Petri nets

- **Petri nets** are mathematical models for describing systems with **concurrency** and **resource sharing**

- they facilitate many **automatic analyses** of interest for concurrent systems

- rich, intuitive **graphical notation** for choice, concurrent execution, interaction with the environment, ...
Petri nets - the origins

• proposed by Carl Adam Petri in his famous thesis *Kommunikation mit Automaten* (1962)

• aimed for a system architecture that could be expanded indefinitely
  => no central components
  => in particular, no central, synchronising clock
  => actions with *locally confined* causes/effects

• original presentation omitted the graphical representation
Next on the agenda

1. modelling concepts: *cookies for everyone!*

2. synchronisation problems as Petri nets

3. Petri net analyses

4. true concurrency semantics; unfoldings
Let’s design a cookie vending machine

coin slot

compartment
Let’s design a cookie vending machine

*coin slot*

*compartment*
Let’s design a cookie vending machine

coin slot  compartment
Let’s design a cookie vending machine
**Terminology**

- **Place**
- **Tokens**
- **Transition (with precondition)**
- **Marking (distribution of tokens)**
Let's design a cookie vending machine

- **coin slot**
- **compartment**
Let’s design a cookie vending machine

transition $t$ is \textit{enabled} it can \textit{occur} and change the \textit{marking}
Let’s design a cookie vending machine

transition $t$ is \textit{enabled}

it can \textit{occur} and change the \textit{marking}
Let’s look inside

cash box?
finitely many cookies?
Let’s look inside

coin slot → a → signal → b → compartment

storage

cash box
Let’s look inside

coin slot → storage

signal

cash box

compartment
Let’s look inside

- **coin slot**
- **signal**
- **cash box**
- **storage**
- **compartment**
Let’s look inside

- coin slot
- signal
- cash box
- storage
- compartment
Let’s open it up to the world
Let’s open it up to the world
Let’s open it up to the world

- $\epsilon$ denotes a transition that once enabled, need not actually occur
- We assume that other enabled transitions occur eventually
The ultimate cookie machine (design)
The ultimate cookie machine (design)

- **coin slot**
- **storage**
- **counter**
- **cash box**
- **compartment**

**Actions:**
- **insert**
- **return coin**
- **signal**
- **take**
The ultimate cookie machine (design)
The ultimate cookie machine (design)

conflict! nondeterminism!
The ultimate cookie machine (design)
The ultimate cookie machine (design)

- **Coin slot**: Insert coin
- **Counter**: Return coin
- **Cash box**: Store coins
- **Storage**: Cookie compartment
- **Signal**: Take cookie

Flowchart:
- Start at insert
- Move to a (counter)
- Move to b (signal)
- Move to storage (compartment)
- End at take
The ultimate cookie machine (design)

exercise: strengthen the design such that the coin slot and signal places store at most one token each
if we are interested in only control flow, we can use a special case - elementary Petri nets - where all tokens are simply black dots

assume all edges to be labelled by: “●“

henceforth, we assume all Petri nets to be elementary
Elementary cookie vending machine

- **Coin slot**
- **Insert**
- **Return coin**
- **Signal**
- **Cash box**
- **Storage**
- **Compartment**
- **Take**
Petri nets: definition

• an (elementary) Petri net consists of a net structure:
  \[ N = (P, T, F) \]
  with finite sets \( P \) and \( T \) of places and transitions, \( F \) an edge relation \( F \subseteq (P \times T) \cup (T \times P) \) and an initial marking \( M_0 : P \to N \)

• transitions marked with \( \epsilon \) are cold

• markings have the form \( M : P \to N \); each place \( p \) holds \( M(p) \) tokens
Petri nets: definition

• the **preset** of a transition $t$ is the set of places $p$ connected by edges from $p$ to $t$ (postset defined analogously)

• a transition is **enabled** if $M(p) \geq 1$ for all places $p$ in the preset

• an enabled transition can **occur**, removing a token from each place in the preset and adding one to each place in the postset
Next on the agenda

1. modelling concepts: *cookies for everyone!*

2. synchronisation problems as Petri nets

3. Petri net analyses

4. true concurrency semantics; unfoldings
Producer-consumer problem

Producers

Buffer

Consumers

store \( (\text{buffer}, \text{int}) \)

consume \( (\text{buffer}) \)

243
46
71
97
Producer-consumer problem

wait

produce

consume

wait

34
Producer-consumer problem
Producer-consumer problem
Producer-consumer problem

- Produce
- Buffer space
- Buffer count
- Consume
- Wait

Diagram showing the relationships between produce, buffer space, buffer count, consume, and wait.
Producer-consumer problem
Producer-consumer problem

