Making Petri nets friendlier to designers

Jordi Cortadella
(Universitat Politècnica de Catalunya, Barcelona)

STRUCTURE 2017
Outline

• Friendly specification models for asynchronous circuits

• Mining friendly process specifications
Friendly specification models for asynchronous circuits

Jordi Cortadella, Alberto Moreno, Danil Sokolov, Alex Yakovlev, and David Lloyd.
Specification of an asynchronous circuit

Minimalist Petri net (Signal Transition Graph)
Asynchronous controllers

Source: Sokolov et al., Towards asynchronous power management, FTFC 2014.
Specification formalisms: process algebras

**CCS:**

Set Env = { rinp, routp, ainp, aoutp };

agent Env-L = rin.'rinp.Env-Lw;
agent Env-Lw = ainp.(ain.Env-L + ainp.Env-Lw);
agent Env-R = routp.(rout.aout.'aoutp.Env-R + routp.Env-R);
agent Implementation = (Env-L | Circuit | Env-R) \ Env;

**Balsa:**

procedure buffer
  (input i: byte;
   output o: byte) is
  variable x : byte
begin
  loop
    i -> x ; o <- x
  end
end
Specification formalisms: Burst Mode

No input/output concurrency
Specification formalisms: STGs
STGs are great!

• Any asynchronous behaviour can be specified.

• Implementation properties can be easily checked:
  – Consistency, output persistence, state encoding.

• There is a solid automatic synthesis flow from STGs to circuits (e.g., petrify, MPSAT):
  – State encoding
  – Logic synthesis for speed-independent circuits
  – Logic decomposition and technology mapping
Control specification

FIFO cntrl

Ri --> Ro
Ao --> Ai

Ri+ --> Ro+
Ao+ --> Ai+
Ri- --> Ro-
Ao- --> Ai-
Control specification

FIFO cntrl

Ri → Ri+
Ao → Ro+
Ai → Ai+
Ro → Ri-
Ao → Ai-
Ri+ → Ro+
Ro+ → Ri-
Ro- → Ao-
Ai+ → Ao+
Ai- → Ao-

Friendly Petri nets
Control specification

FIFO cntrl

Ri → Ro
Ao → Ai
Ro → Ri
Ai → Ao
Control specification

Ri+ ↔ Ao+
Ri- ↔ Ao-
Ri ↔ Ro
Ao ↔ Ai

FIFO cntrl

Ri+ → Ro+
Ri- → Ro-
Ri → Ro
Ri- → Ro

Ao+ → Ai+
Ao- → Ai-
Ao+ → Ai+
Ao- → Ai-
Control specification

FIFO cntrl

Ri  →  Ro  ←  Ai
Ao  ←  Ri-  →  Ro-
       ←  Ai-  →  Ao-
Control specification

FIFO cntrl

Ri+ → Ri- 
Ao+ → Ao- 
Ri+ → Ro+ 
Ri- → Ro- 
Ao+ → Ai+ 
Ao- → Ai- 
Ri+ → Ro+ 
Ri- → Ro- 
Ao+ → Ai+ 
Ao- → Ai-
Control specification

FIFO cntrl

Ri → FIFO cntrl → Ro
Ao → FIFO cntrl ← Ai

Ro+ → Ri+ → Ao+ → Ai+
Ro- → Ri- → Ao- → Ai-
Control specification

FIFO cntrl

Ri  →  Ri+
Ao  →  Ro+
Ro  ←  Ai+
Ai  ←  Ao+

Ri-  ←  Ri-
Ao-  ←  Ro-
Ro-  →  Ai-
Ai-  →  Ao-
Control specification

- FIFO cntrl
- Ri → Ro
- Ao ← Ai
- Ri+ → Ro+
- Ao+ ← Ai+
- Ri- → Ro-
- Ao- ← Ai-
Control specification

FIFO cntrl

Ri → FIFO cntrl → Ro
Ao → FIFO cntrl → Ai
Control specification

Ri+ Ao+ Ri- Ao-
Ri Ro Ao Ai
FIFO cntrl

Ro+
Ro-
Ai+
Ao+
Ri-
Ao-
Ai-
Trying to persuade engineers (the cruel reality)

Yeah! Signal Transition Graphs are extremely useful.

They are based on **Petri nets**. They have places, transitions, and some tokens that flow, ...
We need a simpler formalism

STG experts

Non-expert engineers

STGs
Wish list for the specification model

- Easy adoption.
- Powerful
  - maybe a little bit less than STGs
  - but more than Burst Mode
- Efficient: use logic synthesis (Boolean minimization).
- Reuse the existing infrastructure for STGs.
Engineers understand waveforms

Source: Xilinx LogiCORE IP FIFO Generator
Multiple waveforms (modes of operation)
Expressive power

• Causality and concurrency between events.

• Choice (modes of operation):
  – AMBA bus: read/write cycle.

• Proposal (our sacrifice):
  – Concurrency and choice mutually exclusive.
Expressive power

[Petri net diagram]

Not allowed
Example: Buck controller

(a) UV without ZC
(b) UV before ZC
(c) UV after ZC
Mode: UV before ZC
Waveform Transition Graph (WTG)

(nodal states)

choice-free & acyclic
ANALYSIS OF ASYNCHRONOUS CIRCUITS AND PROCESSES

DEFINITION 8.9. 1) A sequence of states \( \alpha^1 \alpha^2 \ldots \alpha^k \) is called complete if it is either finite and ends with a deadlock state, or contains no infinite section where all states have the same variable value.

