LANGUAGES, LOGICS, TYPES AND TOOLS FOR CONCURRENT SYSTEM MODELLING

Ramūnas Gutkovas
ramunas@fct.unl.pt

2016-12-14 NOVA LINCS
A LITTLE ABOUT MYSELF

2007 BSc

2007-2009 Startup

2011 MSc

Uppsala, Sweden

2016 PhD

Kaunas, Lithuania

2007 BSc

Startup 2007-2009
BUGS

OR, WHEN MACHINES GO WRONG
An error has occurred. To continue:

Press Enter to return to Windows, or

Press CTRL+ALT+DEL to restart your computer. If you do this, you will lose any unsaved information in all open applications.

Error: 0E : 016F : BFF9B3D4

Press any key to continue
Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you. (0% complete)

If you'd like to know more, you can search online later for this error: HAL_INITIALIZATION FAILED
EXPENSIVE BUGS

Pentium FDIV bug, 1994
$475 million worth of recalls

Ariane 5 went “poof”, 1996
Integer Overflow
$ 500 million loss

goo.gl/bU36B2
KILLER BUGS

Therac-25, 1980s

Due to a race condition produced a lethal radiation burst
5 killed

Toyota, 2010

Unintended acceleration
Software bug
89 killed
SECURITY

Heartbleed
OpenSSL
A bug that allows to obtain keys
Most of the internet affected
SSL is foundation for ecommerce.

Microsoft calls them 1 million dollar bugs!
Many more:
goo.gl/GVQIIIC

CVE-2016-5195
Dirty Cow
Data race in the linux kernel
since 2007 allows to escalate
privileges
Millions of Android devices vulnerable
UNPRECEDENTED IMPLICATIONS

Celebgate

Celebrity Apple’s iCloud accounts hacked

Influence foreign government elections

Voldemord
STANDARD APPROACH: TESTING

Internal behavior of the code is unknown
Testing can’t be complete

Testing is **essential**, however, it is not **sufficient**!

Suppose `int` is 32 bits

```c
int multiply (int x, int y)
```

Thus, there are $2^{64}$ inputs

Intel Core i7 5960X (8 core)

... can do about $2^{38}$ instructions per second

It would take $2^{26}$ sec ~ 18641 hours ~ **2 years** to test
...program testing can be used to show the presence of bugs, but never to show their absence!

E. W. Dijkstra [EWD303]
CONCURRENT SYSTEMS
Concurrent = Two Queues One Coffee Machine

Parallel = Two Queues Two Coffee Machines

© Joe AmSTRY 2013
THE CHALLENGE

find

An adequate language for describing concurrent systems
A mathematical theory for capturing dynamics, i.e. semantics
Well-founded Verification Technique
CALCULUS OF COMMUNICATING SYSTEMS

\[ a \in \mathcal{A} \quad \text{action} \]

\[ P, Q ::= a.P \quad \text{input} \]
\[ \overline{a}.P \quad \text{output} \]
\[ 0 \quad \text{inaction} \]
\[ \tau.P \quad \text{silent} \]
\[ P | Q \quad \text{parallel} \]
\[ P + Q \quad \text{sum/choice} \]
\[ !P \quad \text{replication} \]

Ex.
\[ a.P | \overline{a}.Q \]

(can be extend with value passing)
OBSERVABLE BEHAVIOUR

- Pay
- Coffee
- Tea
- Pay
- Coffee
- Tea
OBSERVABLE BEHAVIOUR

pay.(coffee.0 + tea.0)

pay

coffee.0 + tea.0

coffee
tea
0
0

pay.coffee.0 + pay.tea.0

pay

coffee.0

coffee
0

tea.0

tea
0
0
There is nothing canonical about the choice of the basic combinators, even though they were chosen with great attention to economy. What characterises our calculus is not the exact choice of combinators, but rather the choice of interpretation and of mathematical framework.

