**Outline**

- Bigraphs
- Bigraphical Reactive Systems
- BPL Tool
- BPL Experiments
- Negative Application Conditions
- BPL Tool Demo

---

**Bigraphs**

—combine place and link graphs

- A bigraph is a place graph (a forest) + a link graph
- Each place graph node has a kind (K, L, M, etc.), that also determines how many ports it has
- Sites (‘[]’) are special leaf nodes representing a “hole”
- Node ports are ordered, but children are not
- The link graph connects ports by hyper edges called links
- We usually illustrate the place graph by nesting its nodes

```
val K2 = active (K2 -: 2)
val M1 = atomic (M1 -: 1)
M1[y1] || L2[y1,y2] o (M1[y1] '[]' '[]')
```

---

**Interfaces**

—Inner and outer faces of bigraphs describe width and names

- Each bigraph has an inner and outer face: \( b : \langle m, X \rangle \rightarrow \langle n, Y \rangle \)
- Inner face \( \langle m, X \rangle \) describes number of sites and inner names
- Outer face \( \langle n, Y \rangle \) describes number of roots and the outer names
- The link graph can connect inner names (as well as ports) to outer names

```
K1[y1] o '[]' || y1/x || L2[y1,y2] o ('[]' '[]' '[]')
: \langle 3, \{x\} \rangle \rightarrow \langle 2, \{y1,y2\} \rangle
```
**Binding Bigraphs**
— a bigraph flavour with link scoping

- Bigraphs come in several flavours; we consider binding bigraphs:
- A link can be **bound** by one of its points, called the **binder**
- The location of the binder defines the **scope** of the link: all other points of the link must lie within the binder's node
- Interfaces are extended to indicate locations of local names

\[
y_1/x_1 \ || \ K_1[y_2] \ || \ L_{10}[\{x_2\}] \ o \ (M_2[y_2,x_2] \ | \ \emptyset)
\]

**Composing Bigraphs**
— if interfaces allow

- Bigraphs \(b_1 : I' \rightarrow J\) and \(b_2 : I \rightarrow I'\) can be composed into \(b_1 \circ b_2\) by putting \(b_2\)'s roots into \(b_1\)'s sites, and merging the names of their common interface \(I'\)

**Bigraph Products**
— if interfaces allow

- The **parallel product** \(b_1 \ || \ b_2\) of bigraphs \(b_1 : I_1 \rightarrow J_1\) and \(b_2 : I_2 \rightarrow J_2\) is computed by juxtaposing the roots and merging global outer names
- Name sets of \(I_1\) and \(I_2\) must be disjoint, and local name sets of \(J_1\) and \(J_2\) must be disjoint
- The **prime product** \(b_1 \ | \ b_2\) is computed from \(b_1 \ || \ b_2\) by merging all the roots

**Bigraphical Reactive Systems**
— a set of rules, each with redex, reactum and instantiation

- A bigraphical reactive system (BRS) is a set of reaction rules
- A rule "\texttt{name} :::: R --[\rho]--> R'\" consists of redex \(R\) and reactum \(R'\) bigraphs, and an instantiation \(\rho\)

"\texttt{REACT2} :::: Sum o (Send2[y_1,y_2,x] `|` \emptyset) `|` Sum o (Get2[x][z_1, z_2] `|` \emptyset) --[0|->0, 1|->2]--> (x/\emptyset || y_1/z_1 || y_2/z_2) o (\emptyset `|` `[z_1, z_2]`)"
Matching and Reaction
—by replacing redex with reactum, and copying parts of the parameter

- \( R \) matches \( a \) if \( \exists \) context \( C \) and parameter \( d \), with \( a = C \circ R \circ d \)
- Reaction replaces redex with reactum, yielding \( a' = C \circ R' \circ d' \)
- \( d' \) is computed from \( d \), copying or discarding parts using \( \rho \)

Objective: create a tool for BRS experiments
- Initial goal: a reference implementation, not efficiency
- Proven correct in great detail
- Simplified matching operation:
  \[
  (a, R \rightarrow [\rho] \rightarrow R') \rightarrow \text{BPL Tool} \rightarrow \text{lazy list of matches } [(C1,d1), (C2,d2), \ldots]
  \]
- Simplified reaction operation:

BPL Tool Matching Engine
—implementing matching, normalisation, renaming, ...

- Matching is based on a system inferring judgments \( a, R \rightarrow C, d \)
- Proven sound and complete (Troels C Damgaard)
- Implemented faithfully, one ML function per inference rule
- Still, implementation is non-trivial:
  > it works on normal forms, so normalisation must be implemented
  > efficient normalisation requires an intricate renaming of names used internally in bigraph terms
  > rules are nondeterministic, so resulting lazy lists must be composed from all combinations of lazy lists derived by subinferences

Detailed reaction operation:

BPL Tool Tactics

We have added basic tactics to the BPL Tool:

\[
t ::= \text{react\_rule \textit{name}} \\
\text{react\_rule\_any} \\
t_1 ++ t_2 \\
\text{TRY } t_1 \text{ ORTHEN } t_2 \\
\text{IF } t_1 \text{ THEN } t_2 \text{ ELSE } t_3 \\
\text{REPEAT } t \\
i \text{TIMES\_DO } t \\
\text{roundrobin} \\
\text{finish} \\
\text{fail}
\]
**BPL Tool Specifications**

