Process Calculi for WSNs and more

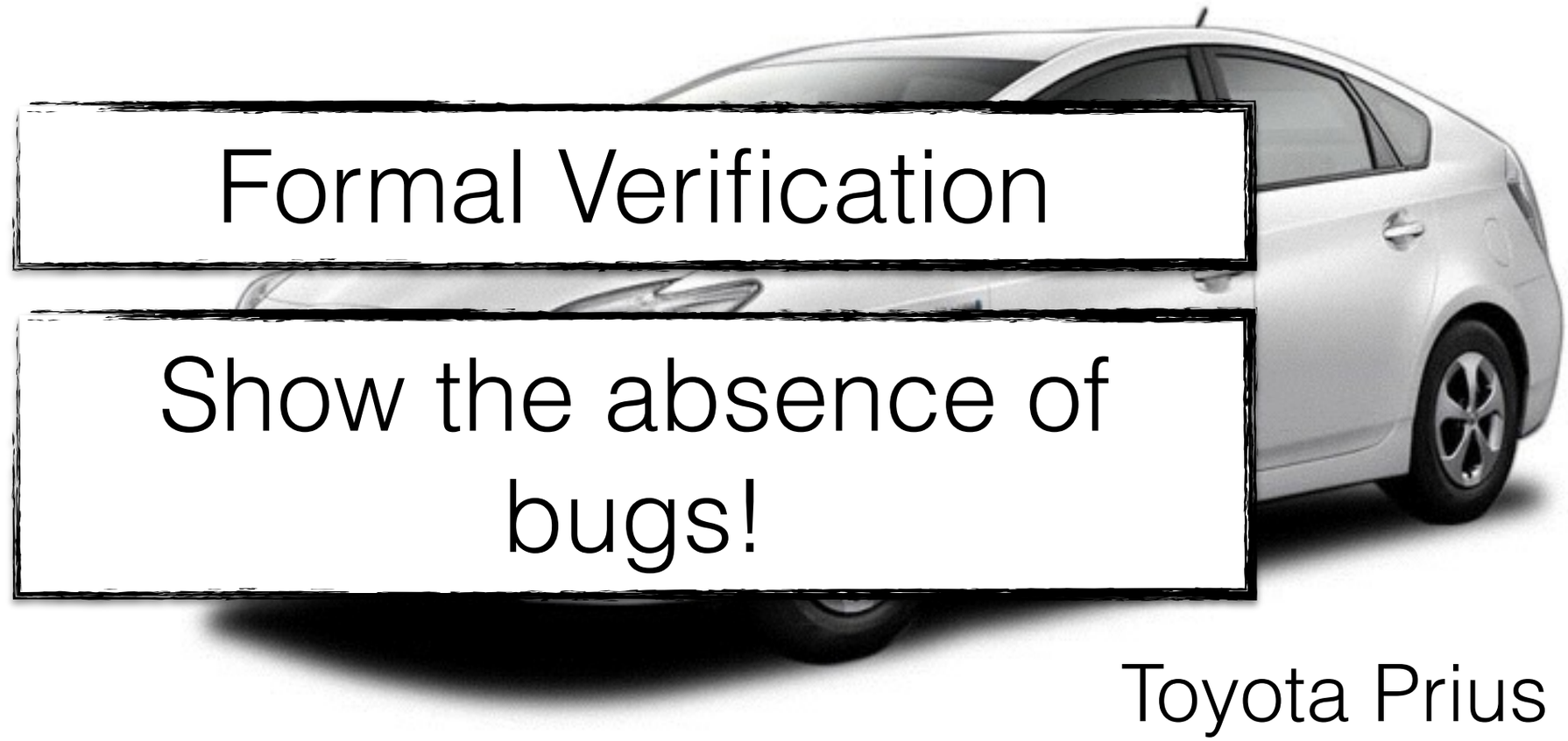
Types and tools

Ramūnas Gutkovas

ProFUN meeting
2014 October 22
Uppsala University

In 2010, Toyota **recalled** 400,000 vehicles

to correct a **software** **“glitch”** in ABS



Toyota Prius

Testing shows the presence, not the absence of bugs!

