HAL
open science

PIC2LNT: Model Transformation for Model Checking an Applied Pi-Calculus

Radu Mateescu, Gwen Salaün

To cite this version:

Radu Mateescu, Gwen Salaün. PIC2LNT: Model Transformation for Model Checking an Applied Pi-Calculus. TACAS - 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems - 2013, Mar 2013, Rome, Italy. pp.192-198, 10.1007/978-3-642-36742-7_14 . hal-00805533

HAL Id: hal-00805533

https://inria.hal.science/hal-00805533

Submitted on 28 Mar 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

PIC2LNT: Model Transformation for Model Checking an Applied Pi-Calculus

Radu Mateescu[1] and Gwen Salaün[2,1]

[1] Inria Grenoble Rhône-Alpes and LIG — CONVECS team, France
[2] Grenoble INP, France
{Radu.Mateescu,Gwen.Salaun}@inria.fr

1 Introduction

The π-calculus [12] was proposed by Milner, Parrow, and Walker about twenty years ago for describing concurrent systems with mobile communication. The π-calculus is equipped with operational semantics defined in terms of LTSs (Labelled Transition Systems). Although a lot of theoretical results have been achieved on this language (see, *e.g.*, [1, chapter 8] for a survey), only a few verification tools have been designed for analysing π-calculus specifications automatically. The two most famous examples are the Mobility Workbench (MWB) [14] and JACK [5], which were developed in the 90s.

Our objective is to provide analysis features for π-calculus specifications by reusing the verification technology already available for value-passing process algebras without mobility. Contrary to existing verification tools for the π-calculus, which rely on specific algorithms and intermediate models, such as HD-automata [5], our approach is based on a novel translation [9] from the finite control fragment of π-calculus to a standard process algebra called LOTOS NT (LNT for short) [3]. LNT is a value-passing process algebra with imperative programming flavour accepted as input by the CADP verification toolbox [8]. It supports the specification of data structures (constructed types, pattern-matching, recursive functions) and concurrent processes. LNT has a user-friendly syntax and a formal operational semantics defined in terms of LTSs. To the best of our knowledge, this is the first π-calculus translation having a standard process algebra as target language.

In this work, we go a step further by extending the original polyadic π-calculus with data-handling features. This results in a general-purpose applied π-calculus, which offers a good level of expressiveness for specifying mobile concurrent systems, and therefore for widening its possible application domains. As language for describing data types and functions, a natural choice was LNT itself: in this way, the data types and functions used in the π-calculus specification can be directly imported into the LNT code produced by translation. We generalized our previous translation [9] to handle applied π-calculus specifications, and we automated it in the tool PIC2LNT 2.0. This enables the analysis of applied π-calculus specifications using all verification tools of CADP, in particular the EVALUATOR 4.0 on-the-fly model checker [11], which evaluates temporal properties involving channel names and data values.

2 Applied Pi-Calculus

We designed our applied π-calculus by extending the original polyadic π-calculus [12] equipped with the early operational semantics defined in [13]. We consider π-calculus specifications satisfying the *finite control* property [4], which amounts to forbid recursive agent calls through parallel composition operators. When the set of channels is bounded, this results in finite-state systems that can be analyzed using existing model checking techniques. We extended the original π-calculus with constructs for manipulating data variables and expressions. Agents can be parameterized by data variables in addition to channel names, and the polyadic communication was extended to handle emission/reception of data values. The guard operator was generalized to handle arbitrary Boolean expressions (in addition to the comparison of channel names), and a new operator was added for declaring and initializing data variables. The replication operator was restricted to a bounded version (in order to satisfy the finite control property), which instantiates n parallel copies of an agent, and therefore enables to describe mobile systems containing a finite amount of dynamic control. Data types and functions are specified in LNT [3] as external modules, which are imported in the applied π-calculus specification. The concrete syntax (which is compatible with MWB for dataless π-calculus specifications) and semantics of the applied π-calculus are described in [10].

