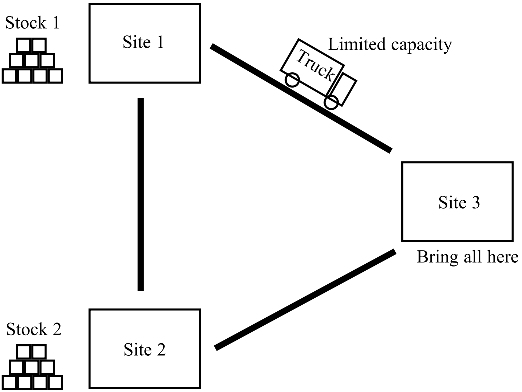
Factored Planning: From Automata to Petri Nets

Loïg Jezequel[1], Eric Fabre[2], Victor Khomenko[3]

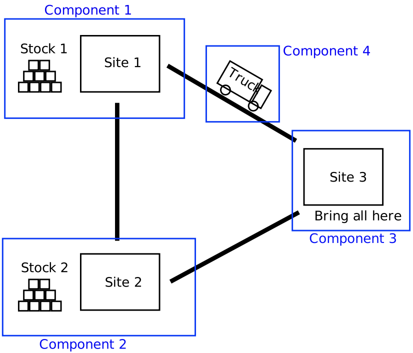
ACSD, July 9, 2013

[1]ENS Cachan Bretagne
[2]INRIA Rennes
[3]Newcastle University



Goal

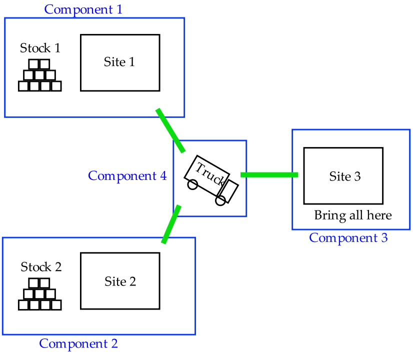
Find a *plan*: a sequence of actions (with minimal cost) moving the system from its initial state to one of its goal states



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

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 $\mathcal{A} = \mathcal{A}_1 || \ldots || \mathcal{A}_n$, find a *word* in $\mathcal{A}$ by local computations

Components ⇒ Automata
Plans ⇒ Words
Interaction ⇒ Synchronous product

New goal

Given $\mathcal{A} = \mathcal{A}_1 || \dots || \mathcal{A}_n$, 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 $\mathcal{A}$

⇒ Easy extraction of a word from $\mathcal{A}$ by local searches

A possibility [Fabre et al. 10]

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

How? Conditional independence like property

$$\Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_1 \times \mathcal{A}_2) \equiv_{\mathcal{L}} \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_1) \times \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_2)$$

Application:

$$\begin{aligned}
\Pi_{\Sigma_1}(\mathcal{A}) &= \Pi_{\Sigma_1}(\mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_3) \\
&\equiv_{\mathcal{L}} \Pi_{\Sigma_1}(\mathcal{A}_1) \times \Pi_{\Sigma_1}(\mathcal{A}_2 \times \mathcal{A}_3) \\
&\equiv_{\mathcal{L}} \mathcal{A}_1 \times \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_2 \times \mathcal{A}_3) \\
&\equiv_{\mathcal{L}} \mathcal{A}_1 \times \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_2 \times \Pi_{\Sigma_2 \cap \Sigma_3}(\mathcal{A}_3))
\end{aligned}$$

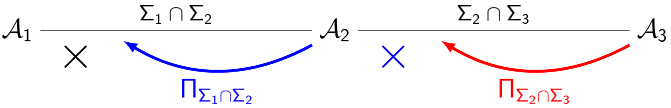
A possibility [Fabre et al. 10]

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

How? Conditional independence like property

$$\Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_1 \times \mathcal{A}_2) \equiv_{\mathcal{L}} \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_1) \times \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_2)$$

Application:



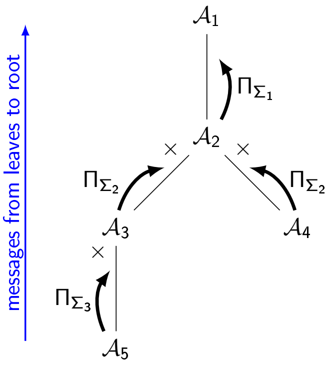
$$\begin{aligned}
\Pi_{\Sigma_1}(\mathcal{A}) &= \Pi_{\Sigma_1}(\mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_3) \\
&\equiv_{\mathcal{L}} \Pi_{\Sigma_1}(\mathcal{A}_1) \times \Pi_{\Sigma_1}(\mathcal{A}_2 \times \mathcal{A}_3) \\
&\equiv_{\mathcal{L}} \mathcal{A}_1 \times \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_2 \times \mathcal{A}_3) \\
&\equiv_{\mathcal{L}} \mathcal{A}_1 \times \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_2 \times \Pi_{\Sigma_2 \cap \Sigma_3}(\mathcal{A}_3))
\end{aligned}$$

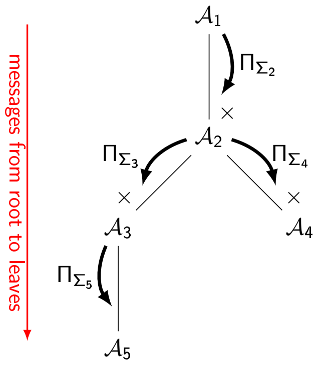
A possibility [Fabre et al. 10]

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



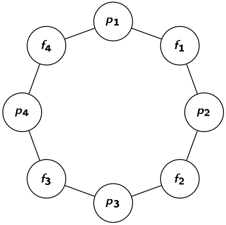


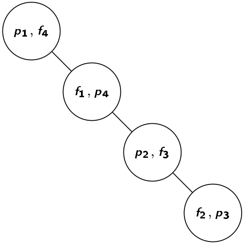
Concurrency in factored planning problems

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

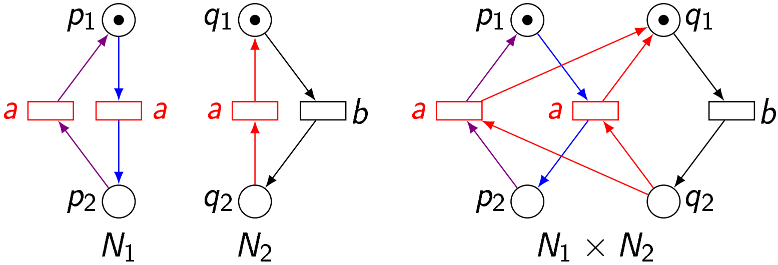
The real purpose of automata

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


p1
q1
a
a
a
b
p2
q2
N1
N2
p1
q1
a
a
b
p2
q2
N1 × N2

$\Pi_\Sigma(N)$: a two step procedure

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

$\Pi_\Sigma(N)$: a two step procedure

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

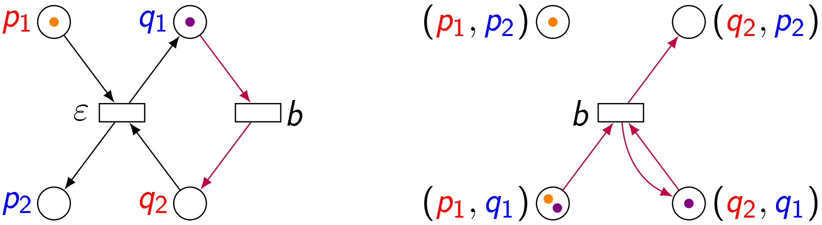
How to remove silent transitions

Use the reachability graph: no more concurrency

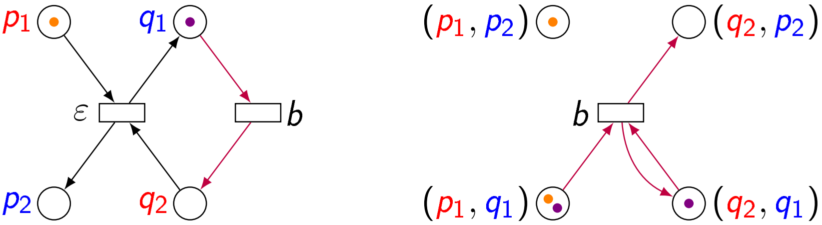
Preservation of concurrency: for restricted class of nets only
[Wimmel 04]