- E. W. Dijkstra

Psi

- 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

$$\overline{M}\ N.P \qquad \underline{M}(\lambda\widetilde{x})N.P$$

object

pattern

Parameters:

M, N: **T** (terms)

φ : **C** (conditions)

Ψ : **A** (assertions)

assertion $(\!|\Psi|\!)$ “facts”

condition

$$\mathbf{case}\ \varphi_1 : P_1 \,[]\, \cdots \,[]\, \varphi_n : P_n$$

like guarded commands, **if-then-else**

the usual: $\mathbf{0}$ $(\nu a)P$ $P \mid Q$ $!\,P$

Cook a psi-calculus

Define terms **T** (e.g. data terms, channels) M, N

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

assertions **A** (statements about e.g. terms) Ψ

can be practically anything

Cook a psi-calculus

Define terms **T**, conditions **C,** assertions **A** M, N φ Ψ

Define substitution on these (satisfy axioms) $[\widetilde{a} := \widetilde{M}]$

Define operators:

$\leftrightarrow : \mathbf{T} \times \mathbf{T} \to \mathbf{C}$	Channel equivalence
$\otimes : \mathbf{A} \times \mathbf{A} \to \mathbf{A}$	Composition
$\mathbf{1} : \mathbf{A}$	Unit assertion
$\vdash \subseteq \mathbf{A} \times \mathbf{C}$	Entailment

(practically anything)

$\prec : \mathbf{T} \times \mathbf{T} \to \mathbf{C}$	Broadcast Output Connectivity
$\succ : \mathbf{T} \times \mathbf{T} \to \mathbf{C}$	Broadcast Input Connectivity

Example

$M \in \mathbf{T}$

$\varphi \in \mathbf{C}$

$\Psi \in \mathbf{A}$

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

$$\overline{\mathsf{init}(1)}\,123.\mathbf{0} \mid$$
$$\underline{\mathsf{init}(2)}\,(\lambda x)x.\mathbf{0} \mid$$
$$\underline{\mathsf{init}(3)}\,(\lambda y)y.\mathbf{case}\ y = 3 : P \mid$$
$$(\!|1 \prec 2, 1 \prec 3|\!)$$

Example

$M \in \mathbf{T}$

$\varphi \in \mathbf{C}$

$\Psi \in \mathbf{A}$

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

Transition relation ~ semantics

α action/behaviour

$$\mathsf{init}(1)123.\mathbf{0} \mid \mathsf{init}(2)(\lambda x)x.\mathbf{0} \mid \mathsf{init}(3)(\lambda y)y.\mathbf{case}\ y = 3 : P \mid (\!| 1 \prec 2, 1 \prec 3 |\!) \xrightarrow{\alpha} \mathbf{0} \mid \mathbf{0} \mid \mathbf{case}\ 123 = 3 : P[y := 123] \mid (\!| 1 \prec 2, 1 \prec 3 |\!)$$

Example

$M \in \mathbf{T}$

$\varphi \in \mathbf{C}$

$\Psi \in \mathbf{A}$

$$M ::= \mathsf{init}(M) \mid a \mid i \in \mathbb{N}$$

$$\varphi ::= M = M' \mid M \prec M'$$

$$\Psi ::= M \prec M', \Psi \mid \epsilon$$

User defined logic

$$\frac{\Psi, M \prec M' \vdash \mathsf{init}(M) \prec \mathsf{init}(M')}{\Psi \vdash M = M' \text{ if } M = M'}$$

$$\mathsf{init}(1)\,123.\mathbf{0} \mid \mathsf{init}(2)\,(\lambda x)x.\mathbf{0} \mid \mathsf{init}(3)\,(\lambda y)y.\mathbf{case}\ y = 3 : P \mid (\!|1 \prec 2, 1 \prec 3|\!)$$

α

$$\mathbf{0} \mid \mathbf{0} \mid \mathbf{case}\ 123 = 3 : P[y := 123] \mid (\!|1 \prec 2, 1 \prec 3|\!)$$

Recent Advancements to Psi

Crypto Example

Term for encryption

$\mathsf{enc}(M, K)$

text

key

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

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

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

We need a way to control what are pattern variables

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

Knowledge of the key

Computation

All names of $\tilde{L}$

must be in $M[\tilde{x} := \tilde{L}]$ if $\tilde{x} \subseteq \mathrm{n}(M)$

Useful computation to have as part of substitution

$$\mathsf{dec}(\mathsf{enc}(M, K), K) \to M$$

However, the substitutions are **not** allowed to **lose names**

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

k does not appear in the result

in [TGC'13]

Generalised Pattern Matching

User defined pattern matchin.
Relaxes requirement on the substitution.

