1 TEMPORAL LOGICS

- Mere navigation in the reachability graph does not make it easy to reach a conviction about the presence of any complicated properties.
- We need a language to express such properties.
- The language needs to possess algorithms to check said properties.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

Introduction

- Safety properties:
  'Nothing evil will happen.'
  - Deadlock-freedom
  - Mutual exclusion.
  - Data security.
  - In general, the system is guaranteed not to go to a certain state.
- Liveness properties:
  'Something good eventually happens.'
  - Fairness.
  - Guaranteed delivery of messages.
  - Guaranteed service for all service requests.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

Introduction

- Others properties:
  - P occurs infinitely often.
  - After P, Q will occur after n steps/seconds.
  - For each state (i.e. marking) the condition C holds.
- How to model this kind of properties in a common language?
- Shared aspects of the language:
  - *Eventually* in the future.
  - *Always* in the future.
- This means we will have quantification over *time*.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

Introduction

- Structure of time:
  1. Linear?
  2. Branching?
  3. Discrete?
  4. Continuous?
  5. Does current moment belong to the future, too?
- Are we talking about states ot events here?

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

Introduction

- Choices:
  - The structure of time is discrete since the state transitions are discrete, and the system has at most an enumerable number of states.
  - Current moment belongs to the future.
  - It is more difficult to make a choice between linear and branching time.
  - It proves out that it is natural to talk about states instead of events.
  - Two logical systems are obtained:
    - LTL (linear temporal logic): Reachability graph is considered a set of *infinite* transition sequences.
    - CTL (computational tree logic): Reachability graph is considered an *infinite* tree.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

LTL

- Atomic formalae: $p \in \Phi$ if p is the name of a place.
- Formulae $Fma(\Phi)$:
  - Atomic formulae: $p \in \Phi \Rightarrow p \in Fma(\Phi)$
  - Formulae: $A \in Fma(\Phi)$ where $A ::= p \mid \bot \mid A_1 \Rightarrow A_2 \mid \Box A$
- $\Box A$ means *always in the future* A.
- The rest of the connectors can be defined as follows:

$$\begin{array}{rcl} \neg A & \text{on} & A \Rightarrow \bot \\ A_1 \vee A_2 & \text{on} & (\neg A_1) \Rightarrow A_2 \\ A_1 \wedge A_2 & \text{on} & \neg(A_1 \Rightarrow \neg A_2) \\ \Diamond A & \text{on} & \neg \Box \neg A \\ \top & \text{on} & \neg \bot \end{array}$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

LTL

- Examples:
  - Guaranteed delivery of all messages:

$$\Box(Sender(\langle 0, rts\rangle) \Rightarrow \Diamond Sender(\langle 1, rts\rangle)) \quad (1)$$

  - The philosophers 1 and 2 cannot eat at the same time:

$$\Box\neg(E(\langle ph_1\rangle) \wedge E(\langle ph_2\rangle))$$

  - q holds in finitely many states.

$$\Diamond\,\Box\,\neg q$$

  - q holds in infinitely many states:

$$\Box\Diamond q$$

  - Safety property:

$$\Box A$$

  - Liveness property:

$$\Diamond A$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

Semantics

- Semantics:
  - The semantics of LTL formulae is defined with respect to infinite sequences.
  - The pair $F = (S, R)$ is a *frame*, where S is a non-empty (enumerable) set (of states) and $R \subseteq S \times S$ (the reachability relation).
  - The following holds w.r.t. the reachability relation: $\forall s \exists! t : (sRt)$, i.e. every state has only one follower.
  - The reflexive and transitive closure of the relation R is denoted R^*.
  - The Φ-*model* is the 3-tuple $M = (S, R, V)$, where $V : \Phi \to 2^S$.
  - $V(p)$ is the set of the states where the atomic formula p holds.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

SEMANTICS

- The formula A holds in the model M at the point s, which is denoted

$$M \models_s A,$$

defined inductively as follows:

$$\begin{aligned}
M \models_s p &\quad \text{iff} \quad s \in V(p) \\
M \not\models_s \bot & \\
M \models_s A_1 \Rightarrow A_2 &\quad \text{iff} \quad M \models_s A_1 \text{ implies } M \models_s A_2 \\
M \models_s \Box A &\quad \text{iff} \quad \forall t \in S, sR^*t \text{ implies } M \models_t A
\end{aligned}$$

