Translating Process Algebra to Uppaal

- Declare processes with channels as parameters.
- Allow linear equations only:

$$X = a! + c? + a!.Y + c?.Z \qquad Z = \delta$$

τ	`tau`
δ	`delta`
- systems are parallel compositions of processes (with channels as arguments).

Example

```
chan a

proc x is
 X = a! . X
in X

proc y(c,d) is
 Y = c? . Y + d? . Z
 Z = delta
in Y

system x || y(a,a)
```

Encapsulation

- Given $H \subset \Sigma$, we add syntax $\partial_H(p)$.
- The operator ∂_H disallows action from H.
- E.g. if $\gamma(a,b) = c$ then

$$a \parallel b = ab + ba + c$$

and

$$\partial_{\{a,b\}}(a \parallel b) = c$$

- We need this operator to express that in Uppaal there is no communication with the outside world.

Transition rules for encapsulation

$$\frac{x \xrightarrow{v} \surd \quad v \notin H}{\partial_H(x) \xrightarrow{v} \surd} \qquad \frac{x \xrightarrow{v} x' \quad v \notin H}{\partial_H(x) \xrightarrow{v} \partial_H(x')}$$

Axioms for encapsulation

D1	$\partial_H(v)$	$=$	v		$v \notin H$
D2	$\partial_H(v)$	$=$	δ		$v \in H$
D3	$\partial_H(\delta)$	$=$	δ		
D4	$\partial_H(x + y)$	$=$	$\partial_H(x) + \partial_H(y)$		
D5	$\partial_H(x \cdot y)$	$=$	$\partial_H(x) \cdot \partial_H(y)$		

Stuttering

- Intuitively means moving between states with the same atomic propositions.
- A run is a sequence of subsets of Prop:

$$S_1' \, S_2' \, S_3' \cdots$$

- Defining

$$S^1 = S, \; S^{n+1} = S \, S^n,$$

  we can uniquely write any run π as

$$S_1^{n_1} \, S_2^{n_2} \, S_3^{n_3} \cdots$$

  where $S_i \neq S_{i+1}$.
- The stutter free variant of π, denoted $E(\pi)$, is

$$S_1 \, S_2 \, S_3 \cdots$$

Stuttering Invariance

- An LTL formula ϕ is stutter invariant if

$$\pi, 0 \models \phi \Leftrightarrow E(\pi), 0 \models \phi$$

- The fragment of LTL, without the next time operator X is stutter invariant. This fragment is denoted LTL-X.

Safety properties

- Intuitively it is a property that says that bad things never happen.
- A safety property is a property that allows failure detection in finitely many steps.
- Given a property.
    - A bad prefix is a finite prefix of a computation for which the property fails, such that any computation starting with the same prefix also fails the property.
    - The property is a safety property is every failing computation has a bad prefix.
- The following class of LTL formula's are safety formula's

$$\phi_s ::= p \mid \neg p \mid \phi_s \wedge \phi_s \mid \phi_s \vee \phi_s \mid X\,\phi_s \mid \phi_s\ R\ \phi_s$$

Liveness properties

- Intuitively it is a property that says that good things happen infinitely often.
- For LTL we can just say that a property is a liveness property if it isn't a safety property.

Fairness

- Intuitively it means that everybody gets their turn.
- Let there be N processes $1 \leq i \leq N$.
- *Weak fairness* If from a certain point in the computation a step is continuously enabled (p_i) then it is executed infinitely often (q_i).

$$\bigwedge_{1 \leq i \leq N} (\Diamond\Box p_i \rightarrow \Box\Diamond q_i) \sim \bigwedge_{1 \leq i \leq N} \Box\Diamond(\neg p_i \vee q_i)$$

- *Strong fairness* If a step is infinitely often enabled then it is infinitely often executed.

$$\bigwedge_{1 \leq i \leq N} (\Box\Diamond p_i \rightarrow \Box\Diamond q_i)$$

Model Checking and Fairness

- To check whether a property holds for fair computations, we can obviously check the formula

$$\text{fairness} \rightarrow \text{property}$$

- Unfortunately, many tools will experience an exponential blow-up in the number of processes.
- If you are lucky, the tool will have support for fairness
- Otherwise, you will have to do the work yourself.

Fairness for continuously enabled processes.

Let process be a variable containing the process that made the last step.
Given a Büchi Automaton and K processes, make $K + 2$ copies $(0, \cdots, K + 1)$ of the Büchi automaton.

- Remove the acceptance conditions from all copies, except copy 0.
- In copy 0, redirect edges starting in accepting states to copy 1.
- In copy i $(1 \leq i \leq K)$ duplicate the edges to the next copy with the extra condition process $= i$ and add the condition process $\neq i$ to every old edge.
- In copy $k + 1$, redirect all edges to copy 0.

Fairness for processes, which once enabled, remain enabled until taken.

Given a Büchi Automaton and K processes, make $K+2$ copies $(0, \cdots, K+1)$. of the Büchi automaton.

- Remove the acceptance conditions from all copies, except copy 0.
- In copy 0, redirect edges starting in accepting states to copy 1.
- In copy i $(1 \leq i \leq K)$ duplicate the edges to the next copy with the extra condition [process $= i$ or process i not enabled] and add the condition [process $\neq i$ and process i enabled] to every old edge.
- In copy $k+1$, redirect all edges to copy 0.

Monitoring fairness of communication

- Given N channels $1, \cdots, N$.
- Suppose that *every* computation has infinitely many receive operations.
- Declare an integer state and a boolean fair.
- Set state to 0 and fair to true.
- For every receive:
  - If state is 0 then set fair to false and state to 1.
  - If state is $1, \cdots, N$ and the receive is on channel state or channel state is empty then increase state.
  - If state is $N + 1$ then set fair to true and state to 0.
- The LTL formula $\Box\Diamond$*fair* expresses fairness of communication.
- The CTL formula `fair --> not fair` and `not fair --> fair` together expresses fairness of all computations.