UPPSALA
UNIVERSITET

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

Johannes Borgström **Ramūnas Gutkovas**
Iona Rodhe Björn Victor

Uppsala University

ProFuN meeting, Dec 18, 2013

Presented at ACSD'13 Barcelona, July

supported by **ProFuN** and UPMARC

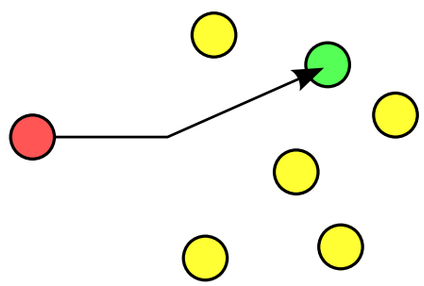
Motivation

- Application Specific Reasoning
- Parametric
- Dynamic Connectivity
- Mobility

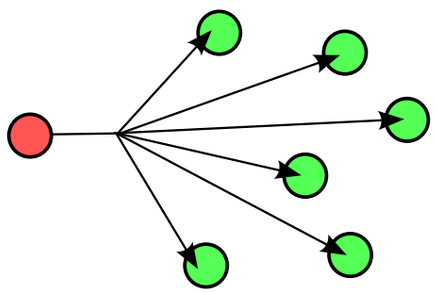
Psi-Calculi Workbench (Pwb) Features

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

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

Assertion

Constraints

Improved Symbolic Behavioral Equivalence and Preorder Checking

$$P \sim Q \qquad\qquad P \lesssim Q$$

Pwb implements Psi-Calculi

[LMCS'11]

- Psi-Calculi is a **parametric** process calculi **framework** on data and logic
- Designed with **applications** in mind (WSNs, Cache Coherence, Security Protocols, etc.)
- The soundness of the **meta-theory** has been **machine-checked** with **Isabelle** (algebraic laws, bisimulation theory, compositional semantics, etc.) inherited by all calculi [to appear in JAR]
- Expressive: captures many other calculi
- Extensions: Higher Order, Sorts, Types, Reliable Br.

Psi-Calculi Workbench via an Example

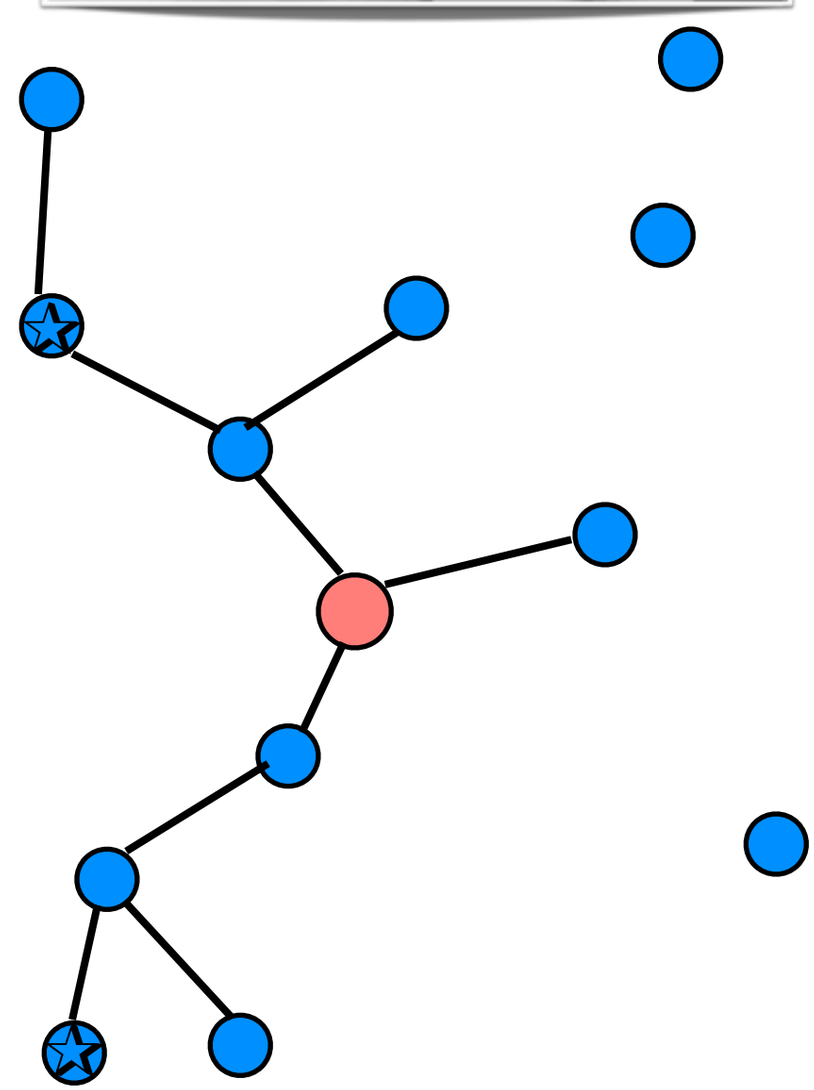
Data Collection in Wireless Sensor Network

