Petri Nets

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

☞ 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 Nets (cont’d)

☞ The dynamic evolution of the system: determined by the firing process of transitions.
   - A transition may fire whenever all its predecessor places are marked.
   - If a transition fires it removes a token from each predecessor place and adds a token to each successor place.

Petri Net Example

A producer and a consumer process communicating through a buffer:
Continuation from previous slide:

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

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 $B'$)

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.

- **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!).

Extended Petri Net Models (cont’d)

• Extended Petri Nets have a larger expressive power than classical Petri Nets.
  - Analysis is more complex; the formal analysis of properties can take unacceptably large amounts of time (memory).
  - Simulation of the Petri Net can be used in order to verify the system and to estimate performance
**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

- 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)
  - 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.
Simultaneous events resulted from 0 delay components:

Alternative 1: C is in vok ed first and then B

Alternative 2: B is in vok ed first and then C

C is first reacting to the event from A and then to that of B

C is reacting to both events

A zero delay process will introduce a "delta delay" on the event. The input and output event are at the same "real time", but a partial order is introduced.

This also solves the problem of zero-delay feedback loops. (delay on all three components is 0, see also FSMs, Fö6 - slide 18)

Not only non-determinism is a problem here, but the output on C can, in general, not be determined.

Some systems reject such models.

If delta delays are introduced (like with VHDL) a deterministic simulation is possible.
Summary

- Discrete Event is a very powerful modelling technique, in terms of expressive power. Models are concurrent and asynchronous. The timing model is also very general. Delays on computation or communication can be arbitrary.

- We pay for the large expressive power by the reduced potential of formal reasoning and efficient synthesis. Discrete event models are mainly used for simulation.

- Simulation is based on maintaining a unique, sorted event queue. This can create problems with simulation efficiency for large system models.

- In order to avoid problems with zero delay computations, “delta delays” are used (e.g. in VHDL).