $(Cfg_2, S_2, \Sigma_2, \Pi_2)$ that capture the semantics of the second language (the axioms of the second language): $\mathcal{T}_2, \longrightarrow_2 \models A_2$.

In order to define the axioms for the aggregated language, we need a way to transform reachability formulae from the two initial signatures into reachability formulae of the target signature. This is performed with the help of the following function:

Definition 10. *We define the function ι_x^i (for $i \in \{1,2\}$ and x a distinguished variable in Var of sort Cfg_i) that takes as input a matching logic formula over $(Cfg_1, S_1, \Sigma_1, \Pi_1)$ (respectively $(Cfg_2, S_2, \Sigma_2, \Pi_2)$) and changes all basic patterns π into $\langle \pi, x\rangle$ (respectively $\langle x, \pi\rangle$) in order to obtain a formula over (Cfg, S, Σ, Π):*

1. $\iota_x^1(\pi) = \langle h_1'(\pi), x\rangle$
2. $\iota_x^2(\pi) = \langle x, h_2'(\pi)\rangle$
3. $\iota_x^i(\varphi_1 \wedge \varphi_2) = \iota_x^i(\varphi_1) \wedge \iota_x^i(\varphi_2)$,
4. $\iota_x^i(\exists y.\varphi_1) = \exists y.\iota_x^i(\varphi_1)$,
5. $\iota_x^i(\neg\varphi_1) = \neg\iota_x^i(\varphi_1)$ *and*
6. $\iota_x^i(P(t_1, \ldots, t_n)) = P(h_i'(t_1), \ldots, h_i'(t_n))$.

Example 8. If φ_1 is the matching formula $\langle \texttt{while (}E\texttt{)}\ S, \sigma\rangle \wedge \mathit{eval}(\sigma, E) \neq_{Int} 0$ and φ_2 is $\langle \texttt{letrec f x = if (}I\texttt{) then }E_1\texttt{ else }E_2\texttt{ in f(x)}\rangle \wedge I <_{Int} 5$, then $\iota_x^i(\varphi_1)$ is $\langle\langle \texttt{while (}E\texttt{)}\ S, \sigma\rangle, x\rangle \wedge \mathit{eval}(\sigma, E) \neq_{Int} 0$ and $\iota_x^i(\varphi_2)$ is the aggregate formula $\langle x, \langle \texttt{letrec f x = if (}I\texttt{) then }E_1\texttt{ else }E_2\texttt{ in f(x)}\rangle\rangle \wedge I <_{Int} 5$.

Next, we show the link, in terms of matching logic formulae, between the aggregated model and the initial model.

Lemma 2. *Let φ_1 and φ_2 be matching logic formulae over $(Cfg_1, S_1, \Sigma_1, \Pi_1)$ and respectively $(Cfg_2, S_2, \Sigma_2, \Pi_2)$. For any aggregated configuration $(\gamma_1, \gamma_2) \in \mathcal{T}_{Cfg}$, for any (well-sorted) valuation ρ, if x is a fresh variable, we have that*

$$\mathcal{T}, (\gamma_1, \gamma_2), \rho \models \iota_x^i(\varphi_i) \text{ iff } \mathcal{T}_i, \gamma_i, \rho\restriction_{h_i'(S_i)} \models \varphi_i.$$

Proof. For simplicity, we assume that $i = 1$ (since the case with $i = 2$ is analogous). We prove the lemma by structural induction on φ_1. We only show the case of negation, the other cases being similar:

– if $\varphi_1 = \neg\varphi_1'$, then

$$\begin{array}{l}\mathcal{T}, (\gamma_1, \gamma_2), \rho \models \iota_x^1(\varphi_1) \text{ iff}\\ \mathcal{T}, (\gamma_1, \gamma_2), \rho \models \neg\iota_x^1(\varphi_1') \text{ iff}\\ \mathcal{T}, (\gamma_1, \gamma_2), \rho \not\models \iota_x^1(\varphi_1') \text{ iff}\\ \mathcal{T}_1, \gamma_1, \rho\restriction_{h_1'(S_1)} \not\models \varphi_1' \text{ iff}\\ \mathcal{T}_1, \gamma_1, \rho\restriction_{h_1'(S_1)} \models \neg\varphi_1'\end{array}$$