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

$\mathbf{X}$ *patterns, ranged over by* X, Y

$$\text{MATCH} : \mathbf{T} \times \mathcal{N}^* \times \mathbf{X} \to \mathcal{P}(\mathbf{T}^*) \quad \textit{Pattern matching}$$

$$\text{VARS} : \mathbf{X} \to \mathcal{P}(\mathcal{P}(\mathcal{N})) \quad \textit{Pattern variables}$$

Signifies which names are patterns

Ex:

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

$\underline{M}(\lambda m)\mathsf{enc}(m, k).P$

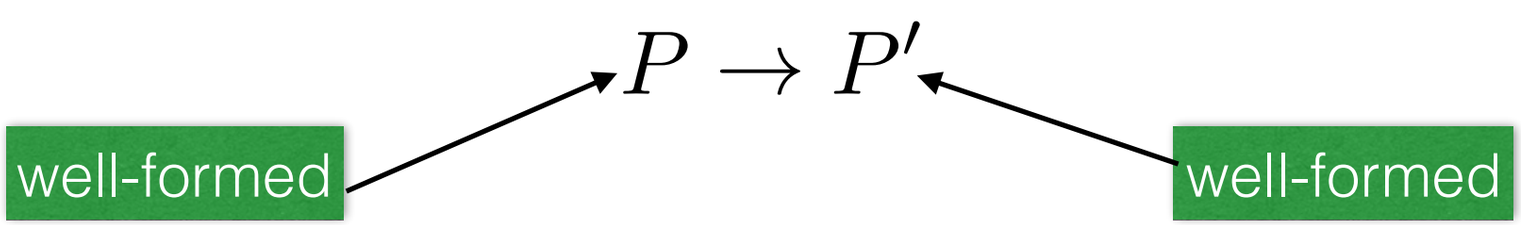
$\underline{M}(\lambda m, k)\mathsf{enc}(m, k).P$

Results

did not break psi

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

- well-formedness of processes is preserved by transitions



Polyadic communication

Polyadic pi-calculus

$$\begin{array}{rc} a(x_1,\ldots,x_n).P & \\ \mid \quad \overline{a}b_1,\ldots,b_n.Q & \longrightarrow \quad P\{b_1,\ldots,b_n/x_1,\ldots,x_n\} \mid Q \end{array}$$

Should be easy to express in Psi

Let's take $\mathbf{T} = \mathcal{N}^*$ ← sequences of names

Substitution needs to be a **total** function

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

Junk

in [TGC'14]

Sorts

a.k.a. Types

Goal: flexible definition of "well-formed"

$\textsc{sort} \quad : \quad \mathcal{N} \cup \mathbf{T} \cup \mathbf{X} \to \mathcal{S}$ name, term, and pattern sorting

is well-sorted iff

substitution	$[\widetilde{a} := \widetilde{N}]$	$\textsc{sort}(a_i) \prec \textsc{sort}(N_i)$
restriction	$(\nu a)P$	$\textsc{sort}(a) \in \mathcal{S}_\nu$
output	$\overline{M}\ N.P$	$\textsc{sort}(M)\ \overline{\propto}\ \textsc{sort}(N)$
input	$\underline{M}(\lambda\widetilde{x})X.P$	$\textsc{sort}(M)\ \underline{\propto}\ \textsc{sort}(X)$

Polyadic Pi-calculus

$$\textsc{sort}(a) = \texttt{chan}$$
$$\textsc{sort}(\tilde{a}) = \texttt{tup}$$
$$\overline{\propto} = \underline{\propto} = \{(\texttt{chan}, \texttt{tup})\}$$

a channel can send/ receive a tuple

$$\textsc{vars}(\langle \tilde{a} \rangle) = \{\tilde{a}\}$$

all names in input pattern must be bound

$$\underline{a}(\lambda \tilde{x})\langle \tilde{x} \rangle.P$$

$$\textsc{match}(\langle \tilde{a} \rangle, \tilde{x}, \langle \tilde{x} \rangle) = \{\tilde{a}\} \text{ if } |\tilde{a}| = |\tilde{x}|$$

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

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

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

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

Possible implementation

$$\mathrm{SrvImp}(c) = c(x).c(y).\mathbf{case}\ x = y : \overline{c}\,\mathrm{true}.\mathbf{0} \parallel x \neq y : \overline{c}\mathrm{false}.\mathbf{0}$$

Session Types

Specification of process that checks equality over a channel of type

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

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

Duals!

Possible implementation

$$\mathrm{SrvImp}(c) = c(x).c(y).\mathbf{case}\ x = y : \overline{c}\,\mathrm{true}.\mathbf{0} \parallel x \neq y : \overline{c}\mathrm{false}.\mathbf{0}$$

$$\mathrm{CltImp}(k) = \overline{k}1.\overline{k}2.k(b).0$$

Session Types

Specification of process that checks equality over a channel of type

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

Possible implementation

$$\mathrm{SrvImp}(c) = c(x).c(y).\mathbf{case}\ x = y : \overline{c}\,\mathrm{true}.\mathbf{0} \parallel x \neq y : \overline{c}\mathrm{false}.\mathbf{0}$$

$$\mathrm{CltImp}(k) = \overline{k}1.\overline{k}2.k(b).0$$

$$c^{+} : \mathsf{CheqEqSrv}$$
$$c^{-} : \mathsf{Clt} = \overline{\mathsf{CheqEqSrv}}$$