- The formula A holds in the model M: $M \models A$ iff $M \models_{s_0} A$.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

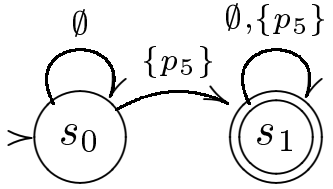
Semantics

- Infinite transition sequences starting from the the initial marking of the reachability graph generate LTL models.
- The states of teh model are markings, and $m_i R m_k$ iff $m_i[t > m_k$.
- The models generated by the reachability graph SG_N will be denoted $\mathcal{M}(SG_N)$.
- The formula A will hold in the net N iff $\forall M \in \mathcal{M}(SG_N) M \models A$.
- Since a reachability graph with no loops generates a superenumerable number of LTL models we conclude that we have not yet found a feasible approach to checking the presence of a property.
- Below we outline a method to check the validity of a formula (Cf. reference (2)).

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

Model checking

- As the number of states is enumerable, the model can be interpreted as the relation $\mathbb{N} \times \Phi \to \{\mathbf{T}, \mathbf{F}\}$ or the relation $\mathbb{N} \to (\Phi \to \{\mathbf{T}, \mathbf{F}\})$.
- The relation $\mathbb{N} \to (\Phi \to \{\mathbf{T}, \mathbf{F}\})$ can be seen as an infinite sequence of symbols with the alphabet $2^{\Phi} \cong (\Phi \to \{\mathbf{T}, \mathbf{F}\})$.
- Hence, an LTL formula A defines the language of infinite sequences: The set of the models for which A holds.
- E.g., $\Diamond p_5$ holds for all sequences, where p_5 sometimes holds. In other words, the formula specifies a language whose alphabet is $\Sigma = \{\emptyset, \{p_5\}\}$ and whose words contain p_5 at least once.
- The words of an alphabet can be recognized with an automaton:



where s_0 is the *initial state* and s_1 is the *accepting state*.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

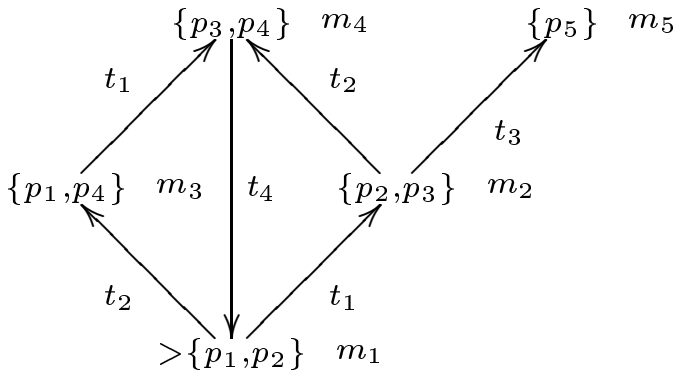
Model checking

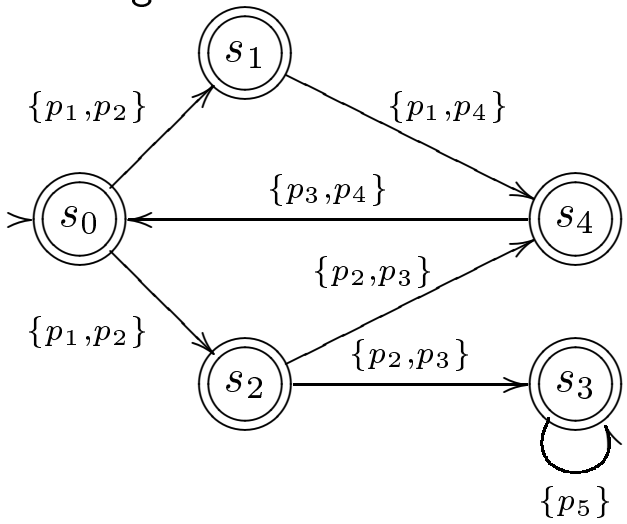
- Since words are infinite sequences the accepting condition of the automaton is as follows: The automaton Q accepts the inifinite word $w = a_0, a_1, \ldots$ iff there exists an infinite sequence $\sigma = s_0, s_1, \ldots$ such that s_0 is the initial state of the automaton, and any s_i is obtained from the state s_{i-1} and the symbol a_{i-1} with the help of a state transition function, and there exists an inifinite set of integers $i \in \mathbb{N}$ such that $s_i = s$, where s is the accepting state.
- This kind of an automaton is called a *Büchi automaton*.
- For any LTL formula A it is possible to construct a Büchi automaton $B(A)$ which only accepts those infinite words for which the formula holds.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

Model checking

- Correspondingly, a reachability graph SG can be considered a Büchi automaton $B(SG)$, where each state is an eaccpting state and the name of the state transition is the set of atomic formulas evaluating to true in the initial state.





