Implementation of Late Name Binding, Asynchronous Pi-Calculus in Maude

Bartosz Zieliński
Department of Computer Science
Faculty of Physics and Applied Computer Science,
University of Łódź
ul. Pomorska nr 149/153, 90-236 Łódź
bzielinski@uni.lodz.pl

Paweł Maślanka
Department of Computer Science
Faculty of Physics and Applied Computer Science,
University of Łódź
ul. Pomorska nr 149/153, 90-236 Łódź
pmaslan@uni.lodz.pl

***Abstract*—We describe a new implementation of asynchronous pi-calculus in Maude which utilizes CINNI — a calculus of explicit substitutions — to cope with name binding. We believe that using late name binding, instead of early, simplifies the implementation, which we intend to use later as part of a workflow specification system.**

Keywords—pi calculus; CINNI; late name binding

I. Introduction

The calculus of mobile processes (pi-calculus, [1]) is one of the most popular proces algebras used in specification of workflows (see e.g., [2], [3], [4]), or security protocols (see e.g., [5]) There are many implementations of pi-calculus which allow actually to run the pi-processes and to explore and analyze the possible execution paths. Here we describe the implementation in the language and term rewriting framework Maude [6]. This is by far not the first implementation of pi-calculus in Maude (see, e.g., [7]). In fact, we follow the basic ideas of [7] such as incorporating transition labels into the process terms (c.f., [8]) and utilizing CINNI [9] — a calculus of explicit substitutions — to deal with name binding. However, instead of early binding semantics, like in [7] we implement a late semantics in a way in which the label part of the process can also bind names, and to the best of our knowledge this is the first such implementation.

II. Asynchronous Pi-Calculus

Let P, Q, … (perhaps with indices) denote pi-process terms and let small latin letters denote channel names. Then the grammar of asynchronous pi-calculus is given by (where I is finite and guards □::=□|□(□)):

$$P ::= 0 \left| \sum_{i\in I} \alpha_i.P_i \right| \overline{a}\langle b\rangle \mid (P \mid Q) \left| (\nu x)P \right| !P \mid [x=y](P,Q),$$

We will write $\sum_{i\in I} \alpha_i.P_i$ using an associative and commutative operator „+". We also consider "|" to be associative and commutative with 0 being the neutral element. Operators (νx) and $a(x)$ bind x in $(\nu x)P$ and $a(x).P$, respectively (and these are the only binding operations). We denote by "$\equiv_\alpha$" the α-equivalence relation on pi-terms. Thus, e.g., if y does not occur free in P we must have that $a(x).P$ and $a(y).([x:=y]P)$ are α-equivalent, where $[x:=y]P$ denotes the capture avoiding substitution of name y for free occurences of x in P. We assume that α-equivalent terms behave identically. Processes evolve by performing internal actions as well as receiving and sending public or private names. **0** is the empty process which does nothing, process $\overline{a}\langle b\rangle$ sends name b on the channel identified by name a. For any $i \in I$, process $\sum_{i\in I} \alpha_i.P_i$ can perform either an internal action (if $\alpha_i = \tau$) or receive value on some channel a (if $\alpha_i = a(x)$ and then become P_i. Here our semantics is late because we do not consider the value actually received as part of the transition. $P \mid Q$ is a parallel composition of processes which can evolve independently or communicate, $!P$ replicates P, $(\nu x)P$ is a restricted process in which x is private (but it can be sent to other process), and finally $[x=y](P,Q)$, evolves as either P or Q depending on whether x is the same name as y or not. Thus, pi-process terms are states in the labeled transition system.

We write $P \xrightarrow{\alpha} Q$ when a process P can transition into Q performing an action with label α There are four kinds of action labels: τ (internal transition), $a(x)$ (value receive), $\overline{a}\langle b\rangle$ (public name sent), $\overline{a}\langle \nu b\rangle$ (private name sent). The possible transitions can be described precisely using rules: we allow those and only those transitions which are provable from the rules. For instance, the first rule below describes communication between parallel processes, the second describes sending of some private name:

$$\frac{P \xrightarrow{\overline{a}\langle b\rangle} P' \quad Q \xrightarrow{a(x)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid [x:=b]Q'}\ \text{FCOM}, \qquad \frac{P \xrightarrow{\overline{a}\langle b\rangle} Q \quad a \neq b}{(\nu b)P \xrightarrow{\overline{a}\langle \nu x\rangle} Q}\ \text{OPEN}.$$

III. Implementation

The main problem of implementing labeled transition system in term rewriting system such as Maude, is that they usually do not support labeled transitions. The fundamental idea of [8] is to incorporate labels into process term — effectively the process term drags with itself the full trace of all actions performed so far by the process. Formally, one extends the grammar of process terms with new syntactic category of action process terms AP:

$$AP :: P \mid \{\alpha\}AP$$

where α is an action label (e.g., τ , $a(x)$, etc.) and P is an ordinary pi-process term. Denote by the unlabelled thick arrow "$\Rightarrow$" the rewriting relation among action process terms. It is defined with the following rules:

$$\frac{P \xrightarrow{\alpha} Q}{P \Rightarrow \{\alpha\}Q}\text{1STEP}, \quad \frac{P \Rightarrow \{\alpha\}Q \qquad Q \Rightarrow AP}{P \Rightarrow \{\alpha\}AP}\text{TRANS}.$$

where P, Q are pi-proces terms, AP is an action process term and α is an action label. We assume that action labels of the form $a(x)$ and $\overline{a}\langle \nu b\rangle$ bind name x in action process terms of the form $\{a(x)\}AP$ and $\{\overline{a}\langle \nu x\rangle\}AP$. Accordingly, we extend α-conversion to action process terms and identify α - equivalent action process terms.

