Petri nets Primer with Workcraft

Alex Yakovlev
ASYNC Summer School 2024
(with thanks to Victor Khomenko)
Petri nets and Async Design – a bit of history

- Carl Adam Petri – 1962 - publishes his dissertation *Kommunikation mit Automaten* (in German), where a model later called Petri nets has been proposed. This mirrors the pioneering work of D. Muller et al. on speed-independent circuits (from the late 1950s).

- In the 1970s Petri nets used in the USA for async design thanks mostly to the efforts of researchers at MIT (J. Dennis, S. Patil, D. Misunas, F. Furtek, P.S. Thiagarajan and many others), some activities in UK, France, Germany. Methods on direct mapping of Petri nets to async circuits are established.


- In 1990s lots of in-depth theory, algorithms and tool developed by the Petrify team, by J. Cortadella (UPC), A. Kondratyev (Aizu, Cadence), M. Kishinevsky (Aizu, Intel), L. Lavagno (PoliTo) and A. Yakovlev (Newcastle) in collaboration with many other researchers. Methods for rigorous SI/QDI synthesis-verification are established.

- In 2000s more theory and tools developed by the Workcraft team at Newcastle, By I. Poliakov, V. Khomenko, A.Mokhov, D. Sokolov, A. Yakovlev in collaboration with many other researchers. Vision of Petri nets as a cornerstone of many interpreted graph models is formed.

- In 2010-2020s, STGs and Workcraft have been widely adopted for async circuit design in industry – especially in analog-mixed signal domain. Corporate tool flows being developed and products such as PMICs coming to market!
Recommended reading (historical overview and key theory of up to 1997):

Example: Producer/Buffer/Consumer

begin parallel
Producer: while true do
produce item
deposit item
Buffer: while true do
deposit item
remove item
Consumer: while true do
remove item
consume item
common action
common action
end parallel
Modelling Producer

**Producer:**

```plaintext
while true do
    produce item
    deposit item
```

**FSM model:**

```
prod dep
```

**PN model:**

```
prod dep
```
Modelling Buffer

Buffer:

\[
\text{while true do}
\]

\[
\text{deposit item}
\]

\[
\text{remove item}
\]

 FSM model:

PN model:
Modelling Consumer

Consumer: while true do
remove item
consume item

FSM model: PN model:

 FSM model: PN model:
Composing Petri net models

Draw the three Petri nets next to each other and glue together boxes with the same label (synchronous communication = join execution of common actions); some (informal) annotation can optionally be added:
Composing Petri net models

Draw the three Petri nets next to each other and glue together boxes with the same label (synchronous communication = join execution of common actions); some (informal) annotation can optionally be added:
Structure of Petri nets

Interpretation of different components:

• circles are called **places** and represent local states

• boxes are called **transitions** and represent actions changing local states

• black dots are called **tokens** and represent the current holding of local states (in general, a place may contain several tokens – e.g. to model a counter)

• **arcs** indicate how executing a transition modifies the state of a Petri net
Transition firing rule

Consider a PN with some marking $M: P \rightarrow \{0, 1, 2, \ldots\}$

A transition $t$ is **enabled** at $M$ if for every place $p$ such that $(p,t) \in F$ we have $M(p) > 0$ (i.e. all its preceding places are marked). **Firing** an enabled transition $t$ removes a token from each place $p$ such that $(p,t) \in F$ (i.e. from each preceding place) and then adds a token to each place $r$ such that $(t,r) \in F$ (i.e. to each succeeding place). 
Firing sequences and reachable markings

**Notation:** $M[t]M'$ means that transition $t$ is enabled at marking $M$, and firing it leads to marking $M'$

Starting from the initial marking $M_0$ we may fire a sequence of transitions $t_1t_2...t_n$ if there are markings $M_1,M_2,...,M_n$ such that $M_0[t_1]M_1[t_2]M_2...M_{n-1}[t_n]M_n$

Then $fs = ℓ(t_1)ℓ(t_2)...ℓ(t_n)$ is a *firing sequence*, and $M_n$ is a *reachable marking*.

Firing sequences represent what an *observer* of the system would see, and reachable markings are the ones the system can find itself in. In the FSM terminology, firing sequences are strings generated/accepted by a PN.
Example
Firing sequence: prod dep rem
Short-hand drawing conventions

• Omit drawing boxes for transitions – draw just labels:

• If a place has one incoming and one outgoing arc, just draw an arc through it:

• Colours of the labels: input, output, internal; the latter behave like outputs but are ignored by the environment
Short-hand drawing conventions: examples

• Producer/Buffer/Consumer system:

• Inverter circuit:
Modelling techniques: basic

Sequential execution of actions

Parallel execution of actions

Synchronisation
(joint execution of an action)

Testing a condition
without consuming a token
Testing for the presence of a token without consuming it is common. Instead of drawing two arcs going in opposite directions one can either draw a single arc with two arrowheads, or just a line without arrowheads, so the following three drawings have the same meaning. Such arcs are called **read arcs**.
Modelling techniques: complementary places

- Places $p$ and $q$ in are complementary iff any transition removing a token from one of them puts a token to the other.
- If $p$ can contain at most one token then creating place $q$ complementary to $p$ such that $q$ is initially marked iff $p$ isn’t does not change the behaviour of the PN.
- Can use a complementary place to test the negation of a condition.

\[
\text{\begin{array}{c}
\text{\textbullet} \quad \text{\textbullet} \quad \text{\textbullet} \\
\text{\textbullet} \quad \text{\textbullet} \quad \text{\textbullet} \\
\end{array}}
\]

\[
\text{\begin{array}{c}
\text{\textbullet} \quad \text{\textbullet} \quad \text{\textbullet} \\
\text{\textbullet} \quad \text{\textbullet} \quad \text{\textbullet} \\
\end{array}}
\]
Modelling techniques: synchronisation

**Example:** producer / 3-slot buffer / consumer:

**Idea:** synchronise the **rem** action of a slot with the **dep** action of the next slot:
Modelling techniques: counters

Idea: use multiple tokens on a place to represent a counter

Example: producer / buffer of capacity 3 / consumer:

Note: the ‘identities’ of items in the buffer are lost!
Modelling techniques: OR-causality

**Motivation:** quickly react to any of several stimuli

**Assumption:** all stimuli are eventually provided (if not, still can use OR-causality, but *arbitration* will be required in the ‘reset’ phase)
Modelling techniques: choices

Free choice

Asymmetric choice

Controlled Choice

Arbitration
Modelling techniques: choices

• Why are there any choices at all?
  • Abstraction of the environment (do not want a detailed model of the rest of the Universe!) – the ‘implementation’ of the environment may well be without choices
  • Resource contention: have to arbitrate between several clients trying to use the same resource
  • Structural choices: not ‘semantical’, e.g. due to modelling concurrency by interleaving; can be removed by restructuring the specification
  • Non-deterministic choices: either not ‘real’ and can be removed (by determinising the specification or making OR-causality explicit) or indicate that the system does not have enough information to perform its function (e.g. more inputs from the environment are required)
Petri net modelling examples
Modelling logic functions, pipelines, counters

Dual-rail AND gate

Dual-rail half-adder

Linear pipeline buffer

Counter (3-stage frequency divider)
Logic Circuit Modelling (cf. Rajit’s production rules)

Level-driven elements

NAND gate

x(=1)   y(=1)
         z(=0)

NOT gate

x(=1)   y(=0)

“Circuit” Petri net equivalents

x=0
y=0
z=0

x=1
y=1
z=1
High-level modelling: Processor Example
High-level modelling: Processor Example

Four-phase arbiter