Lemma 3. *For any pure matching logic formulae φ_1 over $(Cfg_1, S_1, \Sigma_1, \Pi_1)$ and φ_2 over $(Cfg_2, S_2, \Sigma_2, \Pi_2)$, for any aggregated configuration $(\gamma_1, \gamma_2) \in \mathcal{T}_{Cfg}$, for any (well-sorted) valuation ρ, if x is a fresh variable, we have that*

$$\mathcal{T}, (\gamma_1, \gamma_2), \rho \models \iota_x^i(\varphi_i) \text{ iff } \mathcal{T}_i, \gamma_i, \rho\restriction_{h_i'(S_i)} \models \varphi_i \text{ and } \rho(x) = \gamma_{3-i}.$$

Proof. For simplicity, we assume that $i = 1$ (since the case with $i = 2$ is analogous). We prove the lemma by structural induction on φ_1, showing that

$$\mathcal{T}, (\gamma_1, \gamma_2), \rho \models \iota_x^1(\varphi_1) \text{ iff } \mathcal{T}_1, \gamma_1, \rho\restriction_{h_1'(S_1)} \models \varphi_1 \text{ and } \rho(x) = \gamma_2.$$

We distinguish the following cases:

1. if $\varphi_1 = \pi$, then $\iota^1_x(\varphi_1) = (h'_1(\pi), x)$. We have that

$$\begin{array}{ll}
\mathcal{T}, (\gamma_1, \gamma_2), \rho \models \iota^1_x(\varphi_1) & \text{iff} \\
\mathcal{T}, (\gamma_1, \gamma_2), \rho \models (h'_1(\pi), x) & \text{iff} \\
(\gamma_1, \gamma_2) = \rho(h'_1(\pi), x) & \text{iff} \\
\gamma_1 = \rho(h'_1(\pi)) \text{ and } \gamma_2 = \rho(x) & \text{iff} \\
\gamma_1 = \rho\restriction_{h'_1(S_1)}(\pi) \text{ and } \gamma_2 = \rho(x) & \text{iff} \\
\mathcal{T}_1, \gamma_1, \rho\restriction_{h'_1(S_1)} \models \pi \text{ and } \gamma_2 = \rho(x). &
\end{array}$$

2. if $\varphi_1 = \exists y.\varphi'_1$ for a variable y of sort s (with $y \neq x$ since x is a fresh variable) then $\iota^1_x(\varphi_1) = \exists y.\iota^1_x(\varphi'_1)$. We have that

$$\begin{array}{ll}
\mathcal{T}, (\gamma_1, \gamma_2), \rho \models \iota^1_x(\varphi_1) & \text{iff} \\
\mathcal{T}, (\gamma_1, \gamma_2), \rho \models \exists y.\iota^1_x(\varphi'_1) & \text{iff} \\
\text{there is } u \in \mathcal{T}_{1s} \text{ s.t. } \mathcal{T}, (\gamma_1, \gamma_2), \rho[y \mapsto u] \models \iota^1_x(\varphi'_1) & \text{iff} \\
\text{there is } u \in \mathcal{T}_{1s} \text{ s.t. } \mathcal{T}_1, \gamma_1, \rho[y \mapsto u]\restriction_{h'_1(S_1)} \models \varphi'_1 \text{ and } \gamma_2 = \rho[y \mapsto u](x) & \text{iff} \\
\mathcal{T}_1, \gamma_1, \rho\restriction_{h'_1(S_1)} \models \exists y.\varphi'_1 \text{ and } \gamma_2 = \rho(x) & \text{iff} \\
\mathcal{T}_1, \gamma_1, \rho\restriction_{h'_1(S_1)} \models \varphi_1 \text{ and } \gamma_2 = \rho(x). &
\end{array}$$

