Knowledge for the Distributed Implementation of Constrained Systems

Susanne Graf[1] and Sophie Quinton[2]

[1] VERIMAG/CNRS, [2] TU Braunschweig

LCCC Workshop, Lund, April 2013

Distributed Control and Implementation

Problem to be solved:

> *"Given a centralized specification PN and a constraint Ψ,*
> *Derive a distributed implementation I for PN controlled by Ψ"*

Our hypotheses:

- centralized specification *PN*: w.l.g. Petri Nets
- distributed setting: one process per location — can learn about each other only via communication mechanisms provided by the platform
- constraint Ψ: a safety constraint (here: priorities)

Not considered in this talk:

- uncontrollable transitions, data, timing, progress constraints, ...

Our approach to distributed implementation

Knowledge-based presentation for combining control and distribution:

1. Use knowledge to realize a transformation [BBPS09,GPQ10]:

$$PN + \Psi \longrightarrow PN' \text{ enforcing } \Psi$$

2. Derive a distributed implementation I for a PN by means of a protocol Pr [PCT04,BGQ11]:

$$PN' \oplus Pr \longrightarrow I$$

Exist: algorithms/proofs for a particular implementation relation
for a particular platform

Claim: A knowledge-based approach is also interesting for problem (2)

- define more efficient protocols (think in terms of knowledge)
- optimize existing protocols (exploit existing — also application dependent — knowledge)

Outline

1. **Introduction**

2. **Knowledge for Control**
   - One-safe Petri Nets *PN* and control constraints
   - Locality and knowledge
   - Knowledge for Control

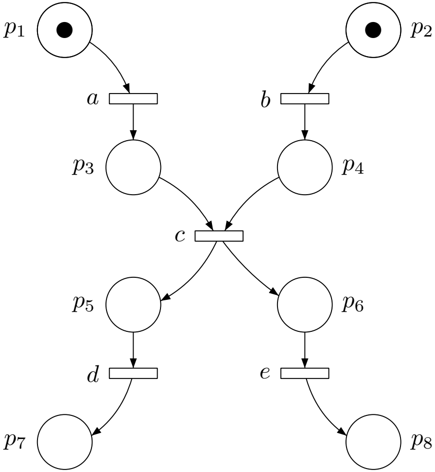
3. **Knowledge for Distributed Implementation**
   - Distributed Setting: implementation relations
   - Knowledge Required in a Distributed Implementation
   - Knowledge and Communication

4. **Discussion**

One-safe Petri Nets

Place/Transition Nets:

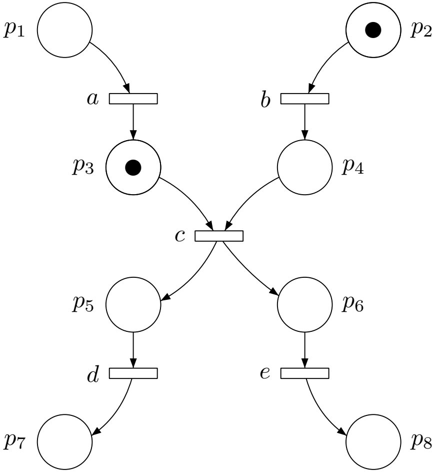
- state s: a set of places
- transition c is enabled (en_c) if $\{p_3, p_4\} \subseteq s$ and leads to $s' = s - \{p_3, p_4\} + \{p_5, p_6\}$.
- a state is reachable if it appears in some execution.
- jointly enabled transitions are independent if they don't share places (e.g. d, e in $\{p_5, p_6\}$)



One-safe Petri Nets

Place/Transition Nets:

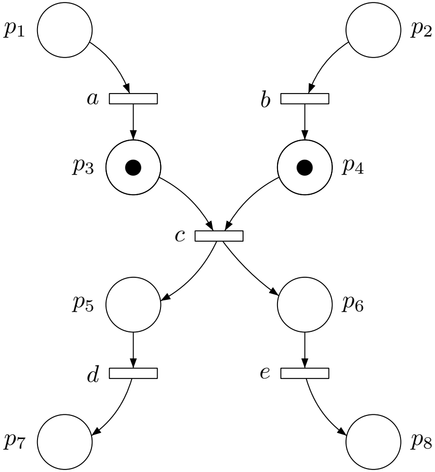
- state s: a set of places
- transition c is enabled (en_c) if $\{p_3, p_4\} \subseteq s$ and leads to $s' = s - \{p_3, p_4\} + \{p_5, p_6\}$.
- a state is reachable if it appears in some execution.
- jointly enabled transitions are independent if they don't share places (e.g. d, e in $\{p_5, p_6\}$)



One-safe Petri Nets

Place/Transition Nets:

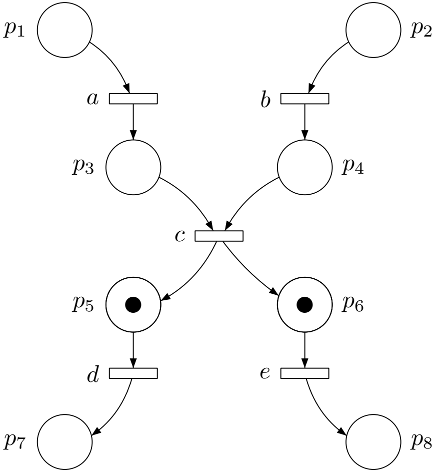
- state s: a set of places
- transition c is enabled (en_c) if $\{p_3, p_4\} \subseteq s$ and leads to $s' = s - \{p_3, p_4\} + \{p_5, p_6\}$.
- a state is reachable if it appears in some execution.
- jointly enabled transitions are independent if they don't share places (e.g. d, e in $\{p_5, p_6\}$)



One-safe Petri Nets

Place/Transition Nets:

