Topics in Concurrency

Lecture 9

Jonathan Hayman

4 March 2015

Petri nets

- Introduced in 1962 (though claimed to have been invented be 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

Multisets generalise sets by allow elements to occur some number of times. ∞-multisets generalise further by allowing infinitely many occurrences.

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

Extend addition:

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

Extend subtraction

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

Extend order:

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

An ∞-multiset over a set X is a function

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

It is a multiset if $f : X \to \omega$.

Operations on ∞-multisets

- $f \leq g$ iff $\forall x \in X. f(x) \leq g(x)$
- $f + g$ is the ∞-multiset such that

$$\forall x \in X.\ (f + g)(x) = f(x) + g(x)$$

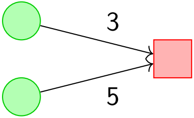
- For g a multiset such that $f \leq g$,

$$\forall x \in X.\ (f - g)(x) = f(x) - g(x)$$

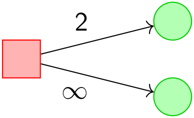
General Petri nets

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 ∞-multiset of conditions $t^\bullet$



- a capacity map Cap an ∞-multiset of conditions, assigning a capacity in ω^∞ to each condition

Dynamics

A marking is an ∞-multiset $\mathcal{M}$ such that

$$\mathcal{M} \leq Cap$$

giving how many tokens are in each condition.

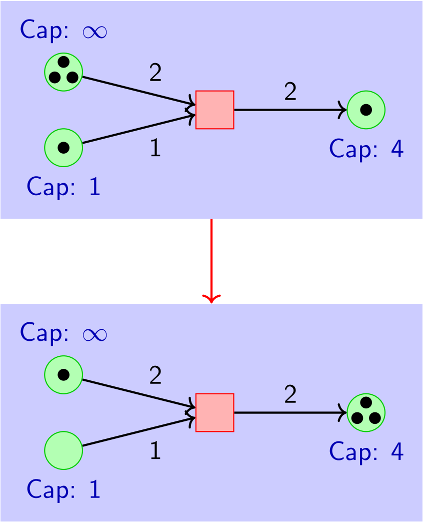
The token game:

For $\mathcal{M}, \mathcal{M}'$ markings, t an event:

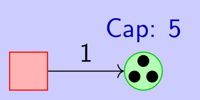
$$\mathcal{M} \xrightarrow{t} \mathcal{M}' \qquad \text{iff} \qquad {}^{\bullet}t \leq \mathcal{M} \quad \& \quad \mathcal{M}' = \mathcal{M} - {}^{\bullet}t + t^{\bullet}$$

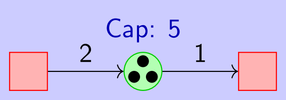
An event t has concession (is enabled) at $\mathcal{M}$ iff

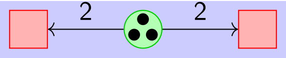
$${}^{\bullet}t \leq \mathcal{M} \quad \& \quad \mathcal{M} - {}^{\bullet}t + t^{\bullet} \leq Cap$$

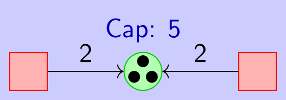

Cap: ∞
2
2
1
Cap: 4
Cap: 1
Cap: ∞
2
2
1
Cap: 4
Cap: 1

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* $e^{\bullet}$ *to any event* e *such that*
$$ {}^{\bullet}e \cup e^{\bullet} \neq \emptyset $$

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

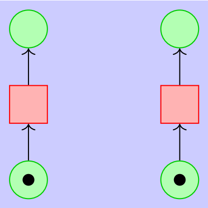
$$\forall b \in B : \; Cap(b) = 1$$

A marking $\mathcal{M}$ *is now a subset of conditions.*

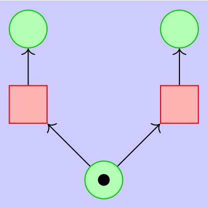
$$\mathcal{M} \xrightarrow{e} \mathcal{M}' \qquad \text{iff} \qquad \begin{array}{rl} & {}^{\bullet}q \subseteq \mathcal{M} \quad \& \quad (\mathcal{M} \setminus {}^{\bullet}e) \cap e^{\bullet} = \emptyset \\ \& & \mathcal{M}' = (\mathcal{M} \setminus {}^{\bullet}e) \cup e^{\bullet} \end{array}$$

