SEMANTICS OF A CONCURRENT LANGUAGE BY MEANS OF DIRECTED TOPOLOGY

Habilitation Defense

Friday the 30th of September 2016

Emmanuel Haucourt

ÉCOLE POLYTECHNIQUE
UNIVERSITÉ PARIS-SACLAY

Summary

Summary

Summary

Summary

Summary

Summary

Summary

Summary

Summary

1. The Language

Targeted software and strategy

- Fine-grained parallel programs (e.g. asynchronous control command systems).

Targeted software and strategy

Program analysis

Targeted software and strategy

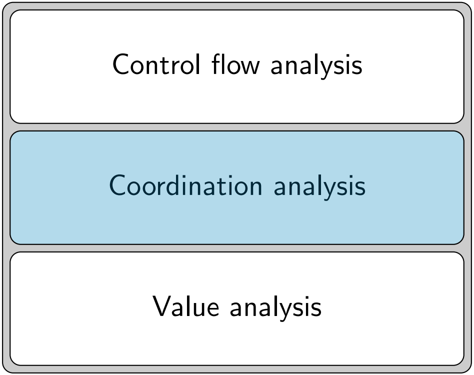
Control flow analysis

Targeted software and strategy

Control flow analysis

Value analysis

Targeted software and strategy



Features of the language

Dijkstra, E.W., *Cooperating sequential processes*, 1968.

Features of the language

Dijkstra, E.W., *Cooperating sequential processes*, 1968.

- shared memory abstract machine (PRAM)
  concurrent read exclusive write (CREW)
- no pointer arithmetics
- no function
- no birth nor death of process at runtime
- tokens are *owned* by processes
- *conservative* processes

Standard examples

```
sem:  1 a
proc:
  p = P(a); V(a)

init:  2p
```

```
var:  x = 0
proc:
  p1 = x:=1 ,
  p2 = x:=2
init:  p1 p2
```

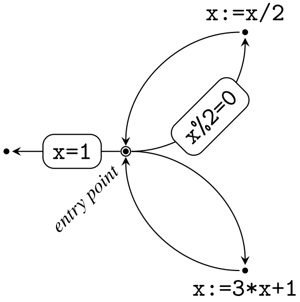
Middle-end representation

- $\mathcal{P} = \{\text{process identifiers}\}$,
  $\mathcal{V} = \{\text{variables}\}$,
  $\mathcal{S} = \{\text{semaphores}\}$,
  $\mathcal{B} = \{\text{barriers}\}$
- init : $\mathcal{V} \to \mathbb{R}$
- arity : $\mathcal{S} \sqcup \mathcal{B} \to \mathbb{N} \cup \{\infty\}$
- proc : $\mathcal{P} \to \{\text{control flow graphs}\}$

Control flow graphs

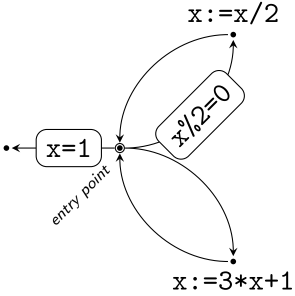
Floyd, R. W., *Assigning meanings to programs*, 1967
Allen, F. E., *Control flow analysis*, 1970

```
var:  x = 7
proc:
p = J(q)+[x<>1]+() ,
q = (x:=x/2; J(p))+[x % 2 = 0]+
           (x:=3*x+1; J(p))
init:  p
```



