Advancing Concurrent System Verification

Type Based Approach and Tools

Ramūnas Gukovas
Licentiate Seminar
2014 October 20
Uppsala University

Supported by ProFuN
In 2010, Toyota recalled 400,000 vehicles to correct a software “glitch” in ABS.

Formal Verification
Show the absence of bugs!

Toyota Prius

Testing shows the presence, not the absence of bugs!
- E. W. Dijkstra
Background
Wireless Sensor Network

ProFuN project
Wireless Sensor Network

ProFuN project
Wireless Sensor Network

ProFuN project
Psi-calculi
Psi-calculi?

- A family of languages, known as process calculi, for modelling concurrent systems
- 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

**Parameters:**
- $\mathcal{M}, \mathcal{N}: \mathbf{T}$ (terms)
- $\varphi: \mathbf{C}$ (conditions)
- $\Psi: \mathbf{A}$ (assertions)

**Syntax Rules:**
- $\overline{M} \mathcal{N}.P$
- $\overline{M}(\lambda \bar{x})\mathcal{N}.P$
- **Case** $\varphi_1: P_1 \quad \cdots \quad \varphi_n: P_n$
- The usual: $0 \quad (\nu a)P \quad P \mid Q \quad !P$

**Labels:**
- Output
- Subjects
- Input
- Object
- Pattern
- Assertion
- “facts”
- Condition

**Examples:**
- Like guarded commands, if-then-else
Cook a psi-calculus

Define terms $\mathbf{T}$ (e.g. data terms, channels)

conditions $\mathbf{C}$ (e.g. for if-then-else)

assertions $\mathbf{A}$ (statements about e.g. terms)

M, N
\mathcal{\varphi}
\Psi

...can be practically anything...
Cook a psi-calculus

Define terms $T$, conditions $C$, assertions $A$

Define substitution on these \[ M, N \varphi \Psi \]

Define operators: $\lhd: T \times T \to C$ Channel equivalence

$\otimes: A \times A \to A$ Composition

$1: A$ Unit assertion

$\vdash \subseteq A \times C$ Entailment

$\prec: T \times T \to C$ Broadcast Output Connectivity

$\succ: T \times T \to C$ Broadcast Input Connectivity

(satisfy axioms)

$[\widetilde{a} := \widetilde{M}]$ (practically anything)
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{align*}
\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 \prec 2, 1 \prec 3)
\end{align*}
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 \( \sim \) semantics

\[ \begin{align*}
\text{init}(1) & 123.0 \mid 0 \\
\text{init}(2) & (\lambda x)x.0 \mid 0 \\
\text{init}(3) & (\lambda y)y.\text{case } y = 3 : P \mid (1 < 2, 1 < 3) \\
\text{case } 123 = 3 & : P[y := 123] \mid (1 < 2, 1 < 3)
\end{align*} \]
**Example**

### Syntax:

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

### Logic:

$$\Psi, M < M' \vdash \text{init}(M) < \text{init}(M')$$

$$\Psi \vdash M = M' \text{ if } M = M'$$

### Example:

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

### Case:

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

**User defined logic**
Advancing Concurrent System Verification

- A tool factory the Psi-Calculi Workbench for concurrent system verification
- Session types for broadcast communication and unreliable systems
- More expressivity: generalised pattern-matching and sorts for psi-calculi
1. INTRODUCTION

The development of concurrent systems is greatly helped by the use of precise and formal models of the system. There are many different formalisms for concurrent systems, often in specialized versions for particular application areas. For such formalism, tool support is necessary for constructing and reasoning about models of non-trivial systems. This paper describes such tool support for a generic semantic framework for process calculi with mobility. Thus, instead of developing a separate tool for each separate process calculus, we develop one single generic tool for a whole family of process calculi with mobility. Thus, instead of developing a separate tool for each separate process calculus, we develop one single generic tool for a whole family of process calculi with mobility.

In this paper we propose a session type system for unreliable broadcast communication. Developing such a system was challenging for two reasons: (i) we needed to extend binary session types to handle asynchrony as well as extending the notion of session endpoint, and (ii) the reactive control flow of a broadcasting system drives us to consider typical patterns of communication interaction rather than communication prefixes. The key ideas are (i) to break the symmetry between the ! and ? ends of points of channel x, allowing x (uniquely owned) to broadcast and !, x to be shared; (ii) to implement (and type) the gap operation in an abstract receive. We retain the standard binary session type constructors.

Session types [7, 6] allow communication protocols to be specified as types and verified by type-checking. Up to now, session type systems have assumed reliable, point to point message passing communication. Reliability is important to maintain send/receive duality, and point to point communication is required to ensure session endpoint linear.

