Unification of Session Trees

Giovanni Bernardi[1], Simon J. Gay[2], and Vasco T. Vasconcelos[3]

[1] IMDEA Software Institute, Spain
[2] University of Glasgow, UK
[3] Universidade de Lisboa, Portugal

Abstract

Session types abstract communication patterns in concurrent programs. Associated with session types there is often a notion of duality that exchanges the point of view from which the communication is observed. A major obstacle to type inference in session-based programming languages is the absence of a unification algorithm for session types. We propose one such algorithm.

1 Motivation

Session types describe sequences of interactions between two partners [5]. They usually comprise five constructors: input and output, internal and external choice, and termination. Many interesting behaviours in concurrent and distributed applications are non-terminating, thus session types also include variables and a binder for recursion. For example, let T be

$$\mu a.\&\{\mathit{add}\colon ?\mathsf{int}.?\mathsf{int}.!\mathsf{int}.a, \mathit{isZero}\colon ?\mathsf{int}.\mathsf{bool}.a, \mathit{quit}\colon \mathsf{end}\}$$

Type T describes a server offering two services: *add* expects two integer values and returns another integer value; *isZero* reads one integer and writes one boolean value. In either case, interaction "goes back to the beginning", allowing the client to select another operation. The *quit*-labelled choice ends the interaction.

To interact correctly with the server behaviour sketched above, a client needs to behave *dually*: where the server offers an external choice ($\&$), the client performs an internal choice ($\oplus$), where the server performs an output ($!$) the client must perform an input ($?$), and conversely in both cases. The type end is self dual. This conveys the intuition for the dualof type operator. For instance we expect that $\mathsf{dualof}\ T = \mu a.\oplus\{\mathit{add}\colon !\mathsf{int}.!\mathsf{int}.?\mathsf{int}.a, \mathit{isZero}\colon !\mathsf{int}.\mathsf{bool}.a, \mathit{quit}\colon \mathsf{end}\}$. The dualof operator is not mere syntactic sugar. Type duality is crucial in the proof of type safety, and type duality also affects the class of safe, typable, processes [2].

For the purpose of type inference and to provide for type abbreviations in programming languages (cf. [4]), we are interested in solving equations such as $!x.!x.y \doteq !(!\mathsf{dualof}\ x.\mathsf{end}).!(!y.\mathsf{end}).?y.\mathsf{end}$, and systems of equations such as $\{x \doteq \mathsf{dualof}\ y, y \doteq !y.\mathsf{end}\}$.

Despite the simple intuition behind duality, in presence of recursive and higher-order terms a term and its dual may be quite different. This happens because of open terms in messages. For example, consider the term $S = \mu a.\mu b.?b.a$, and note that the message of S, namely b, is an open term. The definition of the duality function in [3, Definition 5.11 and Example 5.6] suggests that the dual of S is the a term of the form $\mu a.\mu b.!\mu c.?c.\mu a.\mu b.?b.a.a$. The example shows that a syntactic (term-based) treatment of dualof leads to the expansion of terms, thereby hindering, among other things, proofs of termination of (term-based) unification algorithms. For this reason, in this paper, we work with regular trees, and its associated (finite) graph representation, providing a simple way to define and treat duality.

2 Session trees and the unification problem

Fix $\mathcal{V}$ as a (countably) infinite set of variables. In this abstract we restrict our attention to regular trees as partial functions $s, t \in \{1,2\}^\star \to \{!, ?, \mathsf{dualof}, \mathsf{end}\} \cup \mathcal{V}$, where both ! and ? are of arity two, dualof is of arity one, and end and variables are of arity zero. We denote by ϵ the empty string, and by m, n, p strings in $\{1,2\}^\star$. In examples, we informally use the standard μ-notation to describe trees.

In fact we cannot allow all such functions since we can only give meaning to s when s denotes a contractive function. So we say that s is *contractive* if there exists no p in $\{1,2\}^\star$ such that for every $n \in 1^\star$, we have $t(pn) = \mathsf{dualof}$. An example of a non-contractive tree is $!\mathsf{end}.?\mathsf{end}.\mu a.\mathsf{dualof}\ a$.

