Conditional Constructors – The Matching Calculus

Marko Martin
Technische Universität Darmstadt
MarkoMartin@gmx.net

Mira Mezini
Technische Universität Darmstadt
mezini@cs.tu-darmstadt.de

1. Formalization of Constructor Matching

1.1 Definitions and Notations

We first define the formal elements which will be the basis of the matching calculus:

The set of possible expressions of the target language is denoted by E.

The set of valid variable labels is denoted by Lab.

The set of types is denoted by T.

The mapping of expressions to their most specific type is denoted by function $\tau : E \to T$.

The subtype relation is denoted by $\leq \subseteq T \times T$. It is reflexive, asymmetric, and transitive.

Formal parameters are denoted by pairs $(l, t) \in Lab \times T$; for such pairs, we write $l : t$.

The set of possible single argument patterns is denoted by P and defined as follows. The first element of each pair in P indicates the pattern type: Type 1 are expressions, type 2 are named expressions, and type 3 are template arguments with the specified label.

$$\begin{aligned} P := &\{ (1, e) | \, e \in E \} \cup \\ &\{ (2, p) | \, p \in Lab \times E \} \cup \\ &\{ (3, l) | \, l \in Lab \} \end{aligned}$$

To facilitate reading the calculus rules, we introduce the following notations: For $(1, e) \in P$, we write only e; for $(2, (l, e)) \in P$, we write $l : e$; for $(3, l) \in P$, we write $l*$.

An instance of the constructor matching problem is defined as the input to the matching procedure, i.e., the argument patterns and the formal parameters against which the patterns should be matched, denoted by a pair $((p_1, ..., p_n), (l_1 : t_1, ..., l_k : t_k)) \in P^* \times (Lab \times T)^*$.

For easing readability, we write instead

$$p_1, ..., p_n \doteq l_1 : t_1, ..., l_k : t_k$$

Finally, a few auxiliary definitions. We define the union on partial functions $\mathrm{f}, \mathrm{g} : A \rightharpoonup B$ as follows:

$$\mathrm{f} \cup \mathrm{g} : A \rightharpoonup B : x \mapsto \begin{cases} \mathrm{f}(x) & \text{if } x \in \mathcal{D}(\mathrm{f}) \\ \mathrm{g}(x) & \text{if } x \in \mathcal{D}(\mathrm{g}) \text{ and } x \notin \mathcal{D}(\mathrm{f}) \\ \text{undefined} & \text{otherwise} \end{cases}$$

$\mathcal{D}(\mathrm{f})$ denotes the set of values for which the partial function f is defined. For the empty partial function f with $\mathcal{D}(\mathrm{f}) = \emptyset$, we write $\emptyset$. For the partial function f with finite $\mathcal{D}(\mathrm{f}) = \{x_1, ..., x_n\}$ and values $v_i = \mathrm{f}(x_i)$, we write $\{x_1/v_1, ..., x_n/v_n\}$. For the empty tuple, we write $\emptyset$.

1.2 Constructor Matching Calculus

The constructor matching calculus consists of two components: (1) the calculus language and (2) the calculus rules, whereby the rules eventually describe which words of the calculus language are derivable.

We define the **calculus language** as

$$\mathcal{L} := \left(P^* \times (Lab \times T)^*\right) \times (Lab \rightharpoonup Lab^*)$$

Intuitively, the semantics of an element

$$((p_1, ..., p_n \doteq l_1 : t_1, ..., l_k : t_k), \sigma) \in \mathcal{L}$$

of the calculus language is as follows: The list $p_1, ..., p_n$ of argument patterns matches the formal constructor with the parameters $l_1 : t_1, ..., l_k : t_k$ with a matching function $\sigma \in Lab \rightharpoonup Lab^*$ which provides a mapping from labels of template arguments in $p_1, ..., p_n$ to the labels of matched formal parameters in $l_1, ..., l_k$. The subset of words in $\mathcal{L}$ which is derivable with the matching calculus is denoted with $\bar{\mathcal{L}}$.

The **calculus rules** are specified in the following form:

$$< \text{name} > \frac{< \text{pre}_1 > ... < \text{pre}_n >}{< \text{conclusion} >} < \text{side} >$$

