Tunnelling from CSP to B and Back

Driving the B-Toolkit Animator through a Tunnel

Thai Son Hoang & Ken Robinson
The University of New South Wales, Sydney, Australia

Helen Treharne, Steve Schneider & Neil Evans
Royal Holloway, University of London

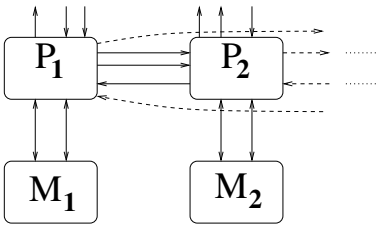
One Day Workshop on B
Royal Holloway, University of London
14th July 2003

Revision: 1.0, July 11, 2003

1. EPSRC Project - **Animating CSP**

Architecture of CSP $\|$ B specifications

CSP for describing flow of control—how the B machine is to be driven.



- Each B machine has a different CSP controller
- Kinds of CSP Event:
  - between a B machine and its controller where CSP events are operation calls.
    $e!v?x$ matches $x \longleftarrow e(v)$.
  - communications between controllers
  - external for a controller

1.1. Controller language

The CSP controller language for driving the B machines is sequential. It is made up of the following clauses:

$$\begin{aligned} P ::= & a \rightarrow P \mid d!v\{E(v)\} \rightarrow P \mid c?x\langle E(x)\rangle \rightarrow P \mid \\ & e!v?x\{E(x)\} \rightarrow P \mid e!v?x\langle E(x)\rangle \rightarrow P \mid \\ & P_1 \,\Box\, P_2 \mid P_1 \sqcap P_2 \mid \sqcap_{x|E(x)} P \mid \\ & \textbf{if } b \textbf{ then } P_1 \textbf{ else } P_2 \mid S(p) \end{aligned}$$

- assumptions are given as $\{E(x)\}$. The process diverges if $E(x)$ fails.
- guards are given as $\langle E(x)\rangle$. Inputs of x which fail $E(x)$ are blocked.

1.2. Joint Animation of CSP and B

Given the preceding proposal to drive B through CSP, it would be useful to be able to drive the B-Toolkit animator from a CSP animator. To achieve this we have constructed a "tunnel" that interfaces to the B-Toolkit animator and to some external program that could be driven by a CSP animator, for example.

2. What is a Tunnel?

A tunnel consists of two channels—implemented by pipes—for connecting the B-Toolkit animator as a server to a client. The two channels consist of:

- an input channel, through which the animator receives machine operation names and arguments from the client and passes them to the B-Toolkit animator;
- an output channel, through which the results of machine operations are delivered to the client.

3. The Role of the Tunnel

The tunnel was devised initially to allow communication between a CSP animator and the B-Toolkit animator. However, the tunnel has been implemented as a general communication mechanism between the B-Toolkit animator and some external tool, so it enables general animator-to-animator communication.

Since B machines contain no operation control flow it is essential that the client source assumes a control role.

4. Understanding the Tunnel

B-Toolkit server end

```
channel tunnelIn : STRING
channel tunnelOut : STRING
channel B-Animator : RESULTS

Tunnel = tunnelIn ?op -> tunnelIn ?args
         -> B-Animator(op,args)? result -> tunnelOut! result
         -> Tunnel
```

The real communication is via characters using a protocol to determine end of construct. Values of arguments and results are communicated as name, value pairs: `name <-- value`.

CSP client end

Suppose a fragment of CSP essentially needs to run the B operation `result <-- op(args)` then the communication is handled as follows:

```
tunnelIn! op -> tunnelIn! args -> tunnelOut? result
```

4.1. Communication from the client tool

In order to communicate with the tunnel from the CSP, or other, animator a communication link will need to be created between the other animator and the two pipes created by the tunnel. The tunnel itself is integrated with the B-Toolkit and nothing extra needs to be done with that end.

5. Using the Tunnel

Establishing the other end There are options in Options/Animator

Tunnel: on/off will enable/disable operation of the animator using the tunnel;

Tunnel driver: the command for executing the tunnel client: for example, a communication link to a CSP animator.

