# T–79.4301 Parallel and Distributed Systems (4 ECTS)

## *Lecture 10*

*27th of November 2008*

Keijo Heljanko

Keijo.Heljanko@tkk.fi



# Other models of Concurrency

- Process algebras - An algebraic way of compactly specifying LTSs. Example specifying two synchronizing LTSs: $I = ((a.(\tau.c.0 + b.0)) \mid\mid (a.b.0))$, where "||" is parallel composition, "." is sequential composition, "+" in non-deterministic choice, and "0" is a deadlocking process. Lots of variants exist, the most well know are CCS and CSP.
- Petri nets - A model of concurrency developed by C.A. Petri in 1962. Also lots of variants exist.
- Extended finite state machines, SMV programs (input language of the NuSMV model checker), . . .



# Petri nets

For another perspective into models of concurrency, consider Petri nets. The class we use are called place/transition nets (P/T-nets). A P/T-net is a tuple $N = (P, T, F, W, M_0)$, where

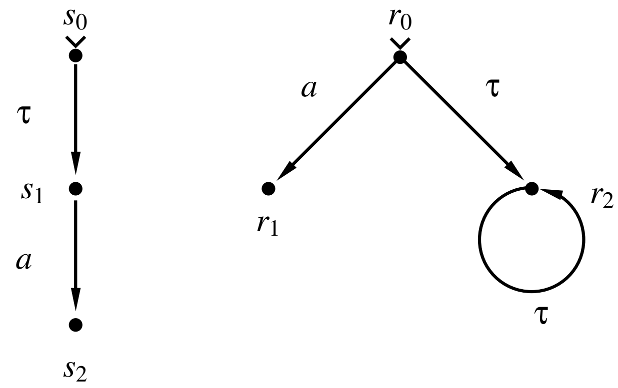
- $P$ is a finite set of places,
- $T$ is a finite set of transitions,
- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation,
- $W : F \mapsto \mathbb{N} \setminus \{0\}$ is the arc weight mapping, and
- $M_0 : P \mapsto \mathbb{N}$ is the initial marking.



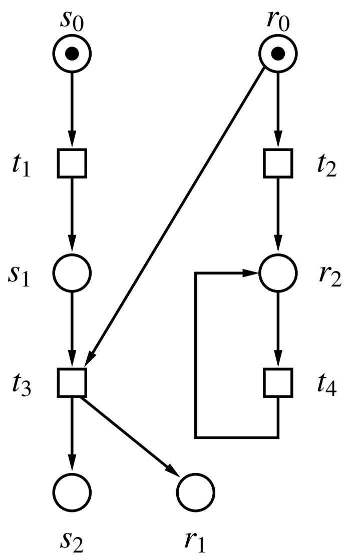
# Running Example

Recall the synchronization of LTSs from Lecture 6:

$L_1 : \quad \Sigma_1 = \{a\} \qquad L_2 : \quad \Sigma_2 = \{a\}$



# Running Example as P/T net






# The running Example

- Places $P = \{s_0, s_1, s_2, r_0, r_1, r_2\}$.
- Transitions $T = \{t_1, t_2, t_3, t_4\}$.
- Flow relation $F = \{(s_0,t_1),(t_1,s_1),(r_0,t_2),(t_2,r_2),$ $(s_1,t_3),(r_0,t_3),(t_3,s_2),(t_3,r_1), (r_2,t_4),(t_4,r_2)\}$.
- Arc weight mapping $W(x,y) = 1$ for all $(x,y) \in F$. We use the convention that only arcs weights $W(x,y) > 1$ are drawn next to the arc $(x,y)$, i.e., the default arc weight is $1$.
- Initial marking $M_0 = \{s_0 \mapsto 1, s_1 \mapsto 0, s_2 \mapsto 0,$ $r_0 \mapsto 1, r_1 \mapsto 0, r_2 \mapsto 0\}$.



# From LTSs to P/T-nets

Intuition behind the mapping:

- Local states of the components are mapped to places.
- Transitions of the Petri net consist of all legal ways of synchronizing the local transitions of the components. (Potential size blow-up here!)
- The flow relation records what is the precondition under which the synchronization can happen, and what is the effect of the synchronization on the state of each component.
- The initial marking records the initial state of the components.



