Process Calculi for WSNs and more

Types and tools

Ramūnas Gutkovas

ProFUN meeting
2014 October 22
Uppsala University
In 2010, Toyota recalled 400,000 vehicles to correct a software “glitch” in ABS.

Formal Verification
Show the absence of bugs!

Testing shows the presence, not the absence of bugs!
- E. W. Dijkstra
A framework for mobile process calculi ("pi-calculus extensions") for **applications**

- Straightforward semantics, **reusable** theory (holds in all psi-calculi)

- **Correct**: machine-checked proofs! (Isabelle with Nominal Package)
Syntax

$$\overline{M} \; N.\; P$$

output

object

subject

input

pattern

assertion

$$\overline{M} (\lambda \tilde{x}) \underbrace{N}.\; P$$

“facts”

condition

case $$\varphi_1 : P_1 \; \cdots \; \varphi_n : P_n$$

the usual:

$$0\; \; (\nu a)P\; \; P \; | \; Q\; \; !P$$

Parameters:

- $$M, N$$: T (terms)
- $$\varphi$$: C (conditions)
- $$\Psi$$: A (assertions)

like guarded commands, if-then-else
Cook a psi-calculus

Define terms $T$ (e.g. data terms, channels)

conditions $C$ (e.g. for if-then-else)

assertions $A$ (statements about e.g. terms)

can be practically anything
Cook a psi-calculus

Define terms \( T \), conditions \( C \), assertions \( A \)

Define substitution on these \((\text{satisfy axioms})\)

Define operators:

\[\Leftrightarrow: T \times T \rightarrow C\] Channel equivalence

\[\otimes: A \times A \rightarrow A\] Composition

\[1: A\] Unit assertion

\[\vdash \subseteq A \times C\] Entailment

Broadcast Output Connectivity

Broadcast Input Connectivity
Example

\[ M ::= \text{init}(M) | a | i \in \mathbb{N} \]
\[ \varphi ::= M = M' | M \prec M' \]
\[ \Psi ::= M \prec M', \Psi | \epsilon \]

\text{init(1)}123.0 | \text{init(2)}(\lambda x)x.0 | \text{init(3)}(\lambda y)y.\text{case } y = 3 : P | (1 \prec 2, 1 \prec 3) \]
Example

\[ M ::= \text{init}(M) \mid a \mid i \in \mathbb{N} \]

\[ \varphi ::= M = M' \mid M \prec M' \]

\[ \Psi ::= M \prec M', \Psi \mid \varepsilon \]

Transition relation \( \sim \) semantics

\[ \text{init}(1)123.0 \mid \text{init}(2)(\lambda x)x.0 \mid \text{init}(3)(\lambda y)y.\text{case } y = 3 : P \mid (1 < 2, 1 < 3) \]

\[ \alpha \]

\[ 0 \mid 0 \]

\[ \text{case } 123 = 3 : P[y := 123] \mid (1 < 2, 1 < 3) \]
Example

\[ M ::= \text{init}(M) \mid a \mid i \in \mathbb{N} \]

\[ \varphi ::= M = M' \mid M \prec M' \]

\[ \Psi ::= M \prec M', \Psi \mid \epsilon \]

\[
\begin{array}{c}
\text{User defined logic}\\
\hline
\Psi, M \prec M' \vdash \text{init}(M) \prec \text{init}(M')\\
\hline
\Psi \vdash M = M' \text{ if } M = M'
\end{array}
\]

\[
\begin{array}{c}
\text{User defined logic}\\
\hline
\text{init}(1)123.0 \mid 0 \mid 0 \\
\text{init}(2)(\lambda x)x.0 \mid 0 \mid 0 \\
\text{init}(3)(\lambda y)y.\text{case } y = 3 : P \mid \text{case } 123 = 3 : P[y := 123] \mid (1 < 2, 1 < 3)
\end{array}
\]

\[
\alpha
\]

\[
\begin{array}{c}
\end{array}
\]
Recent Advancements to Psi
Crypto Example

Term for encryption: $\text{enc}(M, K)$

We need a way to control what are pattern variables.

Knowledge of the key:

$(\nu k)(\overline{M} \text{enc}(a, k).P) \mid M(\lambda x, y)\text{enc}(x, y).Q)$

$\rightarrow (\nu k)(P \mid Q[x := a, y := k])$

$(\nu k)(\overline{M} \text{enc}(a, k).P) \mid M(\lambda x)\text{enc}(x, k).Q)$

$\rightarrow (\nu k)(P \mid Q[x := a])$
Useful computation to have as part of substitution

$$\text{dec}(\text{enc}(M, K), K) \rightarrow M$$

However, the substitutions are not allowed to lose names

$$\text{dec}(\text{enc}(a, b), b)[b := k] \rightarrow a$$

$k$ does not appear in the result
in [TGC’13]

Generalised Pattern Matching

User defined pattern matching. Relaxes requirement on the substitution.