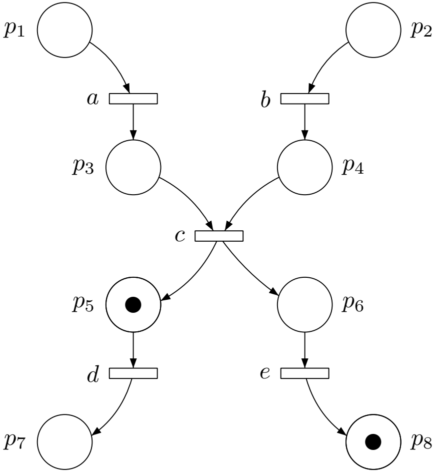
- state s: a set of places
- transition c is enabled (en_c) if $\{p_3, p_4\} \subseteq s$ and leads to $s' = s - \{p_3, p_4\} + \{p_5, p_6\}$.
- a state is reachable if it appears in some execution.
- jointly enabled transitions are independent if they don't share places (e.g. d, e in $\{p_5, p_6\}$)



One-safe Petri Nets

Place/Transition Nets:

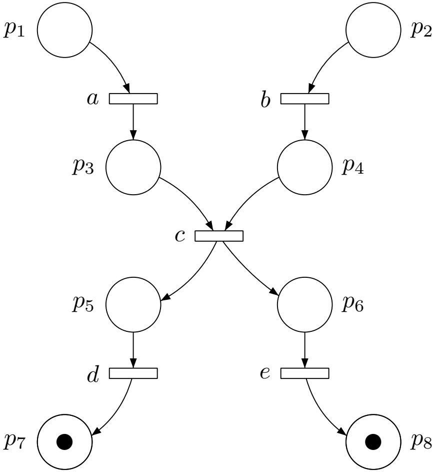
- state s: a set of places
- transition c is enabled (en_c) if $\{p_3, p_4\} \subseteq s$ and leads to $s' = s - \{p_3, p_4\} + \{p_5, p_6\}$.
- a state is reachable if it appears in some execution.
- jointly enabled transitions are independent if they don't share places (e.g. d, e in $\{p_5, p_6\}$)



One-safe Petri Nets

Place/Transition Nets:

- state s: a set of places
- transition c is enabled (en_c) if $\{p_3, p_4\} \subseteq s$ and leads to $s' = s - \{p_3, p_4\} + \{p_5, p_6\}$.
- a state is reachable if it appears in some execution.
- jointly enabled transitions are independent if they don't share places (e.g. d, e in $\{p_5, p_6\}$)



Global constraints — here priorities

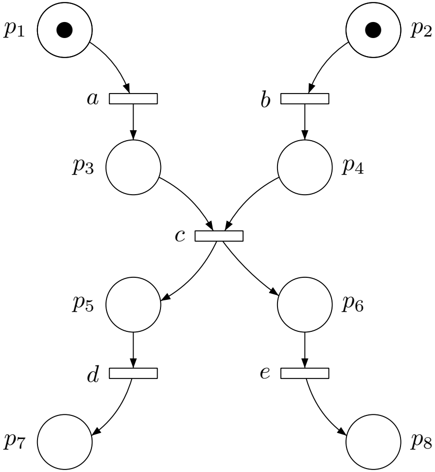
A control constraint Ψ is a set of pairs (state, transition) expressing which transitions are authorized in each state i.e. we assume the centralised control problem to be solved

Running example: priority policies

- a priority policy $\ll$ is a strict partial order on the transitions
- transition t has maximal priority (max_t) in state s if:
  - no transition t' such that $t \ll t'$ is enabled in s

Global constraints — here priorities

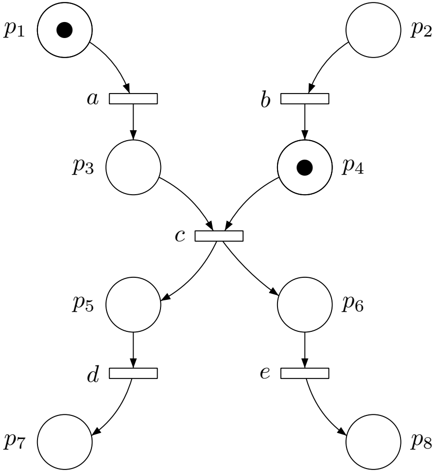
- a prioritized execution of is an execution such that for all $s_i \xrightarrow{t_i} s_{i+1}$, t_i has maximal priority in s_i



$a \ll b$ and $d \ll e$

Global constraints — here priorities

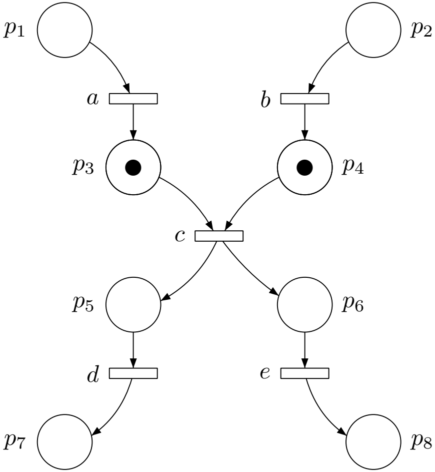
- a prioritized execution of is an execution such that for all $s_i \xrightarrow{t_i} s_{i+1}$, t_i has maximal priority in s_i



$a \ll b$ and $d \ll e$

Global constraints — here priorities

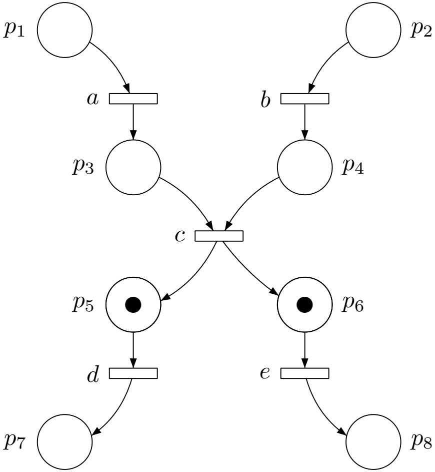
- a prioritized execution of is an execution such that for all $s_i \xrightarrow{t_i} s_{i+1}$, t_i has maximal priority in s_i



