Petri nets

- Introduced in 1962 (though claimed to have been invented by 1939)
- Starting point: think of a transition system where a number of processes can be in a given state and then allow coordination
- **Conditions**: local components of state
- **Events**: transitions and coordination
- Allows study of concurrency of events, reasoning about causal dependency and how the action of one process might conflict with that of another
- The first of a range of models: event structures, Mazurkiewicz trace languages, asynchronous transition systems, . . .
- Many variants with different algorithmic properties and expressivity
Multisets generalise sets by allow elements to occur some number of times. $\omega$-multisets generalise further by allowing infinitely many occurrences.

$$\omega^\infty = \omega \cup \{\infty\}$$

Extend addition:

$$n + \infty = \infty \quad \text{for } n \in \omega^\infty$$

Extend subtraction

$$\infty - n = \infty \quad \text{for } n \in \omega$$

Extend order:

$$n \leq \infty \quad \text{for } n \in \omega^\infty$$

An $\omega$-multiset over a set $X$ is a function

$$f : X \to \omega^\infty$$

It is a multiset if $f : X \to \omega$. 
Operations on $\infty$-multisets

- $f \leq g$ iff $\forall x \in X. f(x) \leq g(x)$
- $f + g$ is the $\infty$-multiset such that
  \[ \forall x \in X. (f + g)(x) = f(x) + g(x) \]
- For $g$ a multiset such that $f \leq g$,
  \[ \forall x \in X. (f - g)(x) = f(x) - g(x) \]
A general Petri net consists of

- a set of conditions $P$
- a set of events $T$
- a pre-condition map assigning to each event $t$ a multiset of conditions $\bullet t$
- a post-condition map assigning to each event $t$ an $\infty$-multiset of conditions $t^*$
- a capacity map $\text{Cap}$ an $\infty$-multiset of conditions, assigning a capacity in $\omega^\infty$ to each condition
Dynamics

A marking is an \( \infty \)-multiset \( M \) such that

\[
M \leq \text{Cap}
\]

giving how many tokens are in each condition.

The token game:

For \( M, M' \) markings, \( t \) an event:

\[
M \xrightarrow{t} M' \quad \text{iff} \quad \bullet t \leq M \quad \& \quad M' = M - \bullet t + t^\bullet
\]

An event \( t \) has concession (is enabled) at \( M \) iff

\[
\bullet t \leq M \quad \& \quad M - \bullet t + t^\bullet \leq \text{Cap}
\]
Further examples
Basic Petri nets

Often don’t need multisets and can just consider sets.

A basic net consists of

- a set of conditions \( B \)
- a set of events \( E \)
- a pre-condition map assigning a subset of conditions \( \bullet e \) to any event \( e \)
- a post-condition map assigning a subset of conditions \( e^* \) to any event \( e \) such that
  \[
  \bullet e \cup e^* \neq \emptyset
  \]

The capacity of any condition is implicitly taken to be 1:

\[
\forall b \in B : \text{Cap}(b) = 1
\]

A marking \( M \) is now a subset of conditions.

\[
M \xrightarrow{e} M' \quad \text{iff} \quad \bullet q \subseteq M \quad \& \quad (M \setminus \bullet e) \cap e^* = \emptyset \\
\quad \& \quad M' = (M \setminus \bullet e) \cup e^*
\]
Concurrent Engineering Concepts

Concurrency

Forwards conflict

Backwards conflict

Contact
Persistent conditions

Between basic and general nets, conditions can be introduced that when they hold persist thereafter.

Useful for modelling broadcast messages.

\[ M \xrightarrow{e} M' \quad \text{iff} \quad \bullet e \subseteq M \land (e \bullet \cap (M \setminus (\text{Persistent} \cup \bullet e))) = \emptyset \land M' = (M \setminus \bullet e) \cup e \bullet \cup (M \cap \text{Persistent}) \]
Persistent conditions

Between basic and general nets conditions can be introduced that when they hold persist thereafter.

Useful for modelling broadcast messages

\[ \mathcal{M} \xrightarrow{e} \mathcal{M}' \quad \text{iff} \quad e \subseteq \mathcal{M} \land (e^\bullet \cap (\mathcal{M} \setminus (\text{Persistent} \cup e^\bullet))) = \emptyset \land \mathcal{M}' = (\mathcal{M} \setminus e^\bullet) \cup e^\bullet \cup (\mathcal{M} \cap \text{Persistent}) \]
Persistent conditions

Between basic and general nets, conditions can be introduced that when they hold persist thereafter.