Concepts

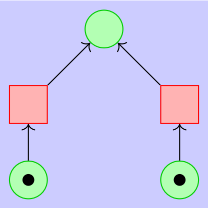
Concurrency



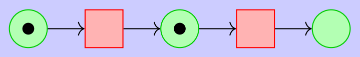
Forwards conflict



Backwards conflict



Contact

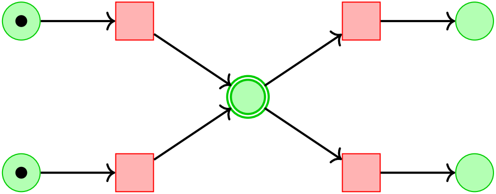


Persistent conditions

Between basic and general nets

conditions  can be introduced that when they hold persist thereafter

Useful for modelling broadcast messages



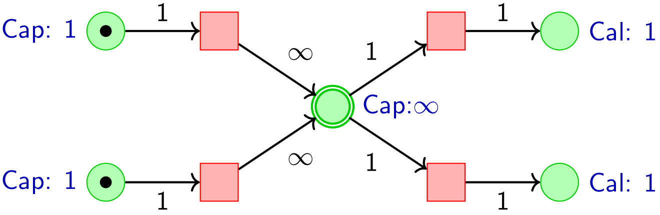
$$\mathcal{M} \xrightarrow{e} \mathcal{M}' \quad \text{iff} \quad \begin{array}{ll} & {}^\bullet e \subseteq \mathcal{M} \;\&\; (e^\bullet \cap (\mathcal{M} \setminus (\mathit{Persistent} \cup {}^\bullet e)) = \emptyset \\ \& & \mathcal{M}' = (\mathcal{M} \setminus {}^\bullet e) \cup e^\bullet \cup (\mathcal{M} \cap \mathit{Persistent}) \end{array}$$

Persistent conditions

Between basic and general nets

conditions  can be introduced that when they hold persist thereafter

Useful for modelling broadcast messages



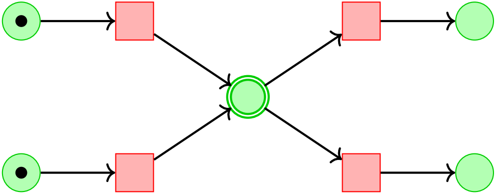
$$\mathcal{M} \xrightarrow{e} \mathcal{M}' \qquad \text{iff} \qquad \begin{array}{ll} & {}^\bullet e \subseteq \mathcal{M} \;\&\; (e^\bullet \cap (\mathcal{M} \setminus (\mathit{Persistent} \cup {}^\bullet e)) = \emptyset \\ \& & \mathcal{M}' = (\mathcal{M} \setminus {}^\bullet e) \cup e^\bullet \cup (\mathcal{M} \cap \mathit{Persistent}) \end{array}$$

Persistent conditions

Between basic and general nets

conditions  can be introduced that when they hold persist thereafter

Useful for modelling broadcast messages



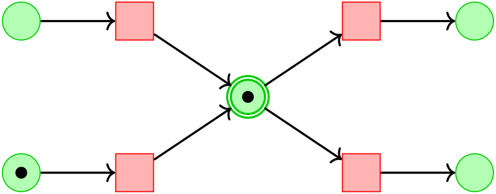
$$\mathcal{M} \xrightarrow{e} \mathcal{M}' \qquad \text{iff} \qquad \begin{array}{rl} & {}^\bullet e \subseteq \mathcal{M} \;\&\; (e^\bullet \cap (\mathcal{M} \setminus (\mathit{Persistent} \cup {}^\bullet e)) = \emptyset \\ \& & \mathcal{M}' = (\mathcal{M} \setminus {}^\bullet e) \cup e^\bullet \cup (\mathcal{M} \cap \mathit{Persistent}) \end{array}$$

Persistent conditions

Between basic and general nets

conditions  can be introduced that when they hold persist thereafter

Useful for modelling broadcast messages



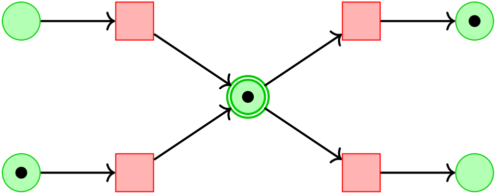
$$\mathcal{M} \xrightarrow{e} \mathcal{M}' \quad \text{iff} \quad \begin{array}{ll} & {}^\bullet e \subseteq \mathcal{M} \ \& \ (e^\bullet \cap (\mathcal{M} \setminus (\mathit{Persistent} \cup {}^\bullet e)) = \emptyset \\ \& & \mathcal{M}' = (\mathcal{M} \setminus {}^\bullet e) \cup e^\bullet \cup (\mathcal{M} \cap \mathit{Persistent}) \end{array}$$

