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
Psi-calculi

- 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

output
subjects
input

\( \overline{M} \ N.\ P \)

object
pattern

assertion

\( \overline{M} (\lambda \vec{x}) \overline{N}.\ P \)

condition

\( \mid \Psi \mid \)

“facts"

\textbf{case} \ \varphi_1 : P_1 \ [\cdots [\varphi_n : P_n]

the usual:

0 \ (\nu a)P \ P \mid Q \ !P

Parameters:

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

like guarded commands, \textbf{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)

$M, N$

$\varphi$

$\Psi$

can be practically anything
Cook a psi-calculus

Define terms $\mathbf{T}$, conditions $\mathbf{C}$, assertions $\mathbf{A}$

Define substitution on these (satisfy axioms)

Define operators:

$\leftrightarrow : \mathbf{T} \times \mathbf{T} \to \mathbf{C}$ Channel equivalence

$\otimes : \mathbf{A} \times \mathbf{A} \to \mathbf{A}$ Composition

$1 : \mathbf{A}$ Unit assertion

$\vdash \subseteq \mathbf{A} \times \mathbf{C}$ Entailment

$\bowtie : \mathbf{T} \times \mathbf{T} \to \mathbf{C}$ Broadcast Output Connectivity

$\bowtie : \mathbf{T} \times \mathbf{T} \to \mathbf{C}$ Broadcast Input Connectivity

$\tilde{a} := \tilde{M}$
Example

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

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

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

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

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

\[ \Psi, M < M' \vdash \text{init}(M) < \text{init}(M') \]
\[ \Psi \vdash M = M' \text{ if } M = M' \]

**User defined logic**

**init(1)** 123.0 | \[123.0 | \]
**init(2)** (λx)x.0 | \[0 | \]
**init(3)** (λy)y.case \[y = 3 : P] | \(\text{case } 123 = 3 : P[y := 123] \mid (1 < 2, 1 < 3)\)
Recent Advancements to Psi
Crypto Example

Term for encryption

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

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

Knowledge of the key

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

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

We need a way to control what are pattern variables
Computation

Useful computation to have as part of substitution

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

However, the substitutions are not allowed to lose names

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

\textit{k} does not appear in the result
Generalised Pattern Matching

User defined pattern matching. Relaxes requirement on the substitution.

- **MATCH**: $T \times N^* \times X \rightarrow \mathcal{P}(T^*)$
  - Pattern matching

- **VARS**: $X \rightarrow \mathcal{P}(\mathcal{P}(N))$
  - Pattern variables

Signifies which names are patterns

Ex:

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

$\overline{M(\lambda \tilde{x})X.P}$
- well-formed if $\tilde{x} \in \text{VARS}(X)$

Ex:

$\overline{M(\lambda m)\text{enc}(m, k).P}$

$\overline{M(\lambda m, k)\text{enc}(m, k).P}$

in [TGC’13]
Results

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

• Well-formedness of processes is preserved by transitions.

\[ P \rightarrow P' \]

Well-formed did not break psi
Polyadic communication

Polyadic pi-calculus

\[ a(x_1, \ldots, x_n).P \begin{array}{c} \mid \\bar{a}b_1, \ldots, b_n.Q \end{array} \rightarrow P\{b_1, \ldots, b_n/x_1, \ldots, x_n\} \mid Q \]

Should be easy to express in Psi

Let’s take \( T = N^* \)

Substitution needs to be a total function

\[(a, b, c)[a := (c, d)] = ((c, d), b, c) \notin N^*\]
in [TGC’14] a.k.a. Types

Sorts

Goal: flexible definition of “well-formed”

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

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 \(\overline{M} N.P\)
  \(\text{SORT}(M) \backsim \text{SORT}(N)\)

- input \(M(\lambda \tilde{x})X.P\)
  \(\text{SORT}(M) \backsim \text{SORT}(X)\)
Polyadic Pi-calculus

\[
\begin{align*}
\text{SORT}(a) &= \text{chan} \\
\text{SORT}(\tilde{a}) &= \text{tup} \\
\bar{\bar{x}} &= \bar{\bar{x}} = \{(\text{chan, tup})\} \\
\text{VARS}(\langle \tilde{a} \rangle) &= \\{\tilde{a}\} \\
\text{MATCH}(\langle \tilde{a}, \tilde{x}, \langle \tilde{x} \rangle \rangle) &= \{\tilde{a}\} \text{ if } |\tilde{a}| = |\tilde{x}| \\
\langle \tilde{a} \rangle \text{ matches the pattern } \langle \tilde{x} \rangle \text{ binding } \tilde{x}, \text{ then substituting } \tilde{a} \text{ for } \tilde{x}
\end{align*}
\]

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

Formal correspondence of transitions and equivalence
Session types
Broadcast

in [PLACES’14]
Session Types

Specification of process that checks equality over a channel of type

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

Possible implementation

\[
\text{SrvImp}(c) = c(x).c(y).\text{case } x = y : \text{true}.0 \mid x \neq y : \text{false}.0
\]
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

\[ \text{SrvImp}(c) = c(x).c(y).\text{case } x = y : \overline{c}\text{ true.} \text{0 } \mid x \neq y : \overline{c}\text{ false.} \text{0} \]
\[ \text{CltImp}(k) = \overline{k1.\overline{k2}.k(b).} \text{0} \]
Session Types

Specification of process that checks equality over a channel of type

\[
\text{CheqEqSrv} = \dagger[\text{int}].\dagger[\text{int}].\dagger[\text{bool}].\text{end}
\]

\[
\text{Clt} = \dagger[\text{int}].\dagger[\text{int}].\dagger[\text{bool}].\text{end}
\]

Possible implementation

\[
\text{Sr} \text{vImp}(c) = c(x).c(y).\text{case } x = y : \overline{c}\text{true.0} \mid x \neq y : \overline{c}\text{false.0}
\]

\[
\text{CltImp}(k) = \overline{k1}.\overline{k2}.k(b).0
\]

System

\[
(\nu c)(\text{Sr} \text{vImp}(c^+) \mid \text{CltImp}(c^-))
\]

\[
c^+ : \text{CheqEqSrv}
\]

\[
c^- : \text{Clt} = \text{CheqEqSrv}
\]
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

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

\( \overline{c^+ x. c^+(y)}.P \)

- Runtime tracking of session state
- Extended notion of duality

\( c^-(x).\overline{c^-y}.Q_i \)
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
Concurrency Workbench
PiET
ProVerif

But specialised!

Mobility Workbench
Petruchio
PAT3
Psi-Calculi Workbench

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

Communication Primitives

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

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

Symbolic Behavioral Equivalence Checking

\[ P \sim Q \]
Parametric Architecture

Pwb

Command Interpreter

Symbolic Equivalence Checker

Symbolic Execution

Psi Calculi Core

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>  |  (new chan1)  Node<1, chan1, datum1>  |  (new chan2)  Node<2, chan2, datum2>

Node Connectivity for Broadcasting

Sink

0

Node
1
2

graph represented as edge list

(0,1), (0,2), (1,2)
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>))))

broadcasts

---
can unicast
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