R. Milner

[Milner 1980]
Equivalence based on the observable behaviour

**Bisimilarity**

\[ P \sim Q \]

at each state \( P \) can perform all the actions of \( Q \), and vice versa, and states continue to be bisimilar

**Alg. properties**

\[ P|Q \sim Q|P \]
\[ P|(Q|R) \sim (P|Q)|R \]
\[ 0 + P \sim P \]
\[ P + Q \sim Q + P \]
\[ 0|P \sim P \]
\[ 0 + P \sim P \]
\[ 0|P \sim P \]

**Weak bisimilarity**

\[ P \approx \tau.P \]

roughly, ignoring silent actions
COMPOSITIONALITY
(Frege’s principle)

Systems built from smaller systems

Component Modularity
Under all contexts a process’s behaviour is indistinguishable (i.e., bisimilar)

A congruence relation
Equivalence (bisimulation) preserved under all operations

Ex.

if $P \sim Q$
then $R|P \sim R|Q$
VERIFICATION TECHNIQUE

Specification ≈ Implementation
(weakly) bisimilar

Specification = pay.(coffee.0 + tea.0)

Implementation =
    pay.(νinternal)(internal(amount).
    if amount = 50
        then coffee.0 + tea.0
    else coffee.0 + tea.0
    | P)
PI-CALCULUS

[Milner et al. 1991]

CCS

+mobility

spi-calculus

+security

polyadic synch. pi-calculus

+algebraic

polyadic pi-calculus

concurrent constraint calculus

applied pi-calculus

... and myriad of other ‘small’ extensions of pi
PI-CALCULUS

... and myriad of other ‘small’ extensions of pi

[Milner et al. 1991]
Appendix

In this Appendix, we will include the proofs of some of the results stated in the text main of the proofs are by case analysis, and we will give the argument for a few cases or typical cases. Full proofs may be found in [1].

Proof of Lemma 1: The proof by induction on depth of inference. We consider in turn each transition rule as the last rule applied in the inference of the assumption \( P \rightarrow Q \). We give two cases.

(a) \( \text{Tautology} \): \( \alpha \rightarrow \neg \alpha \): Let \( P \vdash Q \), with \( \alpha \in \psi \) (and \( \alpha \neq Q \)). Then \( \neg Q \rightarrow P \) (false), so the conclusion (false) is in \( \psi \). Then \( \neg Q \rightarrow P \) is derived in \( \psi \). This follows from the proof of Lemma 1, which is beyond the scope of this paper.

(b) \( \text{Modus Ponens} \): Let \( P \vdash Q \), with \( P \rightarrow Q \) and \( P \in \psi \). Then \( Q \rightarrow Q \) (true) in \( \psi \). Then \( (P \rightarrow Q) \rightarrow Q \) in \( \psi \). This follows from the proof of Lemma 1, which is beyond the scope of this paper.

**Proof of Theorem 2**: The proof by induction on depth of inference. We consider in turn each transition rule as the last rule applied in the inference of the assumption \( P \rightarrow Q \). We give two cases.

(a) \( \text{Tautology} \): \( \alpha \rightarrow \neg \alpha \): Let \( P \vdash Q \), with \( \alpha \in \psi \) (and \( \alpha \neq Q \)). Then \( \neg Q \rightarrow P \) (false), so the conclusion (false) is in \( \psi \). Then \( \neg Q \rightarrow P \) is derived in \( \psi \). This follows from the proof of Lemma 1, which is beyond the scope of this paper.

(b) \( \text{Modus Ponens} \): Let \( P \vdash Q \), with \( P \rightarrow Q \) and \( P \in \psi \). Then \( Q \rightarrow Q \) (true) in \( \psi \). Then \( (P \rightarrow Q) \rightarrow Q \) in \( \psi \). This follows from the proof of Lemma 1, which is beyond the scope of this paper.

10 pages of proof appendix + 30 main text and proofs
With cheats!

In this Appendix we outline the proofs of some of the results stated in the text. Most of the proofs are by case analysis, and we give the argument for a few crucial or typical cases. Full proofs may be found in [3].

