Factored Planning: From Automata to Petri Nets

Loïg Jezequel¹, Eric Fabre², Victor Khomenko³

ACSD, July 9, 2013

¹ENS Cachan Bretagne
²INRIA Rennes
³Newcastle University
Goal

Find a *plan*: a sequence of actions (with minimal cost) moving the system from its initial state to one of its goal states.
Each component is a planning problem with its own resources and actions.
Each **component** is a planning problem with its own resources and actions.

The components **interact** by resources and/or actions.

**Goal**

Find a set of **compatible** local plans: they can be **interleaved** into a global plan.

Each component is a planning problem with its own resources and actions.

The components interact by resources and/or actions.

Goal

Find a set of compatible local plans: they can be interleaved into a global plan.
Components ⇒ Automata
Plans ⇒ Words
Interaction ⇒ Synchronous product

New goal
Given $A = A_1 \parallel \ldots \parallel A_n$, find a word in $A$ by local computations
Components $\Rightarrow$ Automata
Plans $\Rightarrow$ Words
Interaction $\Rightarrow$ Synchronous product

New goal
Given $\mathcal{A} = \mathcal{A}_1 \parallel \ldots \parallel \mathcal{A}_n$, find a word in $\mathcal{A}$ by local computations

A possibility [Fabre et al. 10]
Compute $\mathcal{A}'_i = \Pi_{\Sigma_i}(\mathcal{A})$ for each $i$ without computing $\mathcal{A}$

Why?
1. any word $w$ of $\mathcal{A}$ can be projected into a word $w_i$ of $\Pi_{\Sigma_i}(\mathcal{A})$
2. any word $w_i$ of $\Pi_{\Sigma_i}(\mathcal{A})$ is the projection of a word $w$ of $\mathcal{A}$

$\Rightarrow$ Easy extraction of a word from $\mathcal{A}$ by local searches
A possibility [Fabre et al. 10]

Compute $A'_i = \Pi_{\Sigma_i}(A)$ for each $i$ without computing $A$

How? Conditional independence like property

$$\Pi_{\Sigma_1 \cap \Sigma_2}(A_1 \times A_2) \equiv_{L} \Pi_{\Sigma_1 \cap \Sigma_2}(A_1) \times \Pi_{\Sigma_1 \cap \Sigma_2}(A_2)$$

Application:

$$A_1 \xrightarrow{\Sigma_1 \cap \Sigma_2} A_2 \xrightarrow{\Sigma_2 \cap \Sigma_3} A_3$$

$$\Pi_{\Sigma_1}(A) = \Pi_{\Sigma_1}(A_1 \times A_2 \times A_3)$$

$$\equiv_{L} \Pi_{\Sigma_1}(A_1) \times \Pi_{\Sigma_1}(A_2 \times A_3)$$

$$\equiv_{L} A_1 \times \Pi_{\Sigma_1 \cap \Sigma_2}(A_2 \times A_3)$$

$$\equiv_{L} A_1 \times \Pi_{\Sigma_1 \cap \Sigma_2}(A_2 \times \Pi_{\Sigma_2 \cap \Sigma_3}(A_3))$$
A possibility [Fabre et al. 10]

Compute $A'_i = \Pi_{\Sigma_i}(A)$ for each $i$ without computing $A$

How? Conditional independence like property

$$\Pi_{\Sigma_1 \cap \Sigma_2}(A_1 \times A_2) \equiv \Pi_{\Sigma_1 \cap \Sigma_2}(A_1) \times \Pi_{\Sigma_1 \cap \Sigma_2}(A_2)$$

Application:

$$\Pi_{\Sigma_1}(A) = \Pi_{\Sigma_1}(A_1 \times A_2 \times A_3)$$
$$\equiv \Pi_{\Sigma_1}(A_1) \times \Pi_{\Sigma_1}(A_2 \times A_3)$$
$$\equiv A_1 \times \Pi_{\Sigma_1 \cap \Sigma_2}(A_2 \times A_3)$$
$$\equiv A_1 \times \Pi_{\Sigma_1 \cap \Sigma_2}(A_2 \times \Pi_{\Sigma_2 \cap \Sigma_3}(A_3))$$
A possibility [Fabre et al. 10]

Compute $\mathcal{A}'_i = \Pi_{\Sigma_i}(\mathcal{A})$ for each $i$ without computing $\mathcal{A}$

