Translating Pi-Calculus into LOTOS NT

Radu Mateescu

INRIA – LIG, France

Gwen Salaün

Grenoble INP – INRIA – LIG, France

Introduction

- We present here a novel translation from pi-calculus to a classical process algebra, namely LOTOS NT
- We focus here on the finite control fragment of the pi-calculus
- LOTOS NT being an input language of the CADP toolbox, our approach allows to verify pi-calculus specifications using all the state-of-the-art verification tools available in CADP
- Our translation is fully automated by the pic2lnt prototype tool

Outline

- Pi-Calculus and LOTOS NT
- Translation
- Prototype Tool
- Case Study: A Dispatcher Service
- Concluding Remarks

Pi-calculus

- We consider the original version of Pi-calculus equipped with the early operational semantics
- For the sake of simplicity, we focus on the monadic Pi-calculus, but our translator accepts a polyadic Pi-calculus
- Grammar of Pi-calculus:

$$P ::= 0 \mid \mathrm{tau}.P \mid \underline{x}y.P \mid x(y).P \mid P_1|P_2 \mid P_1+P_2 \mid$$
$$(\mathrm{nu}\ x)P \mid [x=y]P \mid [x!=y]P \mid A(x_1,\ldots,x_r)$$

- Agents do not contain recursive calls through the parallel composition operator (finite control property)

LOTOS NT

- LOTOS NT is a value-passing process algebra with user-friendly syntax and operational semantics

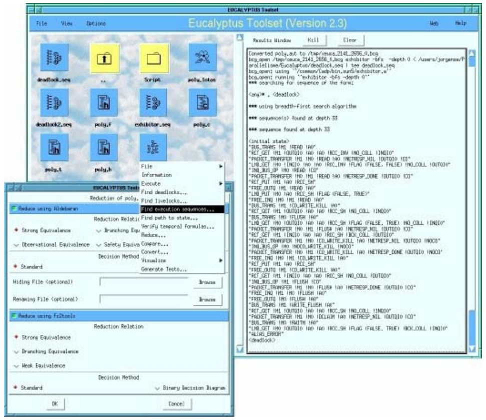
- The specification language consists of two parts:
  - A functional language to describe data types
  - An imperative-like formalism to specify processes

- Grammar of the behavioural LOTOS NT fragment we use:

B ::= **stop** | G(!E, ?X) **where** E' | **if** E **then** B **end if**
| **var** x:T in x:=E ; B **end var** | **hide** G **in** B **end hide**
| P [G_1,..,G_m] (E_1,..,E_n) | **select** B_1 [] ... [] B_n **end select**
| **par** G **in** B_1 || ... || B_n **end par**

- Verification using CADP through a translation to LOTOS

Construction and Analysis of Distributed Processes (CADP)



- Design of asynchronous systems
  - Concurrent processes
  - Message-passing communication
  - Nondeterminism

- Formal approach rooted in concurrency theory: process calculi, Labeled Transition Systems, temporal logics

- Many verification techniques: simulation, model and equivalence-checking, compositional verification, test case generation, performance evaluation, etc

- Numerous practical applications, *e.g.*, telecommunications, middleware and software architectures, hardware

Pi-calculus versus LOTOS NT

Differences

Binary rendez-vous	Multi-way rendez-vous
Unidirectional communication	Bidirectional communication
Mobile channels	Static channels
Dynamic creation of processes	Static network of processes
Names only	Constructed datatypes
Action prefix	Symmetric sequential compo.

Similarities

Choice, recursion

Binary parallel composition

Outline

- Pi-Calculus and LOTOS NT
- Translation
- Prototype Tool
- Case Study: A Dispatcher Service
- Concluding Remarks

Channel Names (1/2)

- Two classes of channels, public (G_{pub}) and private (G_{priv}), used to model non-synchronized communications
- We cannot use LOTOS NT static gates to represent mobile communication
- We represent Pi-calculus channel names as values of a LOTOS NT datatype Chan
- We model channel mobility between Pi-calculus agents by communicating values of this type along gates

Channel Names (2/2)