2) Let \( W \) be a set of states reachable from the states of \( A \). Then \( W \) is called nodal for \( A \) if all complete sequences beginning in the states of \( W \) are contained in the set of states of \( A \).
Nodal state in reachability graph

- Only nodal states can have choice
- For timed verification: all counters to zero
Silent transitions might be acceptable

Causality &
Concurrency
(choice-free)
WTG of a buck controller

WF1

WF2

WF3

early_zc

late_or_no_zc

s0

s1

oc_handling

Structure 2017

Friendly Petri nets
Concurrency and choice

[Petri net diagram]

Structure 2017

Friendly Petri nets 35
Concurrency and choice
Concurrency and choice

Structure 2017

Friendly Petri nets
WTG for the VME bus controller

WF read

WF write

Structure 2017

Friendly Petri nets 38
WTG: expressive power
BM vs. WTG vs. STG

BM

WTG

STG

Structure 2017

Friendly Petri nets
Structural properties of SC WTGs

- Live
- Safe
- Abstraction: one State Machine
- Every cycle of the RG contains at least one nodal state

Reachability (conjecture): $M = M_0 + A\sigma$

Negative result (not tragic): Safe Marked Graphs $\not\subset$ WTG

(choice-free & acyclic)
Not every safe Marked Graph is a WTG

Every state is at one side of some diamond

No nodal state
Mining friendly process specifications

Javier de San Pedro and Jordi Cortadella. Mining structured Petri nets for the visualization of process behavior. 31st ACM Symposium on Applied Computing, April 2016.
Mining specifications

Log:
abcdedeb
acdbecb
acbdeab
abceab
adec
acbec
adea
From LTS to Labelled Petri Net

Overfitting vs. Underfitting

Flower models (≈Σ*)
Our proposal

Few models, each is low-complexity
Extracting well-structured slices

Captures more than 90% of log

Most frequent behavior

Less frequent

Slicing

Structure 2017

Friendly Petri nets

47
Mining specifications

Log:

a b c d e a b
c d b e c b
a c b d e a b
a b c e a
d e c
a b c e c
a d e a

Structure 2017

Friendly Petri nets
Slicing specifications

Log:
abcdab
acdbecb
acbdab
abcea
adec
acbec
adea

Best and Devillers:
• Characterisation of the state spaces of marked graph Petri nets
• Synthesis of live and bounded persistent systems

Petrify → Synthesis of free-choice Petri nets
Slicing specifications

Log:

a b c d e a b
c a c b
a c b d e a b
a b c e a
a d e c
a c b e c
a d e a
Slicing specifications

**Log:**
- abcdeab
- acdbecb
- acbdeab
- abcea
- adec
- acbec
- adea

---

Monolithic Petri net

71% 29%

---

Friendly Petri nets
SAT model

Log:
• a b c e
• a c b e
• a d b c e

- One Boolean variable per LTS transition
  - $t_i$ is true if corresponding arc is selected in slice
- Every valid assignment forms an LTS slice (subset)
Example: forward persistence

- Forward persistence:
  - If $a$ and $b$ exist in a LTS slice, then $a$ and $b$ must exist too.
  - $t_1 \land t_2 \rightarrow t_3 \land t_4$

Log:
- $a \ b \ c \ e$
- $a \ c \ b \ e$
- $a \ d \ b \ c$

SAT constraints:
- $t_2 \land t_3 \rightarrow t_4 \land t_5$
- $t_8 \land t_9 \rightarrow t_{10} \land t_{11}$
- $\neg t_3 \lor \neg t_7$
Example: forward persistency

Structure 2017 Friendly Petri nets 54
Example: incidenttelco

Petri net mined with *ILP miner*:

- 448 arcs
- 9800 crossings
- 100% fitness
Example: *incidenttelco*

- 65 slices required for 100% fitness
- However, with the first slice only:

<table>
<thead>
<tr>
<th>1st slice</th>
<th>Original</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 arcs</td>
<td>448 arcs</td>
</tr>
<tr>
<td>Planar (0 cros.)</td>
<td>9800 crossings</td>
</tr>
<tr>
<td>92.6% fitness</td>
<td>100% fitness</td>
</tr>
</tbody>
</table>

Covers most behaviours
Other examples

• Similar results in other benchmarks
  – Only 1-3 slices required for >90% of behaviour
  – Every slice is low visual complexity

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># of slices</th>
<th># of crossings</th>
</tr>
</thead>
<tbody>
<tr>
<td>documentflow</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>fhmilu</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>fhmn5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>incidenttelco</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>kim</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>purchasetopay</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>receipt</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>tsl</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th># of slices</th>
<th># of crossings</th>
</tr>
</thead>
<tbody>
<tr>
<td>documentflow</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>fhmilu</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>fhmn5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>incidenttelco</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>kim</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>purchasetopay</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>receipt</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>tsl</td>
<td>10</td>
<td>9</td>
</tr>
</tbody>
</table>
Conclusions

• We need to expand the social circle of Petri nets:
  – Asynchronous circuits
  – Process/Specification mining
  – ... and many other domains

• Simple structures contribute to sympathise with Petri nets:
  – For specification (WTG)
  – For analysis and visualisation (Petri net slices)

• The structural theory of Petri nets is an essential guidance:
  – “Nice” commonly means “visually friendly”
  – Correlation between structural properties and visualisation