Useful for modelling broadcast messages.

\[
M \stackrel{e}{\rightarrow} M' \quad \text{iff} \quad \bullet e \subseteq M \land (e^\bullet \cap (M \setminus (\text{Persistent} \cup \bullet e))) = \emptyset
\]

\&

\[M' = (M \setminus \bullet e) \cup e^\bullet \cup (M \cap \text{Persistent})\]
Persistent conditions

Between basic and general nets

conditions \( \otimes \) can be introduced that when they hold \textit{persist} thereafter

Useful for modelling broadcast messages

\[ M \xrightarrow{e} M' \iff \bullet e \subseteq M \land (e \cdot \cap (M \setminus (\text{Persistent} \cup \bullet e))) = \emptyset \land M' = (M \setminus \bullet e) \cup e \cdot \cup (M \cap \text{Persistent}) \]
Persistent conditions

Between basic and general nets

conditions can be introduced that when they hold persist thereafter

Useful for modelling broadcast messages

\[
M \xrightarrow{e} M' \quad \text{iff} \quad \bullet e \subseteq M \land (e^* \cap (M \setminus (\text{Persistent} \cup \bullet e))) = \emptyset
\]

\[\land M' = (M \setminus \bullet e) \cup e^* \cup (M \cap \text{Persistent})\]
Persistent conditions

Between basic and general nets

conditions \( \bigcirc \) can be introduced that when they hold persist thereafter

Useful for modelling broadcast messages

\[ M \xrightarrow{e} M' \text{ iff } \begin{align*}
\bullet e & \subseteq M & (e^* \cap (M \setminus (\text{Persistent} \cup \bullet e))) = \emptyset \\
& \& M' = (M \setminus \bullet e) \cup e^* \cup (M \cap \text{Persistent})
\end{align*} \]
Persistent conditions

Between basic and general nets

conditions can be introduced that when they hold persist thereafter

Useful for modelling broadcast messages

\[ M \xrightarrow{e} M' \quad \text{iff} \quad \bullet e \subseteq M \& (e^* \cap (M \setminus (\text{Persistent} \cup \bullet e))) = \emptyset \]
\[ \quad \& \quad M' = (M \setminus \bullet e) \cup e^* \cup (M \cap \text{Persistent}) \]
Modelling cryptographic protocols and event-based reasoning
Cryptographic protocols

- Protocols that use crytosystems to achieve some security goal across a distributed network
- Difficult and important to get right
- Security properties are subtle and hard to express
- Must reason about processes in an adverse environment:
  - Asynchronous communication
  - Dolev-Yao attacker (idealised cryptographic primitives)

- \(\Rightarrow\) a language to represent protocols
- with a Petri net semantics
- Analysis based on causal dependency: event-based reasoning
Public key cryptography:

- for each entity/participant/agent $A$, there is a key $Pub(A)$ and a key $Priv(A)$.
- $Pub(A)$ is intended to be known by everybody: it is public
- $Priv(A)$ is intended to be known only by $A$: it is private
- Any agent can encrypt using a key that it knows
- To decrypt a message encrypted under $Pub(A)$ it is necessary to know $Priv(A)$
- To decrypt a message encrypted under $Priv(A)$ it is necessary to know $Pub(A)$

Will also allow symmetric keys e.g. $Key(A, B)$. 
The goal of the NSL protocol: two agents use public-key cryptography to ensure

- **authentication**: For A as the initiator: upon completion of the protocol, A can demonstrate that B generated the messages that A received following the protocol in response to A’s request

- **shared secret**: if two entities complete the protocol with each other, at the end they both know a value not known to any potential attacker (e.g. to be used in more efficient symmetric-key cryptographic operations)

Formally, the correctness properties are subtle (e.g. what if B chose to release its private key?)
The protocol

(1) A $\rightarrow$ B: $\{m, A\}_{Pub(B)}$
(2) B $\rightarrow$ A: $\{m, n, B\}_{Pub(A)}$
(3) A $\rightarrow$ B: $\{n\}_{Pub(B)}$

- $m$ and $n$ are nonces: randomly-generated (very) long integers
- Only $B$ can decrypt the message sent in (1)
- $A$ knows that only $B$ can have sent the message in (2)
- $B$ knows that only $A$ can have sent the message in (1)
- the nonces $m$ and $n$ are shared secrets

But these properties are informal and approximate, and we’ve only described what’s *supposed* to happen . . .
The original protocol

Original protocol introduced by Needham and Schröder in 1978 contained a flaw revealed (and fixed) by Lowe in 1995 [using CSP]:

*Man-in-the-middle attacker* $E$ *convinces* $A$ *to start communication with* $E$ *and uses the messages generated by* $A$ *to follow the protocol with* $B$, *posing as* $A$.