**X** patterns, ranged over by **X, Y**

MATCH : **T** × **N**^* × **X** → **P**(**T**^*)

VARS : **X** → **P**(**P**(**N**))

Signifies which names are patterns

Ex:

\[ \text{VARS}(\text{enc}(m, k)) = \{m\} \]

\[ M(\lambda m)\text{enc}(m, k).P \]

\[ M(\lambda m, k)\text{enc}(m, k).P \]
Results

• Previous Psi results hold: compositional semantics, behavioural equivalence is a congruence

• well-formedness of processes is preserved by transitions

\[ P \rightarrow P' \]
Polyadic communication

Polyadic pi-calculus

\[
a(x_1, \ldots, x_n).P \mid \overline{ab}_1, \ldots, b_n.Q \rightarrow P\{b_1, \ldots, b_n/x_1, \ldots, x_n\} \mid Q
\]

Should be easy to express in Psi

Let’s take \( T = N^* \)

Substitution needs to be a total function

\[
(a, b, c)[a := (c, d)] = ((c, d), b, c) \notin N^*
\]
Goal: flexible definition of “well-formed”

\[
\text{SORT} : \mathcal{N} \cup \mathcal{T} \cup \mathcal{X} \rightarrow \mathcal{S}
\]

name, term, and pattern sorting

is well-sorted iff

substitution \[\tilde{a} := \tilde{N}\]

\[\text{SORT}(a_i) \prec \text{SORT}(N_i)\]

restriction \[(\nu a)P\]

\[\text{SORT}(a) \in \mathcal{S}_\nu\]

output \[\overline{M} \ N. P\]

\[\text{SORT}(M) \prec \text{SORT}(N)\]

input \[\overline{M}(\lambda \tilde{x})X. P\]

\[\text{SORT}(M) \prec \text{SORT}(X)\]
Polyadic Pi-calculus

\[ \text{SORT}(a) = \text{chan} \]
\[ \text{SORT}(\tilde{a}) = \text{tup} \]
\[ \overline{\alpha} = \alpha = \{(\text{chan}, \text{tup})\} \]

\[ \text{VARS}(\langle \tilde{a} \rangle) = \{\tilde{a}\} \]
all names in input pattern must be bound

\[ \text{MATCH}(\langle \tilde{a} \rangle, \tilde{x}, \langle \tilde{x} \rangle) = \{\tilde{a}\} \text{ if } |\tilde{a}| = |\tilde{x}| \]

\[ \langle \tilde{a} \rangle \text{ matches the pattern } \langle \tilde{x} \rangle \text{ binding } \tilde{x}, \text{ then substituting } \tilde{a} \text{ for } \tilde{x} \]

\[ c(\lambda \tilde{x})\langle \tilde{x} \rangle.P \xrightarrow{c \tilde{a}} P[\tilde{x} := \tilde{a}] \]

Formal correspondence of transitions and equivalence

A channel can send/receive a tuple

_\alpha(\lambda \tilde{x})\langle \tilde{x} \rangle.P_
Session types
Broadcast

in [PLACES’14]
Session Types

Specification of process that checks equality over a channel of type

CheqEqSrv = ?[int].?[int].![bool].end

Possible implementation

SrvImp(c) = c(x).c(y).\texttt{case } x = y : \texttt{true}.0 \parallel x \neq y : \texttt{false}.0
Session Types

Specification of process that checks equality over a channel of type

CheqEqSrv = ?[int].?[int].![bool].end
Clt = ![int].![int].?[bool].end

Possible implementation

SrvImp(c) = c(x).c(y).case x = y : c true.0 | x ≠ y : cfalse.0
CltImp(k) = k1.k2.k(b).0

Duals!
Session Types

Specification of process that checks equality over a channel of type

\[ \text{CheqEqSrv} = ?[\text{int}].?[\text{int}].![\text{bool}].\text{end} \]
\[ \text{Clt} = ![\text{int}].![\text{int}].?[\text{bool}].\text{end} \]