The preconditions $< \text{pre}_i >$ as well as the conclusion must be elements of the calculus language $\mathcal{L}$ whereas the side conditions are independent of the calculus language. The following rules are defined for the constructor matching calculus:

$$\text{type}\ \frac{}{(e \doteq l : t)\,,\emptyset}\ \tau(e) \leq t \tag{1}$$

$$\text{name}\ \frac{(e \doteq l : t)\,,\sigma}{(l : e \doteq l : t)\,,\sigma} \tag{2}$$

$$\text{empty}\ \frac{}{(\emptyset \doteq \emptyset)\,,\emptyset} \tag{3}$$

$$\text{template}\ \frac{}{(l{*} \doteq l_1 : t_1, ..., l_n : t_n)\,, \{l/(l_1, ..., l_n)\}}\ \begin{array}{l} n \geq 0 \\ (S1) \end{array} \tag{4}$$

$$\text{compos}\ \frac{\begin{array}{l}(\quad p_1 \quad \doteq \quad l_1 : t_1, \quad ..., l_k : t_k), \sigma_1 \\ (p_2, ..., p_m \doteq l_{k+1} : t_{k+1}, ..., l_n : t_n), \sigma_2\end{array}}{(p_1, ..., p_m \doteq l_1 : t_1, ..., l_n : t_n)\,, \sigma_1 \cup \sigma_2}\ \begin{array}{l} m \geq 2 \\ n \geq k \geq 0 \\ (S1) \\ (S2) \\ (S3) \end{array} \tag{5}$$

$$\begin{array}{l}
(S1) : \forall i, j \in \{1, ..., n\} : i \neq j \Rightarrow l_i \neq l_j \\
(S2) : \forall i \in \{k+1, ..., n\} : \forall \sigma'_1, \sigma'_2 \in Lab \rightharpoonup Lab^* : \\
\quad \begin{pmatrix} ((\quad p_1 \quad \doteq \quad l_1 : t_1, \quad ..., l_i : t_i), \sigma'_1) \notin \bar{\mathcal{L}} \vee \\ ((p_2, ..., p_m \doteq l_{i+1} : t_{i+1}, ..., l_n : t_n), \sigma'_2) \notin \bar{\mathcal{L}} \end{pmatrix} \\
(S3) : \mathcal{D}(\sigma_1) \cap \mathcal{D}(\sigma_2) = \emptyset
\end{array}$$

The rule *type* (1) describes the usual method parameter matching as known from most object-oriented languages: Expression e can be used for a parameter with type t if the type of e is t itself or a subtype. *name* (2) extends this for named arguments which only match a parameter with the same name. *empty* (3) defines the empty argument list to match the empty parameter list. The rule *template* (4) expresses that a single template argument matches an arbitrary list of formal parameters as long as the labels of these parameters are mutually different $(S1)$. *compos* (5) allows lists of arguments to match lists of formal parameters whose labels must be disjunct again. The side condition $(S2)$ ensures that the first argument pattern always matches the longest possible sequence of formal parameters (cf. the example in the following subsection and Figure **??** for an intuition of the rationale for this side condition), and $(S3)$ essentially expresses that each template argument may occur only once in a pattern list.

1.3 Properties of the Matching Calculus

The following two theorems lay down two key properties of the constructor matching calculus.

THEOREM 1 (Rule uniqueness). *For each instance $\mathcal{E}$ of the constructor matching problem, there is exactly one calculus rule which might produce the calculus word $(\mathcal{E}, \sigma)$ with some matching function σ. The instances $\mathcal{E}_1, ..., \mathcal{E}_n$ of constructor matching problems in the preconditions $(\mathcal{E}_1, \sigma_1)$, ..., $(\mathcal{E}_n, \sigma_n)$ of the only applicable rule are uniquely determined by the instance $\mathcal{E}$ if $(\mathcal{E}, \sigma)$ is derivable.*

Proof. Let $\mathcal{E} = p_1, ..., p_m \doteq l_1 : t_1, ..., l_n : t_n$. There are three cases to distinguish for the value of m:

1. $m = 0$: Then, the only possible rule is *empty* due to the empty argument list.
2. $m = 1$: Then, for every argument pattern type of p_1, there is exactly one possible rule (*type* for an expression, *name* for a named expression, and *template* for a template argument).
3. $m > 1$: Then, the only possible rule is *compos*, which is the only one producing a list of argument patterns.

The constructor matching problems in the preconditions of all rules in a derivation are uniquely determined by the produced constructor matching problem instance $\mathcal{E}$:

- Rules *type*, *empty*, and *template* do not have preconditions. Hence, the statement holds trivially.
- For rule *name*, it is $\mathcal{E} = l : e \doteq l : t$, which uniquely determines the constructor matching problem in the precondition as $e \doteq l : t$.
- For rule *compos*, let $\mathcal{E} = p_1, ..., p_m \doteq l_1 : t_1, ..., l_n : t_n$. The constructor matching problem instances in the preconditions have the form $\mathcal{E}_1 = p_1 \doteq l_1 : t_1, ..., l_k : t_k$ and $\mathcal{E}_2 = p_2, ..., p_m \doteq l_{k+1} : t_{k+1}, ..., l_n : t_n$ with fulfilled side conditions of the rule. In order to show the uniqueness of $\mathcal{E}_1$ and $\mathcal{E}_2$, we have to show that k is uniquely determined by $\mathcal{E}$ and the side conditions of the rule. We will show this by contradicition: Let $k' \neq k$ and $(p_1 \doteq l_1 : t_1, ..., l_{k'} : t_{k'})\,, \sigma'_1$ and $(p_2, ..., p_m \doteq l_{k'+1} : t_{k'+1}, ..., l_n : t_n)\,, \sigma'_2$ be the preconditions of a valid application of rule *compos* which produces the constructor matching problem $\mathcal{E}$ in its conclusion. From the side conditions, we know that $0 \leq k' \leq n$. For k', we distinguish two cases:
  - $k' < k$: This violates side condition $(S2)$ because we know that, for $i = k \in \{k'+1, ..., n\}$, the preconditions $(p_1 \doteq l_1 : t_1, ..., l_i : t_i)\,, \sigma_1$ and $(p_2, ..., p_m \doteq l_{i+1} : t_{i+1}, ..., l_n : t_n)\,, \sigma_2$ are derivable ($\in \bar{\mathcal{L}}$) for some σ_1, σ_2.
  - $k' > k$: Then, $\mathcal{E}$ could not have been derived because side condition $(S2)$ would have been violated. Analogously to the above case, there would be an $i \in k+1, ..., n$, namely k', such that $(p_1 \doteq l_1 : t_1, ..., l_i : t_i)\,, \sigma_1$ and $(p_2, ..., p_m \doteq l_{i+1} : t_{i+1}, ..., l_n : t_n)\,, \sigma_2$ would be derivable with $\sigma_1 = \sigma'_1$ and $\sigma_2 = \sigma'_2$.

Consequently, it must be $k' = k$.

THEOREM 2 (Matching uniqueness). *If, for an instance $\mathcal{E}$ of the constructor matching problem, the calculus word $(\mathcal{E}, \sigma)$ is derivable, then the matching function σ is unique, i.e., there is no $\sigma' \neq \sigma$ such that $(\mathcal{E}, \sigma')$ is derivable.*

Proof. From Theorem 1, we know that, if $(\mathcal{E}, \sigma)$ has been derived, the rules which have been applied in this derivation are uniquely determined because, in each step of the derivation, the instance $\mathcal{E}$ of the constructor matching problem in the conclusion uniquely determines 1. the rule which has to be applied in this step and 2. the instances $\mathcal{E}_1, ..., \mathcal{E}_n$ of the constructor matching problem in the preconditions.

For uniqueness of the matching function σ for a given instance $\mathcal{E}$ of the constructor matching problem, we have to show that a derivation with given instances of the constructor matching problem in preconditions and conclusions uniquely determines the matching σ in the last conclusion of the derivation. We show this by induction over the derivation of the matching calculus:

Induction hypothesis. The matching function σ in the last conclusion of a derivation of the matching calculus is uniquely determined by the instance $\mathcal{E}$ of the constructor matching problem in this conclusion.

Base case. The shortest possible derivation only consists of one rule application. The preconditions of this rule must be empty; otherwise, the derivation would not be valid. Therefore, we have to distinguish three possible rules: For rules *type* and *empty*, the matching function in the conclusion is empty and the hypothesis holds trivially. For rule *template* and $\mathcal{E} = l{*} \doteq l_1 : t_1, ..., l_n : t_n$, the matching function is $\{l / (l_1, ..., l_n)\}$ and, hence, uniquely determined by $\mathcal{E}$.

Induction step. Let the preconditions of the last derivation rule application be $(\mathcal{E}_1, \sigma_1), ..., (\mathcal{E}_n, \sigma_n)$ with $n > 0$ (because $n = 0$ would be the base case). From the induction hypothesis, we know that $\sigma_1, ..., \sigma_n$ are uniquely determined by the corresponding constructor matching problem instance in $\mathcal{E}_1, ..., \mathcal{E}_n$. From Theorem 1, we know that $\mathcal{E}_1, ..., \mathcal{E}_n$ are uniquely determined by $\mathcal{E}$. It remains to show that all rules with non-empty preconditions uniquely determine the matching function σ in their conclusion by the matching functions in their preconditions: Rule *name* just copies the matching function from the precondition. Rule *compos* applies the union of partial functions to the matching functions σ_1 and σ_2 in its preconditions. Hence, in both rules, the resulting matching function in the conclusion is indeed uniquely determined by the functions in the preconditions.

Summary of the induction step in the syntax $x \xrightarrow{\text{uniquely determines}} y$:

$$\mathcal{E} \xrightarrow{\text{Theorem 1}} \mathcal{E}_1, ..., \mathcal{E}_n \xrightarrow{\text{ind. hypothesis}} \sigma_1, ..., \sigma_n \xrightarrow{\text{rule}} \sigma$$