\[
\begin{align*}
A &\rightarrow B : \{m, A\}_{Pub(B)} \\
B &\rightarrow A : \{m, n\}_{Pub(A)} \\
A &\rightarrow B : \{n\}_{Pub(B)}
\end{align*}
\]
The original protocol

Original protocol introduced by Needham and Schröder in 1978 contained a flaw revealed (and fixed) by Lowe in 1995 [using CSP]:

*Man-in-the-middle attacker* $E$ *convinces* $A$ *to start communication with* $E$ *and uses the messages generated by* $A$ *to follow the protocol with* $B$, *posing as* $A$.

\[
\begin{align*}
A \rightarrow B &: \{m, A\}_{Pub(B)} \\
B \rightarrow A &: \{m, n\}_{Pub(A)} \\
A \rightarrow B &: \{n\}_{Pub(B)}
\end{align*}
\]
The original protocol

Original protocol introduced by Needham and Schröder in 1978 contained a flaw revealed (and fixed) by Lowe in 1995 [using CSP]:

*Man-in-the-middle attacker* $E$ *convinces* $A$ *to start communication with* $E$ *and uses the messages generated by* $A$ *to follow the protocol with* $B$, *posing as* $A$.

```
A → B :  \{m, A\}_{Pub(B)}

B → A :  \{m, n\}_{Pub(A)}
```

```
A → B :  \{n\}_{Pub(B)}
```
The original protocol

Original protocol introduced by Needham and Schröder in 1978 contained a flaw revealed (and fixed) by Lowe in 1995 [using CSP]:

*Man-in-the-middle attacker* $E$ *convinces* $A$ *to start communication with* $E$ *and uses the messages generated by* $A$ *to follow the protocol with* $B$, *posing as* $A$.

\[
\begin{align*}
A & \rightarrow B : \{m, A\}_{Pub(B)} \\
B & \rightarrow A : \{m, n\}_{Pub(A)} \\
A & \rightarrow B : \{n\}_{Pub(B)}
\end{align*}
\]
The original protocol

Original protocol introduced by Needham and Schröder in 1978 contained a flaw revealed (and fixed) by Lowe in 1995 [using CSP]:

*Man-in-the-middle attacker* $E$ *convinces* $A$ *to start communication with* $E$ *and uses the messages generated by* $A$ *to follow the protocol with* $B$, *posing as* $A$.

\[
\begin{align*}
A &\rightarrow B : \{m, A\}_{Pub(B)} \\
B &\rightarrow A : \{m, n\}_{Pub(A)} \\
A &\rightarrow B : \{n\}_{Pub(B)} \\
E &\rightarrow A : \{m, A\}_{Pub(E)} \\
B &\rightarrow A : \{m, A\}_{Pub(B)} \\
A &\rightarrow B : \{m, n\}_{Pub(A)} \\
E &\rightarrow B : \{n\}_{Pub(E)} \\
\end{align*}
\]
The original protocol

Original protocol introduced by Needham and Schröder in 1978 contained a flaw revealed (and fixed) by Lowe in 1995 [using CSP]:

*Man-in-the-middle attacker* $E$ *convinces* $A$ *to start communication with* $E$ *and uses the messages generated by* $A$ *to follow the protocol with* $B$, *posing as* $A$.

\[
\begin{align*}
A &\rightarrow B : \{m, A\}_{Pub(B)} \\
B &\rightarrow A : \{m, n\}_{Pub(A)} \\
A &\rightarrow B : \{n\}_{Pub(B)}
\end{align*}
\]
We take an infinite set of names

\[
\text{Names} = \{m, n, \ldots, A, B, \ldots\}
\]

with name variables

\[
x, y, \ldots, X, Y
\]

Messages shall be ranged over by message variables

\[
\psi, \psi', \psi_1, \ldots
\]

Indices shall be used to identify components of parallel compositions

\[
i \in \text{Indices}
\]

Messages can contain free variables \(\rightsquigarrow\) messages as patterns on input
SPL syntax

**Name expressions**
\[ v :: = n | A | \ldots | x | X \]

**Key expressions**
\[ K :: = Pub(v) | Priv(v) | Key(v, v') \]

**Messages**
\[ M :: = \psi | v | k | M_1, M_2 | \{M\}_k \]

**Processes**
\[ p :: = \text{out new } \xi M.p \]
\[ \text{in pat } \eta \xi \psi M.p \]
\[ \parallel_{i \in I} p_i \]
Conventions

- `out M.p` where the list of `new` variables is empty
- `in M.p` where the lists of name and message variables are precisely the free name and message variables in `M`
- `nil` is the empty parallel composition, which may be freely omitted
- Use infix notation for finite parallel composition: `p_1 || p_2` is `||_{i\in\{1,2\}} p_i`
- Replication of a process `!p` is `||_{i\in\omega} p`