- The answer to the question "**Does the formula A hold in the net N?**" is obtained by computing the intersection (product) automaton of the nets $B(\neg A)$ and $B(SG_N)$. If the insection automaton only accepts the empty language then the formula A holds.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

PROD

- In PROD the verification of LTL formulae is done by performing model checking with an on-the-fly tester.
- While generating the reachability graph PROD checks the validity of a formula.
- If the formula does not hold, a counter example is obtained. If the formula is valid no output is given.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

EXTENSIONS

- It is often useful to express the operators 'before' or 'until'.
- The language of logics can be extended with the connective *until*: $M \models_s A\mathcal{U}B$ iff there exists t, such that sR^*t and $M \models_t B$ and $M \models_u A$ for all u such that sR^*u and uR^*t.
- The requirement that reply v does not arrive before the question k can now be expressed as follows:

$$\Diamond v \Rightarrow (\neg v)\mathcal{U}k$$

- The state where q holds must be preceded by a state where p holds:

$$\neg q\mathcal{U}p \vee \Box p$$

- The connective Until is an extension to the expressive power of the logics because of the following:

$$\begin{aligned} \Diamond A &\equiv \top\mathcal{U}A \\ \Box A &\equiv \neg(\top\mathcal{U}\neg A) \end{aligned}$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

EXTENSIONS

- Often the operator *next* expressing the 'next state' is defined:

$$M \models_i \bigcirc A \text{ iff } M \models_j A \text{ where} iRj.$$

- The following holds for the operator $\bigcirc$:

$$\begin{aligned} \Box A &\equiv (A \wedge \bigcirc \Box A) \\ \Diamond A &\equiv (A \vee \bigcirc \Diamond A) \\ A \mathcal{U} B &\equiv (B \vee (A \wedge \bigcirc (A \mathcal{U} B)) \end{aligned}$$

- Hence, we will have also the following:

$$\begin{aligned} \Diamond \Box \neg A &\equiv \Box \neg A \vee \bigcirc \Diamond \Box \neg A \\ &\equiv (\neg A \wedge \bigcirc \Box \neg A) \vee \bigcirc \Diamond \Box \neg A, \end{aligned}$$

tai

$$\begin{aligned} \Box \Diamond A &\equiv \Diamond A \wedge \bigcirc \Box \Diamond A \\ &\equiv ((A \vee \bigcirc \Diamond A) \wedge \bigcirc \Box \Diamond A. \end{aligned}$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

About the expressive power

- Consider the property p which holds in every odd state (the states are here labelled with integers):
- Example:

		0	1	2	3	4	5	6	7	8	9	10 ...
s_1	=	p	$\neg p$	p	$\neg p$	p	$\neg p$	p	$\neg p$	p	$\neg p$	$p \ldots$
s_2	=	p	p	p	$\neg p$	p	$\neg p$	p	$\neg p$	p	$\neg p$	$p \ldots$

but not

		0	1	2	3	4	5	6	7	8	9	10 ...
s_3	=	$\neg p$	$\neg p$	p	$\neg p$	p	$\neg p$	p	$\neg p$	p	$\neg p$	$p \ldots$
s_4	=	p	p	p	$\neg p$	$\neg p$	$\neg p$	p	$\neg p$	p	$\neg p$	$p \ldots$

- This kind of properties cannot be modelled with the given connectives.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

About the expressive power

- Example: The following formulae

$$p \wedge \Box(p \Rightarrow \bigcirc\neg p) \wedge \Box(\neg p \Rightarrow \bigcirc p)$$

or

$$p \wedge \Box(p \Rightarrow \bigcirc \bigcirc p)$$

do not hold in the above models.

- In general, the following statement is true:

  Any formula, comprising atomic formulae and at most n instances of operator $\bigcirc$, evalutate to the same Boolean value for all sequences

$$\sigma_k = p^k(\neg p)p^\omega,$$

  where $k > n$.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

References

(1) Z. Manna & A. Pnueli: *The Temporal Logic of Reactive and Concurrent Systems: Specification*

Use of temporal logics based proofs in the specification and verification of systems.