# From LTSs to P/T-nets

- Given $L = L_1 || L_2 || \cdots || L_n$ with $L_i = (\Sigma_i, S_i, S_i^0, \Delta_i)$, we get a P/T-net $N_L$ as follows:
- $P = S_1 \cup S_2 \cup \cdots \cup S_n$,
- $T \subseteq \Delta_1 \cup \{-\} \times \Delta_2 \cup \{-\} \times \cdots \times \Delta_n \cup \{-\}$ (to be defined on the next slide),
- $F$ is the smallest relation satisfying for every (P/T-net) transition $g \in T$:
  - For all $1 \leq i \leq n, t_j = (p,l,p') \in \Delta_i$: If $g = (\ldots, t_j, \ldots)$ then $(p,g) \in F$ and $(g,p') \in F$.
- $M_0(p) = 1$ if $p \in S_1^0 \cup S_2^0 \cup \cdots \cup S_n^0$, and $M_0(p) = 0$ otherwise.

# From LTSs to P/T-nets (cnt.)

- For all $x \in \Sigma \cup \{\tau\}$ and all $g \in \Delta_1 \cup \{-\} \times \Delta_2 \cup \{-\} \times \cdots \times \Delta_n \cup \{-\}$ the (P/T-net) transition $g = (t_1, t_2, \ldots, t_n) \in T$ iff:
  - $x = \tau$: there is $1 \leq i \leq n$ such that $t_i = (s_i, \tau, s_i') \in \Delta_i$ and $t_j = -$ for all $1 \leq j \leq n$, when $j \neq i$.
  - $x \neq \tau$: for every $1 \leq i \leq n$: $t_i = (s_i, x, s_i') \in \Delta_i$, when $x \in \Sigma_i$ and $t_i = -$, when $x \notin \Sigma_i$.

Finally we define $W(x,y) = 1$ for all $(x,y) \in F$.



# From LTSs to P/T-nets (cnt.)

- We now claim that reachability graphs of $L = L_1 || L_2 || \cdots || L_n$ and $N_L$ are the same.
- However, to do so we have to define the behavior of P/T-nets.



# Behavior of P/T-nets

- The state of a P/T-net consist of a *marking* $M : P \mapsto \mathbb{N}$, which tells for each place how many *tokens* (drawn as black dots) it contains.
- The notation $M(p)$ denotes the number of tokens in place $p$.
- In our running example $M(p) \leq 1$ for all places $p \in P$, i.e., each place contains at most one token. However, this is not required in general.



# Behavior of P/T-nets

- The *preset* of a node $x \in P \cup T$ is denoted by ${}^\bullet x$ and defined to be: ${}^\bullet x = \{y \in P \cup T \mid (y,x) \in F\}$. The preset of a node consist of those nodes from which an arc to $x$ exist. In our running example ${}^\bullet t_3 = \{s_1, r_0\}$.
- The *postset* of a node $x \in P \cup T$ is denoted by $x^\bullet$ and defined to be: $x^\bullet = \{y \in P \cup T \mid (x,y) \in F\}$. The postset of a node consist of those nodes to which an arc from $x$ exist. In our running example $t_3{}^\bullet = \{s_2, r_1\}$.

# Enabling of transitions

- To simplify definitions, we extend $W(x,y)$ to all pairs $(x,y) \in (P \cup T) \times (T \cup P)$ as follows: if $(x,y) \notin F$ then $W(x,y) = 0$.
- A transition $t \in T$ is enabled in marking $M$, denoted $t \in enabled(M)$, iff for all $p \in P : M(p) \geq W(p,t)$. (All places $p$ which are in the preset of $t$ contain at least the number of tokens specified by $W(p,t)$.)



# Firing of transitions

- The marking $M'$ reached after firing $t$, denoted $M' = \mathit{fire}(M,t)$, is defined for all $p \in P$ as: $M'(p) = M(p) - W(p,t) + W(t,p)$. (First remove as many tokens as given by $W(p,t)$ from all places in the preset of $t$, and then add as many tokens for all places in the postset of $t$ as denoted by $W(t,p)$.)



# Reachability graph

Analogous to the similar definition for LTSs (from end of Lecture 5): Reachability graph $G = (V, E, M_0)$ is the graph with the smallest sets of nodes $V$ and edges $E$ such that:

- $M_0 \in V$, where $M_0$ is the initial marking of the net $N$, and
- if $M \in V$ then for all $t \in enabled(M)$ it holds that $M' = \mathit{fire}(M,t) \in V$ and $(M, t, M') \in E$.



