

Programming Language Aggregation with Applications in Equivalence Checking [★]

Ștefan Ciobâcă¹, Dorel Lucanu¹, Vlad Rusu², and Grigore Roșu^{1,3}

¹ “Alexandru Ioan Cuza” University, Romania

² Inria Lille, France

³ University of Illinois at Urbana-Champaign, USA

Abstract. We show that, given the operational semantics of two programming languages L and R , it is possible to construct an aggregate language, in which programs consist of pairs of programs from L and respectively R . In the aggregate language, a program $P = (P_L, P_R)$ takes a step from either P_L or P_R . The main difficulty is how to aggregate the two languages so that data such as integers or booleans that are common to both languages has the same representation in the aggregate language. The aggregation is based on the pushout theorem, which allows us to construct a model of the aggregate language from models of the initial languages, while making sure that the interpretation of common data such as integers is the same.

The main application of the aggregation result is in equivalence checking. It is possible to check for example that two programs P_L and P_R (written in L and respectively R) compute the same result by checking the partial correctness of the pair (P_L, P_R) in the aggregate language.

1 Introduction and Preliminaries

This article explains how to construct a single language that is capable of executing pairs of programs written in the two languages. The challenge is how to achieve that generically, where the two languages are given by their formal semantics, without relying on the specifics of any particular language. The aggregated language must be capable of independently “executing” pairs of programs in the original languages.

Let S be a set of **sorts**, Σ an S -sorted algebraic signature (i.e., an indexed set $\Sigma = \cup_{w \in S^*, s \in S} \Sigma_{w,s}$, where $\Sigma_{w,s}$ is the set of function symbols of arity w with a result of sort s) and Π an indexed set $\Pi = \cup_{w \in S^*} \Pi_w$ of predicate symbols. Then $\Phi = (S, \Sigma, \Pi)$ is called a **many-sorted FOL signature**.

Definition 1. Let h be a morphism from $\Phi = (S, \Sigma, \Pi)$ to $\Phi' = (S', \Sigma', \Pi')$ and $\mathcal{T}'$ be a model of $\Phi' = (S', \Sigma', \Pi')$. We define $\mathcal{T}'|_h$ (the **reduct** of $\mathcal{T}'$ through h) to be the model of Φ such that:

1. $\llbracket s \rrbracket_{\mathcal{T}'|_h} = \llbracket h(s) \rrbracket_{\mathcal{T}'}$ for all $s \in S$.
2. $\llbracket f \rrbracket_{\mathcal{T}'|_h}(e_1, \dots, e_n) = \llbracket h(f) \rrbracket_{\mathcal{T}'}(e_1, \dots, e_n)$ for all $f \in \Sigma_{s_1, \dots, s_n, s}$ and for all $e_1 \in \llbracket s_1 \rrbracket_{\mathcal{T}'|_h}, \dots, e_n \in \llbracket s_n \rrbracket_{\mathcal{T}'|_h}$.

[★] 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.

3. $(e_1, \dots, e_n) \in \llbracket p \rrbracket_{\mathcal{T}' \upharpoonright_h}$ iff $(e_1, \dots, e_n) \in \llbracket h(p) \rrbracket_{\mathcal{T}'}$ for all $p \in \Pi_{s_1, \dots, s_n}$ and for all $e_1 \in \llbracket h(s_1) \rrbracket_{\mathcal{T}'}, \dots, e_n \in \llbracket h(s_n) \rrbracket_{\mathcal{T}'}$.

Theorem 1 (Pushout).

Let Φ_R, Φ_L and Φ_0 be three FOL signatures, h_R a morphism from Φ_0 to Φ_R and h_L a morphism from Φ_0 to Φ_L . There exists a tuple (h'_L, Φ', h'_R) , called the **pushout** of

$$\Phi_L \xleftarrow{h_L} \Phi_0 \xrightarrow{h_R} \Phi_R,$$

where h'_L is a morphism from Φ_L to Φ' and h'_R a morphism from Φ_R to Φ' such that the following conditions hold:

1. (commutativity) $h'_L(h_L(x)) = h'_R(h_R(x))$ for all $x \in \Phi_0$ and
2. (minimality) if there exist Φ'' and morphisms h''_L (from Φ_L to Φ'') and h''_R (from Φ_R to Φ'') such that $h''_L(h_L(x)) = h''_R(h_R(x))$ for all $x \in \Phi_0$ then there exists a morphism h from Φ' to Φ'' .

Furthermore, the pushout is unique (up to renaming).

$$\begin{array}{ccc} (S_0, \Sigma_0, \Pi_0) & \xrightarrow{h_R} & (S_R, \Sigma_R, \Pi_R) \\ h_L \downarrow & & \downarrow h'_R \\ (S_L, \Sigma_L, \Pi_L) & \xrightarrow{h'_L} & (S', \Sigma', \Pi') \end{array}$$

Fig. 1. Push-out diagram assumed throughout the paper.

