Petri Nets

1. Basic Petri Net Model
2. Properties and Analysis of Petri Nets
3. Extended Petri Net Models

Petri Nets
☞ Systems are specified as a directed bipartite graph.
   The two kinds of nodes in the graph:
   1. Places: they hold the distributed state of the system expressed by the presence/absence of tokens in the places.
   2. Transitions: denote the activity in the system
   ☞ The state of the system: captured by the marking of the places (number of tokens in each place)

Petri Net Example
A producer and a consumer process communicating through a buffer:
Petri Net Example (cont'd)

Continuation from previous slide:

Notice that the buffer is considered to be infinite (tokens can accumulate in place $B$).

Petri Net Example (cont'd)

Here we have the same model as on the previous slides, but with limited buffer. The buffer size is three (number of initial tokens in $B'$).

- Nr. of tokens in $B'$: how many free slots are available in the buffer;
- Nr. of tokens in $B$: how many messages (tokens) are in the buffer.

Total number of tokens in $B$ and $B'$ is constant.

Some Features and Applications of Petri Nets

- Intuitive.
  Easy to express concurrency, synchronisation, nondeterminism.

  Nondeterminism is an important difference between Petri nets and dataflow!

- As an uninterpreted model, Petri Nets can be used for several, very different classes of problems.
  - Uninterpreted model: nothing has to be specified related to the particular activities associated to the transitions.
Some Features and Applications of Petri Nets (cont’d)

- Petri Nets have been intensively used for modeling and analysis of industrial production systems, information systems, but also
  - Computer architectures
  - Operating systems
  - Concurrent programs
  - Distributed systems
  - Hardware systems

Properties and Analysis of Petri Nets

☞ Several properties of the system can be analysed using Petri nets:

- Boundedness: the number of tokens in a certain place does not exceed a given limit.
  - If this limit is 1, the property is sometimes called safeness.
  - You can check that available resources are not exceeded.

Properties and Analysis of Petri Nets (cont’d)

- Liveness:
  - A transition $t$ is called live if for every possible marking there exists a chance for that transition to become enabled.
  - The whole net is live, if all its transitions are live.
  - Liveness is important in order to check that a system is not deadlocked.

- Reachability: given a current marking $M$ of the net, and another marking $M'$, does there exist a sequence of transitions by which $M'$ can be obtained?
  - You can check that a certain desired state (marking) is reached.
  - You can check that a certain undesired state is never reached.

Mathematical tools are available for analysis of Petri Nets.

☞ The properties discussed above can be formally verified.

☞ Petri nets (like dataflow systems) are asynchronous concurrent.

- Events can happen at any time.
- There exists a partial order of events:
Extended Petri Net Models

Basic Petri Net models have a limited expressive power.

☞ Timed Petri Nets
- Transitions have associated times (time intervals)
- Tokens are carrying time stamps.
  - With timed Petri nets we can model the timing aspects

☞ Coloured Petri Nets
- Tokens have associated values
- Transitions have associated functions
  - Coloured Petri Nets are similar to dataflow models (but also capture nondeterminism).
Summary

- Petri Nets are a mixture of dataflow and state-based model. Places hold the distributed state of the system (represented by the marking); transitions denote the activity of the system.
- Petri nets elegantly capture concurrency, synchronisation, and nondeterminism.
- A large class of problems can be solved using Petri Net modelling; system properties like boundedness, liveness, and reachability can be formally analysed.
- Petri nets, like dataflow, are asynchronous, concurrent models. Events can happen at any time; there exists a partial order of events.

Summary (cont’d)

- Basic Petri Nets are limited in their expressive power - in timed Petri Nets an explicit notion of time has been introduced;
  - in coloured Petri nets tokens have associated values.
- Formal reasoning about extended Petri Net models is very difficult because of complexity issues. Simulation of the models is often used for system validation.