—30 thousand lines of code provide Web GUI and CLI

A few facts about the current BPL Tool state:

- It has a prototype web interface at http://tiger.itu.dk:8080/bplweb/
  - for demos and small experiments: matching & single-stepping reactions
- It has a command line interface
  - for systematic and larger experiments: matching, single-stepping reactions, tactics-driven multistep reactions
- Both interfaces allow for visualisation
- It consists of more than 30 thousand lines of code:
  - kernel: 22,000 lines of SML code
  - library: 13,000 lines of SML code
  - web interface: 2,100 lines of Ruby code
- SML code can run on SMLNJ, MosML or MLton

**BRS Uses**

—modelling calculi and ubiquitous systems

Bigraphical reactive systems can be used for

- modelling calculi
  - \( \pi \)
  - ambient calculus
  - \( \lambda \)
  - BPEL
- modelling location based systems: place graph models location, link graph models connectivity

**BPL Tool Experiments**

We have modelled/are currently trying to use the BPL Tool for modelling

- the mobile phone example (Milner’s \( \pi \) book)
- platographical models (with Ebbe Elsborg et al.)
- the GeoCast protocol (Henning Niss)
- the Insider Problem (Eske Bentzon)
- IEEE 802.11 MAC 4-way handshake (Eske Bentzon)
- the ARAN protocol (Eske Bentzon)
- BPEL (with Thomas Hildebrandt, Mikkel Bundgaard, Espen Højsgaard)

**Negative Application Conditions**

—detecting node absence is hard using bigraphs

Bigraph redexes are good at detecting node presence
Bigraph redexes are bad at detecting node absence

Standard trick: segregate nodes into located classes

“M is not present in K”:

\[
\text{notR} = \begin{array}{c}
\text{nodes}
\end{array}
\]

“M is present in K”:

\[
R = \begin{array}{c}
\text{M}
\end{array}
\]
Negative Application Conditions
—detecting link absence is even harder using bigraphs

- Even worse is detecting link absence: $y_1$ can be connected to $y_2$ in the context $\mathbf{R} = KLM$.
- $y_1$ is not connected to $L$ (but possibly others)

Typical trick: remove $L$ nodes outside $K$ iteratively, then check that $K$ contains no $L$ node.
- This requires intermediate representations and iteration.

BPL Tool Demo
—modelling mobile phones in the $\pi$ calculus in the BPL Tool

- As an example, we model a mobile phone connected via a transmitter to a control centre.
- Two actions are possible: The car can talk, or the control centre can switch transmitters.
- System is defined in the $\pi$ calculus.

Mobile Phones Example Definitions

$\text{Car}(\text{talk}, \text{switch}) = \overline{\text{talk}}.\text{Car}(\text{talk}, \text{switch}) + \text{switch}(t, s).\text{Car}(t, s)$

$\text{Trans}(\text{talk}, \text{switch}, \text{gain}, \text{lose}) =$
$\text{talk}.\text{Trans}(\text{talk}, \text{switch}, \text{gain}, \text{lose})$
$+ \text{lose}(t, s).\overline{\text{switch}}(t, s).\text{IdTrans}(\text{gain}, \text{lose})$

$\text{IdTrans}(\text{gain}, \text{lose}) = \text{gain}(t, s).\text{Trans}(t, s, \text{gain}, \text{lose})$

$\text{Control}(\text{lose}_1, \text{talk}_2, \text{switch}_2, \text{gain}_2, \text{lose}_2, \text{talk}_1, \text{switch}_1, \text{gain}_1) =$
$\text{lose}_1(\text{talk}_2, \text{switch}_2).\overline{\text{gain}_2}(\text{talk}_2, \text{switch}_2).$
$\text{Control}(\text{lose}_2, \text{talk}_1, \text{switch}_1, \text{gain}_1, \text{lose}_1, \text{talk}_2, \text{switch}_2, \text{gain}_2)$

Negative Application Conditions Using Tactics
—IF-THEN-ELSE tactics provide shortcut

Using the IF $t_1$ THEN $t_2$ ELSE $t_3$ tactic, we can emulate negative application conditions:

```scala
val testrule = "testrule" ::: R ----|> R
val realrule = "realrule" ::: notR --[rho]--|> R'
val tactic = IF react_rule "testrule" THEN fail ELSE react_rule "realrule"
```
Mobile Phones Example Definition
—in π, BPL Tool, and visual syntax

\[ \text{Car} \langle \text{talk}, \text{switch} \rangle = \text{talk}.\text{Car} \langle \text{talk}, \text{switch} \rangle + \text{switch}(t,s).\text{Car} \langle t,s \rangle \]

val DEF_car =
"DEF_car" :::
\text{Car} \langle \text{talk}, \text{switch} \rangle ----> |\text{Sum} \circ (\text{Send0}[\text{talk}] \circ \text{Car}[\text{talk},\text{switch}]
  '+' \text{Get2}[\text{switch}][[t],[s]]
  \circ (\langle [t,s] \rangle \text{Car}[t,s])\)

\text{Car}
\begin{array}{c}
\text{talk} \\
\text{switch}
\end{array}
\quad \rightarrow \quad
\begin{array}{c}
\text{talk} \\
\text{switch}
\end{array}
\text{Car}
\text{Car}
\begin{array}{c}
\text{Send0} \\
\text{Get2}
\end{array}
\text{Sum}