In this paper we propose a session type system for unreliable broadcast communication. Developing such a system was challenging for two reasons: (i) we needed to extend binary session types to handle asynchrony as well as extending the notion of session endpoint, and (ii) the reactive control flow of a broadcasting system drives us to consider typical patterns of communication interaction rather than communication prefixes. The key ideas are (i) to break the symmetry between the ! and ? ends of points of channel x, allowing x (uniquely owned) to broadcast and !, x to be shared; (ii) to implement (and type) the gap operation in an abstract receive. We retain the standard binary session type constructors.

We use ψ-calculi [1] as the underlying process framework, and specifically we use the extension of the ψ-calculus family with broadcast semantics [2]. ψ-calculi provide a parametric process calculus framework for extending the semantics of the π-calculus with arbitrary data structures and logical assertions. Expressing our work in the ψ-calculus framework allows us to avoid defining a new operational semantics, instead defining the semantics of our broadcast session calculus by translation into a broadcast ψ-calculi.

Establishing a link between session types and ψ-calculi is therefore another contribution of our work.

1. INTRODUCTION

There is today a growing number of high-level constructs in the area of concurrency. Examples include type systems, communication with pattern matching, encryption primitives, concurrent constraints, nondeterminism, process creation, and dynamic connection topologies. Several such formalisms, e.g. the applied π-calculus, are extensions of the π-calculus; a growing number is geared towards particular applications or computational paradigms.

Our goal is a unified framework to represent different process calculi and notions of computation. To this end, we extend our previous work on ψ-calculi with novel abstract patterns and pattern matching, and add sorts to the data terms language, giving sufficient criteria for subject reduction to hold. Our framework can accommodate existing process calculi; the resulting transition systems are isomorphic to the originals up to strong bisimulation. We also demonstrate different notions of computation on data terms, including cryptographic primitives and broadcast calculus with erasure choice. Finally, we prove standard correctness and structural properties of bisimulation, substantial parts of the proof have been machine-checked using Nominal Isabelle.

1. INTRODUCTION

There is today a growing number of high-level constructs in the area of concurrency. Examples include type systems, communication with pattern matching, encryption primitives, concurrent constraints, nondeterminism, process creation, and dynamic connection topologies. Several such formalisms, e.g. the applied π-calculus, are extensions of the π-calculus; a growing number is geared towards particular applications or computational paradigms.

Our goal is a unified framework to represent different process calculi and notions of computation. To this end, we extend our previous work on ψ-calculi with novel abstract patterns and pattern matching, and add sorts to the data terms language, giving sufficient criteria for subject reduction to hold. Our framework can accommodate existing process calculi; the resulting transition systems are isomorphic to the originals up to strong bisimulation. We also demonstrate different notions of computation on data terms, including cryptographic primitives and broadcast calculus with erasure choice. Finally, we prove standard correctness and structural properties of bisimulation, substantial parts of the proof have been machine-checked using Nominal Isabelle.
Contributions
Tools

Tool is essential for verifying non-trivial systems!

Many tools
mCRL2
ABC
SBC
PiET
Concurrency Workbench
ProVerif
Mobility Workbench
Petruchio
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

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

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, sinkChan>
| Node<1, chan1, datum1>  |
| Node<2, chan2, datum2>  |

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>))))
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
Session Types

Specification of process that checks equality over a channel of type `int`:

```plaintext
CheqEqSrv = {[int].?[int].![bool].end
```

Possible implementation:

```plaintext
SrvImp(c) = c(x).c(y).case x = y : true.0 | x ≠ y : 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 : \overline{c} true.0 \mid x \neq y : \overline{c} false.0
CltImp(k) = \overline{k}1.\overline{k}2.k(b).0
```
Session Types

Specification of process that checks equality over a channel of type

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

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

Possible implementation

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

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

System

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

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

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

$$(\nu c)(c^+(x) . c^+(y) . 0 \mid c^- 2.0)$$

Process no longer consistent with the type!
Results

• We are the first to introduce session types to unreliable and broadcast systems

• Well-typed processes always transition to well-typed processes

• Well-typed process does not reduce to an error
Crypto Example

Term for encryption: $\text{enc}(M, K)$
- Text: $\lambda a, k.\text{enc}(a, k)$
- Key: $\lambda x, y.\text{enc}(x, y)$

Knowledge of the key:
- $M(\lambda x, y.\text{enc}(x, y)).Q$
- $\rightarrow (\nu k)(P \mid Q[x := a, y := k])$

We need a way to control what are pattern variables:
- $\rightarrow (\nu k)(P \mid Q[x := a])$
Computation

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
Generalised Pattern Matching

User defined pattern matching. Relaxes requirement on the substitution.

\[ \text{MATCH} : \mathbf{T} \times \mathcal{N}^* \times \mathbf{X} \rightarrow \mathcal{P}(\mathbf{T}^*) \]

\[ \text{VARS} : \mathbf{X} \rightarrow \mathcal{P}(\mathcal{P}(\mathcal{N})) \]