Data Collection in Wireless Sensor Networks (TAG)

(Madden et. al. '02)

- Network consists of a set of **nodes** and one distinguished node **sink**
- Protocol has two phases:

1. **Establishment of a routing tree** (rooted at sink): nodes wirelessly broadcast a special initialization message.
2. **Data collection:** nodes send and forward data via established route using (reliable) unicast messages



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>  ;
```

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>  ;
```

1. Route Tree Establishment

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>  ;
```

1. Route Tree Establishment

2. Data Collection

Specification in Pwb

Node Behavior

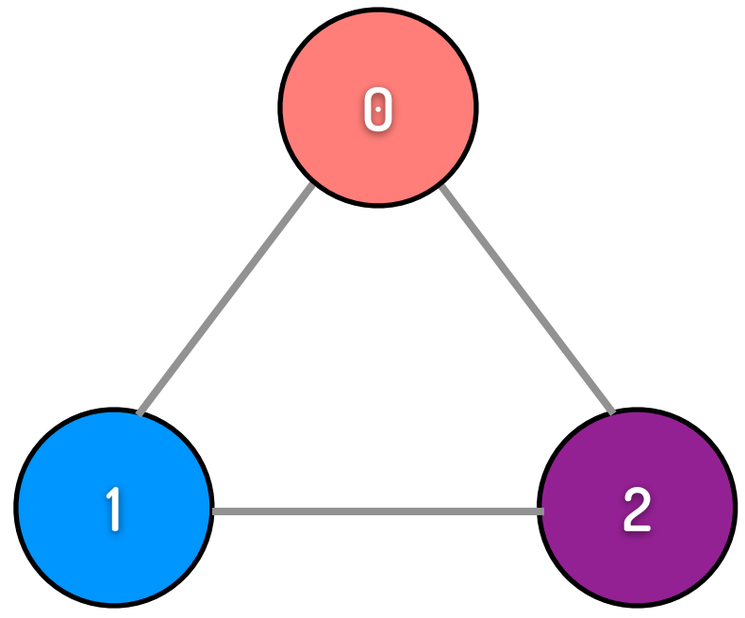
```
Sink(nodeId, sinkChan) <=
    '"init(nodeId)"! <sinkChan> .
    ! "data(sinkChan)"(x). ProcData<x> ;

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)