3. if $\varphi_1 = \varphi'_1 \wedge \varphi'_2$ for a pure formula φ'_1 and a patternless formula φ'_2, then $\iota^1_x(\varphi_1) = \iota^1_x(\varphi'_1) \wedge \iota^1_x(\varphi'_2)$. We have that

$$\begin{array}{ll}
\mathcal{T}, (\gamma_1, \gamma_2), \rho \models \iota^1_x(\varphi_1) & \text{iff} \\
\mathcal{T}, (\gamma_1, \gamma_2), \rho \models \iota^1_x(\varphi'_1) \wedge \iota^1_x(\varphi'_2) & \text{iff} \\
\mathcal{T}, (\gamma_1, \gamma_2), \rho \models \iota^1_x(\varphi'_1) \text{ and } \mathcal{T}, (\gamma_1, \gamma_2), \rho \models \iota^1_x(\varphi'_2) & \text{iff} \\
\mathcal{T}_1, \gamma_1, \rho\restriction_{h'_1(S_1)} \models \varphi'_1 \text{ and } \rho(x) = \gamma_2 \text{ and } \mathcal{T}_1, \gamma_1, \rho\restriction_{h'_1(S_1)} \models \varphi'_2 & \text{iff} \\
\mathcal{T}_1, \gamma_1, \rho\restriction_{h'_1(S_1)} \models \varphi'_1 \wedge \varphi'_2 \text{ and } \rho(x) = \gamma_2 & \text{iff} \\
\mathcal{T}_1, \gamma_1, \rho\restriction_{h'_1(S_1)} \models \varphi_1 \text{ and } \rho(x) = \gamma_2. &
\end{array}$$

In the following subsections, we define three types of aggregations for these sets of axioms, for each type of language aggregation.

The Axioms for the Asynchronous Product with Interleaving Semantics We let

$$\begin{aligned}
A_1 \otimes_a A_2 = \{\iota^1_y(\varphi_1) \Rightarrow \iota^1_y(\varphi'_1) \mid \varphi_1 \Rightarrow \varphi'_1 \in A_1\} \cup \\
\{\iota^2_x(\varphi_2) \Rightarrow \iota^2_x(\varphi'_2) \mid \varphi_2 \Rightarrow \varphi'_2 \in A_2\}
\end{aligned}$$

where x is a fresh variable of sort Cfg'_1 and y is a fresh variable of sort Cfg'_2. The intuition is that x captures any left-hand side and allow the right-hand to take a step while y captures any right-hand side and allow the left-hand side to take a step. We show formally that $A_1 \otimes_a A_2$ is indeed a formal specification of the language $(\mathcal{T}, \longrightarrow_1 \otimes_a \longrightarrow_2)$:

Theorem 3 (correctness for asynchronous product).
Let $A = A_1 \otimes_a A_2$ and $\longrightarrow = \longrightarrow_1 \otimes_a \longrightarrow_2$. We have that

$$(\mathcal{T}, \longrightarrow) \models A.$$

Proof. We have to show that $(\mathcal{T}, \longrightarrow) \models A$. By definition, $(\mathcal{T}, \longrightarrow) \models A$ if, for any reachability rule $\varphi \Rightarrow \varphi' \in A$, we have that $(\mathcal{T}, \longrightarrow) \models \varphi \Rightarrow \varphi'$. Let $\varphi \Rightarrow \varphi' \in A$ be an arbitrary reachability rule. We show that $(\mathcal{T}, \longrightarrow) \models \varphi \Rightarrow \varphi'$.