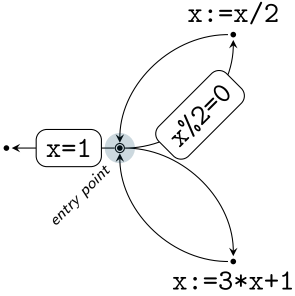
An Execution Trace

on a control flow graph



An Execution Trace

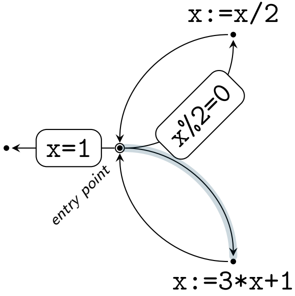
on a control flow graph



the current value of x is 7

An Execution Trace

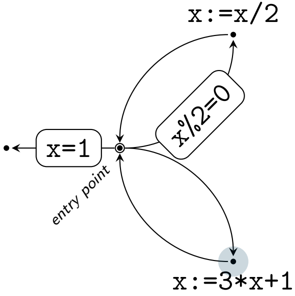
on a control flow graph



the current value of x is 7

An Execution Trace

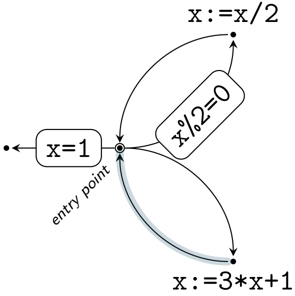
on a control flow graph



the current value of `x` is 22

An Execution Trace

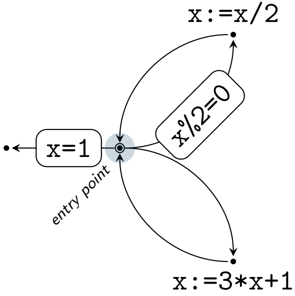
on a control flow graph



the current value of `x` is 22

An Execution Trace

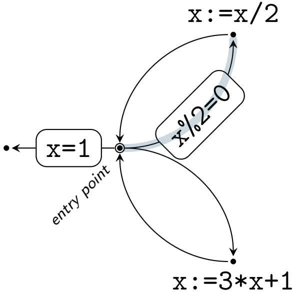
on a control flow graph



the current value of x is 22

An Execution Trace

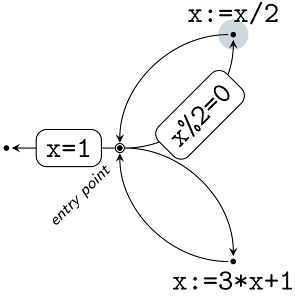
on a control flow graph



the current value of x is 22

An Execution Trace

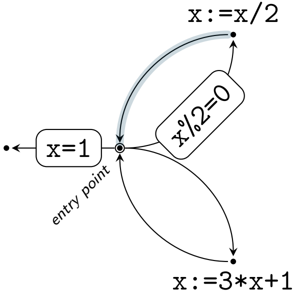
on a control flow graph



the current value of x is 11

An Execution Trace

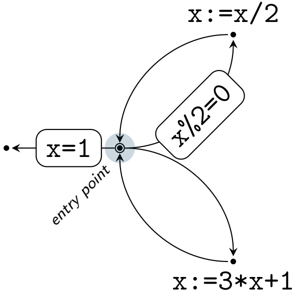
on a control flow graph



the current value of `x` is 11

An Execution Trace

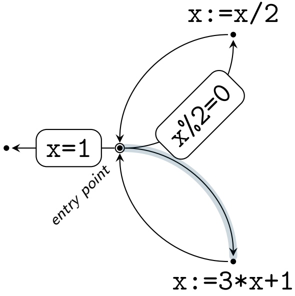
on a control flow graph



the current value of x is 11

An Execution Trace

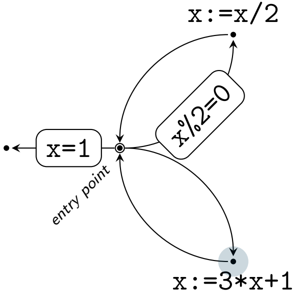
on a control flow graph



the current value of x is 11

An Execution Trace

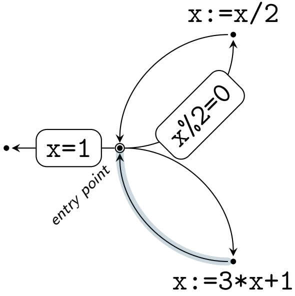
on a control flow graph



the current value of x is 34

An Execution Trace

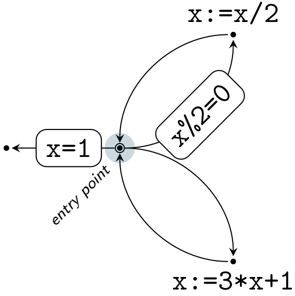
on a control flow graph



the current value of x is 34

An Execution Trace

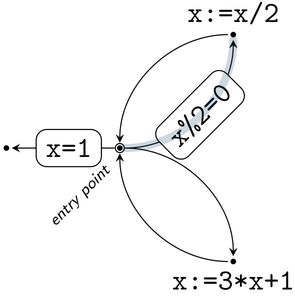
on a control flow graph



the current value of x is 34

An Execution Trace

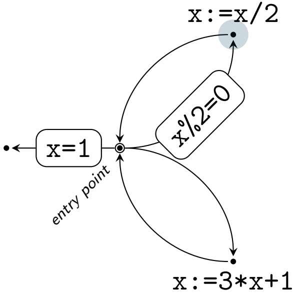
on a control flow graph



the current value of `x` is 34

An Execution Trace

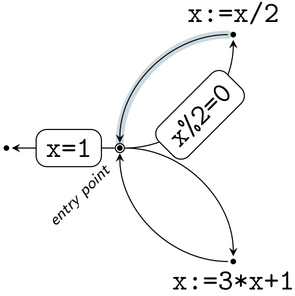
on a control flow graph



the current value of x is 17

An Execution Trace

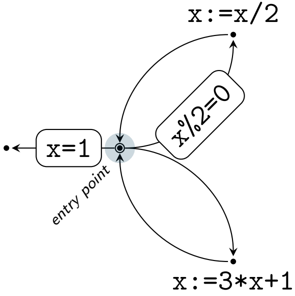
on a control flow graph



the current value of x is 17

An Execution Trace