(2) P. Wolper: *Temporal Logic*. Chapter 4 in A. Thayse, *From Modal Logic to Deductive Databases*.

Model checking algorithms.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

Fairness

Consider the mutex algorithm given below with the help of temporal logics:

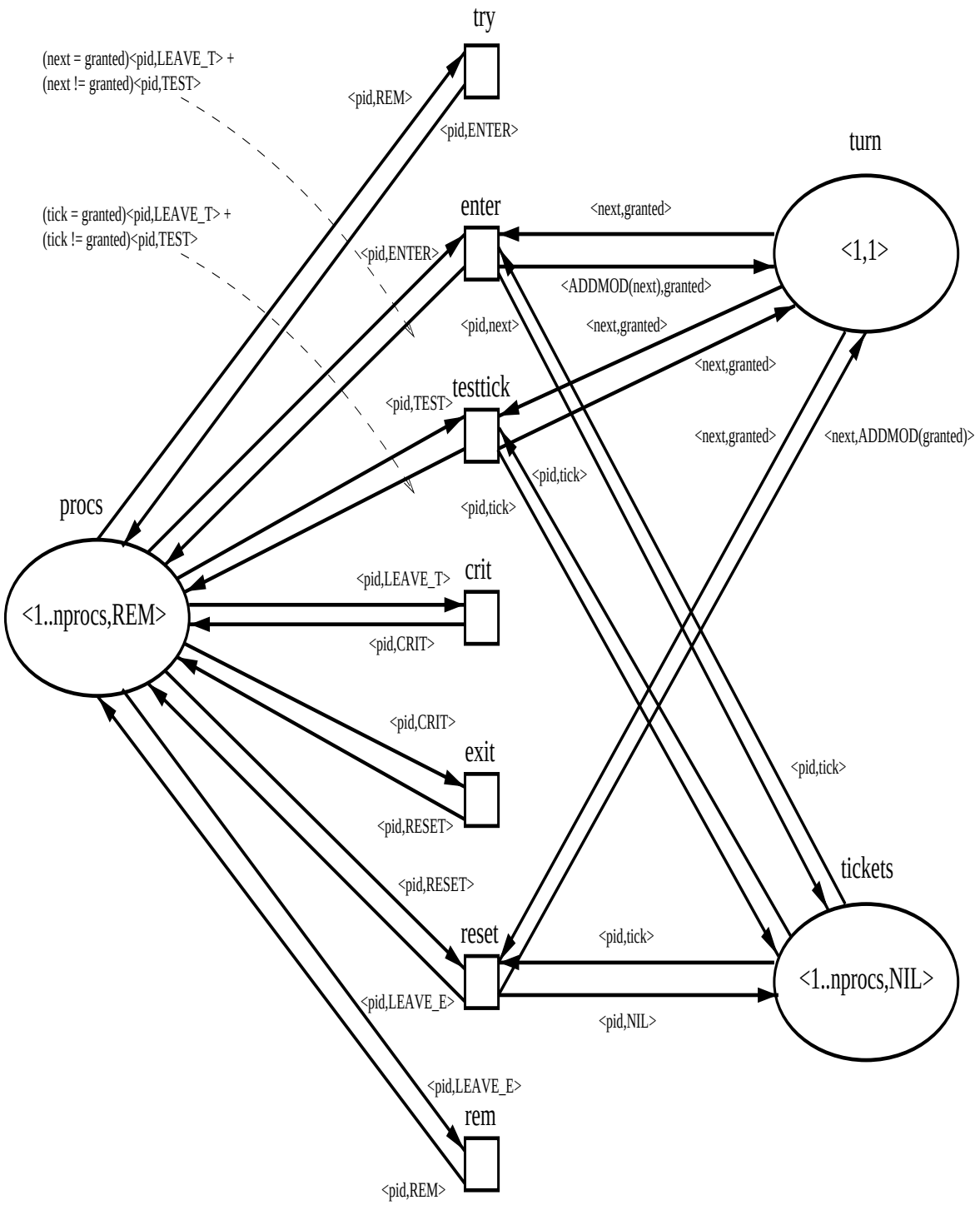
TicketME algorithm (informal):

A shared variable holds a pair $\langle next, granted \rangle$ of values in $\{1, \ldots, n\}$, initially $\langle 1, 1 \rangle$. The *next* component represents the next "ticket" into the critical section, while the *granted* component represents the last "ticket" that has been granted permission to enter the critical section. When a process enters the trying section, it "takes a ticket", that is, it copies and increments the *next* component modulo n. When a process's ticket is equal to the *granted* component, it goes to the critical region. When process exits the critical section, it increments the *granted* component modulo n.