$a \ll b$ and $d \ll e$

Global constraints — here priorities

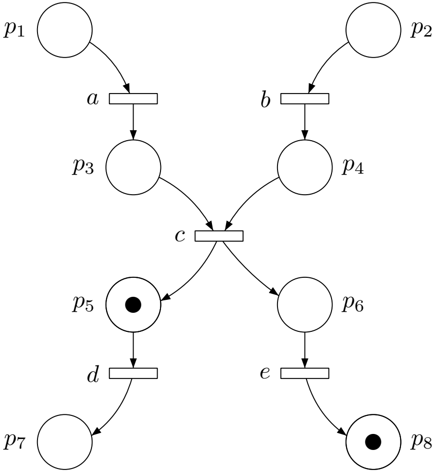
- a prioritized execution of is an execution such that for all $s_i \xrightarrow{t_i} s_{i+1}$, t_i has maximal priority in s_i



$a \ll b$ and $d \ll e$

Global constraints — here priorities

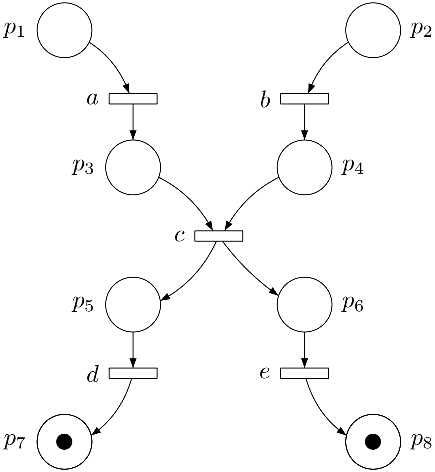
- a prioritized execution of is an execution such that for all $s_i \xrightarrow{t_i} s_{i+1}$, t_i has maximal priority in s_i



$a \ll b$ and $d \ll e$

Global constraints — here priorities

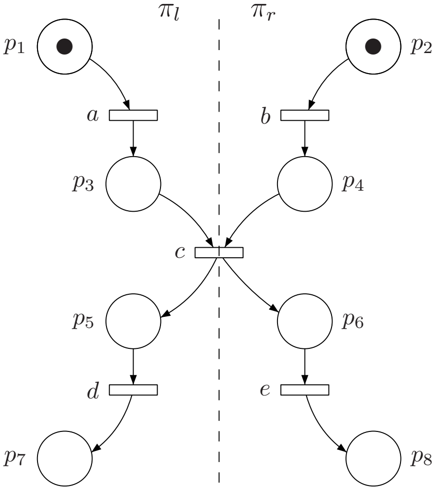
- a prioritized execution of is an execution such that for all $s_i \xrightarrow{t_i} s_{i+1}$, t_i has maximal priority in s_i
- *independent* transitions may not be independent any more (e.g. d, e in $\{p_5, p_6\}$)
- Question of [GPQ-CAV10]: can we transform the controlled system $(PN, \ll)$ into a Petrinet? which can be analyzed, implemented "as usually"



$a \ll b$ and $d \ll e$

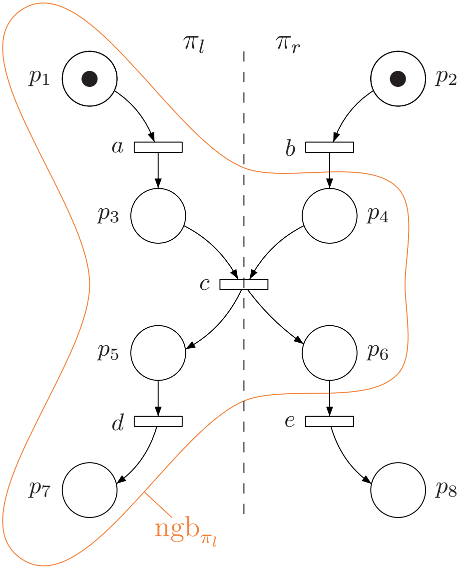
Compositional setting

- a process or thread π is a set of places $P_\pi \subseteq P$ (exactly 1 token) and the corresponding transitions $T_\pi \subseteq T$



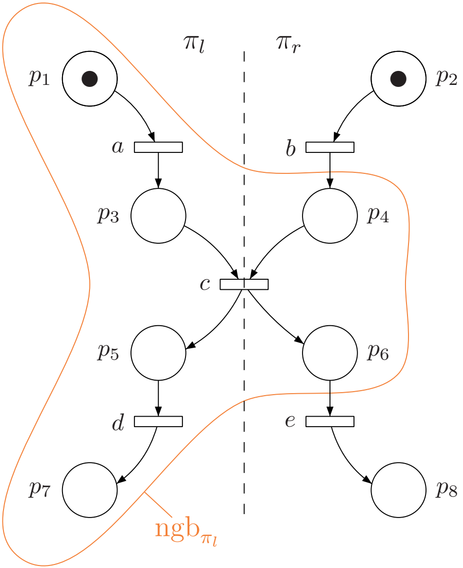
Compositional setting

- a process or thread π is a set of places $P_\pi \subseteq P$ (exactly 1 token) and the corresponding transitions $T_\pi \subseteq T$
- the neighborhood ngb_π of π is $\bigcup_{t \in T_\pi} ({}^\bullet t \cup t^\bullet)$
- the set of local states of π is $\{s \cap \mathrm{ngb}_\pi \mid s \in S\}$
  the local state corresponding to s is denoted $s|_\pi$