Pwb Features

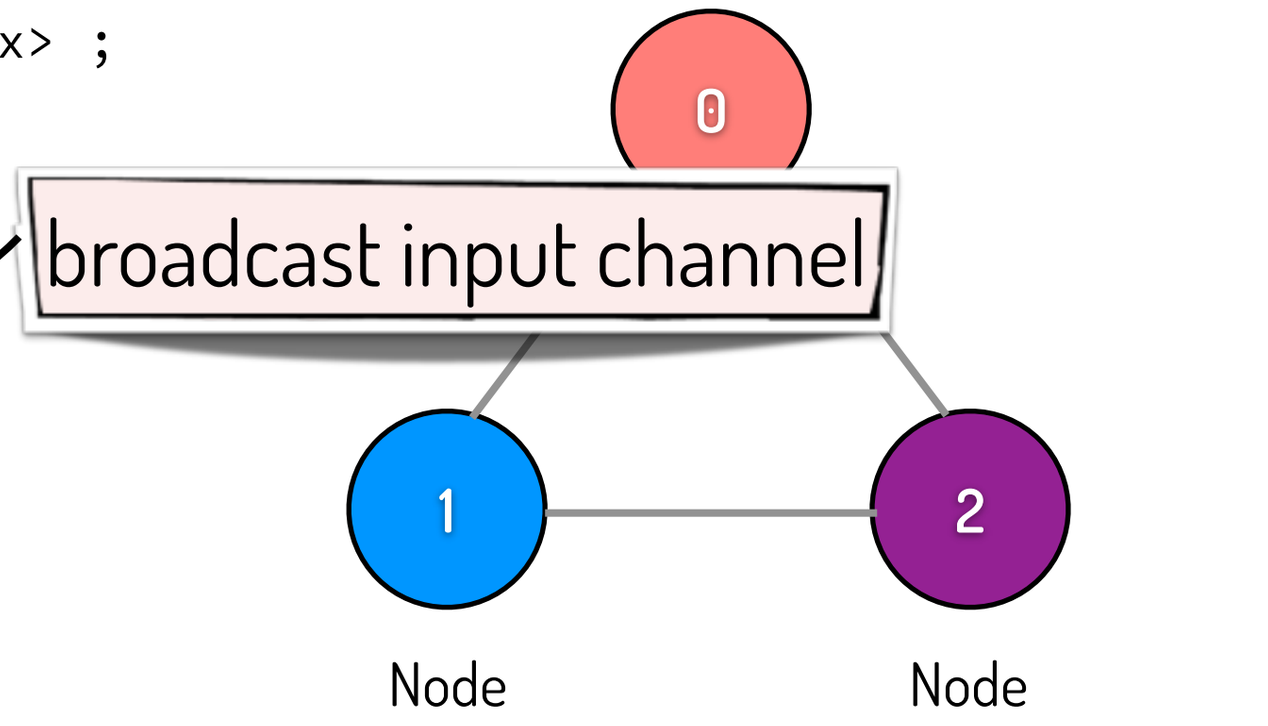
broadcast output channel

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>  ;
```

Node Connectivity for Broadcasting



System

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

graph represented as edge list

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

all channels are structured

Pwb Features

broadcast output channel

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>  ;
```