Discrete Event Models

1. What Is a Discrete Event Model?
2. Discrete Event Simulation
3. Efficiency of Discrete Event Simulation
4. Potential Ambiguities in Discrete Event Simulation

Discrete Event Models

- The system is a collection of processes that respond to events.
  - Each event carries a time-stamp indicating the time at which the event occurs.
  - Time-stamps are totally ordered.
  - A Discrete Event (DE) simulator maintains a global event queue sorted by the time-stamps. The simulator also keeps a single global time.
### Discrete Event Simulator

- The discrete event model has been mainly used for system simulation.
  - Several languages have been developed for system modeling based on the discrete event model. Most well known:
    - VHDL, Verilog (both used for hardware modeling), SystemC
  - Efficient way to simulate distributed systems.
    - In general, efficient for large systems with autonomous components, with relatively large idle times. Systems with non-regular, possibly long times between different activities.

Why is this the case?

Because DE simulation will only consider the particular times when a change in the system (an event) occurs. This is opposed to, for example, cycle-based models, where all clock-ticks are considered.

### Discrete Event Simulation (cont'd)

- Efficiency related problems:
  - Keeping the sorted event-queue is time-consuming.
  - As the activity of the simulated system increases (a lot of events at a very high number of time moments have to be considered) the overhead becomes high ⇒ simulation is slow.

- Event driven models are primarily employed for simulation.
  - Functional verification
  - Performance evaluation

- Both synthesis and formal verification are very difficult (complex) with DE models:
  - The classical trade-off between expressive power and the possibility of formal reasoning and efficient synthesis.
A Problem with Discrete Event Simulation

Have a look at the diagram in slide 22: In the third step we have “Activate and run all processes which are sensible to the updated events.”

- If we have a set of several such processes ready to run, in which order should they be activated?

The only acceptable answer: The order should not matter; whatever the order, the simulation result has to be the same! Otherwise, different simulators might provide different simulation outputs for the same model!

A Problem with Discrete Event Simulation (cont’d)

Process A

wait on S1

S1<=5 after 5

wait ...

Process B

wait on S1

S2<=18 after 10

At time 10 Process A executes

S1 5

S2 18

At time 15

S1 5

S2 18

Processes B and C are ready to execute

Scenario 1

S1<=5 after 5

wait ...

Process A

wait on S1

S2<=18 after 10

Process B

wait on S1, S2

Process C

wait on S1, S2

• B executes, S1=5, S2=0, t=15

• C executes, S1=5, S2=0, t=15

• C executes, S1=5, S2=18, t=25

Scenario 2

S1<=5 after 5

wait ...

Process A

wait on S1

S2<=18 after 10

Process B

wait on S1, S2

Process C

wait on S1, S2

• C executes, S1=5, S2=0, t=15

• B executes, S1=5, S2=0, t=15

• C executes, S1=5, S2=18, t=25

The two scenarios are identical!!! How fine!!!
A Problem with Discrete Event Simulation (cont’d)

We have changed the example! No after clause!

Process A
wait on S1
S2<=18

Process B
wait on S1, S2

Process C

Scenario 1
B first, then C

• B executes, S1=5, S2=0, t=15

At time 15 Process A executes

15
S1
0
S2
18

Processes B and C are ready to execute

Scenario 2
C first, then B

• C executes, S1=5, S2=0, t=15

• B executes, S1=5, S2=0, t=15

The two scenarios are different!!! How bad!!!

We get into this problem due to the zero delay assignment to S2 in Process B! The new event is queued at the current simulation time (15), which leads to the potential ambiguity.

Such an ambiguity creates problems. For example, different simulators will produce different output for the same model with identical inputs.

A possible solution (used, for example, in VHDL and SystemC): A zero delay event will be registered at a time which is infinitesimally delayed relative to the current time.

A “delta delay” will be introduced on the event ⇒ the new event will be consumed in the following simulation cycle and not the current one.
A Problem with Discrete Event Simulation (cont'd)

Scenario 1
B first, then C
• C executes, S1=5, S2=0, t=15
• C executes, S1=5, S2=0, t=15
At time 15(+) \( \delta \)

Scenario 2
C first, then B
• C executes, S1=5, S2=0, t=15
• B executes, S1=5, S2=0, t=15
At time 15(+) \( \delta \)

The two scenarios are identical!!! How fine!!!