Possible implementation

\[ \text{SrvImp}(c) = c(x).c(y).\text{case } x = y : \bar{c}\text{true.0 } \parallel x \neq y : \bar{c}\text{false.0} \]
\[ \text{CltImp}(k) = \bar{k}1.\bar{k}2.\bar{k}(b).0 \]

System

\[ (\nu c)(\text{SrvImp}(c^+) \mid \text{CltImp}(c^-)) \]
Session Types

- Structured Description of a protocol
- Specifies direction and data carried over channel
- Abstract specification
- Safety: progress, session fidelity
Broadcast Session Types

- First Application of session types to Unreliable and Broadcast communication systems
- Types for scatter & gather communication pattern
Scatter & Gather

Type

$c^+ : ![\text{int}].?[\text{int}].T$

$c^+ x. c^+(y).P$

- Runtime tracking of session state
- Extended notion of duality

$c^-(x).c^-y.Q_i$
Unreliability

Let process recover

\((\nu c)(c^+(x).c^+(y).0 \mid \overline{c^{-2.0}})\)

Process no longer consistent with the type!
Psi-calculi Workbench

to appear in ACM transactions on embedded systems
Tools

Tool is essential for verifying non-trivial systems!

Many tools

mCRL2

ABC  SBC  PiET

Concurrency Workbench

ProVerif

Mobility Workbench

Petruochio

PAT3

But specialised!
Psi-Calculi Workbench

- Tool factory: define your own tool!
- Based on the parametric psi-calculi framework
Features

Communication Primitives
- Unicast
- Wireless Broadcast

Parametric On

Data Structures
- e.g., Names, Bits, Vectors, ADTs, Trees, ...

Logics
- e.g., EUF, FOL, Equational Theory, ...

Logical Assertions
- e.g., Knows a secret, Connectivity, Constraints...
Pwb Functionality

Symbolic Execution

\[ \Psi \triangleright P \xrightarrow{\alpha} P' \]

Symbolic Behavioral Equivalence Checking

\[ P \sim Q \]

Symbolic Constraints
Parametric Architecture

Supporting library
Parametric Architecture

User Supplied
- Pretty Printer
- Parser
- Equivalence Constraint Solver
- Execution Constraint Solver
- Data
- Logics
- Assertions

Pwb
- Command Interpreter
- Symbolic Equivalence Checker
- Symbolic Execution
- Psi Calculi Core

Supporting library
Data Collection in Wireless Sensor Networks

1. Routing tree
2. Data collection
Specification in Pwb

Node Behavior

Sink(nodeId, sinkChan) <=
  "init(nodeId)"! <sinkChan> .
  ! "data(sinkChan)"(x). ProcData<x> ;

Node(nodeId, nodeChan, datum) <=
  "init(nodeId)"? (chan) .
  "init(nodeId)"! <nodeChan> .
  "data(chan)"<datum> .
  ! "data(nodeChan)"(x).
  "data(chan)"<x> ;

System

(new sinkChan)  Sink<0, sinkChan> | graph represented as edge list
(new chan1)     Node<1, chan1, datum1> | (0,1), (0,2), (1,2)
(new chan2)     Node<2, chan2, datum2>
Example Transition

(new sinkChan) Sink<0, sinkChan>
(new chan1) Node<1, chan1, datum1>
(new chan2) Node<2, chan2, datum2>

"init(0)"!(new sinkChan)sinkChan
true

(!("data(sinkChan)"(gnb). ProcData<gnb>))
(((new chan1)(
  "init(1)"!<chan1>.
  "data(sinkChan)"<datum1>.
  !("data(chan1)"(gnb). 
  "data(sinkChan)"<gnb>)))
| ((new chan2)(
  "init(2)"!<chan2>.
  "data(sinkChan)"<datum2>.
  !("data(chan2)"(gnb). 
  "data(sinkChan)"<gnb>))))
Example Summary

- Executable model of an aggregation-tree building protocol
- Connectivity graph expressed as an assertion (possible to add and remove edges at runtime)
- Mix of wireless broadcast and reliable unicast communication
Getting the tool

http://www.it.uu.se/research/group/mobility/applied/psiworkbench

Dependency: polyml
Current Work: SHIA[CCS’06]

- Case study in Pwb
- Large system
- Cryptography, Arithmetic, Equations
- Infrastructure: better framework for constraint solvers, term algebras, verification
- Goal to check safety properties (deadlock freedom) and security property “optimal security” [ccs06]
Conclusion

- A parametric verification tool the Psi-Calculi Workbench
- Session types for broadcast communication and unreliable systems
- More expressivity: generalised pattern-matching and sorts
Thank you for listening