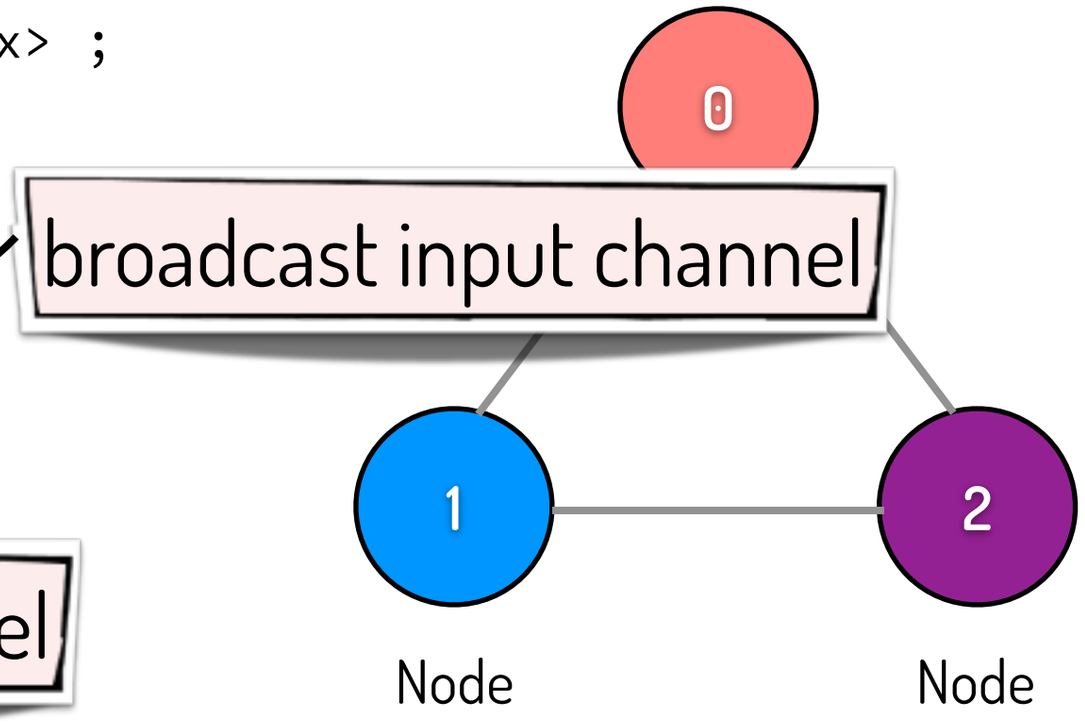
input channel

output channel

ys

```
(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)

Pwb Features

all channels are structured

broadcast output channel

process definitions

Env. assertions

Node Behavior

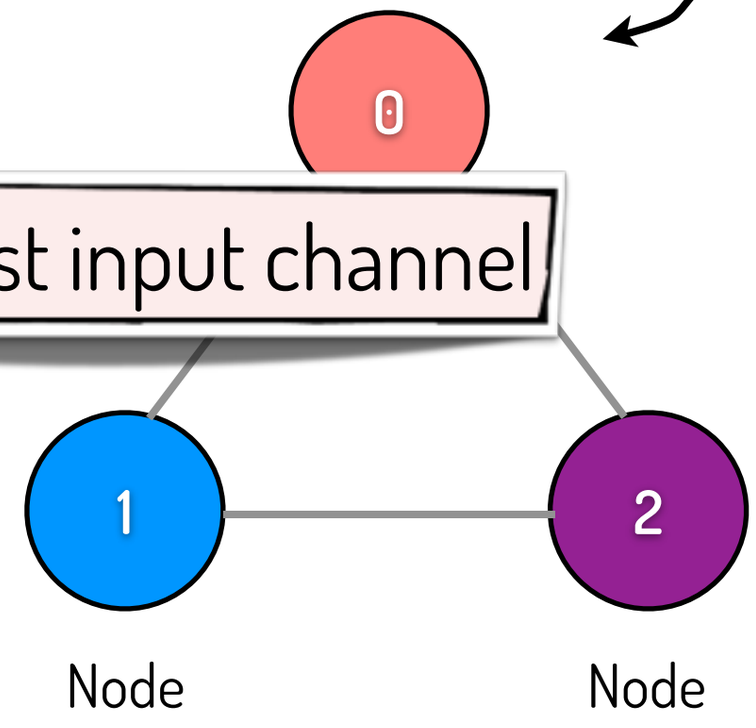
```
Sink(nodeId, sinkChan) <=
    '"init(nodeId)"!<sinkChan> .
    "data(sinkChan)"(x). ProcData<x> ;

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

input channel

output channel

Broadcast Connectivity for Sinks




```
(new sinkChan)
(new chan1)
(new chan2)
```

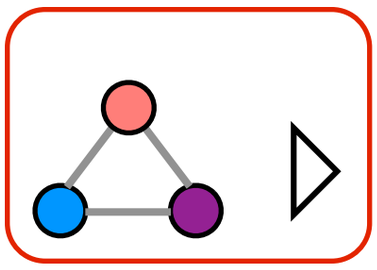
```
Sink<0, sinkChan>
Node<1, chan1, datum1>
Node<2, chan2, datum2>
```

process invocations

as edge list

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

Establishment of a Routing Tree (1)