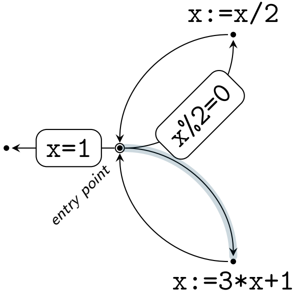
on a control flow graph



the current value of x is 17

An Execution Trace

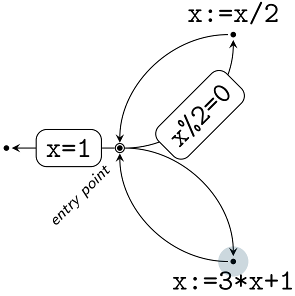
on a control flow graph



the current value of x is 17

An Execution Trace

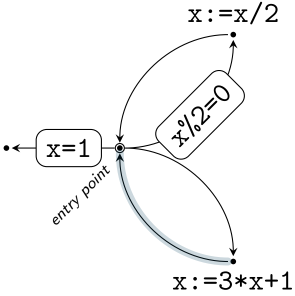
on a control flow graph



the current value of x is 52

An Execution Trace

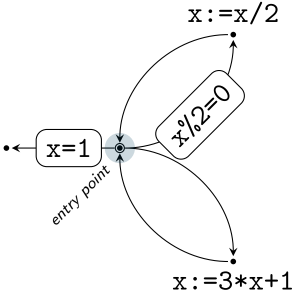
on a control flow graph



the current value of `x` is 52

An Execution Trace

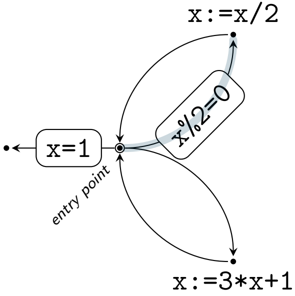
on a control flow graph



the current value of `x` is 52

An Execution Trace

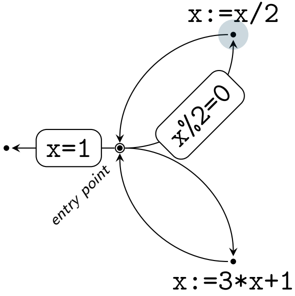
on a control flow graph



the current value of x is 52

An Execution Trace

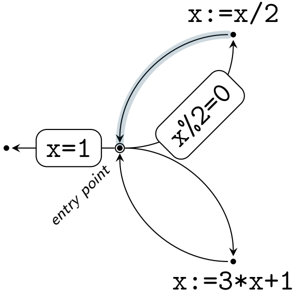
on a control flow graph



the current value of x is 26

An Execution Trace

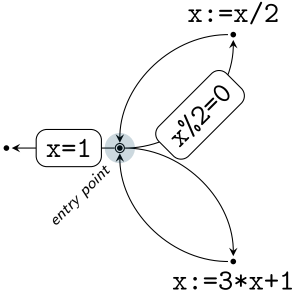
on a control flow graph



the current value of `x` is 26

An Execution Trace

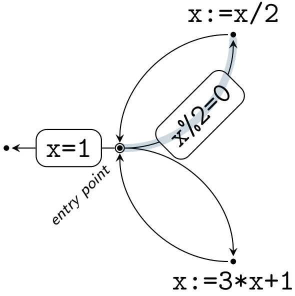
on a control flow graph



the current value of x is 26

An Execution Trace

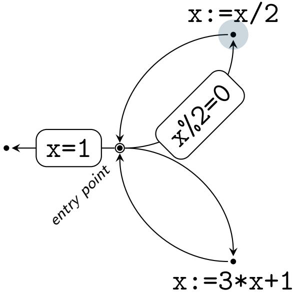
on a control flow graph



the current value of x is 26

An Execution Trace

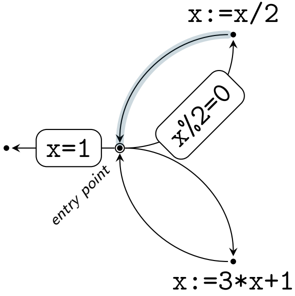
on a control flow graph



the current value of x is 13

An Execution Trace

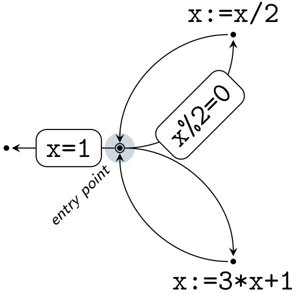
on a control flow graph



the current value of `x` is 13

An Execution Trace

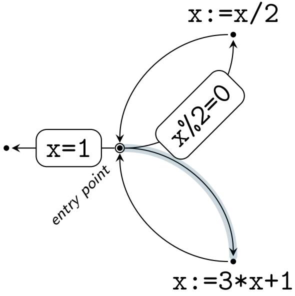
on a control flow graph



the current value of x is 13

An Execution Trace

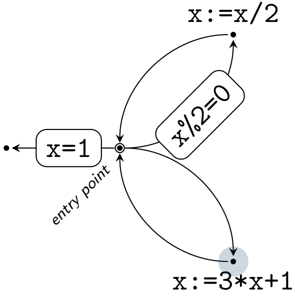
on a control flow graph



the current value of `x` is 13

An Execution Trace

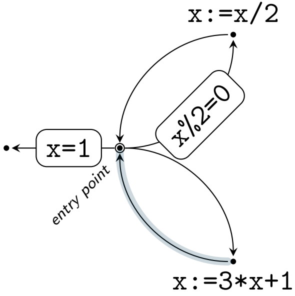
on a control flow graph



the current value of x is 40

An Execution Trace

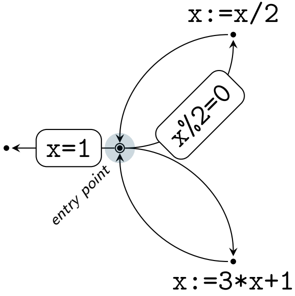
on a control flow graph



the current value of x is 40

An Execution Trace

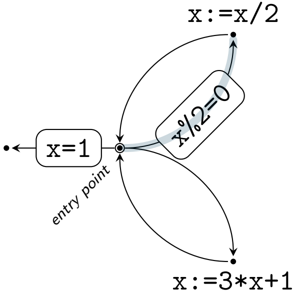
on a control flow graph



the current value of x is 40

An Execution Trace