Transition contraction: efficient in practice
[André 82] [Vogler and Kangsah 07]

Contraction of a silent transition t, only when ${}^\bullet t \cap t^\bullet = \emptyset$



Contraction of a silent transition t, only when ${}^\bullet t \cap t^\bullet = \emptyset$



Language and safeness preserving contraction of t

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

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

or $|{}^\bullet t| = 1$ and $({}^\bullet t)^\bullet = \{t\}$

Benchmark selection

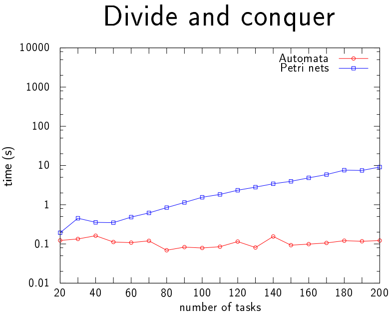
- From Corbett96
- Scale well (number of components vs. size of components)
- Tree shape (manually obtained)

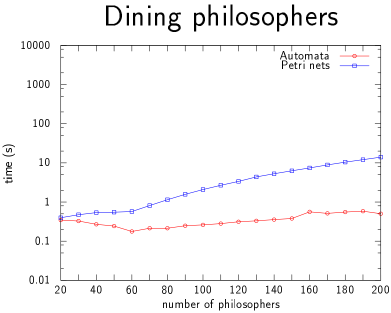
Benchmark set

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

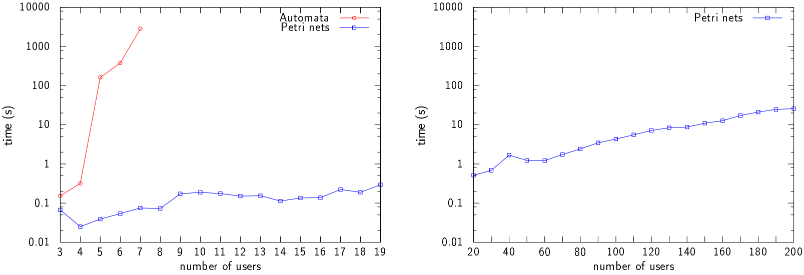
What we compare

Times spent to compute updated automata/Petri nets

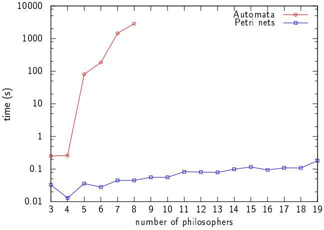

Divide and conquer
Automata
Petri nets
time (s)
10000
1000
100
10
1
0.1
0.01
20 40 60 80 100 120 140 160 180 200
number of tasks

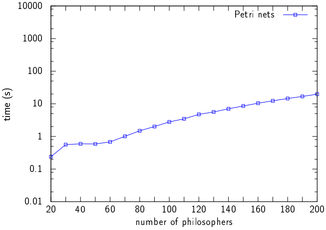

Dining philosophers
Automata
Petri nets
time (s)
10000
1000
100
10
1
0.1
0.01
20 40 60 80 100 120 140 160 180 200
number of philosophers

Token-ring mutual exclusion protocol

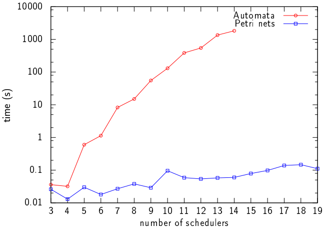


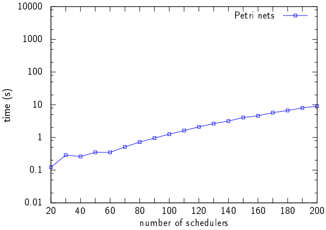
Dining philosophers with a dictionary





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