Compositional setting

- a process or thread π is a set of places $P_\pi \subseteq P$ (exactly 1 token) and the corresponding transitions $T_\pi \subseteq T$
- the neighborhood ngb_π of π is $\bigcup_{t \in T_\pi}({}^\bullet t \cup t^\bullet)$
- the set of local states of π is $\{s \cap \mathrm{ngb}_\pi \mid s \in S\}$ the local state corresponding to s is denoted $s|_\pi$



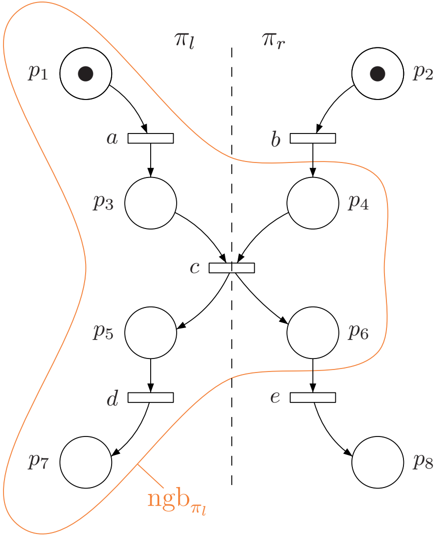
Definition of Knowledge

- π knows a property φ in a local $s|_\pi$ if φ holds in all reachable s' such that $s'|_\pi = s|_\pi$

$$s|_\pi \models K_\pi\varphi$$

- by extension: π knows φ in a global s if $s|_\pi \models K_\pi\varphi$

Stability Property:
if $s|_\pi \models K_\pi\varphi$, then
$s|_\pi \models K_\pi\varphi\,\mathcal{U}ntil\,\neg(s|_\pi)$

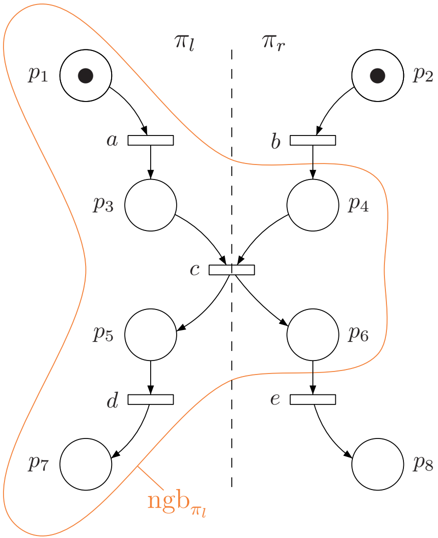


$a \ll b$ and $d \ll e$

Knowledge to support transitions

What are useful knowledge properties?

- transition a can be fired in s if $en_a \wedge \neg en_b$
- en_a is a local condition, always *known* in π_l: $s_{\pi_l} \models K_{\pi_l} en_a$ or $s_{\pi_l} \models K_{\pi_l} \neg en_a$
- are there local states s_{π_l} in which also $\neg en_b$ holds ?

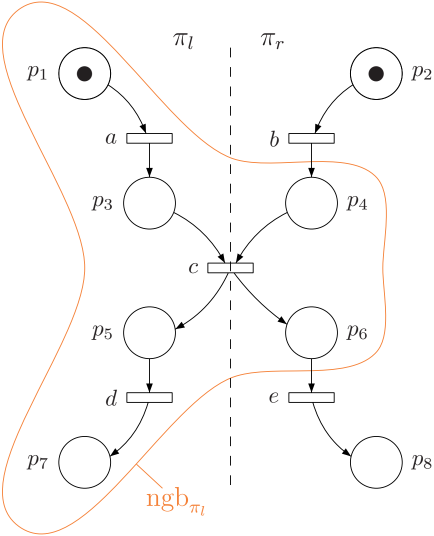


$a \ll b$ and $d \ll e$

Knowledge to support transitions

Useful knowledge for the example:

- $\{p_1\} \models K_{\pi_l} en_a$ but
  $\{p_1\} \not\models K_{\pi_l} \neg en_b$

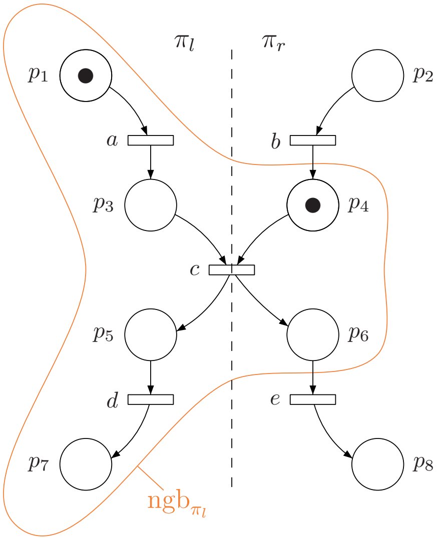


$a \ll b$ and $d \ll e$

Knowledge to support transitions

Useful knowledge for the example:

- $\{p_1\} \models K_{\pi_l} en_a$ but
  $\{p_1\} \not\models K_{\pi_l} \neg en_b$
- $\{p_1, p_4\} \models K_{\pi_l} en_a$ and
  $\{p_1, p_4\} \models K_{\pi_l} \neg en_b$