The equality relation on trees, $=_T$, we are interested in is more generous (equates more trees) than partial function equality. For s and t arbitrary trees, we expect the following six equalities and implications to hold:

$$\mathsf{dualof}(!s.t) =_T ?s.\mathsf{dualof}\ t \qquad \mathsf{dualof}(?s.t) =_T !s.\mathsf{dualof}\ t$$
$$\mathsf{dualof}\ \mathsf{dualof}\ s =_T s \qquad \mathsf{dualof}\ \mathsf{end} =_T \mathsf{end}$$
$$\mathsf{dualof}\ s =_T t \Rightarrow s =_T \mathsf{dualof}\ t \qquad s =_T \mathsf{dualof}\ s \Rightarrow s =_T \mathsf{end}$$

We now formally define what it means for two trees to be equivalent. The definition generalises [1, Definition 2.11] by dropping the acyclicity condition and adding a relation corresponding to the dualof operator. We first define transition relations on strings denoting paths in trees, and then define relations of equality and duality, coinductively. The definitions are inspired by weak bisimulation. As the transition relations are deterministic, it would be possible to use trace-based definitions, but we find the bisimulation style convenient for proofs.

Let $L = \{\, d, e \,\}$, let α range over L, and let $\bar{\cdot}$ exchange d and e. We inductively define the transition relation $\to \subseteq \{1,2\}^\star \times L \times \{1,2\}^\star$. The intuition is that if $m \xrightarrow{e} n$ then m and n will be equivalent, and if $m \xrightarrow{d} n$ then m and n will be dual. Let t be the tree.

- $m \xrightarrow{e} m$
- If $t(m) = \mathsf{dualof}$ then $m \xrightarrow{d} m1$.
- If $m \xrightarrow{\alpha} n$ and $n \xrightarrow{\alpha} p$ then $m \xrightarrow{e} p$.
- If $m \xrightarrow{\alpha} n$ and $n \xrightarrow{\overline{\alpha}} p$ then $m \xrightarrow{d} p$.
- If $t(m) = \mathsf{end}$ then $m \xrightarrow{d} m$.

A pair of relations $(\mathcal{E}, \mathcal{D})$ between the nodes of s and the nodes of t is an *equivalence-duality* if whenever $(m, n) \in \mathcal{E}$ then the following conditions are true:

1. If $s(m) = \mathsf{dualof}$ then either (1) there exists n' such that $n \xrightarrow{e} n'$ and $(m1, n') \in \mathcal{D}$, or (2) there exists n' such that $n \xrightarrow{d} n'$ and $(m1, n') \in \mathcal{E}$;
2. If $s(m) = \mathsf{end}$ then either (1) there exists n' such that $n \xrightarrow{e} n'$ and $t(n') = \mathsf{end}$, or (2) there exists n' such that $n \xrightarrow{d} n'$ and $t(n') = \mathsf{end}$;
3. If $s(m) = !$ then either (1) there exists n' such that $n \xrightarrow{e} n'$ and $t(n') =!$ and $(m1, n1) \in \mathcal{E}$ and $(m2, n2) \in \mathcal{E}$, or (2) there exists n' such that $n \xrightarrow{d} n'$ and $t(n') =?$ and $(m1, n1) \in \mathcal{E}$ and $(m2, n2) \in \mathcal{D}$;
4. If $s(m) = ?$ then either (1) there exists n' such that $n \xrightarrow{e} n'$ and $t(n') =?$ and $(m1, n1) \in \mathcal{E}$ and $(m2, n2) \in \mathcal{E}$, or (2) there exists n' such that $n \xrightarrow{d} n'$ and $t(n') =!$ and $(m1, n1) \in \mathcal{E}$ and $(m2, n2) \in \mathcal{D}$;

5. If $s(m) = x$ (variable) then there exists n' such that $n \xrightarrow{e} n'$ and $t(n') = x$;
6. Symmetrical conditions for $t(n)$;

and whenever $(m, n) \in \mathcal{D}$ then the following conditions are true:

1. If $s(m) = \mathsf{dualof}$ then either (1) there exists n' such that $n \xrightarrow{e} n'$ and $(m1, n') \in \mathcal{E}$, or (2) there exists n' such that $n \xrightarrow{d} n'$ and $(m1, n') \in \mathcal{D}$;
2. If $s(m) = \mathsf{end}$ then either (1) there exists n' such that $n \xrightarrow{e} n'$ and $t(n') = \mathsf{end}$, or (2) there exists n' such that $n \xrightarrow{d} n'$ and $t(n') = \mathsf{end}$;
3. If $s(m) = !$ then either (1) there exists n' such that $n \xrightarrow{e} n'$ and $t(n') = ?$ and $(m1, n1) \in \mathcal{E}$ and $(m2, n2) \in \mathcal{D}$, or (2) there exists n' such that $n \xrightarrow{d} n'$ and $t(n') = !$ and $(m1, n1) \in \mathcal{E}$ and $(m2, n2) \in \mathcal{E}$;
4. If $s(m) = ?$ then either (1) there exists n' such that $n \xrightarrow{e} n'$ and $t(n') = !$ and $(m1, n1) \in \mathcal{E}$ and $(m2, n2) \in \mathcal{D}$, or (2) there exists n' such that $n \xrightarrow{d} n'$ and $t(n') = ?$ and $(m1, n1) \in \mathcal{E}$ and $(m2, n2) \in \mathcal{E}$;
5. If $s(m) = v$ (variable) then there exists n' such that $n \xrightarrow{d} n'$ and $t(n') = v$;
6. Symmetrical conditions for $t(n)$.

By monotonicity and Knaster-Tarski, we have the largest equivalence-duality pair. We say that two trees, s and t, are *equivalent* if their root nodes are in the first component of the pair; we say that s and t are *dual* if their root nodes are in the second component.

For example, we can show that $?(\mu a.!a.\mathsf{end}).\mathsf{end} =_T \mathsf{dualof}\mu a.!a.\mathsf{end}$. The terms $?(\mu a.!a.\mathsf{end}).\mathsf{end}$ and $\mathsf{dualof}\mu a.!a.\mathsf{end}$ are represented respectively by the tree s and t, defined by:

$$\begin{array}{llll} s(\epsilon) = ? & & t(\epsilon) = \mathsf{dualof} & \\ s(1^n) = ! & n \geqslant 1 & t(1^n) = ! & n \geqslant 1 \\ s(1^n 2) = \mathsf{end} & n \geqslant & t(1^n 2) = \mathsf{end} & n \geqslant 1 \end{array}$$

Let $\mathcal{E} = \{ (\epsilon, \epsilon) \} \cup \{ (1^m, 1^n) \mid m, n \geqslant 1 \}$, and $\mathcal{D} = \{ (\epsilon, 1^n) \mid n \geqslant 1 \} \cup \{ (1^m, \epsilon) \mid m \geqslant 1 \}$. To show that the root of s is equivalent to the root of t, it suffices to check that the relations $\mathcal{E}$ and $\mathcal{D}$ are an equivalence-duality pair. This is routine work.

A *unification problem* is a finite set of equations $\Gamma = \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$. A *unifier* of Γ is a substitution σ such that $s_1\sigma =_T t_1\sigma$, ..., $s_n\sigma =_T t_n\sigma$. The set of unifiers for a given unification problem may have more than one element. For example, the unification problem $\{\mathsf{dualof}(!\mathsf{end}.x) \doteq ?\mathsf{dualof}\, y.!\mathsf{end}.\mathsf{end}\}$ has many unifiers, including $\sigma_1 = \{x \mapsto ?\mathsf{end}.\mathsf{end}, y \mapsto \mathsf{dualof}\,\mathsf{end}\}$ and $\sigma_2 = \{x \mapsto \mathsf{dualof}(!\mathsf{end}.\mathsf{end}), y \mapsto \mathsf{end}\}$.