System

$$(\nu c)(\mathrm{SrvImp}(c^{+}) \mid \mathrm{CltImp}(c^{-}))$$

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

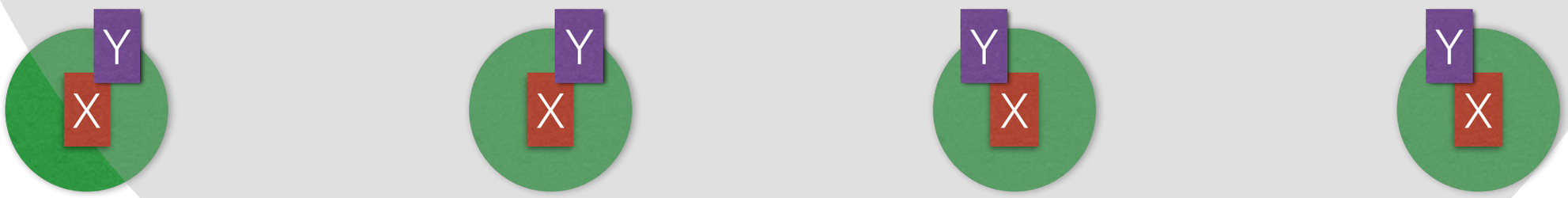
Scatter & Gather

Type

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

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

- Runtime tracking of session state
- Extended notion of duality



$c^-(x).\overline{c^-}y.Q_i$

Unreliability

Let process recover

$$(\nu c)\ P \bowtie R$$

$$(\nu c)(c^+(x).c^+(y).\mathbf{0} \mid \overline{\overline{c^-}2}.\mathbf{0})$$

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

ProVerif

Concurrency Workbench

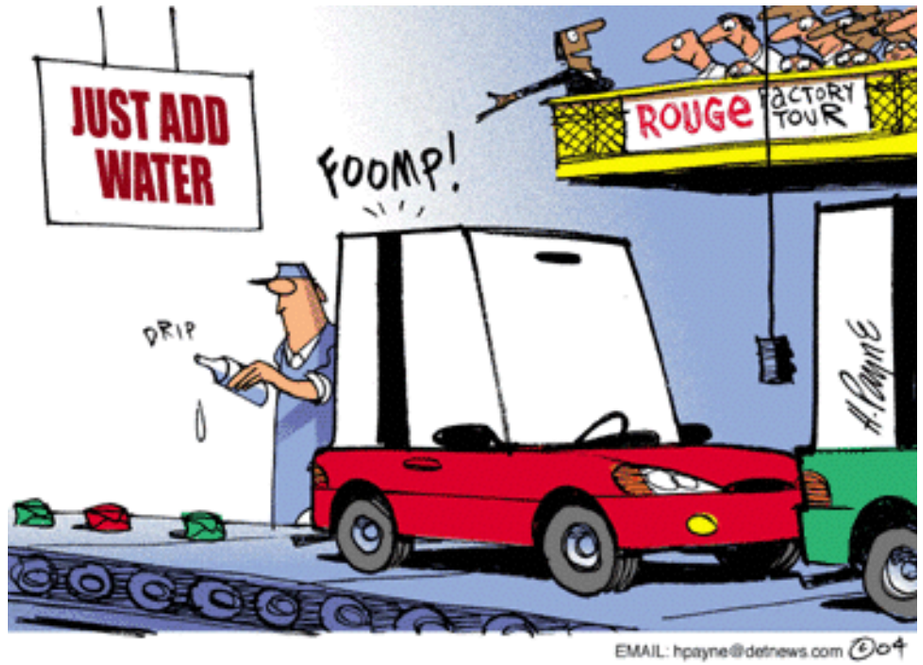
But specialised!

Mobility Workbench

Petruchio

PAT3

Psi-Calculi Workbench

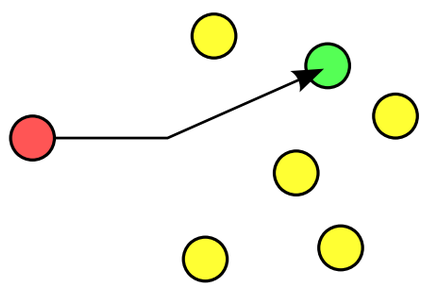


- Tool factory: define your own tool!
- Based on the parametric psi-calculi framework