As $\varphi \Rightarrow \varphi' \in A = A_1 \otimes_a A_2$, it follows that $\varphi \Rightarrow \varphi' = \iota^1_y(\varphi_1) \Rightarrow \iota^1_y(\varphi'_1)$ for some $\varphi_1 \Rightarrow \varphi'_1 \in A_1$ and a fresh variable y or that $\varphi \Rightarrow \varphi' = \iota^2_x(\varphi_2) \Rightarrow \iota^2_x(\varphi'_2)$ for some $\varphi_2 \Rightarrow \varphi'_2 \in A_2$ and a fresh variable x. Since the two cases are analogous, we deal only with the first and we assume therefore that $\varphi \Rightarrow \varphi' = \iota^1_y(\varphi_1) \Rightarrow \iota^1_y(\varphi'_1)$ for some $\varphi_1 \Rightarrow \varphi'_1 \in A_1$ and a fresh variable y. Therefore it remains to show that $(\mathcal{T}, \longrightarrow) \models \iota^1_y(\varphi_1) \Rightarrow \iota^1_y(\varphi'_1)$.

Let ρ be an arbitrary valuation. Let $(\gamma_1, \gamma_2) \in \mathcal{T}_{Cfg}$ be an arbitrary configuration such that $\mathcal{T}, (\gamma_1, \gamma_2), \rho \models \iota^1_y(\varphi_1)$. By Lemma 3, we have $\mathcal{T}_1, \gamma_1, \rho\restriction_{h'_1(S_1)} \models \varphi_1$ and $\rho(x) = \gamma_2$. Given that $\varphi_1 \Rightarrow \varphi'_1 \in A_1$ and $\mathcal{T}_1, \longrightarrow_1 \models A_1$, there exists $\gamma'_1 \in \mathcal{T}_{1\,Cfg}$ such that $\gamma_1 \longrightarrow_1 \gamma'_1$ and $\mathcal{T}_1, \gamma'_1, \rho\restriction_{h'_1(S_1)} \models \varphi'_1$. By Lemma 3, we have that $\mathcal{T}, (\gamma'_1, \gamma_2), \rho \models \iota^1_y(\varphi'_1)$. But, by the definition of $\longrightarrow$ ($\longrightarrow = \longrightarrow_1 \otimes_a \longrightarrow_2$), we also have that $(\gamma_1, \gamma_2) \longrightarrow (\gamma'_1, \gamma_2)$.

We have shown that for an arbitrary valuation ρ and for an arbitrary configuration $(\gamma_1, \gamma_2) \in \mathcal{T}_{Cfg}$ such that $\mathcal{T}, (\gamma_1, \gamma_2), \rho \models \iota^1_y(\varphi_1)$, there exists a configuration (γ'_1, γ_2) such that $(\gamma_1, \gamma_2) \longrightarrow (\gamma'_1, \gamma_2)$ and $\mathcal{T}, (\gamma'_1, \gamma_2), \rho \models \iota^1_y(\varphi'_1)$. But this means that $(\mathcal{T}, \longrightarrow) \models \iota^1_y(\varphi_1) \Rightarrow \iota^1_y(\varphi'_1)$, which is what we had to show.

The Axioms for the Parallel Product We let

$$A_1 \otimes_p A_2 = \{\iota^1_y(\varphi_1) \wedge \iota^2_x(\varphi_2) \Rightarrow \exists x.\exists y.(\iota^1_y(\varphi'_1) \wedge \iota^2_x(\varphi'_2)) \mid \varphi_1 \Rightarrow \varphi'_1 \in A_1, \varphi_2 \Rightarrow \varphi'_2 \in A_2\}$$

where x is a fresh variable of sort Cfg'_1 and y is a fresh variable of sort Cfg'_2. The intuition is that x captures any left-hand side and y captures any right-hand side. We show formally that $A_1 \otimes_p A_2$ is indeed a formal specification of the language $(\mathcal{T}, \longrightarrow_1 \otimes_p \longrightarrow_2)$:

Theorem 4 (correctness for parallel product).
Let $A = A_1 \otimes_p A_2$ and $\longrightarrow = \longrightarrow_1 \otimes_p \longrightarrow_2$. We have that

$$(\mathcal{T}, \longrightarrow) \models A.$$

Proof. We have to show that $(\mathcal{T}, \longrightarrow) \models A$. By definition, $(\mathcal{T}, \longrightarrow) \models A$ if, for any reachability rule $\varphi \Rightarrow \varphi' \in A$, we have that $(\mathcal{T}, \longrightarrow) \models \varphi \Rightarrow \varphi'$. Let $\varphi \Rightarrow \varphi' \in A$ be an arbitrary reachability rule. We show that $(\mathcal{T}, \longrightarrow) \models \varphi \Rightarrow \varphi'$.