Paraphrased from: *Nancy A. Lynch: Distributed Algorithms, 1996, (ISBN: 1-55860-348-4).*

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

Ticket Mutex Pr/T net



©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

Verification of properties

Let Ticket-ME be a model of the algorithm for two processes N_{T_2}. The following properties can be verified:

- The processes are mutually exclusive:

$$N_{T_2} \models \Box(\neg(procs(\langle 1, CRIT\rangle) \wedge procs(\langle 2, CRIT\rangle)))$$

- There exists an execution where a process is able to enter the critical section (verification with a counter argument):

$$N_{T_2} \not\models \Box(\neg(procs(\langle 1, CRIT\rangle)))$$

$$N_{T_2} \not\models \Box(\neg(procs(\langle 2, CRIT\rangle)))$$

- If the process wishes to enter the critical section, it will eventually succeed in it (this is something we cannot verify in the model):

$$N_{T_2} \models \Box(procs(\langle 1, ENTER\rangle) \Rightarrow \Diamond procs(\langle 1, CRIT\rangle))$$

$$N_{T_2} \models \Box(procs(\langle 2, ENTER\rangle) \Rightarrow \Diamond procs(\langle 2, CRIT\rangle))$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

Fairness

Why cannot a process succeed in entering the critical section whenever it wants to do so?

The reason for this is that the model ignores *fairness* between the processes. In the model it possible to find executions where one of the processes does not do any steps after a certain number of steps.

Example:

Let us consider an execution where the fired transitions (indexed with the number of the executing process) are as follows:

$$try_1, try_2, enter_1, enter_2, crit_1, testtick_2, testtick_2, \ldots$$

In this execution, the process 1 enters the critical section but never leaves it. The process 2 is in a state where it is continuously checking for permission to enter the critical section with the transition $testtick_2$ while the process 1 is forever unable to leave the critical section with the transition $exit_1$.

Because of fair scheduling by the operating system, this kind of situations do not necessarily appear in 'real systems' so they are often removed from the model.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

Fairness

So now we need to define which executions are fair.

Fairness depends on the underlying model and it is beyond the scope of this course to go deeper into different definitions of fairness.

Since fairness is a property of executions we can employ LTL temporal logics to define fairness.

This means that instead of verifying the formula given below:

$$N \models property$$

we verify the following formula:

$$N \models fairness \Rightarrow property,$$

where $property$ is the LTL formula to be verified and $fairness$ is the *fairness assumption*.

The fairness assumption is an LTL formula which evaluates to true for all fair traces and false to to all unfair traces. All unfair executions are distinguished.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

Fairness

In our example we will use the following definition for fairness:

Fair runs are the infinite runs in which both processes fire infinitely many transitions.

In other words, we forbid executions where one of the processes does not execute at all after a certain time point.