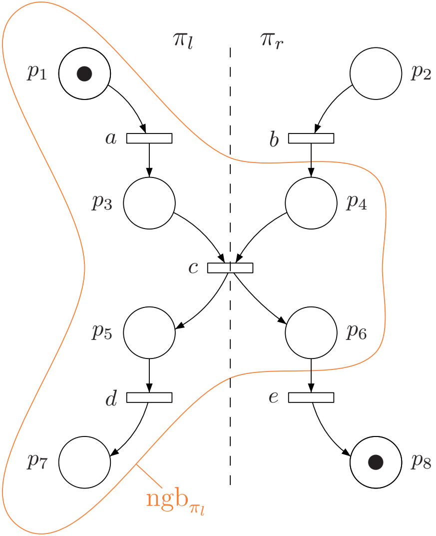


$a \ll b$ and $d \ll e$

Knowledge to support transitions

Useful knowledge for the example:

- $\{p_1\} \models K_{\pi_l} en_a$ but
  $\{p_1\} \not\models K_{\pi_l} \neg en_b$
- $\{p_1, p_4\} \models K_{\pi_l} en_a$ and
  $\{p_1, p_4\} \models K_{\pi_l} \neg en_b$
- $\{p_5, p_6\} \models K_{\pi_l} en_d$ but
  $\{p_5, p_6\} \models K_{\pi_l} en_e$
- $\{p_5\} \models K_{\pi_l} en_d$ and
  $\{p_5\} \models K_{\pi_l} \neg en_e$



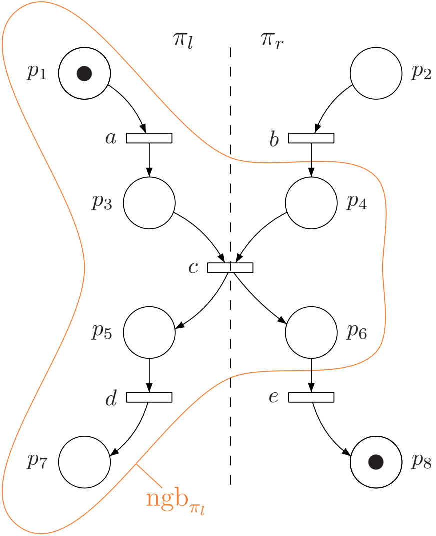
$a \ll b$ and $d \ll e$

Knowledge to support transitions

We can define a Petri net with local conditions for the controlled system $(PN, \ll)$. It is like PN, but

- allows a only in the local state $\{p_1, p_4\}$ of π_l
- allows d only in the local state $\{p_5\}$ of π_l

Allows compositional analysis: A (partial) state s knows en_t iff one of the threads involved in t knows en_t (disjunctive control)



$a \ll b$ and $d \ll e$

Outline

1. **Introduction**

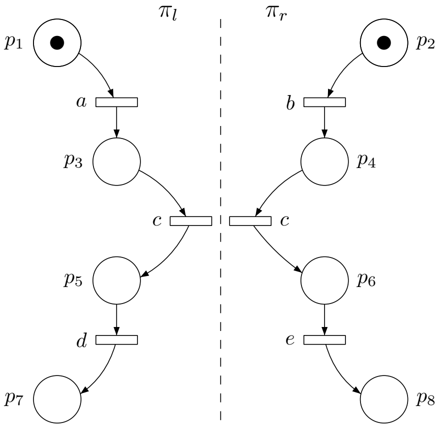
2. **Knowledge for Control**
   - One-safe Petri Nets *PN* and control constraints
   - Locality and knowledge
   - Knowledge for Control

3. **Knowledge for Distributed Implementation**
   - Distributed Setting: implementation relations
   - Knowledge Required in a Distributed Implementation
   - Knowledge and Communication

4. **Discussion**

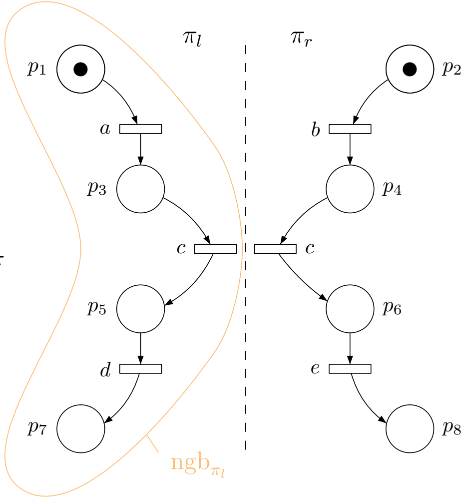
Distributed setting

- a process π is a set of places $P_\pi \subseteq P$ (exactly 1 token) and T_π contains for each transition in which π is involved, a corresponding local transition



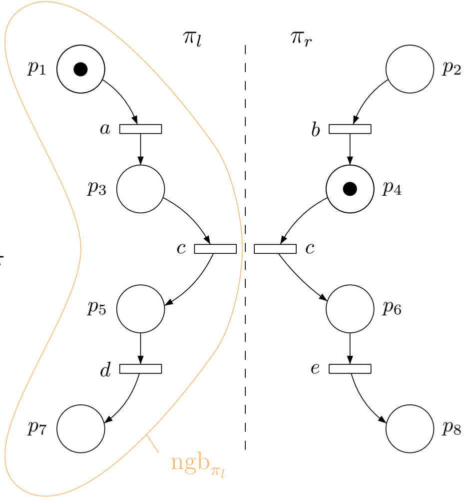
Distributed setting

- a process π is a set of places $P_\pi \subseteq P$ (exactly 1 token) and T_π contains for each transition in which π is involved, a corresponding local transition
- the neighborhood ngb_π of π is exactly the set of local places P_π



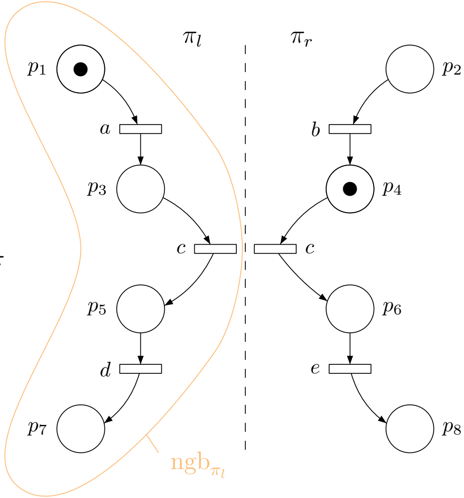
Distributed setting

- a process π is a set of places $P_\pi \subseteq P$ (exactly 1 token) and T_π contains for each transition in which π is involved, a corresponding local transition
- the neighborhood ngb_π of π is exactly the set of local places P_π