How? Generalization

*Message passing algorithms*: proceed by successive refinements

\[
\begin{align*}
\mathcal{A}_1 & \xrightarrow{\Pi_{\Sigma_1}} \mathcal{A}_2 \\
\mathcal{A}_2 & \xrightarrow{\Pi_{\Sigma_2}} \mathcal{A}_3 \\
\mathcal{A}_2 & \xrightarrow{\Pi_{\Sigma_2}} \mathcal{A}_4 \\
\mathcal{A}_1 & \xrightarrow{\Pi_{\Sigma_2}} \mathcal{A}_5 \\
\mathcal{A}_3 & \xrightarrow{\Pi_{\Sigma_3}} \mathcal{A}_4 \\
\mathcal{A}_3 & \xrightarrow{\Pi_{\Sigma_3}} \mathcal{A}_5 \\
\mathcal{A}_5 & \xrightarrow{\Pi_{\Sigma_5}} \mathcal{A}_3 \\
\mathcal{A}_5 & \xrightarrow{\Pi_{\Sigma_5}} \mathcal{A}_4
\end{align*}
\]
Concurrency in factored planning problems

Global concurrency: between components (private actions)

Local concurrency: internal to a component

Remark: local concurrency is not anecdotal
Concurrency in factored planning problems

Global concurrency: between components (private actions)
Local concurrency: internal to a component

Concurrency in networks of automata

Global concurrency: taken into account
Local concurrency: ignored!
Concurrency in factored planning problems

Global concurrency: between components (private actions)
Local concurrency: internal to a component

Concurrency in networks of automata

Global concurrency: taken into account
Local concurrency: ignored!

Networks of Petri nets

Global concurrency: taken into account
Local concurrency: taken into account
The real purpose of automata

Implementation of product and projection of regular languages with finite objects

Our goal

More efficient implementation by taking local concurrency into account:

- product of languages $\Rightarrow$ product of Petri nets
- projection of languages $\Rightarrow$ projection of Petri nets
Languages, automata, Petri nets
Product of Petri nets
Projection of Petri nets

Product of Petri nets

\[ N_1 \times N_2 \]
$\Pi_\Sigma(N)$: a two step procedure

1. Replace the transitions with label not in $\Sigma$ by silent transitions
2. Remove silent transitions (optimisation purpose)
Π_{Σ}(N): a two step procedure

1. Replace the transitions with label not in Σ by silent transitions
2. Remove silent transitions (optimisation purpose)

How to remove silent transitions

Use the reachability graph: no more concurrency

Preservation of concurrency: for restricted class of nets only
[Wimmel 04]

Transition contraction: efficient in practice
[André 82] [Vogler and Kangsah 07]
Contraction of a silent transition $t$, only when $\cdot t \cap t^\bullet = \emptyset$

```
Languages, automata, Petri nets
Product of Petri nets
Projection of Petri nets
Conclusions
```

```
$p_1 \bullet$   $q_1 \bullet$
\vspace{-2cm}
\hspace{-3cm}$\varepsilon$
\hspace{3cm}$b$
\vspace{-2cm}
$p_2$   $q_2$

(p_1, p_2) \bullet
\vspace{-2cm}
\hspace{-3cm}b
\vspace{-2cm}
(q_2, p_2)

(p_1, q_1) \bullet
\vspace{-2cm}
\hspace{-3cm}(q_2, q_1)
```
Contraction of a silent transition $t$, only when $\cdot t \cap t^\bullet = \emptyset$

Language and safeness preserving contraction of $t$

$|t^\bullet| = 1$, $\cdot (t^\bullet) = \{t\}$ and $M^0(p) = 0$ with $t^\bullet = \{p\}$

or $|\cdot t| = 1$, $\cdot (t^\bullet) = \{t\}$ and $\forall p \in t^\bullet$, $M^0(p) = 0$

or $|\cdot t| = 1$ and $(\cdot t)^\bullet = \{t\}$
Benchmark selection
- From Corbett96
- Scale well (number of components vs. size of components)
- Tree shape (manually obtained)

Benchmark set
- Dining philosophers
- Dining philosophers with a dictionary
- Divide and conquer
- Milner’s cyclic scheduler
- Token-ring mutual exclusion protocol

What we compare
Times spent to compute updated automata/Petri nets
Divide and conquer

Dining philosophers
Token-ring mutual exclusion protocol
Dining philosophers with a dictionary

Milner’s cyclic scheduler
Contribution

- Networks of automata \(\Rightarrow\) networks of Petri nets for planning
- Experimental comparison: Petri nets can bring an important efficiency gain by handling local concurrency
- Extension to weighted systems (in the paper)
**Contribution**

- Networks of automata $\Rightarrow$ networks of Petri nets for planning
- Experimental comparison: Petri nets can bring an important efficiency gain by handling local concurrency
- Extension to weighted systems (in the paper)

**Possible future work**

- Compare transition contraction without and with weights
- Relax the conditions for transition contraction with weights