We present below a code sample to illustrate our applied π-calculus on a load balancing system, which is a networking method to distribute workloads across multiple servers. The specification (*Main* agent) given below consists of five agents: a client, the load balancer, and three servers. The client corresponds to a possible environment and is used to simulate various scenarios. The load balancer receives new tasks (*task*) with a private name (*com*), and then interacts with the three servers to know their current load. To do so, a public channel (*e.g.*, *al* for the first server) is used for sending the request and receiving the result. The load balancer compares the different loads and forwards the private name originally submitted by the client to the server with the minimum load. A server has three possible behaviours: it can be asked by the load balancer to return its current load; it can receive a request for a new task (reception of a private name from the load balancer and interaction with the client on this private channel to receive the new load); or it can execute part of its work if its total load is greater than zero. We can see with this simple example how data expressions (natural numbers, comparison, addition, etc.) appear as parameters of agents and channels to specify loads and their manipulation.

$$
\begin{array}{l}
Main = \\
\quad (\nu\ task, al, ar, bl, br, cl, cr)\ (\ Client(task) \mid LoadBalancer(task, al, ar, bl, br, cl, cr) \mid \\
\quad Server(al, ar, 0\ of\ Nat) \mid Server(bl, br, 0\ of\ Nat) \mid Server(cl, cr, 0\ of\ Nat)\ )
\end{array}
$$

$$
\begin{array}{l}
Client(task) = \\
\quad (\nu\ com_1)\ \overline{task}\langle com_1\rangle.\overline{com}_1\langle 2\ of\ Nat\rangle.(\nu\ com_2)\ \overline{task}\langle com_2\rangle.\overline{com}_2\langle 1\ of\ Nat\rangle. \\
\quad (\nu\ com_3)\ \overline{task}\langle com_3\rangle.\overline{com}_3\langle 1\ of\ Nat\rangle.(\nu\ com_4)\ \overline{task}\langle com_4\rangle.\overline{com}_4\langle 2\ of\ Nat\rangle.0
\end{array}
$$

$$
\begin{array}{l}
LoadBalancer(task, al, ar, bl, br, cl, cr) = \\
\quad task(com).\ \overline{al}.al(v_1 : Nat).\ \overline{bl}.bl(v_2 : Nat).\ \overline{cl}.cl(v_3 : Nat). \\
\quad (\quad [(v_1 \leq v_2)\ and\ (v_1 \leq v_3)]\ \overline{ar}\langle com\rangle.LoadBalancer(task, al, ar, bl, br, cl, cr) \\
\quad\ + [(v_2 \leq v_1)\ and\ (v_2 \leq v_3)]\ \overline{br}\langle com\rangle.LoadBalancer(task, al, ar, bl, br, cl, cr) \\
\quad\ + [(v_3 \leq v_1)\ and\ (v_3 \leq v_2)]\ \overline{cr}\langle com\rangle.LoadBalancer(task, al, ar, bl, br, cl, cr)\ )
\end{array}
$$

$$
\begin{array}{l}
Server(ld, rq, totalload : Nat) = \\
\quad ld.\overline{ld}\langle totalload\rangle.Server(ld, rq, totalload) \\
\quad + rq(req).req(newload : Nat).Server(ld, rq, totalload + newload) \\
\quad + [totalload > 0]\ \overline{execute}\langle ld, totalload\rangle.Server(ld, rq, totalload - 1)
\end{array}
$$

3 Translation to LNT

Most of the π-calculus constructs are translated quite straightforwardly into LNT because of its high level of expressiveness. Nevertheless, we faced some subtle difficulties in obtaining a translation as succinct as possible while still preserving the LTS semantics, *i.e.*, mapping each transition of a π-calculus agent to a transition of the resulting LNT process. One of the main problems was to encode the binary, unidirectional, and mobile communication of π-calculus into a specification language enabling multi-way and bidirectional communication on static channels.

Since mobile communication cannot be described directly using LNT static channels, we overcome this issue by exploiting the data types and synchronization features of LNT. We represent π-calculus channel names as values of a LNT datatype *Chan*, which defines all the public and private names appearing in the specification. Then, we model channel mobility between π-calculus agents by communicating *Chan* values along LNT channels. Binary unidirectional communications and two-among-n synchronizations, which cannot be directly described in LNT, are encoded by means of dedicated LNT channels (one for each π-calculus parallel composition operator), on which the sender and receiver are indicated explicitly using process identifiers and placeholders. Communication on a π-calculus channel is translated in LNT as a choice on all LNT channels connecting the current agent to its environment. The translation of the original π-calculus to LNT is detailed in [9].

4 Tool Support and Verification with CADP

The translation from our applied π-calculus to LNT has been automated by the translator PIC2LNT 2.0, implemented using the SYNTAX+TRAIAN compiler construction technology [7]. The tool consists of about $2,300$ lines of SYNTAX code, $4,800$ lines of LOTOS NT code, and 700 lines of C code[3].

Figure 1 gives an overview of the complete tool chain. Given a specification in applied π-calculus, possibly containing data types and functions described in

[3] The version 1.0 of PIC2LNT, which handled the original π-calculus (without data manipulation), consisted of about $3,700$ lines of code.