Distributed setting

- a process π is a set of places $P_\pi \subseteq P$ (exactly 1 token) and T_π contains for each transition in which π is involved, a corresponding local transition
- the neighborhood ngb_π of π is exactly the set of local places P_π
- everything else is unchanged

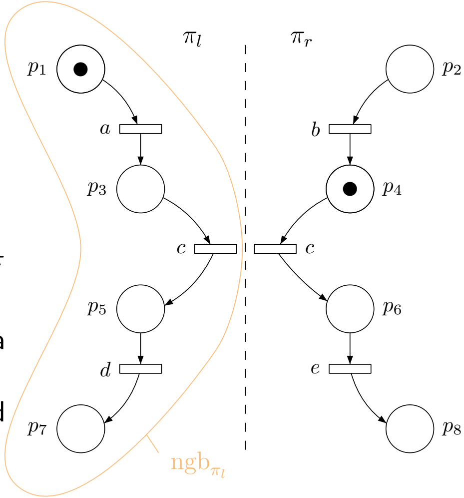


Distributed setting

- a process π is a set of places $P_\pi \subseteq P$ (exactly 1 token) and T_π contains for each transition in which π is involved, a corresponding local transition
- the neighborhood ngb_π of π is exactly the set of local places P_π
- everything else is unchanged

We have now a new Petri net with a **different transition set**.

Question: how to relate distributed and centralized executions ?



Implementation relations $\preceq$

$\preceq$ must support the methodology:

(1) verify φ on PN

(2) guarantee φ on I by construction

Implementation relations $\preceq$

$\preceq$ must support the methodology:

(1) verify φ on PN

(2) guarantee φ on I by construction

Minimal Requirements on $\preceq$: *sequential consistency*

(1) transition correctness (projection of global traces)

(2) atomicity (all π choose the same trace)

Implementation relations $\preceq$

$\preceq$ must support the methodology:

(1) verify φ on PN

(2) guarantee φ on I by construction

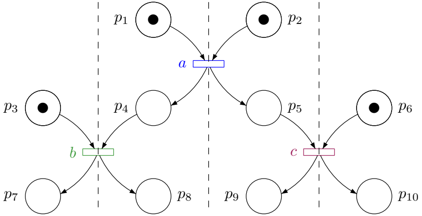
Minimal Requirements on $\preceq$: *sequential consistency*

(1) transition correctness (projection of global traces)
(2) atomicity (all π choose the same trace)

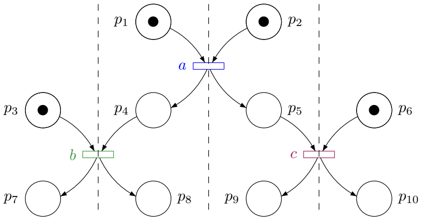
Additional Constraints:

(3) synchronization constraints (e.g. synchronize before/after joint transitions
(4) progress constraints
(5) constraints imposed by Ψ ($\ll$)

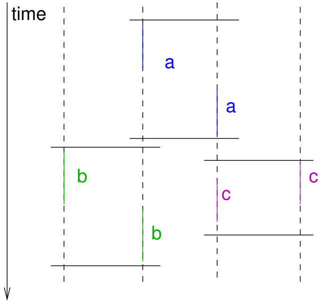
Illustrating Implementation relations



Illustrating Implementation relations

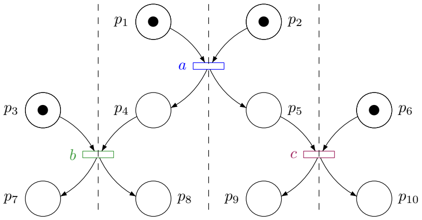


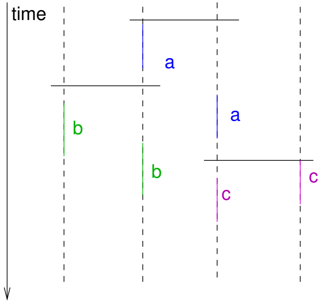
$\preceq_{ss}$: requires synchronization before and after transitions



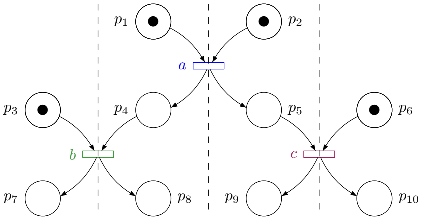
Illustrating Implementation relations

$\preceq$: requires synchronization only before transitions

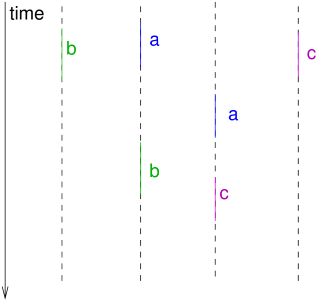




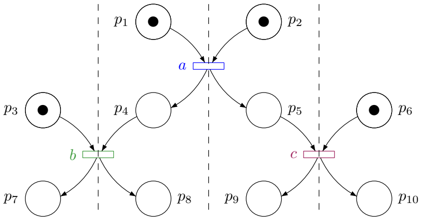
Illustrating Implementation relations



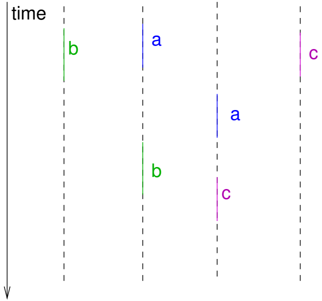
$\preceq_{ns}$: requires no synchronization