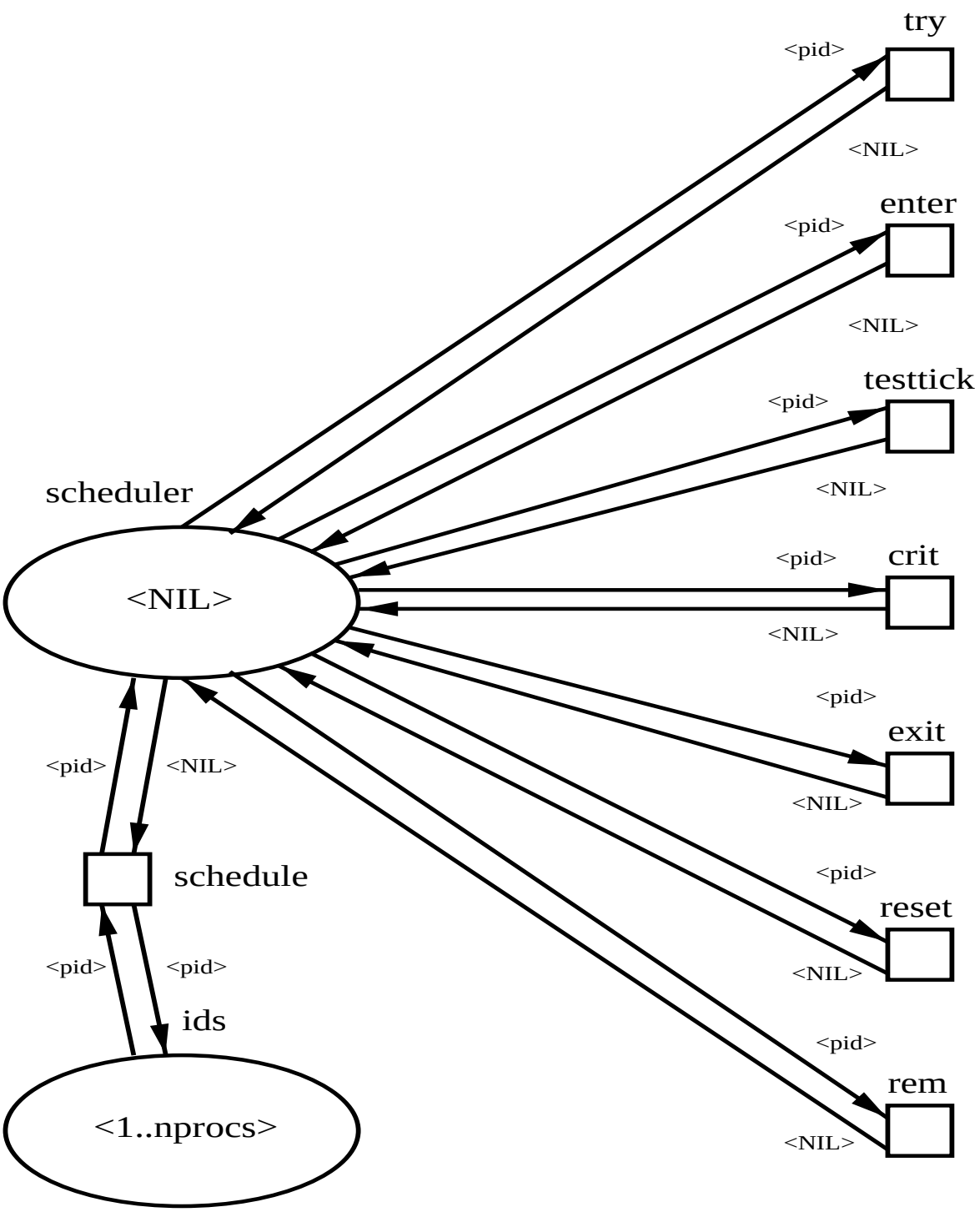
The logics we employ does not say anything about transitions so we have to modify the model to be able to do the same thing with the states of the model.

Process scheduling is added to the model by inserting two new places, one transition, and arcs from all existing transitions. Let the new model be N'_{T_2}. (See the next page.)

In the executions of the system every other transition now is the $schedule$ transitio.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

Inserting the scheduler to the Ticket-ME model



©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000

Fairness

The fairness assumption we wanted is easy to describe with the following LTL formula.

$$fairness \equiv \Box\Diamond scheduler(\langle 1 \rangle) \wedge \Box\Diamond scheduler(\langle 2 \rangle)$$

The formula accepts only such runs where both processes are scheduled infinitely.

Now the wanted liveness properties can be verified:

$$N'_{T_2} \models (\Box\Diamond scheduler(\langle 1 \rangle) \wedge \Box\Diamond scheduler(\langle 2 \rangle)) \Rightarrow (\Box(procs(\langle 1, ENTER \rangle) \Rightarrow \Diamond procs(\langle 1, CRIT \rangle)))$$

$$N'_{T_2} \models (\Box\Diamond scheduler(\langle 1 \rangle) \wedge \Box\Diamond scheduler(\langle 2 \rangle)) \Rightarrow (\Box(procs(\langle 2, ENTER \rangle) \Rightarrow \Diamond procs(\langle 2, CRIT \rangle)))$$

Fairness is not needed when checking the safety properties. (All executions which violate a safety property can be considered as terminating [abortion at violation], and the execution can only be fair when it is infinite.)

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997-2000