As $\varphi \Rightarrow \varphi' \in A = A_1 \otimes_p A_2$, it follows that there exist $\varphi_1 \Rightarrow \varphi_1 \in A_1$, $\varphi_2 \Rightarrow \varphi'_2 \in A_2$, fresh variables x and y such that $\varphi \Rightarrow \varphi' = \iota^1_y(\varphi_1) \wedge \iota^2_x(\varphi_2) \Rightarrow \iota^1_y(\varphi'_1) \wedge \iota^2_x(\varphi'_2)$.

Let ρ be an arbitrary valuation. Let (γ_1, γ_2) be an arbitrary configuration such that $\mathcal{T}, (\gamma_1, \gamma_2), \rho \models \iota^1_y(\varphi_1) \wedge \iota^2_x(\varphi_2)$. By the semantics of $\wedge$, we have that $\mathcal{T}, (\gamma_1, \gamma_2), \rho \models \iota^1_y(\varphi_1)$ and $\mathcal{T}, (\gamma_1, \gamma_2), \rho \models \iota^2_x(\varphi_2)$. By Lemma 3 it follows that $\mathcal{T}_1, \gamma_1, \rho\restriction_{h'_1(S_1)} \models \varphi_1$, $\rho(y) = \gamma_2$, $\mathcal{T}_2, \gamma_2, \rho\restriction_{h'_2(S_2)} \models \varphi_2$ and $\rho(x) = \gamma_1$.

Because $\mathcal{T}_1, \gamma_1, \rho\restriction_{h'_1(S_1)} \models \varphi_1$, $\varphi_1 \Rightarrow \varphi'_1 \in A_1$ and $\mathcal{T}_1, \longrightarrow_1 \models A_1$, we obtain that there exists γ'_1 such that $\gamma_1 \longrightarrow_1 \gamma'_1$ and $\mathcal{T}_1, \gamma'_1, \rho\restriction_{h'_1(S_1)} \models \varphi'_1$. Similarly, there exists γ'_2 such that $\gamma_2 \longrightarrow_2 \gamma'_2$ and $\mathcal{T}_2, \gamma'_2, \rho\restriction_{h'_2(S_2)} \models \varphi'_2$.

By Lemma 3, it follows that $\mathcal{T}, (\gamma'_1, \gamma'_2), \rho[x \mapsto \gamma_1][y \mapsto \gamma_2] \models \iota^1_y(\varphi'_1)$ and $\mathcal{T}, (\gamma'_1, \gamma'_2), \rho[x \mapsto \gamma_1][y \mapsto \gamma_2] \models \iota^2_y(\varphi'_2)$, which implies that $\mathcal{T}, (\gamma'_1, \gamma'_2), \rho \models \exists x.\exists y.(\iota^1_y(\varphi'_1) \wedge \iota^2_y(\varphi'_2))$. By the definition of $\longrightarrow$, we also have $(\gamma_1, \gamma_2) \longrightarrow (\gamma'_1, \gamma'_2)$.

We have started with an arbitrary valuation ρ and an arbitrary configuration (γ_1, γ_2) such that $\mathcal{T}, (\gamma_1, \gamma_2), \rho \models \iota^1_y(\varphi_1) \wedge \iota^2_x(\varphi_2)$ and we have shown that there exists a configuration (γ'_1, γ'_2) such that $(\gamma_1, \gamma_2) \longrightarrow (\gamma'_1, \gamma'_2)$ and $\mathcal{T}, (\gamma'_1, \gamma'_2), \rho \models \exists x.\exists y.\iota^1_y(\varphi'_1) \wedge \iota^2_y(\varphi'_2)$. Therefore $(\mathcal{T}, \longrightarrow) \models \iota^1_y(\varphi_1) \wedge \iota^2_x(\varphi_2) \Rightarrow \exists x.\exists y.\iota^1_y(\varphi'_1) \wedge \iota^2_y(\varphi'_2$, which is what we had to show.