The following type Chan is generated for (nu x)(ab.cx.0)

```
type Chan is
    a, b, c, x(id:Nat) with "==", "!="
end type
```

```
function new_id () : Nat is
    !external null
end function
```

```
function is_public (ch:Chan) : Bool is
        case ch in
                a|b|c → return true
                | any → return false
        end case
end type
```

Action Prefix (1/2)

- The translation takes as input:
  - A Pi-calculus agent P,
  - The gates <u>G</u> on which P communicates with its environment, and
  - A natural k (pid) identifying the concurrent activity
- Communication on a channel x is translated using a choice on all gates <u>G</u> connecting the term P to its environment
- Binary unidirectional communications are encoded using different gate names (one for each |) and identifying explicitly the sender and receiver using placeholders

Action Prefix (2/2)

t($\underline{x}$y.P,{G_1,...,G_n,G_{pub},G_{priv}},k) =

select var r:Nat **in**

G_1 (!x,!y,!k,?r) [] ... [] G_n (!x,!y,!k,?r) []

G_{pub} (!x,!y,!true) **where** is_public(x) []

G_{priv} (!x,!y,!true) **where** not(is_public(x))

end select; t(P,{G_1,...,G_n,G_{pub},G_{priv}},k)

---

t(x(y).P,{G_1,...,G_n,G_{pub},G_{priv}},k) =

select var s:Nat, y:Chan **in**

G_1 (!x,?y,?s,!k) [] ... [] G_n (!x,?y,?s,!k) []

G_{pub} (!x,?y,!false) **where** is_public(x) []

G_{priv} (!x,?y,!false) **where** not(is_public(x))

end select; t(P,{G_1,...,G_n,G_{pub},G_{priv}},k)

Sum, Match, Mismatch, Parallel, Channel Creation

t(P_1+P_2,<u>G</u>,k) = **select** t(P_1,<u>G</u>,k) [] t(P_2,<u>G</u>,k) **end select**

---

t([x=y]P,<u>G</u>,k) = **if** x==y **then** t(P,<u>G</u>,k) **end if**

t([x!=y]P,<u>G</u>,k) = **if** x!=y **then** t(P,<u>G</u>,k) **end if**

---

t(P_1|P_2,<u>G</u>,k) = **hide** G_{new} **in par** G_{new} **in**
t(P_1,<u>G</u> U {G_{new}},2k) || t(P_2,<u>G</u> U {G_{new}},2k+1)
end par end hide

---

t((nu x)P,<u>G</u>,k) = **var** x:Chan **in** x:=x(new_id()); t(P,<u>G</u>,k) **end var**

Agent Definition / Instantiation, Main Specification

$t(A(x_1,\ldots,x_r)=P,\underline{G},k)$ = **process** A_d [$\underline{G}$] ($x_1,\ldots,x_r$:Chan, k:Nat) **is**

$t(P,\underline{G},k)$

end process

$t(A(y_1,\ldots,y_r),\underline{G},k) = A_d\ [\underline{G}]\ (y_1,\ldots,y_r,k)$

---

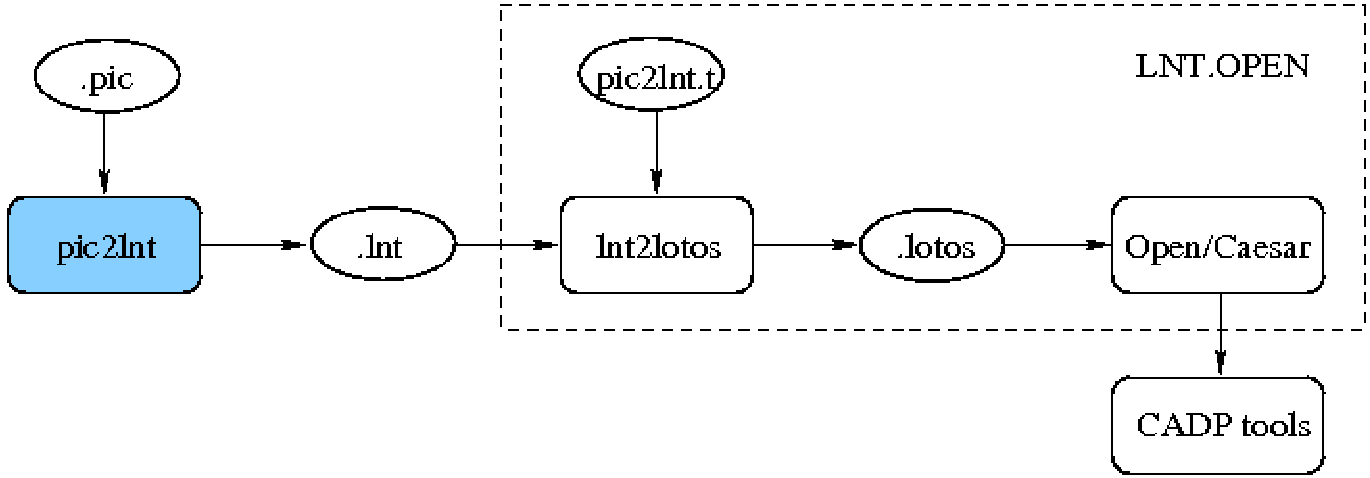
pic2Int(P) = **par** G_{priv} **in** $t(P,\{G_{pub},G_{priv}\},1)$ || **stop end par**