Let $\mathcal{X}$ be a set of variables. We say that a substitution σ is *more general on* $\mathcal{X}$ than substitution θ if there exists a substitution λ such that $x\theta =_T x\sigma\lambda$ for all $x \in \mathcal{X}$. In this case we write $\theta \leqslant_T^{\mathcal{X}} \sigma$ and say that θ is an instance of σ on $\mathcal{X}$. In our example, unifier σ_1 is both more general (on $\{x, y\}$) and less general than σ_2. Take for λ the identity substitution. Then $x\sigma_1 = ?\mathsf{end}.\mathsf{end} =_T \mathsf{dualof}(!\mathsf{end}.\mathsf{end}) = x\sigma_2\lambda$ and $x\sigma_2 = \mathsf{dualof}(!\mathsf{end}.\mathsf{end}) =_T ?\mathsf{end}.\mathsf{end} = x\sigma_1\lambda$, and similarly for variable y. However, variable renaming cannot identify $?\mathsf{end}.\mathsf{end}$ and $\mathsf{dualof}(!\mathsf{end}.\mathsf{end})$.

Regular trees and the particular form of our equations seem to preclude using the standard theory of equational unification. Take the $\mathcal{G}$ calculus [1]. Even if the occurs-check built into the Variable Elimination rule could be alleviated, we still have the problem that $\mathcal{G}$ does not yield, in general, a decision procedure even for unitary theories whose unification problem is decidable. Furthermore, narrowing and the $\mathcal{B}$ calculus cannot help: there seems to be no obvious convergent rewrite system representing the system of equations above, obtained by appropriately orienting the equations. For

```
Unify (s:node, t:node):
  UnifClosure(s, t)
  FindSolution(s)

UnifClosure (s:node, t:node):
  ExpandDual(s)
  ExpandDual(t)
  s := Find(s)
  t := Find(t)
  if s and t are the same node then {Trivial}
    {Do nothing}
  else if s = f(s1,...,sn) and t = f(t1,...,tn) then {Decomposition}
    Union(s, t)
    for i := 1 to n do
      UnifClosure(si, ti)
  else if s = x or t = x {Variable Elimination}
    Union(s, t)
  else if s = dualof t or dualof s = t
    {x = dualof x ⇒ x = end}
    Union(s, t)
    Union(t, end)
  else if s = dualof s1 {dualof x = t ⇒ x = dualof t}
    Union(s,t)
    u := dualof t; dual(s) := u; dual(s1) = u
  else if t dualof t1
    UnionClosure(t,s)
  else
    exit with failure {Symbol clash}

ExpandDual (s:node):
  {Post: find(s) = dualof(t) ⇒ t = x}
  s := Find(s)
  if s = dualof(t) then
    t := Find(t)
    if dual(t) != ⊥ then {t has known dual}
      Union(s, dual(t))
    else if t = !(t1, t2) then {dualof !t1.t2 = ?t1.dualof t2}
      u := ?(t1, dualof(t2))
      Union (s, u); dual(t) := u
    else if t = ?(t1, t2) then {dualof ?t1.t2 = !t1.dualof t2}
      u := !(t1, dualof(t2))
      Union (s, u); dual(t) := u
    else if t = dualof(t1) then {dualof dualof t = t}
      Union(s, t1)
      ExpandDual(t1)
    else {t = end} {dualof end = end}
      Union(s, end)

FindSolution (s:node):
  {Always succeeds; builds a substitution on trees}
  ...
```

Figure 1: Unification of session graphs

example, while solving the system of equations $\{\mathsf{dualof}(!\mathsf{end}.x) \doteq ?(\mathsf{dualof}\, y).!\mathsf{end}.\mathsf{end}\}$ we crucially use the dualof involution rule from right to left (that is $x \to \overline{\overline{x}}$) in order to get rid of the dualof operator on variables x and y. The lack of a convergent set of rewrite rules prevents using results in the literature, including [7, 8].

3 Unification of session graphs

Term graphs were developed to speed up the unification process. Here we try the approach to provide our problem with an algorithm. We follow the presentation of Baader and Snyder [1].