Persistent conditions

Between basic and general nets

conditions  can be introduced that when they hold persist thereafter

Useful for modelling broadcast messages



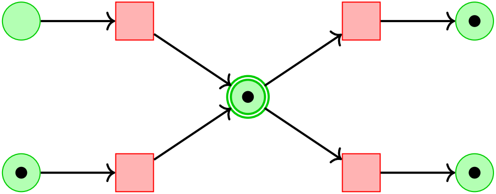
$$\mathcal{M} \xrightarrow{e} \mathcal{M}' \quad \text{iff} \quad \begin{array}{ll} & {}^\bullet e \subseteq \mathcal{M} \;\&\; (e^\bullet \cap (\mathcal{M} \setminus (\mathit{Persistent} \cup {}^\bullet e)) = \emptyset \\ \& & \mathcal{M}' = (\mathcal{M} \setminus {}^\bullet e) \cup e^\bullet \cup (\mathcal{M} \cap \mathit{Persistent}) \end{array}$$

Persistent conditions

Between basic and general nets

conditions can be introduced that when they hold persist thereafter

Useful for modelling broadcast messages



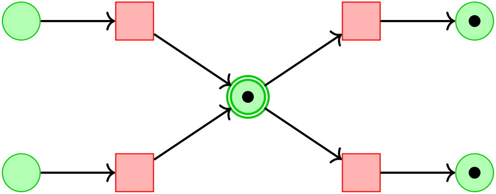
$$\mathcal{M} \xrightarrow{e} \mathcal{M}' \qquad \text{iff} \qquad \begin{array}{rl} & {}^\bullet e \subseteq \mathcal{M} \;\&\; (e^\bullet \cap (\mathcal{M} \setminus (\mathit{Persistent} \cup {}^\bullet e)) = \emptyset \\ \& & \mathcal{M}' = (\mathcal{M} \setminus {}^\bullet e) \cup e^\bullet \cup (\mathcal{M} \cap \mathit{Persistent}) \end{array}$$

Persistent conditions

Between basic and general nets

conditions  can be introduced that when they hold persist thereafter

Useful for modelling broadcast messages



$$\mathcal{M} \xrightarrow{e} \mathcal{M}' \quad \text{iff} \quad \begin{array}{rl} & {}^\bullet e \subseteq \mathcal{M} \;\&\; (e^\bullet \cap (\mathcal{M} \setminus (\mathit{Persistent} \cup {}^\bullet e)) = \emptyset \\ \& & \mathcal{M}' = (\mathcal{M} \setminus {}^\bullet e) \cup e^\bullet \cup (\mathcal{M} \cap \mathit{Persistent}) \end{array}$$

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)

- $\rightsquigarrow$ a language to represent protocols
- with a Petri net semantics
- Analysis based on causal dependency: event-based reasoning

Public-key cryptography

Public key cryptography:

- for each entity/participant/agent A, there is a key $Pub(\mathrm{A})$ and a key $Priv(\mathrm{A})$.
- $Pub(\mathrm{A})$ is intended to be known by everybody: it is public
- $Priv(\mathrm{A})$ 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 $Pub(\mathrm{A})$ it is necessary to know $Priv(\mathrm{A})$
- To decrypt a message encrypted under $Priv(\mathrm{A})$ it is necessary to know $Pub(\mathrm{A})$

Will also allow symmetric keys e.g. $Key(\mathrm{A}, B)$.

The Needham-Schröder-Lowe Protocol

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

$$\begin{array}{lll} (1) & \mathsf{A} \longrightarrow \mathsf{B}: & \{m, \mathsf{A}\}_{Pub(B)} \\ (2) & \mathsf{B} \longrightarrow \mathsf{A}: & \{m, n, \mathsf{B}\}_{Pub(A)} \\ (3) & \mathsf{A} \longrightarrow \mathsf{B}: & \{n\}_{Pub(B)} \end{array}$$

- *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.

A E B

$\mathsf{A} \longrightarrow \mathsf{B} : \; \{m, \mathsf{A}\}_{Pub(B)}$

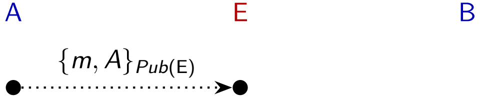
$\mathsf{B} \longrightarrow \mathsf{A} : \; \{m, n\}_{Pub(A)}$

$\mathsf{A} \longrightarrow \mathsf{B} : \; \{n\}_{Pub(B)}$

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.



