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
Today’s 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
Let’s design a cookie vending machine

coin slot  compartment
**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 enabled
it can occur and change the marking
Let’s design a cookie vending machine

transition $t$ is enabled
it can occur and change the marking

coin slot    compartment
Let's design a cookie vending machine

- **coin slot**
- **compartment**

**transition t is enabled**
- it can occur and change the marking

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

- Coin slot
- Signal
- Cash box
- Storage
- Compartment
Let’s look inside

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

coin slot

storage

signal

cash box

compartment
Let’s look inside

coin slot → signal → cash box → compartment

storage

1
Let’s open it up to the world
Let’s open it up to the world

\[
\begin{align*}
\varepsilon & \xrightarrow{1} \text{coin slot} \\
& \xrightarrow{1} a \quad \text{signal} \\
& \xrightarrow{1} b \quad \text{compartment} \\
& \xrightarrow{1} \varepsilon \quad \text{take}
\end{align*}
\]

insert

storage

cash box
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

- **The ultimate cookie machine**
- **insert**
- **coin slot**
- **signal**
- **counter**
- **cash box**
- **storage**
- **compartment**
- **return coin**
- **take**

Diagram:
- **ε**
- **a**
- **b**
- **storage**
- **compartment**
The ultimate cookie machine

- **The ultimate cookie machine**
  - **coin slot**
    - **insert coin**
    - **return coin**
  - **storage**
  - **cash box**
  - **counter**
  - **signal**
  - **compartment**
  - **take**
The ultimate cookie machine
The ultimate cookie machine

conflict! nondeterminism!
The ultimate cookie machine

- **coin slot**
- **storage**
- **insert**
- **return coin**
- **counter**
- **cash box**
- **signal**
- **compartment**
- **take**
The ultimate cookie machine

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

Transitions:
- **insert**
- **return coin**
- **signal**
- **take**

States:
The ultimate cookie machine

exercise: strengthen the design such that the coin slot and signal places store at most one token each
Elementary Petri nets

• 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

counter

cash box

storage

signal

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 \( M_0: P \to \mathbb{N} \)

• markings have the form \( M: P \to \mathbb{N} \); 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

store (buffer, int)

243
46
71
97

consume (buffer)

Producers

Buffer

Consumers
Producer-consumer problem

wait

produce

consume

wait
Producer-consumer problem

- Produce
- Wait
- Buffer space
- Buffer count
- Consume
- Wait
Producer-consumer problem

![Diagram of producer-consumer problem](image-url)
Producer-consumer problem

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

Diagram: A diagram illustrating the producer-consumer problem with states for produce, consume, buffer space, buffer count, and wait, connected with arrows representing the flow between states.
Producer-consumer problem

wait

buffer space

buffer count

produce

consume

wait
Producer-consumer problem

wait
produce
buffer space
buffer count
consume
wait
Producer-consumer problem

- **Produce**
- **Buffer space**
- **Buffer count**
- **Consume**
- **Wait**
Mutual exclusion

$\epsilon \rightarrow waiting_1 \rightarrow CR_1 \rightarrow local_1 \rightarrow \epsilon$

$\epsilon \rightarrow waiting_2 \rightarrow CR_2 \rightarrow local_2 \rightarrow \epsilon$
Mutual exclusion
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

• several tools are available
  => http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html
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:
$$( M(\text{wait}_1) \ M(\text{CR}_1) \ M(\text{loc}_1) \ M(\text{sem}) \ M(\text{wait}_2) \ M(\text{CR}_2) \ M(\text{loc}_2) )$$

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

- (0 0 1 1 0 0 1)
- (0 0 1 0 0 1 0)
- (0 1 0 0 0 0 1)
- (1 0 0 0 0 0 1)
- (0 0 1 1 1 0 0)
- (1 0 0 1 0 0 1)
- (0 1 0 0 1 0 0)
- (1 0 0 0 0 1 0)
- (1 0 0 1 1 0 0)
- (1 0 0 1 0 0 1)
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:

  ![Petri net diagram]

  - 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 more 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

• explicitly represents concurrency and causal dependence between different behaviours

• 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

\[\text{Diagram showing states and transitions}\]
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 the unfolding 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:

```
• a
• a
• ...
• a_n
```

*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

• 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

• for finite unfoldings, compute and explore the whole structure

• for infinite unfoldings, only a finite prefix is computed and explored
can “x” ever occur?
can “x” ever occur?
can "x" ever occur?
can “x” ever occur?
can “x” ever occur?
can “x” ever occur?
can “x” ever occur?
can “x” ever occur?
can “x” ever occur?

need we compute further?
Complete finite prefix

• a **complete finite prefix** is a finite part of an unfolding that is **sufficient for deciding certain questions** about the original Petri net
  => e.g. executability, repeated executability, livelock, ...

• challenge is to determine **when to “stop” unfolding** without information loss
  => *outside scope of this lecture; see Esparza & Heljanko (2008)*

• previous slide gave a complete finite prefix
  => no “x” in the prefix; hence “x” can never occur in the original Petri net

• complete finite prefixes **can** be exponentially more concise than an interleaving-based 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
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

- “A False History of True Concurrency”
  => [http://dx.doi.org/10.1007/978-3-642-16164-3_13](http://dx.doi.org/10.1007/978-3-642-16164-3_13)
  => [https://www7.in.tum.de/~esparza/Talks/Impstrueconc.pdf](https://www7.in.tum.de/~esparza/Talks/Impstrueconc.pdf)
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 interleavings)