PSI-CALCULUS FRAMEWORK

Data structures

Logics

Logical environment

Psi Framework

Syntax

Pi-calculus like semantics

Bisimulation

Congruence

Weak bisimulation

Weak congruence

Bengtson and Pohjola

[Bengtson et al. 2011]
A significant modelling flexibility of psi-calculi comes from the fact that $\Psi$ can be seen as a two-valued logic. In the above example, we interpreted the assertions as a set of names that are known to be equivalent, and the channel equivalence condition as an equality query. We could as well take the assertions to be sets of equations on terms, and the conditions to be also equations, then the $\vdash$ can be defined to be a proof derivation of this equational logic. We can take this even further, we could define assertions to be sets of predicate formulas (including the universal and existential quantifiers), and likewise conditions to be formulas, then $\vdash$ could be defined as a validity relation of the predicate logic or proof derivation relation. Thus, in psi-calculi it is quite straightforward to reuse already developed theories, e.g., of data structures, cryptographic primitives, etc.

Psi-calculi has also been extended to encompass more process calculi: the higher order communication [25], and unreliable broadcast communication [7]. The full syntax of psi-calculi is given in Figure 2.3, where we use $\exists x$ and $\forall N$ to denote arbitrary sequences $x_1, \ldots, x_n$ and $N_1, \ldots, N_n$. Psi-calculi are a major part of this thesis. The Psi-calculi framework is the main subject of Paper II and Paper IV. The logic of Paper I arose from considering the generalised actions with multiple binders and the behavioural
(ν secret)(
  (hash(⟨secret, message⟩) = x) |
  a⟨message, x⟩ |
  a(y).

  case hash(⟨secret, fst(y)⟩) = snd(y) : b YES
  [hash(⟨secret, fst(y)⟩) ≠ snd(y) : b NO

  Verify

generate a key
sign a message
send MAC
receive MAC
Languages, Logics, Types and Tools for Concurrent System Modelling

RAMŪNAS GUTKOVAS
Expanding generality of Psi-calculi with a type-system

Providing a verification calculus for psi-calculus, and others

Tool support for psi-calculi
SORT SYSTEM FOR PSI
A direct encoding of a process calculus to a Psi-calculus

No elaborate encodings
No superfluous data terms
No superfluous behaviour

Many calculi were not representable

Unsorted polyadic pi-calculus    Sorted polyadic pi-calculus
LINDA pattern matching    Polyadic synchronisation pi-calculus
Value-passing CCS

Goal: extend psi-calculi to be capable of representing new calculi!
SYMmetric CRYPTO

Computation

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

makes sense when it is typed

\begin{align*}
(\nu a, k)(\overline{a} \text{ “fooban”}. \text{0} & | \quad \mu (\lambda y) y . \overline{c} \text{ dec}(y, k). \text{0}) \\
\tau & \rightarrow \quad (\nu a, k)(\text{0} & | \quad \overline{c} \text{ dec}(\text{“fooban”}, k). \text{0})
\end{align*}

\begin{align*}
(\nu a, k)(\overline{a} \text{ enc}(M, k). \text{0} & | \quad \mu (\lambda y) y . \overline{c} \text{ dec}(y, k). \text{0}) \\
\tau & \rightarrow \quad (\nu a, k)(\text{0} & | \quad \overline{c}M)
\end{align*}
SORT SYSTEM

Set of sorts $S$

Sort assigning to params function $\text{Sort}(X) \in S$

Sorting relations for substitution and processes:

- can send
- can receive
- can restrict
- can substituted

Consider only well sorted substitutions

Sanity check: A well-sorted substitution preserves well-sortedness of a process.
RESULTS

All the standard algebraic laws of bisimulation are preserved:

- Weak bisimulation
- Weak congruence
- Bisimulation
- Congruence

All the mentioned calculi are **directly representable**:

- Unsorted polyadic pi-calculus
- LINDA pattern matching
- Sorted polyadic pi-calculus
- Polyadic synchronisation pi-calculus
- Value-passing CCS
MODAL LOGICS FOR PSI
MODAL LOGICS

Find grained properties of a system

- Deadlock freedom
- Eventually coffee machine produces coffee
- A malicious message is eventually rejected

Process $P$ is a model of modal logical formula $\varphi$

$P \models \varphi$

Formula $\varphi$ is true for $P$
MODAL LOGICS

Concurrent System Models
CCS
Value-Passing CCS
Spi-calculus
Applied pi-calculus
Fusion calculus
Multi-labelled Nominal transition systems