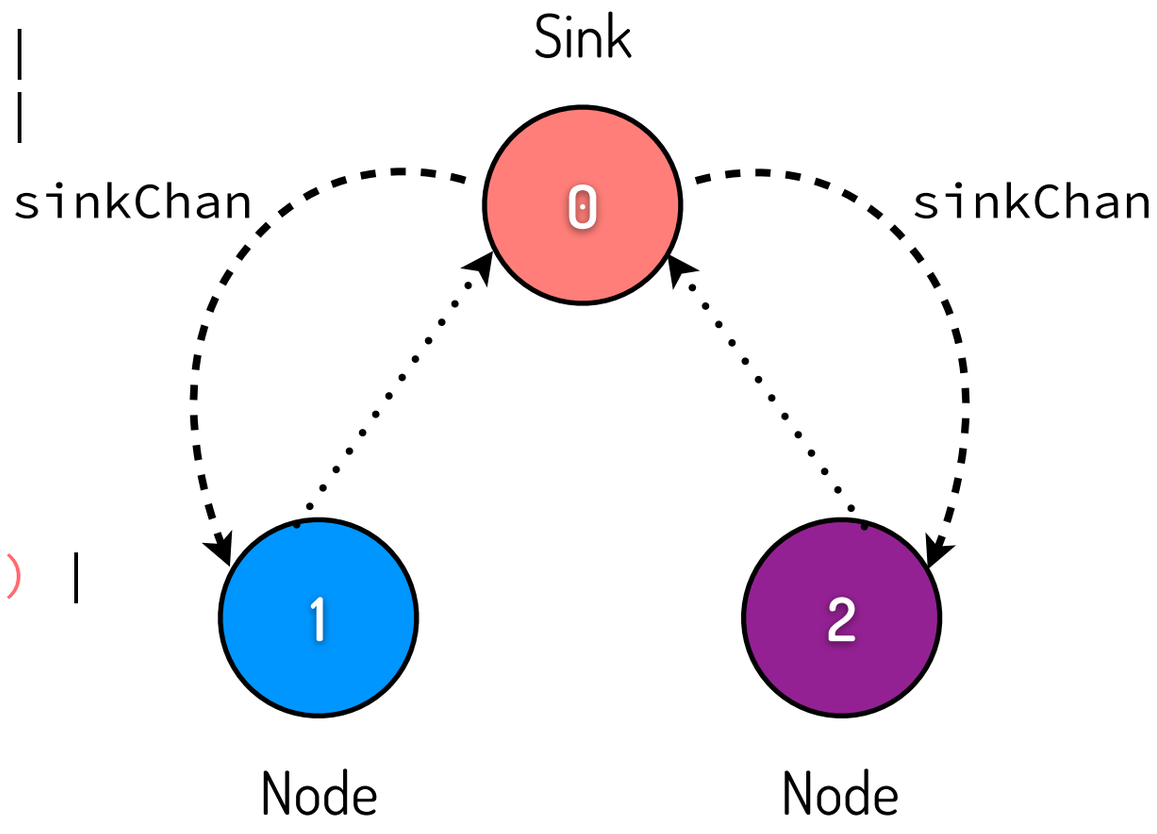
connectivity as current assertion

```
(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>))))
```