Illustrating Implementation relations



$\preceq_{ns}$: requires no synchronization



Even to achieve this loosest implementation relation, one has to *control* local processes

Knowledge for enabling transitions

For $\preceq$, the enabling condition go_t^{π} for a local transition t:

1. t is enabled (in the sense of Petri net) or already partially executed:

$$in_t = \forall\pi' \in proc(t) \,.\, (en_t^{\pi'} \vee done_t^{\pi'})$$

Knowledge for enabling transitions

For $\preceq_{ss}$ and $\preceq$, the enabling condition go_t^π for a local transition t:

1. t is enabled (in the sense of Petri net) or already partially executed:
$$in_t = \forall \pi' \in proc(t) \,.\, (en_t^{\pi'} \vee done_t^{\pi'})$$
2. transitions t' preceding t (in PN) are terminated in all π':
$$\forall t'.(done_{t'}^{\pi} \implies done_{t'}) \; where \; done_{t'} = \forall \pi' \in t' \,.\, done_{t'}^{\pi'}$$

Knowledge for enabling transitions

For $\preceq$, the enabling condition go_t^{π} for a local transition t:

1. t is enabled (in the sense of Petri net) or already partially executed:

$$in_t = \forall \pi' \in proc(t) \ . \ (en_t^{\pi'} \vee done_t^{\pi'})$$

2. t has maximal priority:

$$max_t = \forall t' \ . \ (t \ll t' \implies \neg en_{t'})$$

Knowledge for enabling transitions

For $\preceq$, the enabling condition go_t^{π} for a local transition t:

1. t is enabled (in the sense of Petri net) or already partially executed:

$$in_t = \forall \pi' \in proc(t) \; . \; (en_t^{\pi'} \vee done_t^{\pi'})$$

2. t has maximal priority:

$$max_t = \forall t' \; . \; (t \ll t' \implies \neg en_{t'})$$

3. t has no unresolved conflict:

Knowledge for enabling transitions

For $\preceq$, the enabling condition go_t^{π} for a local transition t:

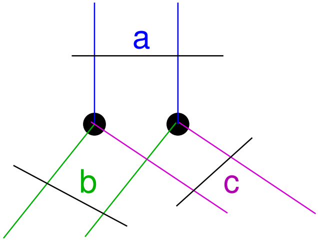
1. t is enabled (in the sense of Petri net) or already partially executed:

$$in_t = \forall \pi' \in proc(t) \,.\, (en_t^{\pi'} \vee done_t^{\pi'})$$

2. t has maximal priority:

$$max_t = \forall t' \,.\, (t \ll t' \implies \neg en_{t'})$$

3. t has no unresolved conflict:



Knowledge for enabling transitions

For $\preceq$, the enabling condition go_t^π for a local transition t:

1. t is enabled (in the sense of Petri net) or already partially executed:

$$in_t = \forall \pi' \in proc(t) \,.\, (en_t^{\pi'} \vee done_t^{\pi'})$$

2. t has maximal priority:

$$max_t = \forall t' \,.\, (t \ll t' \implies \neg en_{t'})$$

3. t has no unresolved conflict:

$$select_t \implies \forall t' \,.\, (\text{potentially in conflict with } t \implies \neg select_{t'})$$

Knowledge for enabling transitions

For $\preceq$, the enabling condition go_t^π for a local transition t:

1. t is enabled (in the sense of Petri net) or already partially executed:

$$in_t = \forall \pi' \in proc(t) \;.\; (en_t^{\pi'} \vee done_t^{\pi'})$$

2. t has maximal priority:

$$max_t = \forall t' \;.\; (t \ll t' \implies \neg en_{t'})$$

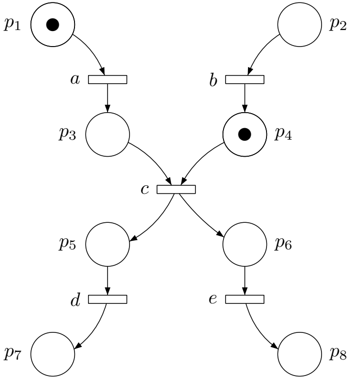
3. t has no unresolved conflict:

$$select_t \implies \forall t' \;.\; (\text{potentially in conflict with } t \implies \neg select_{t'})$$

π must know go_t^π and
π must also know that $\pi' \in proc(t)$ knows or will know $go_t^{\pi'}$

Petri net Knowledge preserved in a distributed setting

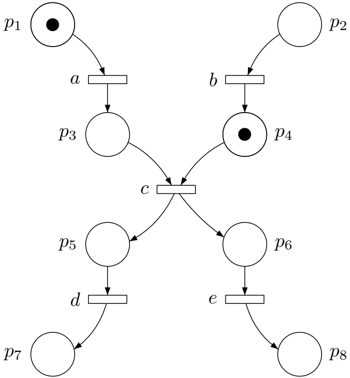
Can we use the knowledge computed on the Petri net ?



$a \ll b$ and $d \ll e$

Petri net Knowledge preserved in a distributed setting

Can we use the knowledge computed on the Petri net ?



$a \ll b$ and $d \ll e$

What can and cannot be preserved:

- non enabledness of transitions (useful for priorities)
- we can transform Petri net knowledge by weakening it with the incertainty induced by $\preceq$
- impossible: knowledge for achieving synchronization

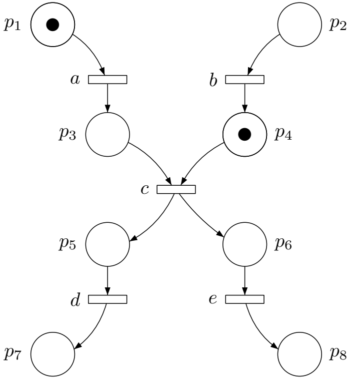
Petri net Knowledge preserved in a distributed setting

Can we use the knowledge computed on the Petri net ?