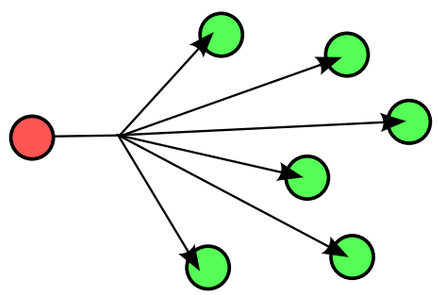
Features

Communication Primitives

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

$$\Psi \triangleright P \xrightarrow{\alpha} P'$$

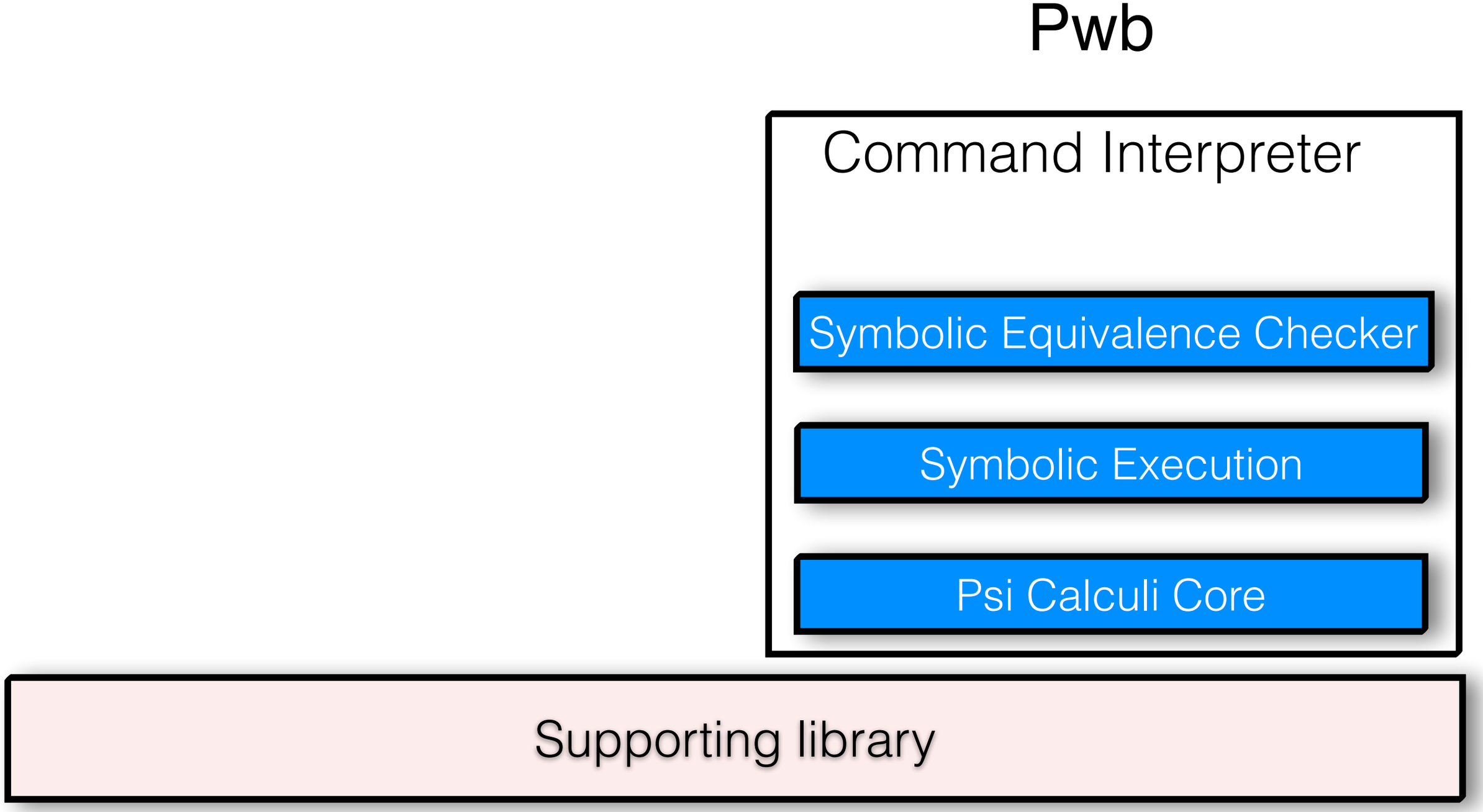
C

Symbolic Constraints

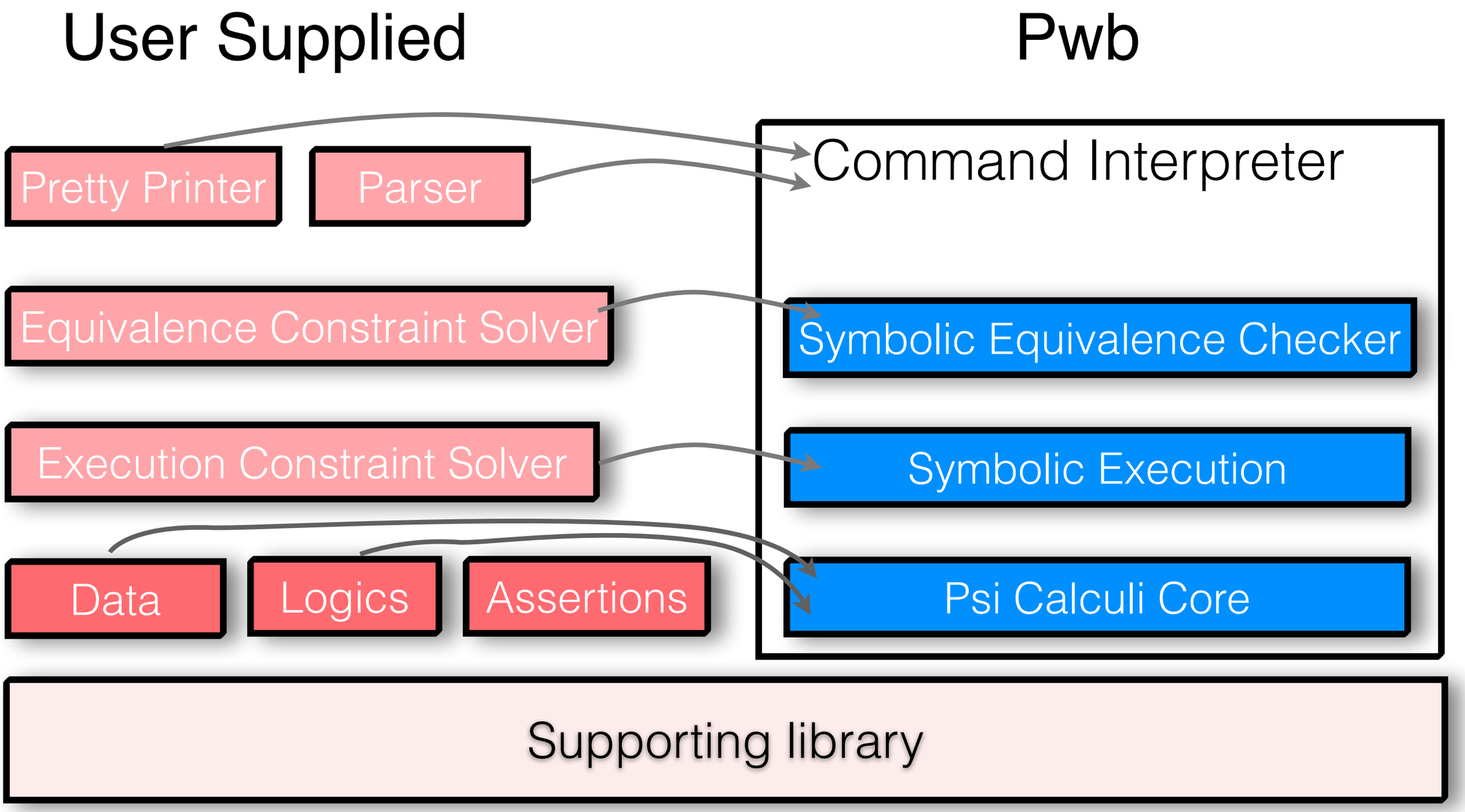
Symbolic Behavioral Equivalence Checking