- **Produce**
- **Buffer space**
- **Buffer count**
- **Consume**
- **Wait**
Producer-consumer problem
Mutual exclusion
Mutual exclusion

\[ \text{waiting}_1 \quad \text{local}_1 \quad \text{CR}_1 \quad \varepsilon \quad \text{semaphore} \quad \text{CR}_2 \quad \varepsilon \quad \text{local}_2 \quad \text{waiting}_2 \]
Next on the agenda

1. modelling concepts: *cookies for everyone!* ✓

2. synchronisation problems as Petri nets ✓

3. Petri net analyses

4. true concurrency semantics; unfoldings
Modelling power vs. analysability

• many properties of interest for concurrent systems can be automatically determined for Petri nets
  => but can be very expensive in the general case

• properties include:
  => $k$-boundedness (i.e. no place ever has more than $k$ tokens)
  => liveness
  => reachability
Reachability problem

• the problem to decide whether some marking $M$ can be derived from the initial marking

• starting point: construct a reachability graph from the initial marking
  => i.e. a transition system completely describing its behaviour
  => nodes denote markings
  => edges denote occurrences

• (more sophistication is needed when reachability graphs are not finite)
Reachability graph for our semaphore

Express marking $M$ as a vector:

$( M(wait_1) M(CR_1) M(loc_1) M(sem) M(wait_2) M(CR_2) M(loc_2) )$

i.e. $(0 0 1 1 0 0 1)$
Reachability graph for our semaphore

- Prove that \((0 1 0 0 0 1 0)\) is unreachable
- Prove that \(M(CR_1) + M(CR_2) + M(sem) = 1\)
Reachability graph for our semaphore
Deciding reachability is expensive

- reachability is an important analysis

- decidable, but **expensive** in the general case
  => EXPSPACE-hard
  => reachability graph not always finite

- part II of Reisig (2013) treats the problem with more sophistication than we have
Next on the agenda

1. modelling concepts: *cookies for everyone!*  

2. synchronisation problems as Petri nets  

3. Petri net analyses  

4. true concurrency semantics; unfoldings
The problem of interleaving semantics

• consider the following Petri net:

• its reachability graph contains $2^n$ states
  => state explosion problem
  => due to interleaving of occurrences
  => unnecessary: ordering of occurrences here immaterial!
Interleaving vs. true concurrency semantics

• an **interleaving** semantics imposes a **total ordering** on sequences of occurrences
  => completely described by a **reachability graph**
  => nodes denote markings; edges denote occurrences
  => state explosion!

• a **true concurrency** semantics instead models time as a **partial order**
  => two or more occurrences can happen **simultaneously**
  => completely described by a so-called **unfolding**
Unfoldings are compact representations of concurrency

- an unfolding of a Petri net $N$ is a Petri net that is more “tree like” - but represents the same behaviour

- idea: analyse the unfolding of a Petri net itself, rather than an underlying transition system (as in the interleaving semantics)
Example: an unfolding
Example: an unfolding
Example: an unfolding
Example: an unfolding
Example: an unfolding
Example: an unfolding
Example: an unfolding
Example: an unfolding
Constructing an unfolding

• assumption: Petri nets are 1-bounded
  => possible to generalise to other Petri net variants

• steps to construct an unfolding $N'$ from a Petri net $N$:

  (1) initialise $N'$ with the places in $N$ containing tokens in the initial marking
  (2) if a reachable* marking in $N'$ enables a transition $t$ in $N$, then disjointly add $t$ to $N'$ and:
      => link it to the corresponding preset
      => disjointly add the postset of $t$
  (3) iterate step 2

*checking reachability is far easier for this special net class
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Returning to our small example

• construct an unfolding of the following Petri net:
Returning to our small example

• construct an unfolding of the following Petri net:

the unfolding is just the Petri net itself!

=> size $O(n)$

=> whereas interleaving yields $2^n$ reachable states
Petri net analysis using unfoldings

• suppose we want to know if some transition \( t \) in a Petri net \( N \) can occur (i.e. a liveness property)

• compute an answer by exploring the unfolding of \( N \) until either:
  => a transition labelled \( t \) is found; or
  => it can be concluded that no such transition occurs

• important to note that only a finite prefix of the unfolding is explored
  => Esparza & Heljanko (2008) cover this important part (that we omit)
Next on the agenda

1. modelling concepts: *cookies for everyone!* ✓
2. synchronisation problems as Petri nets ✓
3. Petri net analyses ✓
4. true concurrency semantics; unfoldings ✓
Main sources for this lecture

• Understanding Petri Nets (2013)
  => by Wolfgang Reisig
  => chapters 1-3

• Unfoldings (2008)
  => by Javier Esparza & Keijo Heljanko
  => chapters 1-3

• both available online (see the course webpage)
Summary

• **Petri nets** facilitate a graphical, intuitive means of modelling concurrent and distributed systems

• **Automatic analyses** exist for reachability, boundedness, liveness, ... but are **expensive** in the general case

• **Unfoldings** (based on true concurrency) may give a more compact representation of concurrency than **reachability graphs** (based on interleaving)