

Labelled transition systems, CCS, and the Mobility Workbench

Model-based Design of Distributed and Mobile Systems
Lecture 2: Spring 2004

Thomas Hildebrandt

hildeitu.dk

Theory Department
IT University of Copenhagen

Overview of the Lecture

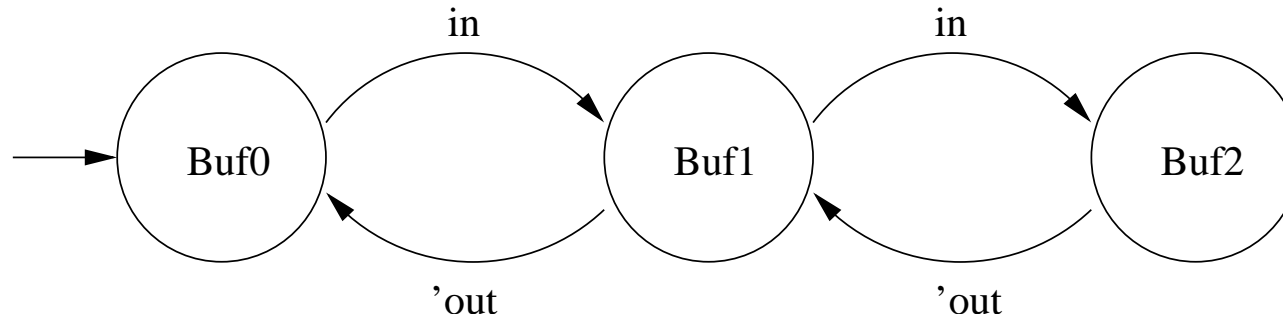
The structure of the lecture:

- Introduction to CCS
- Labelled Transition Systems
- Presentation of a fragment of CCS:
 - Syntax
 - Semantics (informal and formal)
 - Examples

Labelled Transition System

- The behaviour of a process is often defined using a *Labelled Transition System* (LTS)
- A labelled transition system is basically a directed graph with labels on the edges.
 - States = Vertices
 - Labelled transitions = Edges with labels

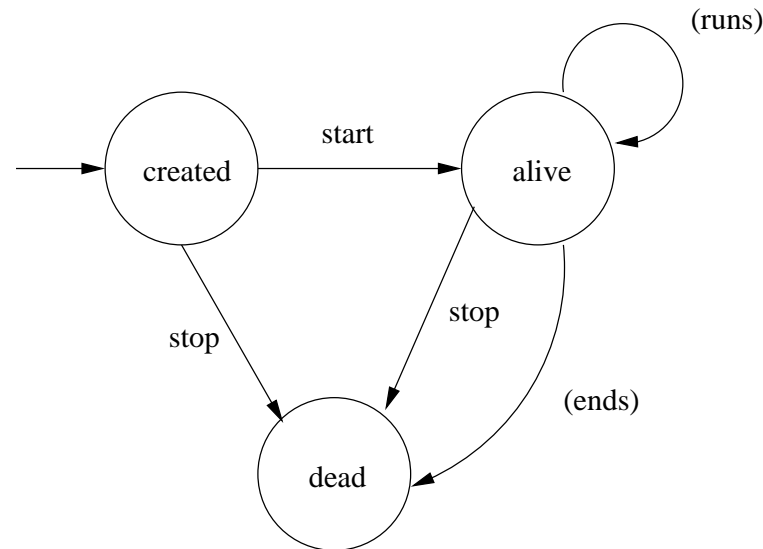
Example of a Transition System



- 3 states: Buf0, Buf1, and Buf2
- 2 labels: in (an input label) and 'out (an output label)
- Even though the *state space* is finite the behavior of the process is infinite. Possible traces:
 - in
 - in, in
 - in, 'out
 - ...