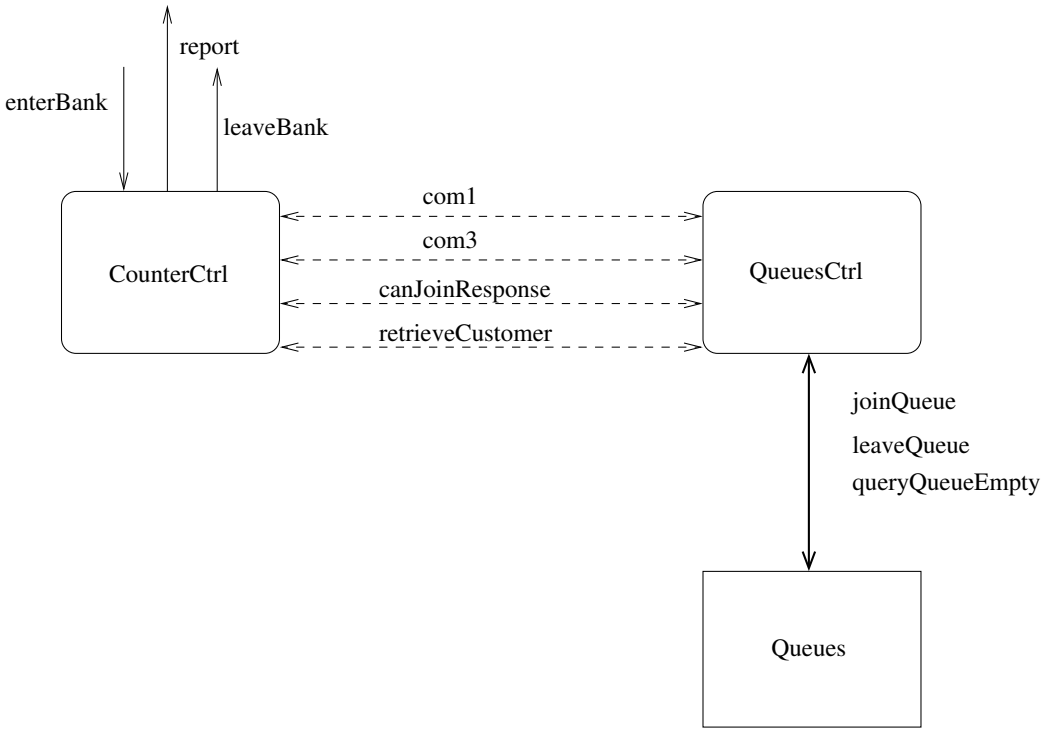
Animation selection If the Tunnel option is on then the animation selection menu will contain Tunnel allowing the animator to be run in tunnel server mode.

5.1. B-Toolkit Animator Communication

The following table describes the animator communications and identifies those handled by the tunnel and those handled by manual interaction through the standard GUI.

Machine parameters	manual
Context	manual
Script initialisation	manual
Operation selection	tunnel
Operation arguments	tunnel
Non-determinism resolution	manual
Outputs	tunnel

5.2. Bank System Architecture



6. Exploring traces

One possible CSP animator is provided by ProBE (Formal Systems). ProBE provides the capability of exploring multiple traces, but that raises some problems for the B animator. Consider the following fragment of B:

```
MACHINE     Simple
VARIABLES   xx,yy
INVARIANT   xx : NAT & yy : NAT
INITIALISATION xx := 0 || yy := 0
OPERATIONS
  op1 xx:=1 || yy :=1;
  incop  xx := xx + 1 || yy := yy +1
```

and the following fragment of CSP:

$$P = a \rightarrow (op1 \rightarrow b \rightarrow incop \rightarrow P) \\ \Box \\ (incop \rightarrow P)$$

- Option 1
  - Exploring traces linearly
  - Need to ensure consistency of B when we jump back to the beginning of a branch. We can do this by undoing the operation calls.
- Option 2
  - Extend the underlying B animator to keep a complex history of the state.

7. Ongoing - Dealing with outputs

- Linking the outputs of a B operation and controlling how the CSP animator should behave is not trivial.
- E.g. If the B operation outputs a value $joinQueue$ from the Bank how do we represent this in ProBE?

- Should it offer the two possibilities in the branch and then after the tunnel retrieves the result puts up a message saying this was not a possible output and backtrack along the exploration?
- Or should we pre-run the operation before deciding what to offer in ProBE as the next possible events?