$\mathsf{A} \longrightarrow \mathsf{B} : \ \{m, \mathsf{A}\}_{Pub(B)}$

$\mathsf{B} \longrightarrow \mathsf{A} : \ \{m, n\}_{Pub(A)}$

$\mathsf{A} \longrightarrow \mathsf{B} : \ \{n\}_{Pub(B)}$

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.

$$\begin{tikzcd} A \arrow[r, dotted, "\{m, A\}_{Pub(E)}"] & E \arrow[r, dotted, "\{m, A\}_{Pub(B)}"] & B \end{tikzcd}$$

$$A \longrightarrow B : \ \{m, A\}_{Pub(B)}$$

$$B \longrightarrow A : \ \{m, n\}_{Pub(A)}$$

$$A \longrightarrow B : \ \{n\}_{Pub(B)}$$

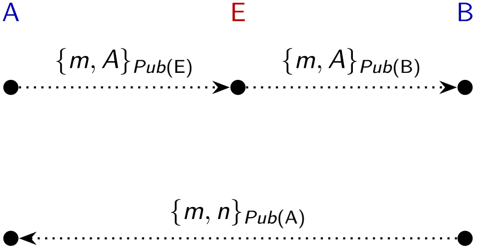
The original protocol

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.

$\mathsf{A} \longrightarrow \mathsf{B} : \ \{m, \mathsf{A}\}_{Pub(B)}$

$\mathsf{B} \longrightarrow \mathsf{A} : \ \{m, n\}_{Pub(A)}$



$\mathsf{A} \longrightarrow \mathsf{B} : \ \{n\}_{Pub(B)}$

The original protocol

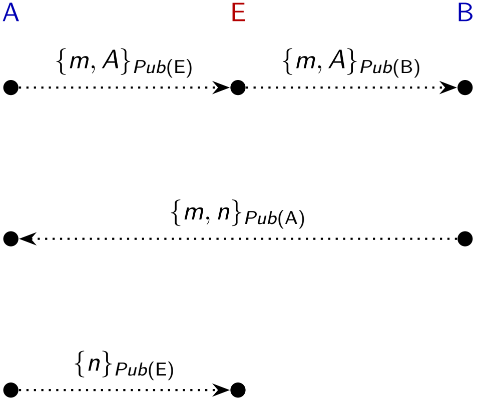
Original protocol introduced by Needham and Schröder in 1978 contained a flaw revealed (and fixed) by Lowe in 1995 [using CSP]:

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.

$\mathsf{A} \longrightarrow \mathsf{B} : \ \{m, \mathsf{A}\}_{Pub(B)}$

$\mathsf{B} \longrightarrow \mathsf{A} : \ \{m, n\}_{Pub(A)}$

$\mathsf{A} \longrightarrow \mathsf{B} : \ \{n\}_{Pub(B)}$



The original protocol

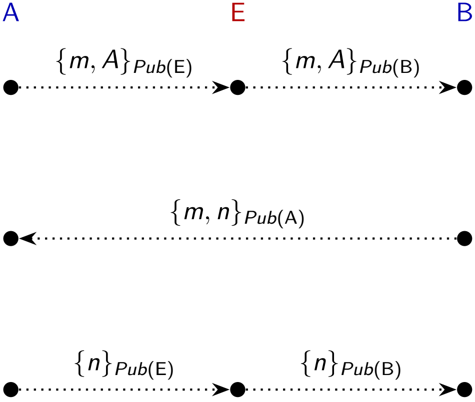
Original protocol introduced by Needham and Schröder in 1978 contained a flaw revealed (and fixed) by Lowe in 1995 [using CSP]:

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.

$\mathsf{A} \longrightarrow \mathsf{B} : \{m, \mathsf{A}\}_{Pub(B)}$

$\mathsf{B} \longrightarrow \mathsf{A} : \{m, n\}_{Pub(A)}$

$\mathsf{A} \longrightarrow \mathsf{B} : \{n\}_{Pub(B)}$



SPL

- We take an infinite set of names

$$\textbf{Names} = \{m, n, \ldots, \mathsf{A}, \mathsf{B}, \ldots\}$$

- 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

$$i \in \textbf{Indices}$$

Messages can contain free variables $\rightsquigarrow$ messages as patterns on input

SPL syntax

Name expressions	$v ::= n \mid A \mid \ldots \mid x \mid X$
Key expressions	$K ::= Pub(v) \mid Priv(v) \mid Key(v, v')$
Messages	$M ::= \psi \mid v \mid k \mid M_1, M_2 \mid \{M\}_k$
Processes	$p ::= \text{out new } \vec{x}\, M.p \mid \text{in pat } \vec{x}\vec{\psi} M.p \mid \|_{i \in I}\, p_i$

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: $p_1 \parallel p_2$ is $\|_{i \in \{1,2\}} \; p_i$
- replication of a process $!p$ is $\|_{i \in \omega} \; p$