LNT libraries, PIC2LNT translates it into an equivalent LNT specification, which is accepted as input by the CADP tools. The resulting LNT specification is connected by the LNT.OPEN tool (via an intermediate translation to LOTOS) to the OPEN/CÆSAR environment [6], which gives access to all the on-the-fly verification tools of CADP. The `pic2lnt_dyn.tnt` file (static code) contains external C functions for generating fresh channel names and process identifiers.

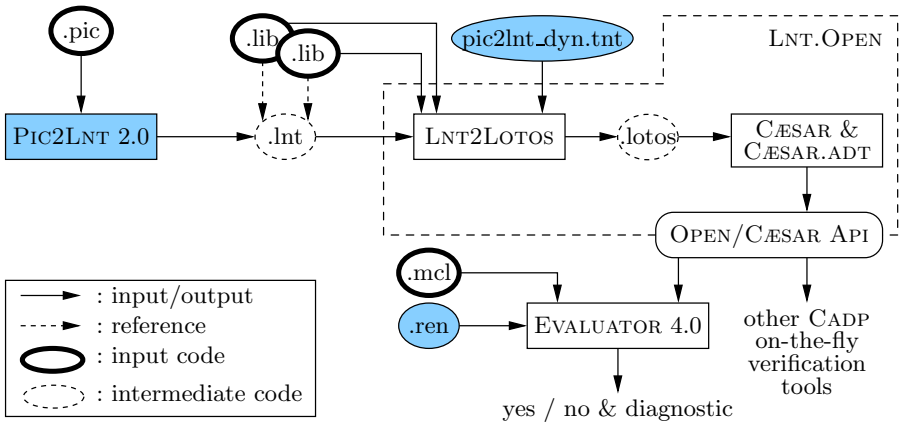

Fig. 1. Overview of the tool chain

As illustrated on Figure 1, one can use the EVALUATOR 4.0 on-the-fly model checker to verify temporal properties specified in MCL [11], an extension of alternation-free μ-calculus with regular expressions, data-based constructs, and fairness operators. MCL is suitable for analyzing applied π-calculus specifications, because the properties can involve both the channel names and the data values transmitted. The LTS actions, which carry additional information introduced during the translation to LNT, are renamed on-the-fly to retrieve the original π-calculus format using a predefined label renaming file.

Going back to the load balancing system specified in Section 2, the LTS of the resulting LNT specification contains $2,007$ states and $5,450$ transitions. As an example, we can check that this LTS satisfies the MCL data-based response property below, which states that every time a server has begun an execution, it will eventually exhaust its workload by executing it one unit at a time:

$$
\begin{array}{l}
[true^*.\{execute\ ?ld{:}String\ ?load{:}Nat\ ...\ where\ load > 1\}] \\
\quad \mu X(crt_load{:}Nat := load - 1).(\\
\qquad \langle true \rangle true \wedge \\
\qquad [\neg\{execute\ !ld\ !crt_load\ ...\}]\ X(crt_load) \wedge \\
\qquad [\{execute\ !ld\ !crt_load\ ...\ where\ crt_load > 1\}]\ X(crt_load - 1) \\
\quad)
\end{array}
$$

The action predicates enclosed between curly braces enable to capture the information present on LTS actions, *i.e.*, the channel names (interpreted as character strings) and the data values transmitted. The box modality matches all sequences that end up, after zero or more steps, with an *execute* action carrying a channel name *ld* and a workload *load*. These values are captured and used later in the parameterized minimal fixed point operator μX, which expresses the inevitable reachability of consecutive *execute* actions that carry decreasing workloads.

The PIC2LNT 2.0 translator is currently provided as a CADP plug-in. The manual page and the executable files for several architectures (Mac computers, PCs running Linux or Windows, Solaris workstations) are available on-line [10].

5 Experimental Evaluation

We applied PIC2LNT 2.0 on a benchmark of π-calculus specifications, which includes most of the examples provided with MWB, as well as applied π-calculus examples that we specified ourselves. Our benchmark currently contains 284 files, totalizing about $5,200$ lines of π-calculus, which were translated in about $50,000$ lines of LNT. This expansion in size, which is negligible given the speed of the LNT compiler, is caused partly by the complexity of the translation (one new LNT channel per parallel composition operator) and partly by the verbosity of LNT w.r.t. the compact algebraic notation of the π-calculus.

The table below shows a few examples from the MWB distribution. For each example, the table gives the number of agents, the size of the specification before and after translation, and some quantitative information (size, time) about the LTS generated using PIC2LNT 2.0 and the CADP exploration tools.