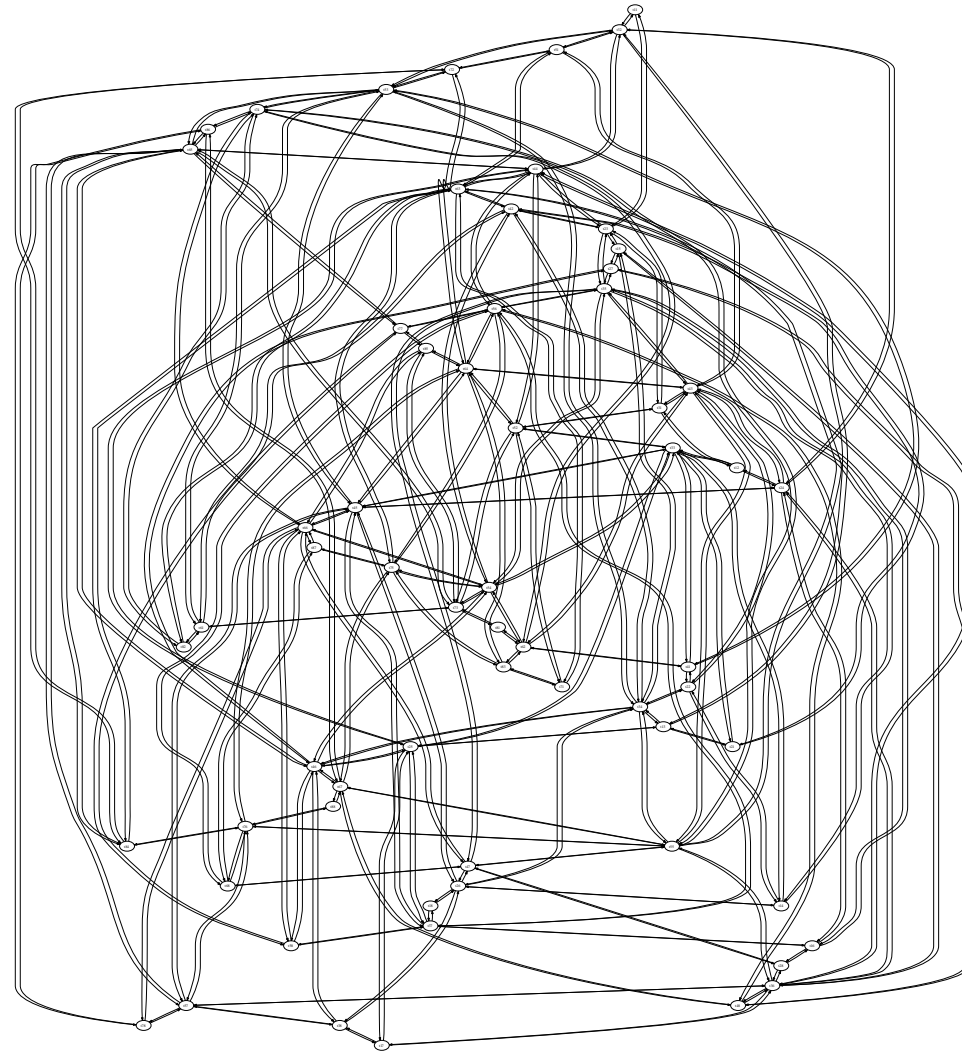
Another Example

- The state machine of the Java Thread from last lecture. In CCS we represents the *silent action* (or *internal action*) by τ .



- A LTS often gives a good intuition about the behaviour of a process, when the state space is small. For larger systems process expressions are preferred

Finite State (L)TS — Chess Knight



Calculus of Communicating Systems (CCS)

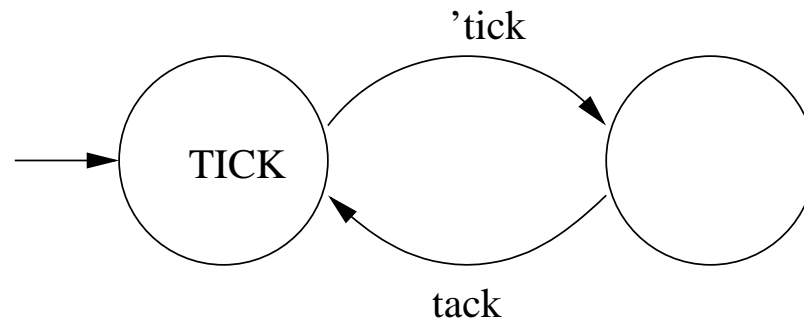
- Robin Milner developed CCS in the late 1970's. The main reference is [Mil89] "Communication and Concurrency"
- CCS is a simple process algebra (calculate and reason about processes)
- The motivation for CCS was to understand *communicating systems* in terms of a few primitive ideas.
- CCS is very low level, focus on theory for reasoning about the behaviour of systems.
- CCS has no primitives/operators for mobility of processes nor any dynamic creation of connections (static connectivity)

Syntax of a fragment of CCS

$P ::=$	0	the inactive process
	$\alpha.P$	perform the action α and continue as P
	$P1 \mid P2$	run $P1$ and $P2$ in parallel
	$P1 + P2$	run either as $P1$ or $P2$
	$Iden\langle nlist \rangle$	run as the process named $Iden$ instantiated with the names in $nlist$
	(P)	parentheses are used for enforcing precedence

Semantics — Action prefix

- The *action prefix* $\alpha.P$ in CCS represents a process which performs the action α and continues as P .
- In CCS there are two kinds of visible actions in CCS:
 - *input* actions α ,
 - *output* actions $'\alpha$, and



agent `TICK(tick,tack) = 'tick.tack.TICK`

Is there another way to define the agent TICK ?

