T–79.4301 Parallel and Distributed Systems (4 ECTS)

T–79.4301 Rinnakkaiset ja hajautetut järjestelmät (4 op)

Lecture 9

7th of April 2008

Keijo Heljanko

Keijo.Heljanko@tkk.fi

Liveness

- Liveness properties are properties of systems that are characterised by the intuitive formulation: “eventually something good happens”.
- Another intuition is the following: For finite state systems all counterexamples demonstrating that a liveness property does not hold are of the form $s^0 \xrightarrow{p} s' \xrightarrow{l} s'$, where l is a non-empty execution of the system starting from state s' and ending in state s', an “nothing good” happens in l.
- Thus, intuitively, liveness properties specify what kinds of loops in the system behavior are allowed for correct implementations.

Liveness - Examples

- All executions of the system will pass through a state where *init_done* holds. (An eventuality property.)
- If a data request is sent to a server, the server will always eventually reply with the data. (A progress property: “always eventually” here means “after and arbitrary long but nevetheless a finite number of time steps”.)

Liveness - Examples (cnt.)

- Both process 0 and process 1 are scheduled infinitely often.
- If both process 0 and process 1 are scheduled infinitely often then the request of process 0 to enter the critical section will always eventually be followed by process 0 entering the critical section. (This is often called model checking under fairness. Namely, if the assumption about fair scheduling holds, then the systems satisfies the required progress property.)
- If process 0 is in the critical section, it will leave the critical section after an unbounded but finite number of time steps.

Liveness

- A practical way of specifying liveness properties is to use the temporal logic LTL (linear temporal logic), or its extension PLTL (linear temporal logic with past).
- In LTL we use operators like:
  - $\mathbf{X}\,\psi_1$ ("next"), the future time correspondent to $\mathbf{Y}\,\psi_1$, and
  - $\psi_1\,\mathbf{U}\,\psi_2$ ("until"), the future time correspondent to $\psi_1\,\mathbf{S}\,\psi_2$.
- The semantics of LTL is outside the scope of this course.

Liveness (cnt.)

- How to specify liveness properties in LTL and how to implement their model checking is covered in the course: T–79.5301 Reactive Systems
  `http://www.tcs.hut.fi/Studies/T-79.5301/`
- Spin has a full blown LTL model checker (as actually most model checkers do these days), so the tool support is available.

Model Based Testing

- Suppose you have verified safety properties of your system implementation G using model checking methods, and you want to implement it as a concrete program P.
- Can we use automated testing to increase our confidence that P satisfies all safety properties proved from the "golden design" model G?
- The answer is yes. The approach presented for doing so is called model based testing (MBT).

Simplified Testing Framework

To keep things simple we add a couple of restrictions needed to keep our intro to MBT short. We also keep the discussion a bit informal.

- Assume G is an LTS with alphabet Σ divided into inputs Σ_I and outputs Σ_O.
- Let both G and P behave in an input-internal-output loop for each test step i as follows:
  1. Wait for an input $a_i \in \Sigma_I$, all inputs are accepted and acted on.
  2. Do some finite sequence of internal τ-moves. (Non-determinism allowed!)
  3. Send an output $b_i \in \Sigma_O$.

Simplified Testing Framework

- Because of the assumptions above, any sequence $a = a_0 a_1 \ldots a_n \in \Sigma_I^*$ is a valid input test sequence for both G and P.
- Now feed the test sequence to P. It produces the output sequence $b = b_0 b_1 \ldots b_n \in \Sigma_O^*$.
- If $a_0 b_0 a_1 b_1 \ldots a_n b_n \notin traces(G)$ the test verdict is fail, otherwise pass.

Test Verdict Computation

- Intuitively, if $a_0 b_0 a_1 b_1 \ldots a_n b_n \notin traces(G)$, then the concrete program P can after some prefix $a_0 b_0 a_1 b_1 \ldots\ldots a_l$ with $l \leq n$ do b_l, and this cannot be matched by any execution of the golden design G.
- However, in this case P might also violate the safety properties proved for G, and therefore we'd better give a fail test verdict.

Test Verdict Computation (cnt.)

- To check whether $a_0 b_0 a_1 b_1 \ldots a_n b_n \notin traces(G)$, we can see $a_0 b_0 a_1 b_1 \ldots a_n b_n$ as an LTS A, and G as the specification LTS, and then check $A \leq_{tr} G$. If $A \leq_{tr} G$ we give test verdict pass, otherwise fail.
- As you may recall, checking $A \leq_{tr} G$ usually involves determinising G.
- Thus if G has $|G|$ states, the determinised version can have exponentially more states, namely $2^{|G|}$.
- By employing the so called on-the-fly determinisation technique, the memory needed to check $A \leq_{tr} G$ can be bounded by the number of states $|G|$.

Model Based Testing

- The first commercial model based testing tools have become available.
  - For example, the testing tools by Conformiq (`http://www.conformiq.com/`) contain automated test generation and execution with MBT techniques.
  - For more on model based testing, see the course: T–79.5304 Formal Conformance Testing `http://www.tcs.hut.fi/Studies/T-79.5304/`

Other models of Concurrency

- Process algebras - An algebraic way of compactly specifying LTSs. Example specifying two synchronizing LTSs: $I = ((a.(\tau.c.0 + b.0)) \;||\; (a.b.0))$, 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

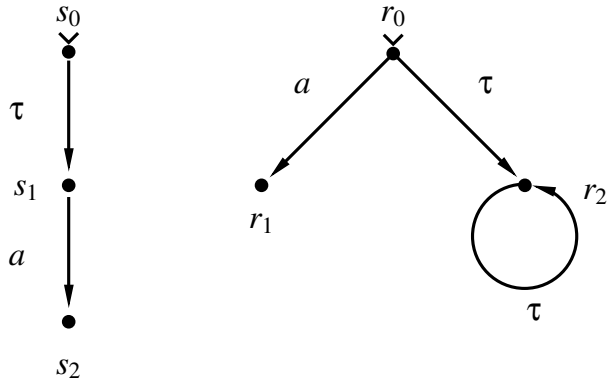
For another perspective into models of concurrency, consider Petri nets. The class we use are called place/transition nets (P/T-nets). A P/T-net is a tuple $N = (P, T, F, W, M_0)$, where

- 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.

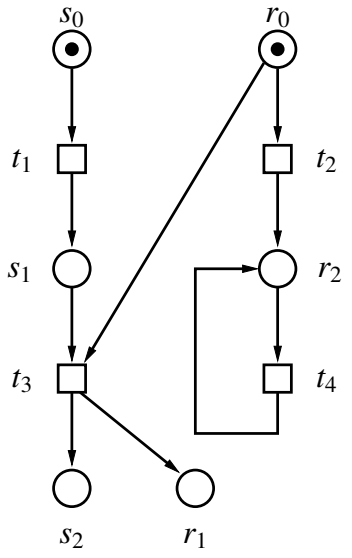
Running Example

Recall the synchronization of LTSs from Lecture 6:

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



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 $t \in \mathit{enabled}(M)$ 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.