Theorem 2. If $\mathcal{T}_R, \mathcal{T}_L, \mathcal{T}_0$ are models of $\Phi_R = (S_R, \Sigma_R, \Pi_R), \Phi_L = (S_L, \Sigma_L, \Pi_L)$ and respectively $\Phi_0 = (S_0, \Sigma_0, \Pi_0)$ such that $\mathcal{T}_R \upharpoonright_{h_R} = \mathcal{T}_L \upharpoonright_{h_L} = \mathcal{T}_0$, there exists a unique model $\mathcal{T}'$ of Φ' such that $\mathcal{T}' \upharpoonright_{h'_L} = \mathcal{T}_L$ and $\mathcal{T}' \upharpoonright_{h'_R} = \mathcal{T}_R$.

2 Matching Logic Syntax and Semantics

We introduce notation used throughout the paper and discuss the recently introduced *matching logic* [3, 2], a language-parametric logic for reasoning about program configurations, and its use in language semantics. Matching logic extends FOL with *basic patterns*, which are open terms (i.e., terms with variables) that can be used as basic formulae in the logic.

Definition 2. A *matching logic signature* is a tuple (Cf, S, Σ, Π) , where the tuple (S, Σ, Π) is a FOL signature and $Cf \in S$ is a distinguished sort.

Matching logic formulae extend FOL formulae with terms of sort Cf as atomic formulae called **basic patterns**:

Definition 3. Given a matching logic signature (Cfg, S, Σ, Π) , the following are **matching logic formulae** (ML formulae) over (Cfg, S, Σ, Π) and the set of sorted variables Var :

$$\varphi ::= \neg\varphi, \varphi \wedge \varphi, \exists x.\varphi, \pi$$

where $\pi \in T_{Cfg}(Var)$ and $x \in Var$.

Models of ML signatures are simply FOL models. In what follows, we fix a model $\mathcal{T}$ of (Cfg, S, Σ, Π) . Elements of $\llbracket Cfg \rrbracket_{\mathcal{T}}$ are called concrete **configurations**. We represent concrete configurations by $\gamma, \gamma', \gamma_1$ and variations thereof. Valuations $\rho : Var \rightarrow \mathcal{T}$ of matching logic are simply valuations of the corresponding first order logic. The satisfaction relation of matching logic is defined between pairs (γ, ρ) of configurations and valuations and ML formulae φ as follows:

Definition 4. The **matching logic satisfaction** relation $\models$ (written as $(\gamma, \rho) \models \varphi$ and read as (γ, ρ) is a **model** of φ) is defined inductively as follows (missing cases are the same as in FOL):

1. $(\gamma, \rho) \models \neg\varphi'$ if it is not true that $(\gamma, \rho) \models \varphi'$
2. $(\gamma, \rho) \models \pi$, where π is a basic pattern if $\rho(\pi) = e$.

Definition 5. A **matching logic semantic domain** for a language is a tuple $(Cfg, S, \Sigma, \Pi, \mathcal{T})$, where (Cfg, S, Σ, Π) is a matching logic signature and $\mathcal{T}$ a matching logic model of (Cfg, S, Σ, Π) .

Definition 6. A **reachability rule** is a pair $\varphi \Rightarrow \varphi'$ of matching logic formulae.

Definition 7. A **matching logic semantics** for a language is a tuple $(Cfg, \Sigma, \Pi, \mathcal{T}, \mathcal{A}, \rightarrow_{\mathcal{T}})$, where $(Cfg, \Sigma, \Pi, \mathcal{T})$ is matching logic semantic domain, $\mathcal{A}$ a set of reachability rules and $\rightarrow_{\mathcal{T}}$ is the transition system generated by $\mathcal{A}$ on $\mathcal{T}$, that is, $\rightarrow_{\mathcal{T}} \subseteq \mathcal{T}_{Cfg} \times \mathcal{T}_{Cfg}$ with $\gamma \rightarrow_{\mathcal{T}} \gamma'$ iff there exist $\varphi \Rightarrow \varphi' \in \mathcal{A}$ and ρ such that $(\gamma, \rho) \models \varphi$ and $(\gamma', \rho) \models \varphi'$.

As discussed in [3], conventional operational semantics of programming languages can be regarded as matching logic semantics: Σ includes the abstract syntax of the language as well as the syntax of the various operations in the needed mathematical domains; $\mathcal{A}$ is the (possibly infinite) set of operational semantics rules of the language; $\mathcal{T}$ is the model of configurations of the language merged together with the needed mathematical domains, and the relation $\rightarrow_{\mathcal{T}}$ is precisely the transition relation defined by the operational semantics.