Outline

- Pi-Calculus and LOTOS NT
- Translation
- Prototype Tool
- Case Study: A Dispatcher Service
- Concluding Remarks

Prototype Tool

- The translation is completely automated by a tool we implemented



- Our benchmark currently contains 160 files:

  2000 lines of Pi-calculus → 23000 lines of LOTOS NT

Outline

- Pi-Calculus and LOTOS NT
- Translation
- Prototype Tool
- Case Study: A Dispatcher Service
- Concluding Remarks

A Dispatcher Service in Pi-Calculus

Main = (nu req, a, b, c)

(Client(req,a,b,c) | Dispatcher(req) | Server(a) | Server(b) | Server(c))

Client (req,a,b,c) = (nu x) (request a.req<a,x>.ClientAux(req,a,a,b,c,x)) +

(nu x) (request b.req<b,x>.ClientAux(req,b,a,b,c,x)) +

(nu x) (request c.req<c,x>.ClientAux(req,c,a,b,c,x))

ClientAux(req,k,a,b,c,x) =

x(info).(x purchase.purchase k.0 + x refuse.refuse k.Client(req,a,b,c))

Dispatcher(req) = req(k,x).k x.Dispatcher(req)

Server(k) = k(x).x info.x(decision).Server(k)

Dispatcher Service in LOTOS NT (1/2)