The Axioms for the General Product For the general product, which allows for both interleaving and parallel steps, we define

$$A_1 \otimes A_2 = A_1 \otimes_a A_2 \cup A_1 \otimes_p A_2.$$

The correctness result for the general product follows quickly from Theorem 3 and Theorem 4.

Theorem 5 (correctness for the general product).
Let $A = A_1 \otimes A_2$ and $\longrightarrow = \longrightarrow_1 \otimes \longrightarrow_2$. We have that

$$(\mathcal{T}, \longrightarrow) \models A.$$

Proof. Let $\varphi \Rightarrow \varphi' \in A$ be an arbitrary rule. We show that $(\mathcal{T}, \longrightarrow) \models \varphi \Rightarrow \varphi'$. Let $\gamma \in \mathcal{T}_{Cfg}$ be an arbitrary configuration and let ρ be an arbitrary valuation such that $\mathcal{T}, \gamma, \rho \models \varphi$.

We distinguish two cases:

1. if $\varphi \Rightarrow \varphi' \in A_1 \otimes_a A_2$, then, by Theorem 3, there exists γ' such that $\gamma(\longrightarrow_1 \otimes_a \longrightarrow_2)\gamma'$ and $\mathcal{T}, \gamma', \rho \models \varphi'$. But, by definition, $\longrightarrow_1 \otimes_a \longrightarrow_2 \subseteq \longrightarrow$. Therefore there exists γ' such that $\gamma \longrightarrow \gamma'$ and $\mathcal{T}, \gamma', \rho \models \varphi'$, which implies $(\mathcal{T}, \longrightarrow) \models \varphi \Rightarrow \varphi'$.
2. if $\varphi \Rightarrow \varphi' \in A_1 \otimes_p A_2$, then, by Theorem 4, there exists γ' such that $\gamma(\longrightarrow_1 \otimes_p \longrightarrow_2)\gamma'$ and $\mathcal{T}, \gamma', \rho \models \varphi'$. But, by definition, $\longrightarrow_1 \otimes_p \longrightarrow_2 \subseteq \longrightarrow$. Therefore there exists γ' such that $\gamma \longrightarrow \gamma'$ and $\mathcal{T}, \gamma', \rho \models \varphi'$, which implies $(\mathcal{T}, \longrightarrow) \models \varphi \Rightarrow \varphi'$.

6 Conclusion and Future Work

In this paper we have shown that if two programming languages are defined using reachability logic axioms and matching-logic based semantics, then we can effectively construct products of the two languages, such that a pair of programs belonging to the product can be executed asynchronously with interleaving, in

parallel (synchronised), or a combination of the two. The construction can be automated in definitional frameworks like $\mathbb{K}$ [15].

A programming language definition consists of a signature and a semantics. In our approach, the signature is a many-sorted first-order signature, that includes both the syntax of the programming languages and the data structures required by the semantics. The category of the many sorted first-order signatures has colimits, in particular pushouts. Moreover, this category has the amalgamation property [9, 16]. The semantics of the programming languages is given by transitions systems. The category of the transition systems has also several nice constructions [19]. We combine these constructions in order to get the definition for aggregated languages. The approach is flexible enough to allow various aggregations. The semantics of a programming language can be specified with (one-step) reachability logic formulae. We show that the specification of the aggregated language can be obtained from the specifications of the components.

We used many-sorted first-order signatures in order to make the presentation easier to follow. However, the syntax of programming languages is usually given by BNF rules, which correspond to order-sorted first-order signatures. Unfortunately, order-sorted first-order logic does not have pushouts of signatures and the amalgamation property. There are several approaches dealing with this issue, see, e.g., [17, 1, 10]. It is challenging to see which one of these is the best candidate for programming languages products and this will be investigated in the future. It is also interesting to see if the formalisation of the matching logic and reachability logic as institutions [2, 3] could help.