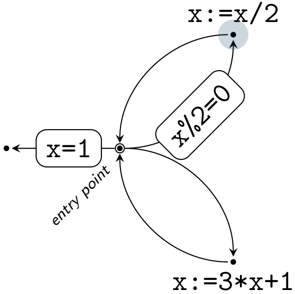
on a control flow graph



the current value of x is 40

An Execution Trace

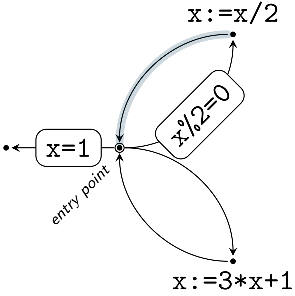
on a control flow graph



the current value of `x` is 20

An Execution Trace

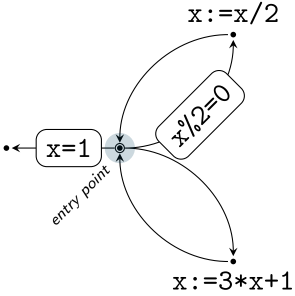
on a control flow graph



the current value of x is 20

An Execution Trace

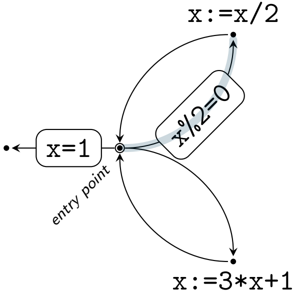
on a control flow graph



the current value of `x` is 20

An Execution Trace

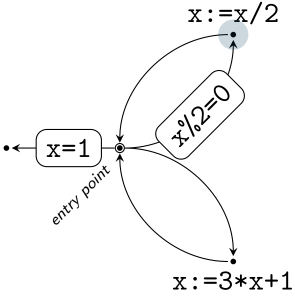
on a control flow graph



the current value of `x` is 20

An Execution Trace

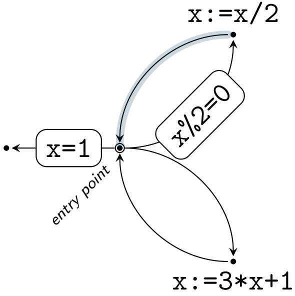
on a control flow graph



the current value of x is 10

An Execution Trace

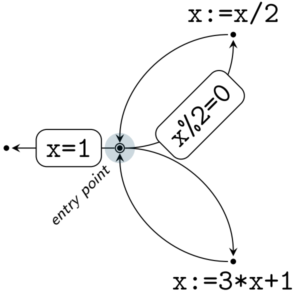
on a control flow graph



the current value of x is 10

An Execution Trace

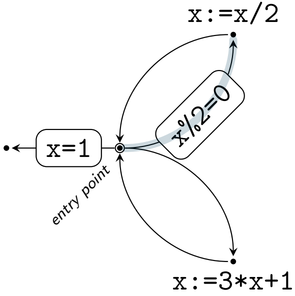
on a control flow graph



the current value of x is 10

An Execution Trace

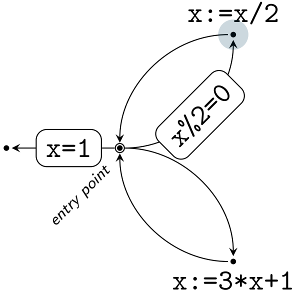
on a control flow graph



the current value of `x` is 10

An Execution Trace

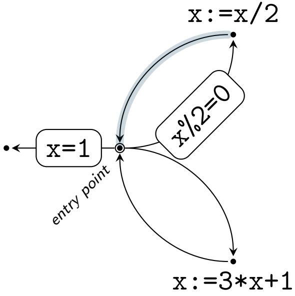
on a control flow graph



the current value of x is 5

An Execution Trace

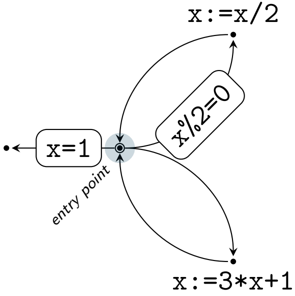
on a control flow graph



the current value of x is 5

An Execution Trace

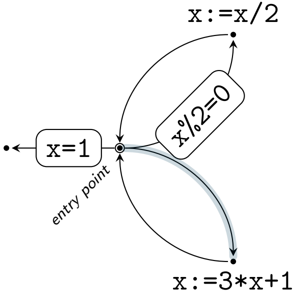
on a control flow graph



the current value of x is 5

An Execution Trace

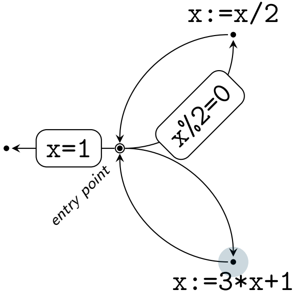
on a control flow graph



the current value of x is 5

An Execution Trace

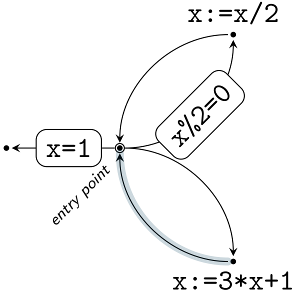
on a control flow graph



the current value of x is 16

An Execution Trace

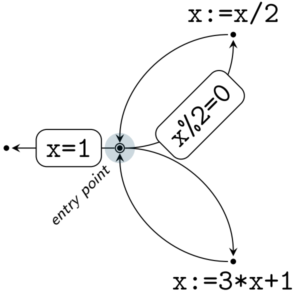
on a control flow graph



the current value of x is 16

An Execution Trace

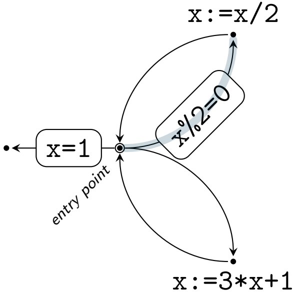
on a control flow graph



the current value of `x` is 16

An Execution Trace

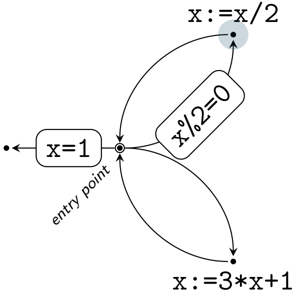
on a control flow graph



the current value of `x` is 16

An Execution Trace