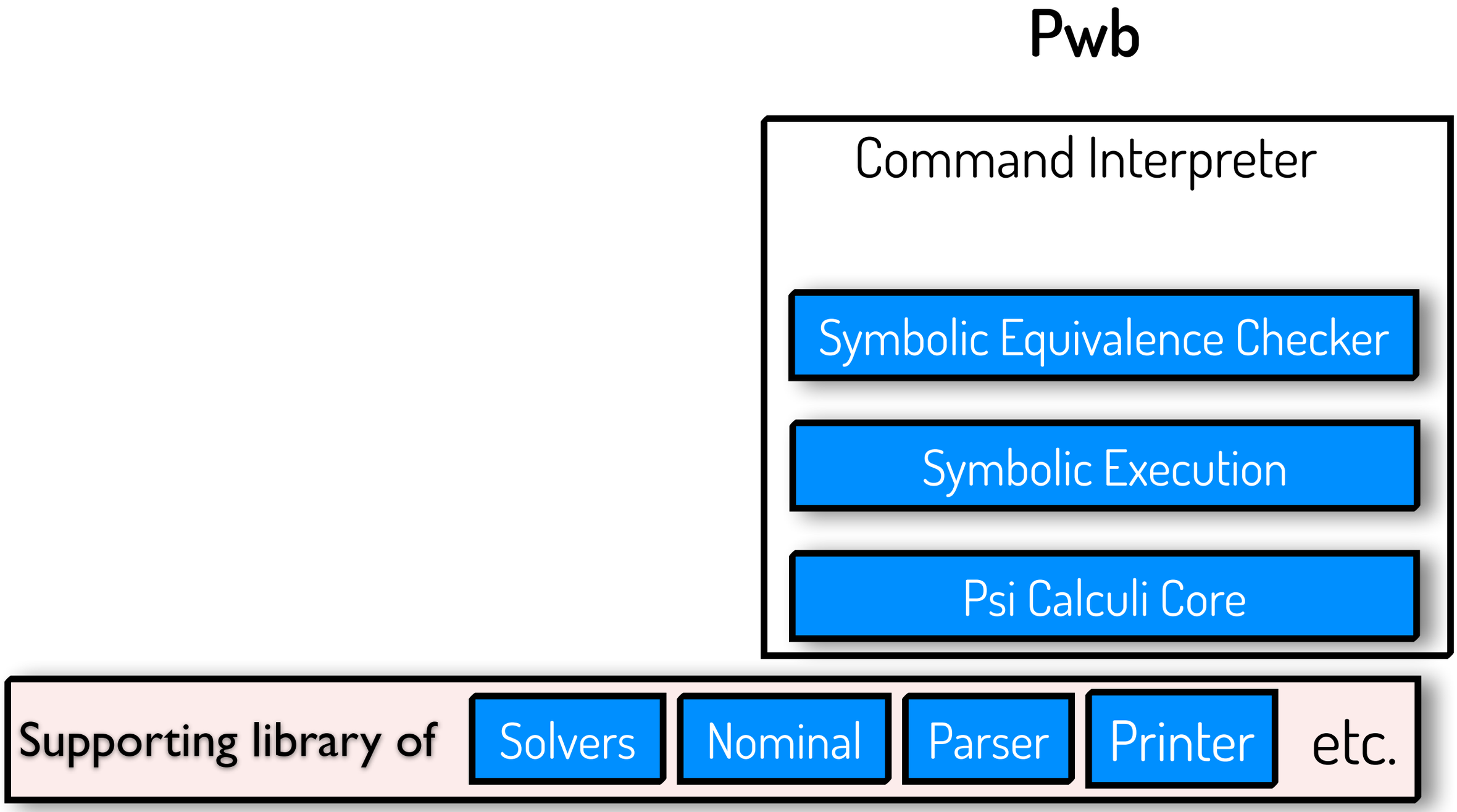


Example Summary

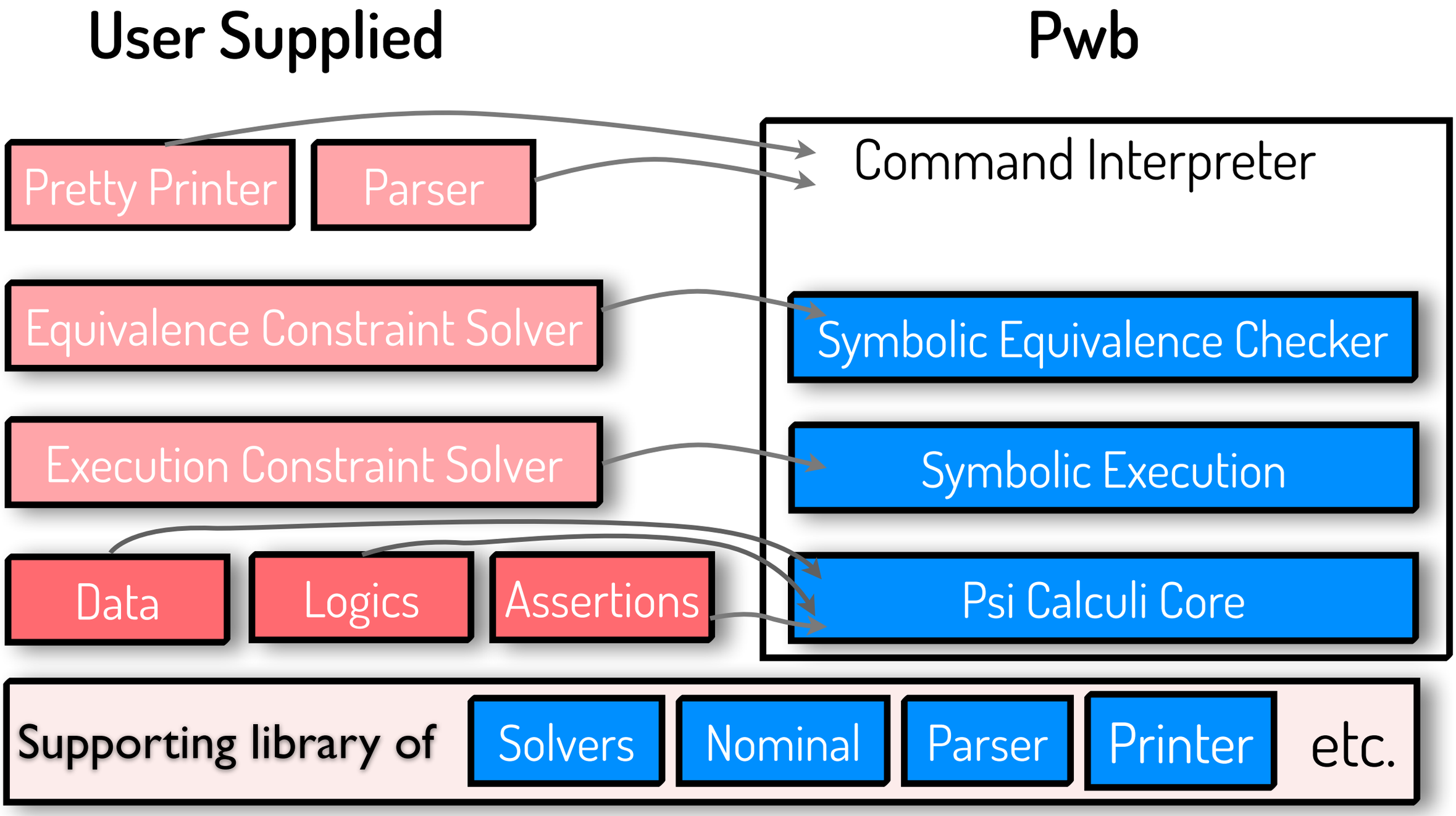
- **Structured channels**
- **Broadcast** and **Unicast** Communication
- Broadcast **Connectivity** as an **assertion**
- Implicitly Parameters of Pwb for WSN

Instantiation of Pwb

Parametric Pwb Architecture



Parametric Pwb Architecture



Implementing the Example in Pwb

Type of Data

```
datatype term
  = Init of term
  | Data of term
  | Name of name
  | Int  of int
```

Type of Logic

```
datatype condition
  = OutputConn of term * term
  | InputConn of term * term
  | ChEq of term * term
```