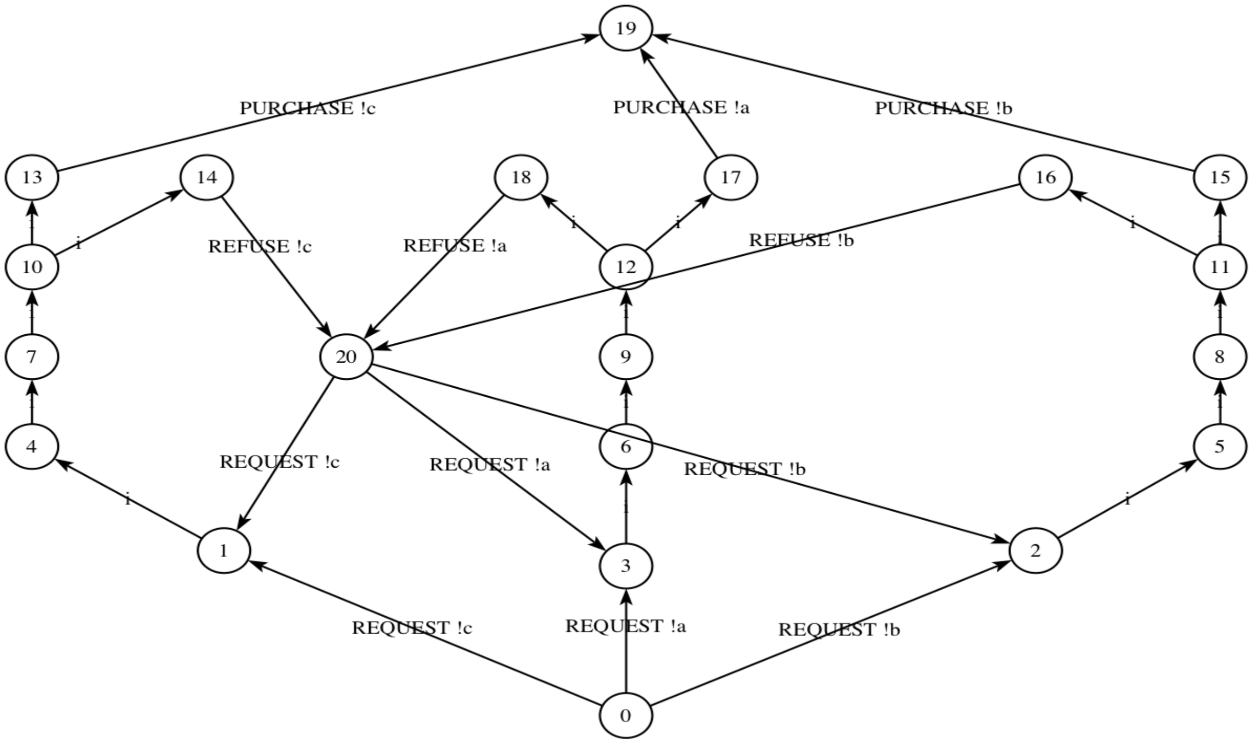
```
process MAIN [PUBLIC,PRIVATE:any] is
 var req, a, b, c:Chan in
  req:=req(new_id()); a:=a(new_id()); b:=b(new_id()); c:=c(new_id());

  hide G0:any in par G0 in hide G1:any in par G1 in
  hide G2:any in par G2 in hide G3:any in par G3 in
     Client_4 [PUBLIC,PRIVATE,G0,G1,G2,G3] (req,a,b,c,2)
   || Dispatcher_4 [PUBLIC,PRIVATE,G0,G1,G2,G3] (req,6)      end par end hide
   || Server_3 [PUBLIC,PRIVATE,G0,G1,G2] (a,14)               end par end hide
   || Server_2 [PUBLIC,PRIVATE,G0,G1] (b,30)                  end par end hide
   || Server_1 [PUBLIC,PRIVATE,G0] (c,31)                     end par end hide
 end var
end process
```

Dispatcher Service in LOTOS NT (2/2)

```
process Dispatcher_4 [PUBLIC,PRIVATE,G0,G1,G2,G3:any] (req:Chan,pid:Nat) is
 select var k,x:Chan, s:Nat in
  G0 (!req, ?k, ?x, ?s, !pid) [] G1 (!req, ?k, ?x, ?s, !pid) []
  G2 (!req, ?k, ?x, ?s, !pid) [] G3 (!req, ?k, ?x, ?s, !pid) []
  PUBLIC (!req, ?k, ?x, !false) where is_public(req) []
  PRIVATE (!req, ?k, ?x, !false) where not(is_public(req))
 end select ;
 select var r:Nat in
  G0 (!k, !x, !pid, ?r) [] G1 (!k, !x, !pid, ?r) []
  G2 (!k, !x, !pid, ?r) [] G3 (!k, !x, !pid, ?r) []
  PUBLIC (!k, !x, !true) where is_public(k) []
  PRIVATE (!k, !x, !true) where not(is_public(k))
 end select ; Dispatcher_4 [PUBLIC,PRIVATE,G0,G1,G2,G3] (req,pid)
end process
```

LTS of the Dispatcher Service



One can use for instance the Evaluator model-checker to check MCL formulas, *e.g.*, "*each request submitted by the client is eventually answered positively*"

Outline

- Pi-Calculus and LOTOS NT
- Translation
- Prototype Tool
- Case Study: A Dispatcher Service
- Concluding Remarks

Concluding Remarks

- We have presented a translation from the finite fragment of the Pi-calculus to LOTOS NT
- This translation makes possible to analyze Pi-calculus specifications using the CADP verification tools
- The translation is fully automated by the pic2lnt tool we implemented and validated on many examples
- Main perspective: extending the Pi-calculus with data-handling features to widen its possible application domains (applied Pi-calculus)