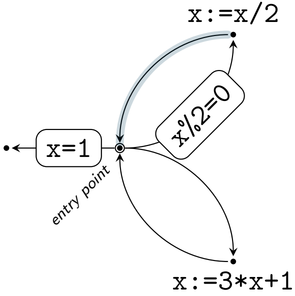
on a control flow graph



the current value of x is 8

An Execution Trace

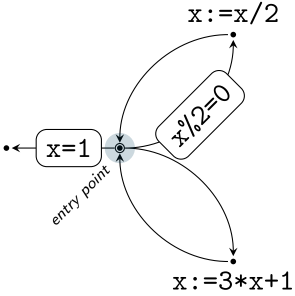
on a control flow graph



the current value of x is 8

An Execution Trace

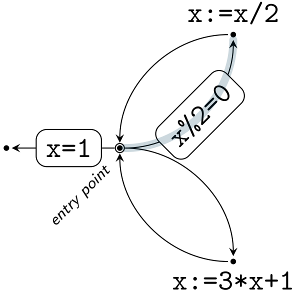
on a control flow graph



the current value of x is 8

An Execution Trace

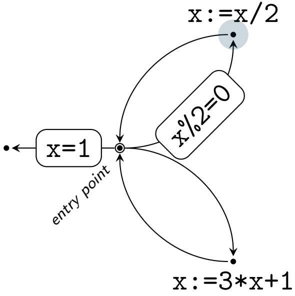
on a control flow graph



the current value of x is 8

An Execution Trace

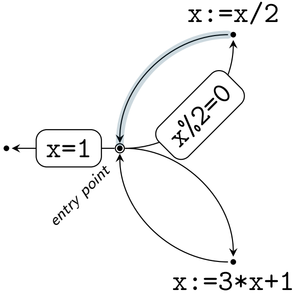
on a control flow graph



the current value of x is 4

An Execution Trace

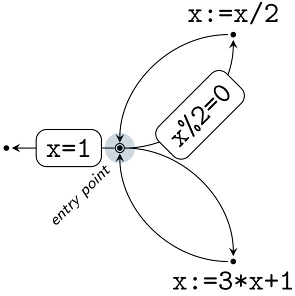
on a control flow graph



the current value of x is 4

An Execution Trace

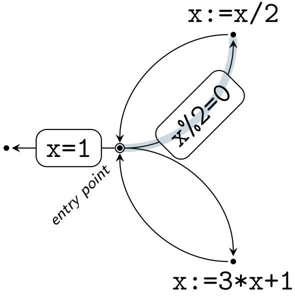
on a control flow graph



the current value of `x` is 4

An Execution Trace

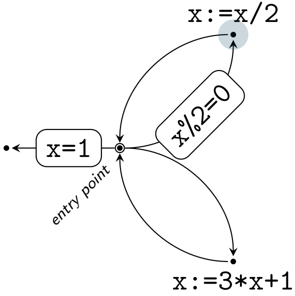
on a control flow graph



the current value of x is 4

An Execution Trace

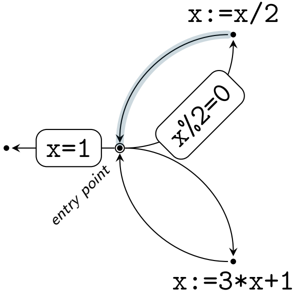
on a control flow graph



the current value of x is 2

An Execution Trace

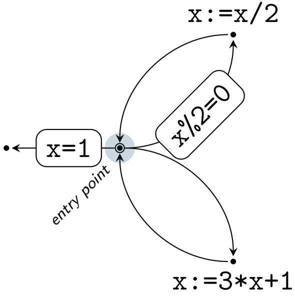
on a control flow graph



the current value of x is 2

An Execution Trace

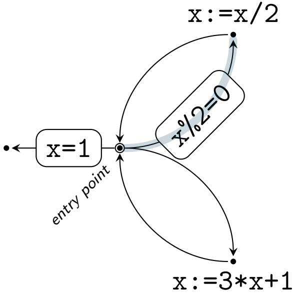
on a control flow graph



the current value of x is 2

An Execution Trace

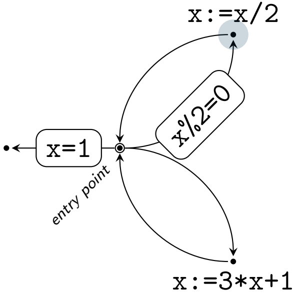
on a control flow graph



the current value of x is 2

An Execution Trace

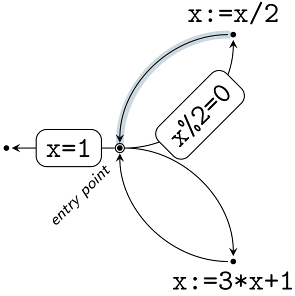
on a control flow graph



the current value of x is 1

An Execution Trace

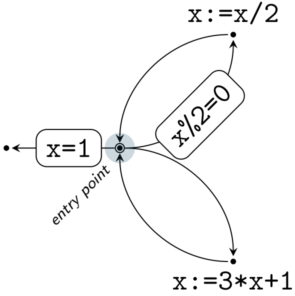
on a control flow graph



the current value of x is 1

An Execution Trace

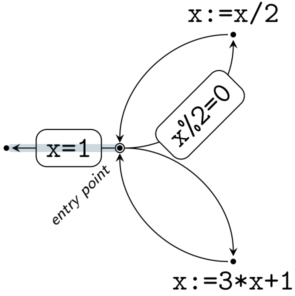
on a control flow graph



the current value of x is 1

An Execution Trace

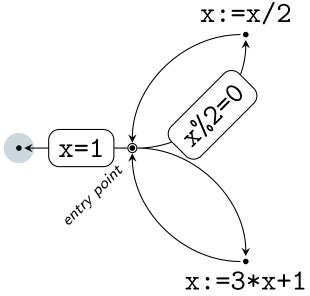
on a control flow graph



the current value of x is 1

An Execution Trace

on a control flow graph



the current value of x is 1

2. Abstract Machine

State (Assume that $\mathcal{P} = \{1, \ldots, N\}$)