An interesting observation can be made for the case where $\longrightarrow = \longrightarrow_1 \otimes \longrightarrow_2$. The transition system $(\mathcal{T}, \longrightarrow)$ is a product in the category of the transitions systems [19]. So, the syntax of the aggregation of the language is defined by a pushout (which is a shared sum) and the semantics by a product.

Language aggregation has uses in proving equivalence properties [7, 6]. We intend to explore its use in proving other kinds of relations and in compiler verification.

References

1. M. Alpuente, S. Escobar, J. Meseguer, and P. Ojeda. Order-Sorted Generalization. *Electr. Notes Theor. Comput. Sci.*, 246:27–38, 2009.
2. C. E. Chiriţă. An Institutional Foundation for the K Semantic Framework. Master's thesis, University of Bucharest, 2014.
3. C. E. Chiriţă and T. F. Şerbănuţă. An Institutional Foundation for the K Semantic Framework. In *this volume*.
4. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott. *All About Maude, A High-Performance Logical Framework*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.
5. Ş. Ciobâcă, D. Lucanu, V. Rusu, and G. Roşu. A Language-Independent Proof System for Mutual Program Equivalence. Technical Report 14-01, Al. I. Cuza Univ., 2014.

6. Ştefan Ciobâcă. Reducing Partial Equivalence to Partial Correctness. In *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2014 16th International Symposium on*, pages 164–171, Sept 2014.
7. Ştefan Ciobâcă, D. Lucanu, V. Rusu, and G. Roşu. A Language-Independent Proof System for Mutual Program Equivalence. In *ICFEM 2014, Luxembourg, Luxembourg, November 3-5, 2014. Proceedings*, pages 75–90, 2014.
8. A. Ştefanescu, Ştefan Ciobâcă, R. Mereuta, B. M. Moore, T. Serbanuta, and G. Roşu. All-Path Reachability Logic. In *RTA-TLCA 2014, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, pages 425–440, 2014.
9. R. Diaconescu. *Institution-independent Model Theory*. Birkhauser Basel, 2008.
10. A. E. Haxthausen and F. Nickl. Pushouts of Order-Sorted Algebraic Specifications. In *AMAST*, pages 132–147, 1996.
11. G. Roşu. Matching Logic: A Logic for Structural Reasoning. Technical Report http://hdl.handle.net/2142/47004, University of Illinois, Jan 2014.
12. G. Roşu. Matching Logic (invited talk). In *26th International Conference on Rewriting Techniques and Applications, RTA 2015, 29 June - 1 July, Warsaw, Poland*, 2015. to appear.
13. G. Roşu, A. Ştefanescu, Ştefan Ciobâcă, and B. M. Moore. One-Path Reachability Logic. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 358–367, 2013.
14. G. Roşu, C. Ellison, and W. Schulte. Matching Logic: An Alternative to Hoare/Floyd Logic. In *AMAST, LNCS 6486*, pages 142–162, 2010.
15. G. Roşu and T.-F. Şerbănuţă. An overview of the K semantic framework. *J. Log. Algebr. Program.*, 79(6):397–434, 2010.
16. D. Sannella and A. Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2012.
17. L. Schröder, T. Mossakowski, A. Tarlecki, P. Hoffman, and B. Klin. Amalgamation in the semantics of CASL. *Theoretical Computer Science*, 331(1):215–247, 2005.
18. T.-F. Şerbănuţă, G. Roşu, and J. Meseguer. A rewriting logic approach to operational semantics. *Inf. Comput.*, 207(2):305–340, 2009.
19. G. Winskel and M. Nielsen. Categories in Concurrency. In *Semantics and Logics of Computation, Publications of the Newton Institute*, pages 299–354. Cambridge University Press, 1997.