3 Aggregation of Matching Logic Semantics

Let $\mathcal{S}_i = (Cfg_i, S_i, \Sigma_i, \Pi_i, \mathcal{T}_i)$, $i \in \{L, R\}$ be the matching logic semantic domains of two languages, (S_0, Σ_0, Π_0) a matching logic signature, h_L and h_R morphisms from (S_0, Σ_0, Π_0) to (S_L, Σ_L, Π_L) and from (S_0, Σ_0, Π_0) to (S_R, Σ_R, Π_R) . Let $\mathcal{T}_L, \mathcal{T}_R, \mathcal{T}_0$ be models of (S_L, Σ_L, Π_L) , (S_R, Σ_R, Π_R) and respectively (S_0, Σ_0, Π_0)

such that $T_L \upharpoonright_{h_L} = \mathcal{T}_0 = T_R \upharpoonright_{h_R}$. Let $(h'_L, (S', \Sigma', \Pi'), h'_R)$ be the pushout of $(S_L, \Sigma_L, \Pi_L) \xleftarrow{h_L} (S_0, \Sigma_0, \Pi_0) \xrightarrow{h_R} (S_R, \Sigma_R, \Pi_R)$. By Theorem 2, there exists a unique (S', Σ', Π') -model $\mathcal{T}'$ such that $\mathcal{T}' \upharpoonright_{h'_L} = \mathcal{T}_L$ and $\mathcal{T}' \upharpoonright_{h'_R} = \mathcal{T}_R$. We now define the **aggregation** of the two matching logic semantic domains. We let $\mathcal{S} = (Cfg, S, \Sigma, \Pi, \mathcal{T})$, where

- Cfg is a new distinguished sort;
- $S = S' \cup \{Cfg\}$
- $\Sigma = \Sigma' \cup \{\langle _, _ \rangle : h_L(Cfg_L) \times h_R(Cfg_R) \rightarrow Cfg, pr_i : Cfg \rightarrow Cfg_i, i \in \{L, R\}\}$;
- $\Pi = \Pi'$;
- $\mathcal{T}_{Cfg} = \mathcal{T}'_{h'_L(Cfg_L)} \times \mathcal{T}'_{h'_R(Cfg_R)}$
- $\mathcal{T}_{\langle _, _ \rangle}(\gamma_L, \gamma_R) = (\gamma_L, \gamma_R)$, $\mathcal{T}_{pr_L}((\gamma_L, \gamma_R)) = \gamma_L$, $\mathcal{T}_{pr_R}((\gamma_L, \gamma_R)) = \gamma_R$.
- $\mathcal{T}_o = \mathcal{T}'_o$ for any other object $o \in S \cup \Sigma \cup \Pi$.

Note that in the aggregated model, sorts such as integers, strings, booleans, etc that are common in both languages are interpreted by the same set.

4 Applications, Discussion, Related Work and Conclusion

The aggregated language is useful, for example, to prove equivalence of programs. If we have a pair of programs $\langle S_L, S_R \rangle$ that both compute the sum of the first n natural numbers:

```

 $S_L \equiv s := 0; \text{ for } i := 1 \text{ to } n \text{ do } s := s + i$ 
 $S_R \equiv \text{let sum } i = \text{if } i == 0 \text{ then } 0 \text{ else } i + \text{sum } (i - 1) \text{ in sum } n$ 

```

We can show that S_L and S_R are equivalent (compute the same result) by showing a partial correctness property of the aggregated program $\langle S_L, S_R \rangle$: $\langle S_L, S_R \rangle \Rightarrow \langle s \mapsto n(n+1)/2, n(n+1)/2 \rangle$. Showing partial correctness can be done using reachability logic [4] or Hoare-like logics over the aggregated language. Based on the aggregated language, we have also developed a method of proving mutual equivalence [1].

References

1. Ștefan Ciobăcă, D. Lucanu, V. Rusu, and G. Roșu. A Language-Independent Proof System for Mutual Program Equivalence (revisited). Technical Report TR 14-01, UAIC Iași, 2014.
2. G. Roșu, C. Ellison, and W. Schulte. Matching logic: An alternative to Hoare/Floyd logic. In *AMAST, LNCS 6486*, pages 142–162, 2010.
3. G. Roșu and A. Stefanescu. Checking reachability using matching logic. In *OOP-SLA*, pages 555–574. ACM, 2012.
4. G. Rosu, A. Stefanescu, Ștefan Ciobăcă, and B. M. Moore. One-path reachability logic. In *LICS*, pages 358–367, 2013.