File name	Description	|Agents|	Nb. of lines .pic | .lnt	LTS $\|S\|/\|T\|$	Time
memcell1	Memory cell	2	7 | 82	10 / 100	0.39s
memcell2	Memory cell	2	7 | 91	91 / 910	0.39s
abp-bv	Alternating bit protocol	7	35 | 257	$1,281$ / $4,320$	1.24s
thandover	Mobile network	6	35 | 257	11 / 18	0.56s
handstrong	Mobile network	9	40 | 318	$39,909$ / $76,679$	0.68s
pbool	Boolean operations	6	38 | 950	4 / 678	1.63s

Our tool support for the applied π-calculus is already used for teaching purposes at the University of Saarbrücken (Germany). It is also currently used for specifying and verifying self-deployment and other self-management protocols designed in the context of the OpenCloudware[4] project, which aims at building an open software engineering platform for the collaborative development of distributed applications to be deployed on multiple cloud infrastructures. Since the applied π-calculus is convenient for specifying many kinds of mobile systems (*e.g.*, Web services, autonomic applications, cloud computing protocols, software architectures, biological systems, cryptographic protocols, etc.), we believe that our tool support can provide a useful service in a wide range of application areas.

[4] `http://opencloudware.org`

6 Concluding Remarks

We introduced in this paper an applied π-calculus equipped with data-handling features, and proposed a translation of this language into the LNT value-passing process algebra. This translation, automated by the PIC2LNT 2.0 tool, enables the analysis of applied π-calculus specifications using all verification tools of CADP. As far as we are aware, this results in one of the few operational frameworks for verifying an applied π-calculus. PROVERIF [2] is an alternative approach focused on the verification of cryptographic protocols and security properties (secrecy, authentication, etc.). In contrast, our solution is independent of any application domain and provides a larger panel of verification techniques.

Acknowledgments. We are grateful to Hubert Garavel for his valuable feedback about the applied π-calculus and the connection of PIC2LNT 2.0 to CADP.

References

1. J. A. Bergstra, A. Ponse, and S. A. Smolka, editors. *Handbook of Process Algebra*. Elsevier, 2001.
2. B. Blanchet and B. Smyth. Proverif: Automatic cryptographic protocol verifier, user manual and tutorial (version 1.86pl3). 2011.
3. D. Champelovier, X. Clerc, H. Garavel, Y. Guerte, V. Powazny, F. Lang, W. Serwe, and G. Smeding. Reference manual of the LOTOS NT to LOTOS translator (version 5.1). Inria/Vasy, 109 pages, 2010.
4. M. Dam. Model checking mobile processes. In *Proc. of CONCUR'93*, volume 715 of *LNCS*, pages 22–36. Springer, 1993.
5. G. L. Ferrari, G. Ferro, S. Gnesi, U. Montanari, M. Pistore, and G. Ristori. An automated based verification environment for mobile processes. In *Proc. of TACAS'97*, volume 1217 of *LNCS*, pages 275–289. Springer, 1997.
6. H. Garavel. OPEN/CÆSAR: An open software architecture for verification, simulation, and testing. In *Proc. of TACAS'98*, volume 1384 of *LNCS*, pages 68–84. Springer, 1998.
7. H. Garavel, F. Lang, and R. Mateescu. Compiler construction using LOTOS NT. In *Proc. of TACAS'02*, volume 2304 of *LNCS*, pages 9–13. Springer, 2002.
8. H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2010: A Toolbox for the Construction and Analysis of Distributed Processes. In *Proc. of TACAS'11*, volume 6605 of *LNCS*, pages 372–387. Springer, 2011.
9. R. Mateescu and G. Salaün. Translating pi-calculus into lotos nt. In *Proc. of IFM'10*, volume 6396 of *LNCS*, pages 229–244. Springer, 2010.
10. R. Mateescu and G. Salaün. Pic2Lnt: A translator from the applied pi-calculus to lotos nt. Available at `http://convecs.inria.fr/software`, 2012.
11. R. Mateescu and D. Thivolle. A model checking language for concurrent value-passing systems. In *Proc. of FM'08*, volume 5014 of *LNCS*, pages 148–164. Springer, 2008.
12. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Inf. Comput.*, 100(1):1–77, 1992.
13. U. Montanari and M. Pistore. Checking bisimilarity for finitary pi-calculus. In *Proc. of CONCUR'95*, volume 962 of *LNCS*, pages 42–56. Springer, 1995.
14. B. Victor and F. Moller. The mobility workbench - a tool for the pi-calculus. In *Proc. of CAV'94*, volume 818 of *LNCS*, pages 428–440. Springer, 1994.