For example, assume that $P \xrightarrow{a(x)} Q$. Because of identification of behaviour of α-equivalent terms we must also have $P \xrightarrow{a(y)} [x := y]Q$ for any name y which is not free in Q. This is evident when we write the transition $P \xrightarrow{a(x)} Q$ with unlabeled rewritings as $P \Rightarrow \{a(x)\}Q$. Indeed, because we extend the α-equivalence to action process terms we have that $\{a(x)\}Q$ and $\{a(y)\}[x := y]Q$ are α-equivalent.

Implementing capture avoiding substitutions and α-conversion necessary to enable some transitions is cumbersome. One way to deal with it is to use de Bruijn indices instead of names. This would however be unreadable for humans. A good balance between readability and implementability is provided by the calculus of explicit substitutions CINNI [9], and we follow [7] in utilizing it. Namely, except for binding constructs, we use channel names of the form a_n, where a is a (user defined) name and n is a natural number.

The meaning of n is as follows: Suppose that a_n occurs in the process or action process term T. Then n is the number of operations binding name a in T which one must pass while going up the parse tree of T before either encountering an operation which binds this a_n or the root of the parse tree.

The signature of action process terms is extended with various substitution operators (including the usual capture avoiding substitution) defined through equations. On the other hand, there is no longer any α-equivalence.

As an example consider the following translation of the OPEN rule mentioned above into unlabeled rewritings with CINNI:

$$\frac{P \Rightarrow \{\overline{y_m}\langle x_0\rangle\}Qy \neq x}{(\nu x)P \Rightarrow \{\overline{y_m}\langle \nu x\rangle\}Q}\text{OPEN}_1, \quad \frac{P \Rightarrow \{\overline{x_{m+1}}\langle x_0\rangle\}Q}{(\nu x)P \Rightarrow \{\overline{x_m}\langle \nu x\rangle\}Q}\text{OPEN}_2$$

Conclusion

We created a new implementation of late binding semantics for asynchronous pi-calculus in Maude. We partly use the ideas from earlier papers ([7], [8]). A novel (to the best of our knowledge) aspect of this implementation is extending α-equivalence to labeled process terms. We found out that implementing late semantics is easier and more natural than late semantics (as in [7]), particularly when using our main idea. We created this implementation to become a part of a system for workflow specification we are currently working on.

References

[1] Robin Milner, "Communicating and Mobile Systems: the Pi-Calculus," Cambridge University Press, 1999

[2] Howard Smith and Peter Fingar, "Workflow is just a pi process", BPTrends, November, pages 1-36, 2003.

[3] Frank Puhlmann and Mathias Weske "A look around the corner: the pi-calculus," In Transactions on Petri Nets and Other Models of Concurrency II, pages 64-78. Springer, 2009.

[4] Bartosz Zieliński, Ścibor Sobieski, Piotr Kruszyński, Maciej Sysak, and Paweł Maślanka, "Object pi-calculus and document workflows," In Model and Data Engineering, pages 227-238. Springer, 2015.

[5] Mark D. Ryan and Ben Smyth, "Applied pi calculus," chapter in Véronique Cortier & Steve Kremer (editors) Formal Models and Techniques for Analyzing Security Protocols, IOS Press, 2011.

[6] Manuel Clavel, Francisco Duran, Steven Eker, Patrick Lincoln, Narciso Marti-Oliet, Jose Meseguer, and Carolyn Talcott, "The maude 2.0 system," In Robert Nieuwenhuis, editor, Rewriting Techniques and Applications (RTA 2003), number 2706 in Lecture Notes in Computer Science, pages 76-87. Springer-Verlag, June 2003.

[7] Prasanna Thati, Koushik Sen, and Narciso Marti-Oliet, "An executable specification of asynchronous pi-calculus semantics and may testing in maude 2.0", Electronic Notes in Theoretical Computer Science, 71:261-281, 2004.

[8] Alberto Verdejo and Narciso Martí-Oliet, "Implementing CCS in Maude 2", Electronic Notes in Theoretical Computer Science, 71:282-300, 2004. WRLA 2002, Rewriting Logic and Its Applications.

[9] Mark-Oliver Stehr, "Cinni-a generic calculus of explicit substitutions and its application to lambda-, sigma- and pi-calculi," Electronic Notes in Theoretical Computer Science, 36:70-92, 2000.