point: a tuple $(p_1, \ldots, p_N)$ s.t. each p_n is either a vertex or an arrow of the n^{th} control flow graph.

State (Assume that $\mathcal{P} = \{1, \ldots, N\}$)

point: a tuple $(p_1, \ldots, p_N)$ s.t. each p_n is either a vertex or an arrow of the n^{th} control flow graph.

context: mapping σ defined over $\mathcal{V} \sqcup \mathcal{S}$ s.t.

- for all $v \in \mathcal{V}$, $\sigma(v) \in \mathbb{R}$, and
- for all $s \in \mathcal{S}$, $\sigma(s)$ is a multiset over $\mathcal{P}$.

We denote by σ_0 the initial context of a program.

state: a point and a context.

Multi-instruction

Cattani, G. L., and Sassone, V., *Higher dimensional transition systems*, 1996
Assume $\mathcal{P} = \{1, \ldots, N\}$

multi-instruction: a partial map μ from $\mathcal{P}$ to {single instructions}

Multi-instruction

Cattani, G. L., and Sassone, V., *Higher dimensional transition systems*, 1996
Assume $\mathcal{P} = \{1, \ldots, N\}$

multi-instruction: a partial map μ from $\mathcal{P}$ to {single instructions}

μ *admissible* in the context σ:

- $\mu(i)$ and $\mu(j)$ do not conflict
- for all $s \in \mathcal{S}$, $|\sigma(s)| + \text{card}\{i \in M \mid \mu(i) = \mathtt{P}(s)\} \leqslant \text{arity}(s)$
- for all $b \in \mathcal{B}$, $\text{card}\{i \in M \mid \mu(i) = \mathtt{W}(b)\} \notin \{1, \ldots, \text{arity}(b)\}$

The context $\sigma \cdot \mu$ is the result of the execution of μ in the context σ.

Paths

path: a sequence of points $p(0), \ldots, p(K)$ s.t. $\forall k \in \{1, \ldots, K\}$ one has

$$\textit{execution: } \forall n \in D_k \; \partial^+ p_n(k-1) = p_n(k)$$

or

$$\textit{branching: } \forall n \in D_k \; p_n(k-1) = \partial^- p_n(k)$$

where $D_k = \left\{ n \in \{1, \ldots, N\} \;\middle|\; p_n(k-1) \neq p_n(k) \right\}$

Execution path

Each path is associated with a sequence of multi-instructions (μ_k) where $\mu_k(n) = \lambda_n(p_n(k))$ for all n such that

$$p_n(k) = \partial^+ p_n(k-1) \text{ or } \lambda_n(p_n(k)) = W(\_)$$

The path is said to be admissible when μ_{k+1} is admissible in the context $\sigma_0 \cdot \mu_1 \cdots \mu_k$ for all k.

It is an execution path when for all k, n :

$$\partial^- p_n(k) = p_n(k-1) \text{ implies } [\![\lambda_n(p_n(k))]\!]_{\sigma \cdot \mu_0 \cdots \mu_{k-1}} \neq 0$$

3. Higher Dimensional Control Flow Structure

Encoding admissibility in a model

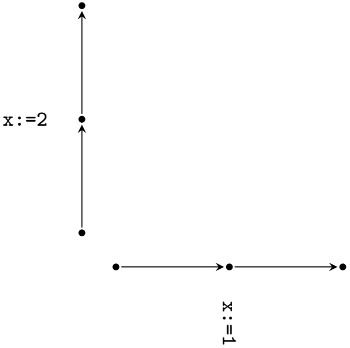
Race condition

write-write conflict

```
var:  x = 0
proc:
   p1 = x := 1,
   p2 = x := 2
init:  p1 p2
```