Signifies which names are patterns

Ex:

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

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

well-formed if \( \tilde{x} \in \text{VARS}(X) \)

Pattern matching

Pattern variables

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

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

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

Should be easy to express in Psi

Let's take \( \mathcal{T} = \mathcal{N}^* \)

Substitution needs to be a \textbf{total} function

\[ (a, b, c)[a := (c, d)] = ((c, d), b, c) \notin \mathcal{N}^* \]
Solution

Allow \((c, d), b, c\)  \[ T = T^* \cup \mathcal{N} \]

Set to error  
\((a, b, c)[a := (c, d)] = \text{error}\)  \[ T = \mathcal{N}^* \cup \{\text{error}\} \]

Allow substitution to be a **partial** function

Better yet! Type to disallow ‘bad’ substitutions from arising.
Sorts

Goal: flexible definition of “well-formed”

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

name, term, and pattern sorting

**is well-sorted iff**

- substitution: \([\tilde{a} := \tilde{N}]\)
  \[\text{SORT}(a_i) \preceq \text{SORT}(N_i)\]
- restriction: \((\nu a)P\)
  \[\text{SORT}(a) \in \mathcal{S}_\nu\]
- output: \(\overline{M} \; N.\; P\)
  \[\text{SORT}(M) \preceq \text{SORT}(N)\]
- input: \(M(\lambda \tilde{x})\; X.\; P\)
  \[\text{SORT}(M) \preceq \text{SORT}(X)\]

a.k.a. Types
Polyadic Pi-calculus

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

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

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

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

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

Formal correspondence of transitions and equivalence
Results

- More **expressive** framework
- Captures many previous process calculi
- Better precision for defining terms
- Well-sortedness is preserved by transitions
- Previous results for psi still hold
- Implemented in Pwb
The Psi-Calculi Workbench: a Generic Tool for Applied Process Calculi

1. INTRODUCTION

Psi-calculi [Bengtson et al. 2011] is a parametric semantic framework based on the calculus of constructions. We use the extension of the framework to broadcast communication, to ensure send/receive duality, and point to point communication. Developing the proof have been machine-checked using Nominal Isabelle. The overall intuition by means of an example.

Personal Contributions

The Psi-Calculi Workbench: a Generic Tool for Applied Process Calculi

Submitted to Special Issue on Application of Concurrency to System Design

Johannes Borgström, Ramūnas Gudkova, Ioana Rodhe and Björn Victor, Uppsala University

The Psi-Calculi Workbench: a Generic Tool for Applied Process Calculi

99999

A Sorted Semantic Framework for Applied Process Calculi

Johannes Borgström, Ramūnas Gudkova, Joachim Parnow, Björn Victor, and Johannes Anis Podelia

Session Types for Broadcasting

Dimitrios Kouzapas
University of Glasgow
dimitrios.kouzapas@glasgow.ac.uk

Ramūnas Gudkova
Uppsala University
ramunas.gudkova@it.uu.se

Simon J. Gay
University of Glasgow
simon.gay@glasgow.ac.uk

Session Types for Broadcasting

Up to now session types have been used under the assumptions of point to point communication, to ensure send/receive duality. In this paper we define a session type theory for broadcast communication semantics that by definition do not assume point to point and reliable communications. Our session framework has been used by

A Sorted Semantic Framework for Applied Process Calculi

A SORTED SEMANTIC FRAMEWORK FOR APPLIED PROCESS CALCULI

JOHANNES BORGSTROM, RAMUNAS GUDKOVA, JOACHIM PARNOW, BJORN VICTOR, AND JOHANNES ANIS PODIOLA

Abstract. Applied process calculi include advanced programming constructs such as type systems, communication with pattern matching, encryption primitives, concurrent constraints, nondeterminism, process creation, and dynamic connection topologies. Several such formalisms, e.g., the applied psi-calculi, are extensions of the type psi-calculi; a growing number is geared towards particular applications or computational paradigms. Our aim is a framework where many such calculi fit and where such results are derived once and for all, eliminating the need for individual proofs about each calculus.

Our effort in this direction is the framework of psi-calculi [BPV11], which provides machine-checked proofs that important meta-theoretical properties, such as compositionality of bisimulation, hold in all instances of the framework. We claim that the theoretical development is more robust than that of other calculi of comparable complexity, since we use a structural operational semantics given by a single inductive definition, and since we have checked most results in the theorem prover Nominal Isabelle [T16b].
Conclusion

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

Made it possible to model more complicated systems

Straightforward protocol specification with safety guarantees

Made psi-calculi more expressive
Future Work

- Algebras of Psi-calculi
- Nominal transition system specification
- Modal logics for Psi
- Models of Psi-calculi
- More case-studies
- Build more complex models out of simpler
- Factory of tool factory
- Fine grained reasoning: safety & liveness
- Efficient representations
- More WSNs
Thank you for listening