$$P \sim Q$$

Parametric Architecture

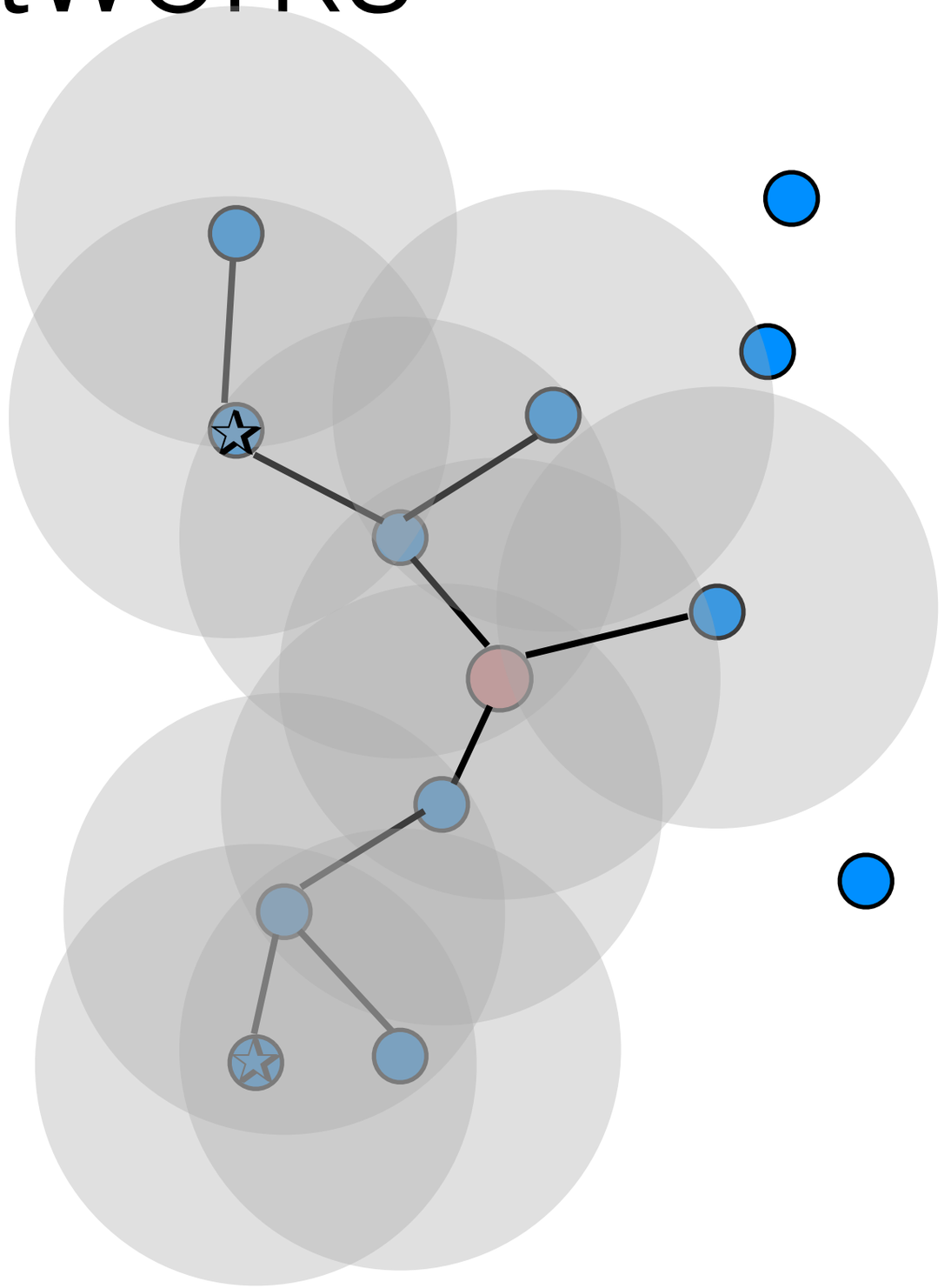


Parametric Architecture



Data Collection in Wireless Sensor Networks

1. Routing tree
2. Data collection



Specification in Pwb

Node Behavior

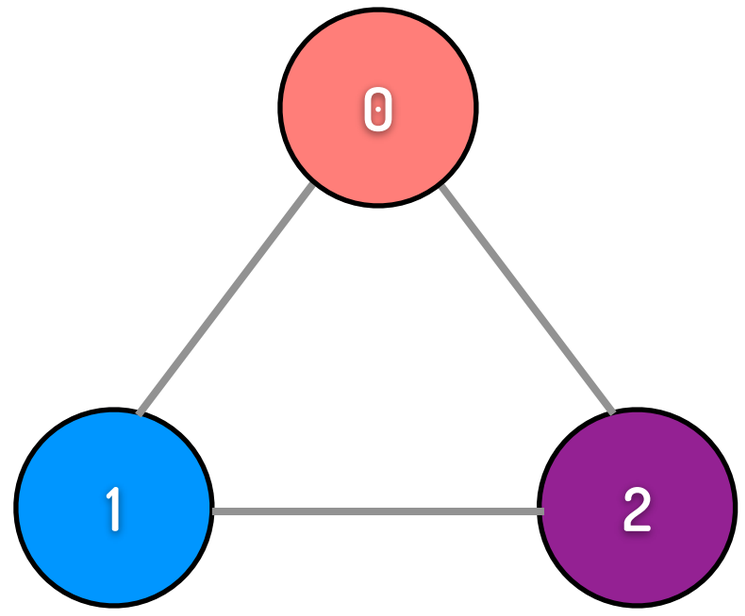
```
Sink(nodeId, sinkChan) <=
    '"init(nodeId)"! <sinkChan> .
    ! "data(sinkChan)"(x). ProcData<x> ;

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