# Reachability graph (cnt.)

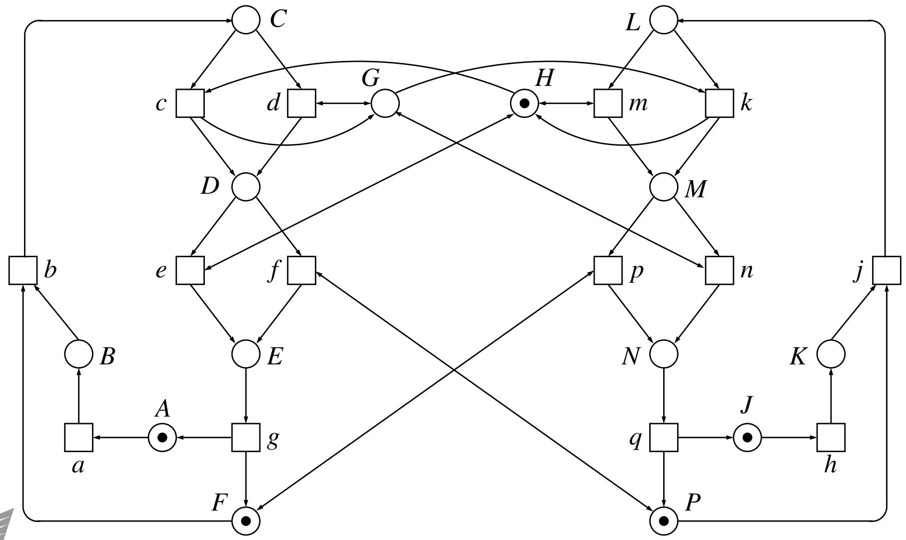
- It is easy to define a P/T-net with an infinite reachability graph.
- A place $p \in P$ is defined to be $k$-bounded iff for all reachable markings $M \in V$ it holds that $M(p) \leq k$.
- A net is defined to be $k$-bounded if all its places are $k$-bounded
- A net is defined to be *unbounded* (i.e., infinite state) iff it is not $k$-bounded for any $k \in \mathbb{N}$.

# P/T-nets and Turing machines

- It is *not* possible to simulate a Turing machine with a P/T-net. Asking whether a marking $M$ is reachable is in fact decidable for P/T-nets (even with infinite reachability graphs).
- The algorithms used are quite involved, and we do not know of an implementation of the theoretical result in question.
- There is a simple (but slow in the worst case) algorithm which can compute which places of the net are unbounded, called the coverability graph algorithm.



# Peterson's Mutex (by W. Reisig)






# From 1-bounded P/T-nets to LTSs

- A $1$-bounded P/T-net $N$ with $|P|$ places can always be converted to a synchronization of LTSs $L_N = L_1 || L_2 || \cdots || L_n$ with $n \leq |P|$ components which have two states each. The reachability graph of $L_N$ will be isomorphic to that of $N$.
- The construction is slightly too complicated to show here. The main trick is to use the set of transitions $T$ as the alphabet $\Sigma$ in $L_N$, and to make each $L_i$ corresponding to a place $p \in P$ synchronize on all labels $t \in {}^\bullet p \cup p^\bullet$.



# From P/T-nets to Promela

- Suppose that the net $N$ we are looking is $255$-bounded. Holzmann suggests the following scheme for translating P/T-nets (with $W(x,y) = 1$ for all $(x,y) \in F$, a restriction which can be easily removed) to Promela as shown in the next two slides.

# From P/T-nets to Promela (cnt.)

```
#define Place byte /* < 256 tokens per place */

Place s0, s1, s2, r0, r1, r2;

#define inp1(x)         (x>0) -> x--
#define inp2(x,y)       (x>0&&y>0) -> x--; y--

#define out1(x)         x++
#define out2(x,y)       x++; y++
```



# From P/T-nets to Promela (cnt.)

```
init
{
          atomic {s0=1;r0=1} /*initial marking*/
          do
/* t1 */  :: atomic { inp1(s0)   -> out1(s1) }
/* t2 */  :: atomic { inp1(r0)   -> out1(r2) }
/* t3 */  :: atomic { inp2(s1,r0)-> out2(s2,r1)}
/* t4 */  :: atomic { inp1(r2)   -> out1(r2) }
          od
}
```



# From P/T-nets to Promela (cnt.)

- Actually, all `atomic` statements of the translation can safely be replaced with `d_step` statements.
- By using the LTS to P/T-net mapping first also LTSs can be translated to Promela.