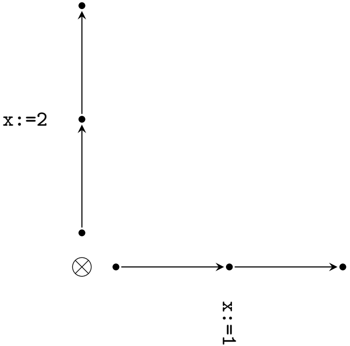
Exhaustive model

tensor product of graphs



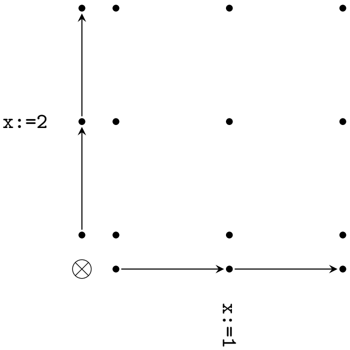
Exhaustive model

tensor product of graphs



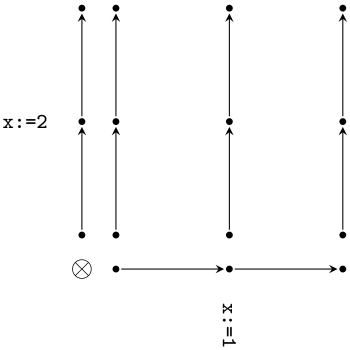
Exhaustive model

tensor product of graphs



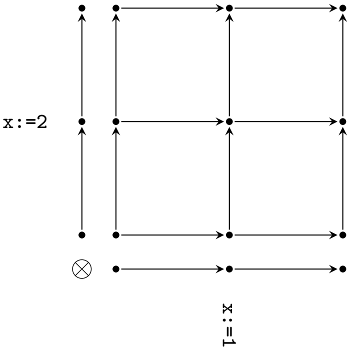
Exhaustive model

tensor product of graphs



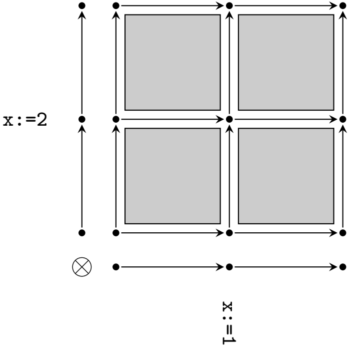
Exhaustive model

tensor product of graphs



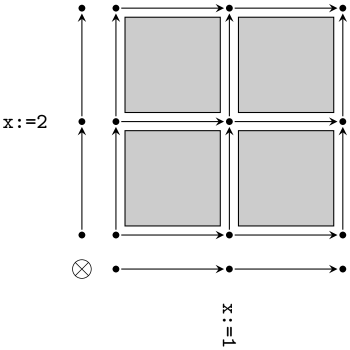
Exhaustive model

tensor product of graphs



Not admissible path

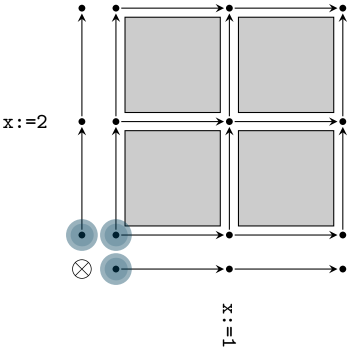
due to race condition



the value of `x` is 0

Not admissible path

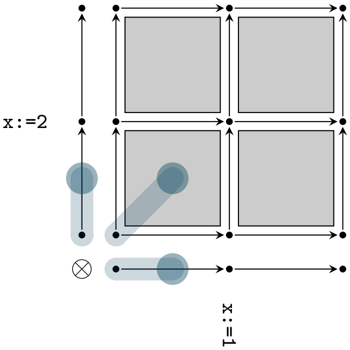
due to race condition



the value of `x` is 0

Not admissible path

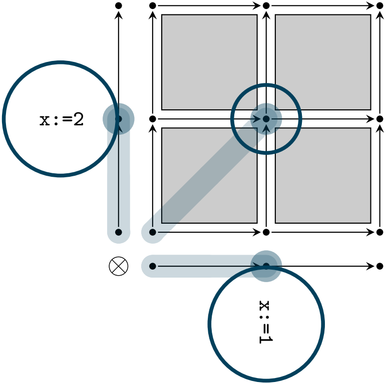
due to race condition



the value of $\mathtt{x}$ is 0

Not admissible path

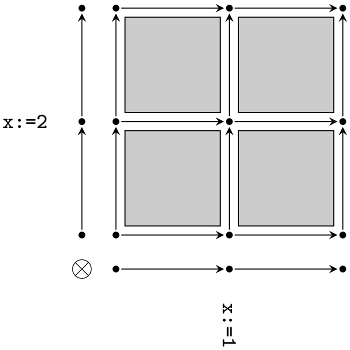
due to race condition



the value of `x` is ?