Type of Assertions

```
datatype assertion = Top
```

Implementing the Example in Pwb

Type of Data

```
datatype term
  = Init of term
  | Data of term
  | Name of name
  | Int  of int
```

Type of Logic

```
datatype condition
  = OutputConn of term * term
  | InputConn of term * term
  | ChEq of term * term
```

Type of Assertions

```
datatype assertion = Top
```

Operations

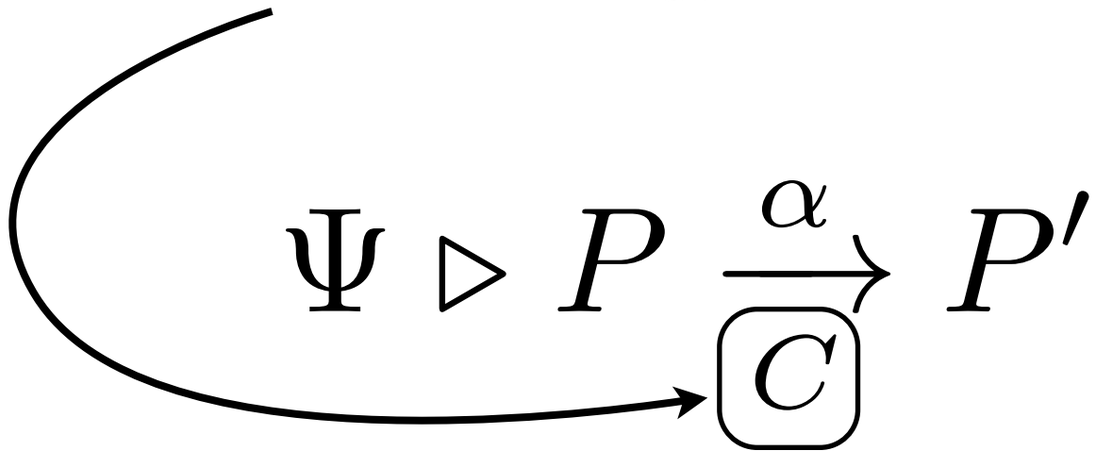
```
chaneq : term*term -> condition
brReceive : term*term -> condition
brTransmit : term*term -> condition
compose : assertion*assertion -> assertion
```

```
substA : (name*term) list -> A -> A
  for A in term, condition, assertion
```

Implementing the Example in Pwb

Execution Constraint Solver

```
solve : condition list -> (string, (substitution*assertion)) either
```



The whole implementation is about **(450 LOC)**

Tool Instance Interface

```
datatype term

datatype condition

datatype assertion
```

```
chaneq : term*term -> condition

brReceive : term*term -> condition

brTransmit : term*term -> condition

compose : assertion*assertion -> assertion

substA : (name*term) list -> A -> A
         for A in term, condition, assertion

solve : condition list ->
        (substitution*assertion) option
```

Future Work

Functionality

- Model Checking
- Behavioral Types
- Instance Algebra

Application

- Bigger Real-World Instances
- Interfacing with Isabelle (generating proofs)

Extensions

- Higher Order Processes
- Reliable Broadcast
- Pattern Matching

Conclusions

- Pwb is a **parametric** tool on data and logics for concurrency
- Pwb is **one tool** for **many calculi** inheriting machine checked proofs
- Pwb provides primitives for both **unicast** and **broadcast** communication
- Pwb provides symbolic **execution** and **equivalence** checking

Session Types for Unreliable Broadcast

Joint work with Dimitris Kouzapas and Simon Gay of Glasgow University

- For Scatter/Gather broadcast protocols
- Protocol is specified as a type
- Type safety ensures recovery of broken sessions and that the processes follow the protocol
- Extension of binary session types to unreliable systems

Where to Get **Pwb**

Pwb Home

http://goo.gl/**ZJPu9**

Pwb on Github

http://goo.gl/**aU40h**

Pwb is free software (GPL)

Runs on UNIX like systems (on Windows, use cygwin)