A *tree graph* is a directed (possibly cyclic) graph whose nodes are labelled with function symbols or variables, whose outgoing edges from any node are ordered, and where the outdegree of any node labelled with symbol f is equal to the arity of f. Variables have outdegree zero.

In such a graph, each node has a natural interpretation as a tree, and we shall speak of nodes and trees as if they were the same. It is possible to create a graph with unique, shared occurrences of variables in $\mathcal{O}(n)$, where n is the number of all characters in the string representing the unification problem. Therefore, we assume that the input to our algorithm is a tree graph representing the two terms to be unified, with unique, shared occurrences of all variables. A *substitution* involving only subterms of a graph can be represented directly by a relation on the nodes of the graph.

The algorithm we propose is adapted from Huet [6]. For each node in the graph we need:

- a *class* pointer, and

- a *dual* pointer.

A representative of an equivalence class is a node whose class pointer points to itself. Dual pointers can be undefined, denoted by $\perp$. A *tree graph* Δ for s and t is initialised as follows:

- The equivalence relation is the identity relation, where each class contains a single node, hence for each node the class pointer is initialised to point to the same node;
- The dual relation is empty, hence for each node the dual pointer is initialised to $\perp$.

The algorithm in Figure 1 iteratively builds the smallest equivalence-duality relations (described in Section 2) such that the equivalence component contains the pairs of terms to be unified. This generalises the concept of unification closure [1]. The algorithm uses the standard union-find procedures, which we omit from our description. Procedure ExpandDual(s) behaves as the identity function when s is not a dualof node. Otherwise, it "expands" dualof nodes using the first four of the equations in Section 2, ensuring that the representative of the equivalence class for s is a dualof node only when s is of the form dualof x. The procedure terminates due to the contractive nature of trees (cf. recursive call). Procedure UnifClosure extends that in [1, 6]. It starts by getting dualof nodes out of the way. After the standard Trivial, Decomposition and Variable Elimination cases, we have cases corresponding to the remaining two implications in Section 2 (read left-to-right and right-to-left).

Proving correctness of the algorithm requires proving that it constructs the smallest equivalence-duality relation containing the terms to be unified, as explained (but not proved) in [1]. As for complexity, in the absence of dualof nodes, Unify is clearly quasi-linear [1, 6]. dualof nodes introduce a fixed number of operations per node, hence we expect the complexity of algorithm to remain quasi-linear.

We have implemented the unification algorithm in Java (11 classes/interfaces, 500 loc, excluding the parser). The algorithm can be exercised online at `http://gloss.di.fc.ul.pt/tryit/tools/Unst`, where a short tutorial can also be found.

References

[1] Franz Baader and Wayne Snyder. Unification theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 445–532. Elsevier and MIT Press, 2001.

[2] Giovanni Bernardi, Ornela Dardha, Simon J. Gay, and Dimitrios Kouzapas. On duality relations for session types. In *TGC*, volume 8902 of *LNCS*, pages 51–66. Springer, 2014.

[3] Giovanni Bernardi and Matthew Hennessy. Using higher-order contracts to model session types. *CoRR*, abs/1310.6176, 2013.

[4] Juliana Franco and Vasco Thudichum Vasconcelos. A concurrent programming language with refined session types. In *Second International Workshop on Behavioural Types*, volume 8368 of *LNCS*, pages 33–42. Springer, 2013.

[5] Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP*, volume 1381 of *LNCS*, pages 22–138. Springer, 1998.

[6] Gérard Huet. *Résolution d'équations dans des langages d'ordre* $1, 2, \ldots, \omega$. Thèse d'État, Université de Paris VII, 1976.

[7] Florent Jacquemard, Michaël Rusinowitch, and Laurent Vigneron. Tree automata with equality constraints modulo equational theories. *Journal of Logic and Algebraic Programmming*, 75(2):182–208, 2008.

[8] Sébastien Limet and Pierre Réty. E-unification by means of tree tuple synchronized grammars. *Discrete Mathematics and Theoretical Computer Science*, 1:69–98, 1997.