# From P/T-nets to Promela (cnt.)

- It may be more efficient to use a Petri net model checker such as PROD (`http://www.tcs.hut.fi/Software/prod/`) to do the model checking as for example the partial order reductions in Spin are not really effective for the model obtained from the translation. (The concurrency of the model is hidden inside the data manipulation of a single process.)
- Another Petri net model checker is Maria (`http://www.tcs.hut.fi/Software/maria/index.en.html`).
- Both of the tools actually use high-level Petri nets, which contain extensions to deal with structured data

# Extending LTSs with Data

- Sometimes it is convenient to extend the LTS model with data in order to more conveniently model Promela like languages.
- We will sketch the idea below in an informal manner.
- The idea is the following: Assume we have a system with $n$ LTS components $L_i$, which manipulate $m$ global variables $x_j$ with a value range $0 \ldots r$.



# ELTSs

- The state vector of the extended LTS system (ELTS) will consist of a tuple $(s_1, s_2, \ldots, s_n, v_1, v_2, \ldots, v_m)$, where $s_i$ is the current local state of the component $L_i$ and $v_j$ is the current value of the global variable $x_j$.
- The initial state will be extended to give initial values for all the global variables.
- For each global variable we can define some operations, for example `inc(x)` to increment the value of global variable $x$, `dec(x)` to decrement it, and expressions like `iszero(x)` to check whether the variable is zero. (Expressions must be side-effect free.)



# ELTSs (cnt.)

- Now we can for each local transition of the LTS add a guard: a list of expressions evaluated using the current values of the global variables. The guard will evaluate to true iff all the expressions evaluate to true.
- A global transition will be enabled iff all guards of all its component transitions evaluate to true.



# ELTSs (cnt.)

- To update the global variables, each local transition is also associated with a list of operations.
- When a global transition is fired, each of the local transitions participating in it will in their turn execute its list of operations on the global variables.
- The state of global variables obtained after all operations have been executed is recorded as the state reached after firing the global transition.

# ELTSs (cnt.)

- It is fairly straightforward to include other data manipulation features of Promela such as FIFOs and all their expression and operations in an ELTS model.
- The part that is hard to faithfully handle using ELTSs are the `atomic` and `d_step` features of Promela.
- There are many variants of the ELTS model, also state machines variants (extended finite state machines, EFSMs) are pretty common in the literature.



# Promela and EFSMs

- Internally inside Spin all Promela programs are first translated into (a Spin variant of) extended finite state machines EFSMs.
- Consider for example the Peterson's Mutex algorithm shown in the next slide.
- Its EFSM can be produced in xspin using the feature "View Spin automaton for each Proctype" available in the run menu. The automaton is shown in the next slide following the Promela code.

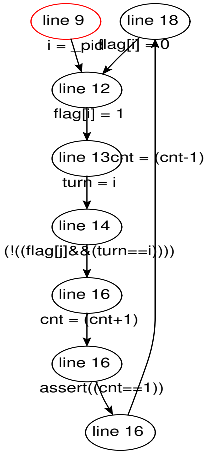


# Promela: Peterson's Mutex

```
bool turn, flag[2]; /* Code reformatted, old line numbers below */
byte cnt;
active [2] proctype P1()
{       pid i, j;
        i = _pid;                               /* line 9 */
        j = 1 - _pid;
again:  flag[i] = true;                         /* line 12 */
        turn = i;
        !(flag[j] && turn == i) ->
                cnt++; assert(cnt == 1); cnt--; /* line 16 */
        flag[i] = false;                        /* line 18 */
        goto again
}
```



# EFSM for Peterson's Mutex



# Notes on the Spin EFSM

- Notice how all the control flow statements have been removed, and all that remains is a state machine with expressions added to the edges. For example, the `goto` on line 19 has been removed. No `goto` statements will exists in any Spin EFSMs. (The picture is incomplete wrt. expressions.)
- Spin has done some internal optimizations. For example, there is no state of the automaton corresponding to line 10 of the program. This optimization is safe because $j$ is a local variable.
- At runtime there are two instances of the same EFSM running.



# Spin EFSMs

The statements appearing on edges of Spin EFSMs are:

- Assignments
- Assertions
- Print statements
- Send or Receive Statements
- Promela expressions (expression statements)

All other features of Promela (if-statements,do-loops, gotos, etc.) are mapped to the structure of the state machine part of the Spin EFSM.