Internal Actions

- The third kind of action in CCS is the *internal action*
- In CCS we represent the internal action or silent action by τ
- This action cannot be observed (other than something happened)
- So for example starting a thread T parallel to the execution of the main program P can be encoded in CCS as $\tau.(T\langle \dots \rangle \mid P)$
- This kind of action is otherwise primarily connected to synchronisation (in the next lecture)

Semantics Continued – Choice or Sum

- Having only prefix, we are only able to describe a single set of *execution traces*.
- The *choice or sum operator* $+$ allows us to implement a choice between two processes.
- $a.P1 + b.P2$ represents a process which can either do an a action and become $P1$ or it can do a b action and become $P2$.
- The operator can be generalised from pairs to an arbitrary number of summands, written Σ .

How does the LTS look for the process ?

agent CHOICE(decrease, increase) = decrease.SMALLER +
increase.LARGER

Non-deterministic Choice or Sum

- The operator can be *non-deterministic* if several of the *guarding actions* are equal.
- The vending machine below can, when given a coin either deliver coffee or tea.

```
agent VM(coin,coffee,tea) = coin.'coffee.VM +  
                             coin.'tea.VM
```

- We can make this vending machine deterministic (so the user can decide) by delaying the choice of beverage.

```
agent VM(coin,coffee,tea) =  
    coin.('coffee.VM + 'tea.VM)
```

Semantics Continued — Process parameters

- Processes in CCS can be *parametric* (and in MWB they have to be)
- The process below defines process P with two parameters *first* and *second*.
- When we use P we can instantiate these names to whatever we like.

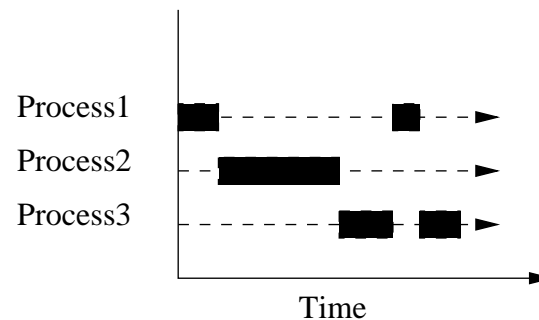
```
agent P(first,second) = first.second.P<first,second>
```

- So the process below is actually equal to the one above, since we have just changed the names of the variables.

```
agent P(f,s) = f.s.P<f,s>
```

True Concurrency vs. Interleaving

- In Java each thread executes as a sequential program and the JVM then *interleaves* the processes
- In CSS we also interleave the actions of the processes



- Alternative to interleaving we have *true concurrency*, where multiple actions can be executed at a time
- The following are examples of true concurrency: the Petri Nets model, the 2 ALU's in Pentium 4

Semantics Continued — Parallel composition

- The parallel composition $P \mid Q$ of the two processes P and Q represents the concurrent execution of the two processes
- The actions of P and Q will be interleaved in some arbitrary order
- So $a.b.0 \mid c.d.0$ have several traces
 - a, b, c, d
 - a, c, d, b
 - ...
- But a will always come before b and c before d

Syntax Examples

For example the following expressions are all *syntactical correct*

- $sit.stand.walk.0$
- $(sit.eat.0 + stand.walk.0) \mid Loop < talk >$ (assuming that $Loop(a) = a.Loop < a >$)
- $hello.0 \mid world.0 + in.0 + ccs.0$

What about these expressions are they syntactical correct ?

- $hello.name + bye.name.0$
- $doSomething.0 + hide.show.0$
- $split.(doA.0 \mid doB.0)$
- 0.0

Transition Semantics of a Fragment of CCS

$$\text{ACT} \frac{}{\alpha.P \xrightarrow{\alpha} P}$$

$$\text{SUM} \frac{P_j \xrightarrow{\alpha} P'_j}{\sum_{i \in I} P_i \xrightarrow{\alpha} P'_j} \quad (j \in I)$$

$$\text{COM1} \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

$$\text{COM2} \frac{P \xrightarrow{\alpha} P'}{Q \mid P \xrightarrow{\alpha} Q \mid P'}$$

$$\text{CON} \frac{P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'} \quad (A \stackrel{\text{def}}{=} P)$$

References

References

- [Mil89] Robin Milner. *Communication and Concurrency*.
Prentice-Hall, 1989.