What can and cannot be preserved:

- non enabledness of transitions (useful for priorities)
- we can transform Petri net knowledge by weakening it with the incertainty induced by $\preceq$
- impossible: knowledge for achieving synchronization

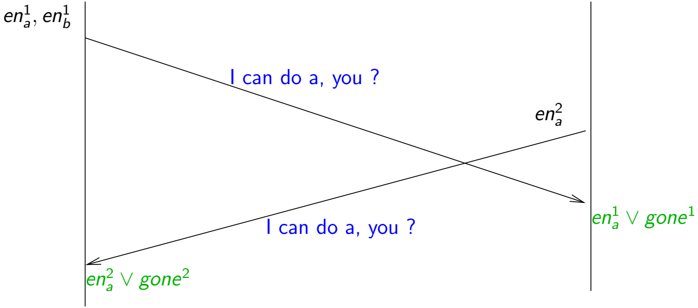
Conclusion: to achieve synchronization, one must communicate



$a \ll b$ and $d \ll e$

Knowledge through communication

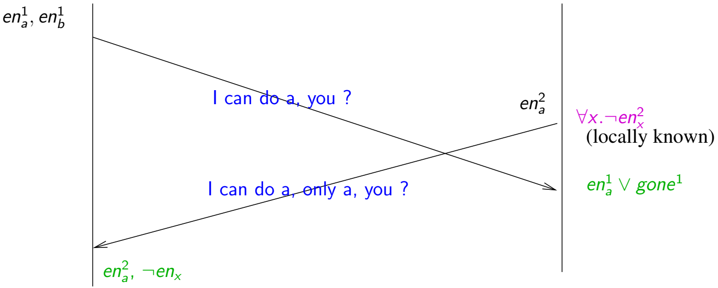
A typical protocol for achieving distributed implementation:



- no information gained – also not with a negative response

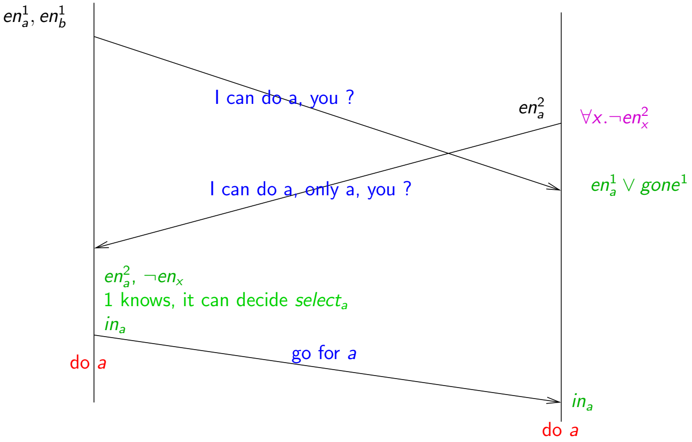
Knowledge through communication

A typical protocol for achieving distributed implementation:



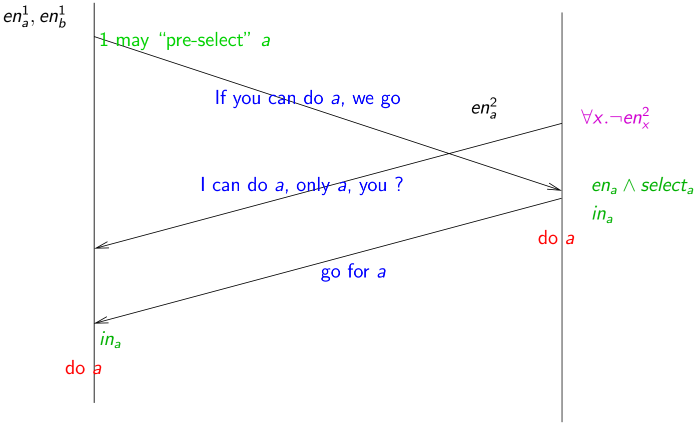
- process 1 can no decide to $select_a$ (if not yet engaged for b)

Knowledge through communication



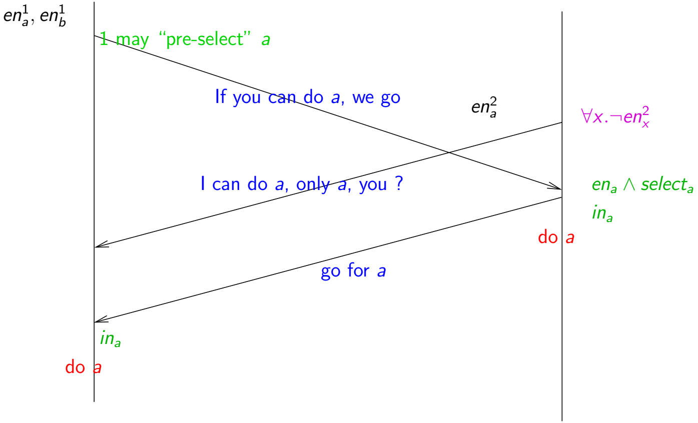
- conveye information providing stronger knowledge — when possible (e.g. information about absence of conflict)

Knowledge through communication



- try to resolve conflicts early

Knowledge through communication



- try to resolve conflicts early
- avoid requesting knowledge that is already available

Discussion

I hope I could convince you that knowledge is a useful tool for reasoning about distribution

Perspectives

- take into account data, timing, ... (discrete and continuous)
- formulate platform characteristics in terms of knowledge
- devise modular proofs for distribution strategies