Psi-calculi framework
Concurrent constraint calc.
Possibly others

Logics
Hennessy, Milner 1985
Hennessy, Liu 1995
Fredrup et al. 2002
Hüttel, Pedersen et al. 2007
Haugstad et al. 2006
De Nicola, Loreti 2008
???
???
Formulas depend on finite number of names

\[ P \models \varphi \iff P \models \varphi \]

\[ P \models \neg A \iff \text{not } P \models A \]

\[ P \models \bigwedge_{i \in I} A_i \iff (\forall i \in I) \ P \models A_i \]

\[ P \models \langle \alpha \rangle A \iff (\exists P') \ P \xrightarrow{\alpha} P', \ P' \models A \]

**Thm.** Adequate for strong bisimilarity.

**What’s new:** finitely supported formulas
EXPRESSIVENESS

Next step
for any action there is a state

Quantifiers
for every value of a domain

Fresh/New
for a state where a name does not appear

Recursion in Logic
\[ \text{rec} X. A \]

Ex.
Eventually get \textbf{coffee} :=
\[ \text{rec } X. \langle \text{coffee} \rangle \text{true } \lor \text{ next step, recurse on } X \]
RESULTS

Adequate Modal Logic for many transition systems

The main proofs are machine checked

Adequate for many variants of bisimilarity:
hyper, open, early, late, weak

Provide an adequate modal logic for
psi-calculi, concurrent constraint calculus,
and others
AUTOMATED TOOLS

Small specification:
WSN secure aggr.
Small spec. in Pwb
20 LOC
only 3 nodes

Property
There is no tempered data that the network accepts

Results in
PSI-CALCULI WORKBENCH

Tool factory: define your own tool!

Based on the parametric psi-calculi framework
PARAMETRIC

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

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

Logical Assertions
- e.g., Knows a secret, Connectivity, …
FEATURES

Communication Primitives

Unicast

Unreliable Broadcast

[ Borgström et al. 2011 ]

Execution of Processes

(Weak) Bisimulation Checking

Pluggable Architecture
EXAMPLE: WSN AGGREGATION

Spatially distr. nodes

Wireless communication

Protocol:

- Establish routing tree
- Forward data
WORKBENCH MODEL

Sink(nodeId, bsChan) <=
  '"init(nodeId)"!<bsChan> .
  ! "data(bsChan)"(x) ;

Node(nodeId, nodeChan, datum) <=
  '"init(nodeId)"?(pChan) .
  '"init(nodeId)"!<nodeChan> .
  '"data(pChan)"<datum> .
  NodeForwardData<nodeChan, pChan> ;

NodeForwardData(nodeChan, pChan) <=
  ! "data(nodeChan)"(x). '"data(pChan)"<x> ;
SYMBOLIC EXECUTION

---|gna!(new bsChan)bsChan-->

**Source:**
System3<d1, d2>

**Constraint:**
- (new chan1, chan2, chanS){"init(0)<gna"} ∧
- (new chanS, chan2, chan1){"gna>init(1)"} ∧
- (new chanS, chan1, chan2){"gna>init(2)"}

**Solution:**
- \([\text{gna := "init(0)"}], 1\)

**Derivative:**
- (!("data(chanS)"(x))) |
  - (((new chan1)(
    "init(1)!<chan1>.
    "data(chanS)"<d1>.
    NodeForwardData<chan1, chanS>
  )) |
  - ((new chan2)(
    "init(2)!<chan2>.
    "data(chanS)"<d2>.
    NodeForwardData<chan2, chanS>
  )))

---

**generated action**

---

**system with 3 nodes**

---

**constraints**

---

**solution**

---

**Execution:**
derived process
ARCHITECTURE

Pwb

Command Interpreter

Symbolic Equivalence gen.

Symbolic Execution

Psi Calculi Core

Supporting library
ARCHITECTURE

Parameters

Pretty Printer  Parser

Plug in external solvers, e.g. SMT solvers
Z3, CVC4, Yices2

Data  Logics  Assertions

Supporting library

Pwb

Command Interpreter

Symbolic Equivalence gen.
Symbolic Execution
Psi Calculi Core
CONCLUSION
A widened applicability of psi-calculi via a type system

A general and powerful modal logic that is applicable to systems such as psi-calculi

Tool support for psi
QUESTIONS