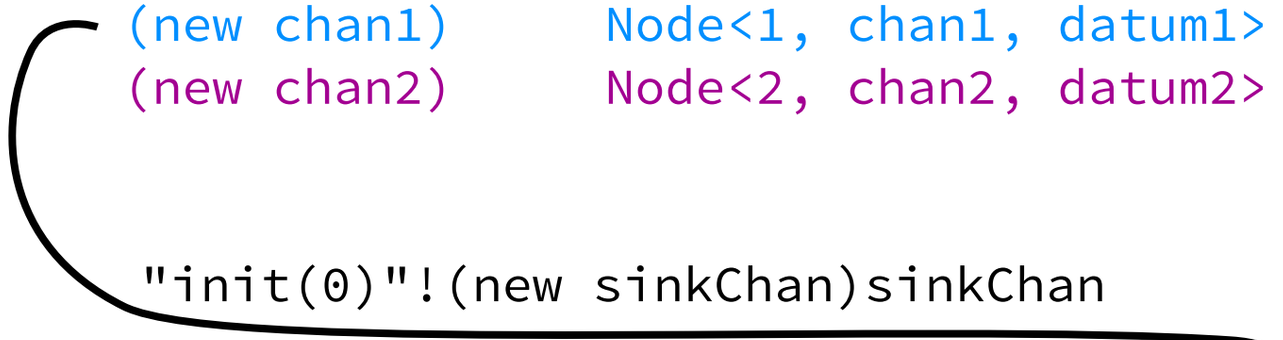


graph represented as edge list

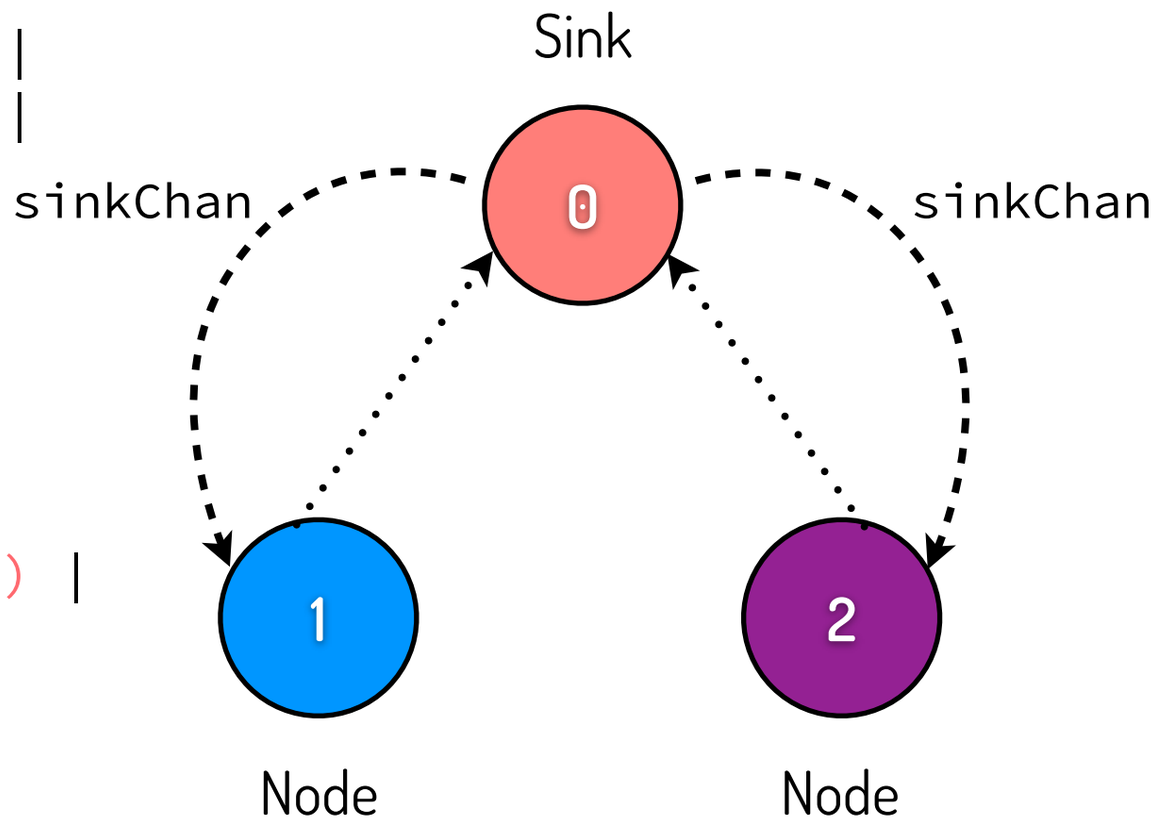
(0,1), (0,2), (1,2)

Example Transition

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




```
(!("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
- Infrastucture: better framework for constraint solvers, term algebras, verification
- Goal to check safety properties (deadlock freedoom) 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