Admissible path

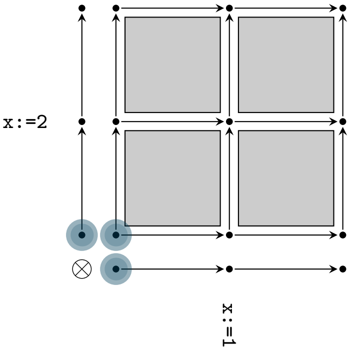
that however meets a forbidden point



the value of x is 0

Admissible path

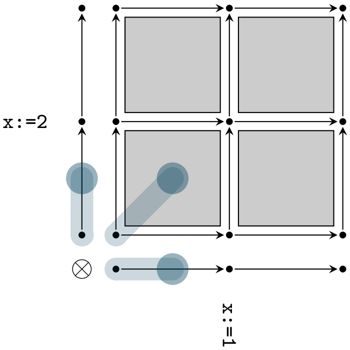
that however meets a forbidden point



the value of x is 0

Admissible path

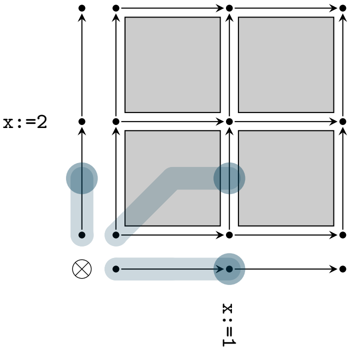
that however meets a forbidden point



the value of x is 0

Admissible path

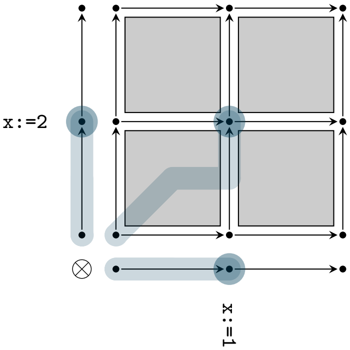
that however meets a forbidden point



the value of x is 1

Admissible path

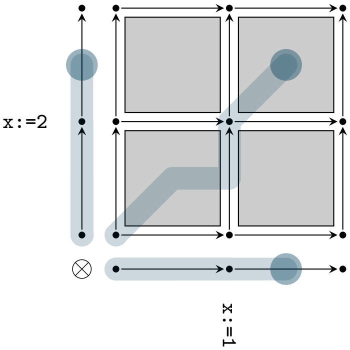
that however meets a forbidden point



the value of x is 2

Admissible path

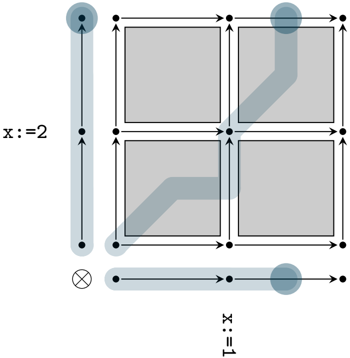
that however meets a forbidden point



the value of x is 2

Admissible path

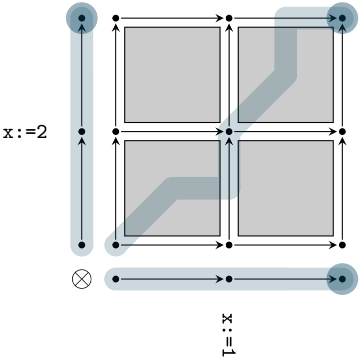
that however meets a forbidden point



the value of x is 2

Admissible path

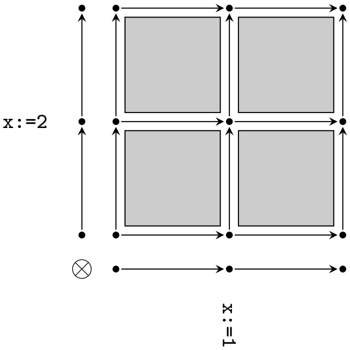
that however meets a forbidden point



the value of `x` is 2

Admissible path

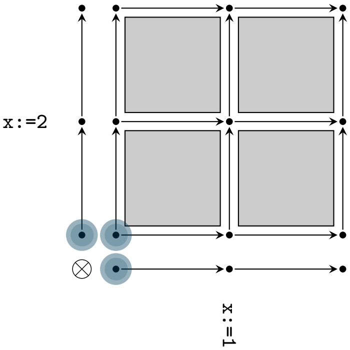
avoiding forbidden points



the value of x is 0

Admissible path

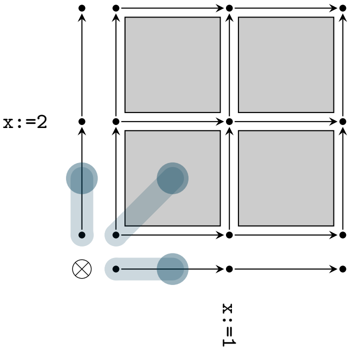
avoiding forbidden points



the value of x is 0

Admissible path

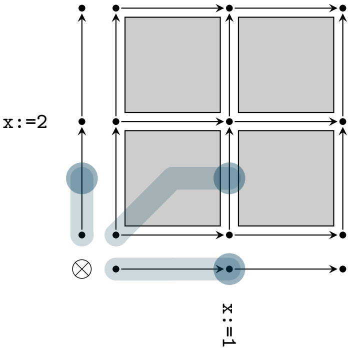
avoiding forbidden points



the value of x is 0

Admissible path

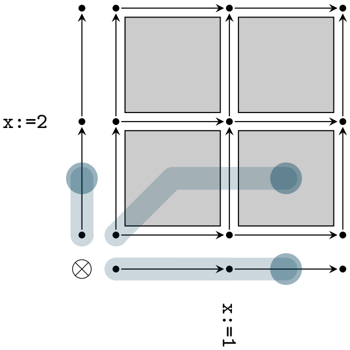
avoiding forbidden points



the value of x is 1

Admissible path

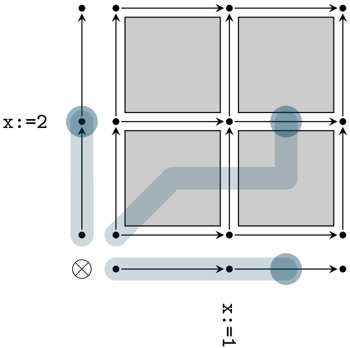
avoiding forbidden points



the value of x is 1

Admissible path

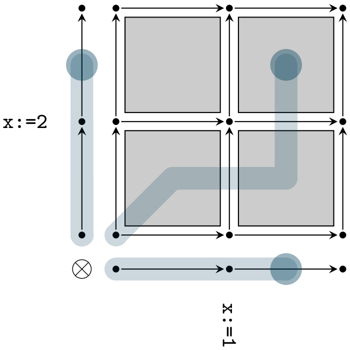
avoiding forbidden points



the value of x is 2

Admissible path

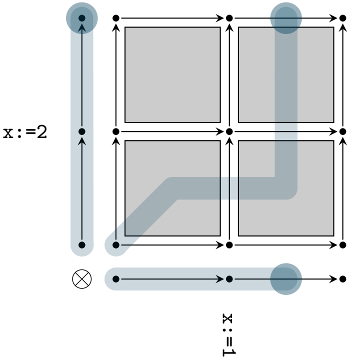
avoiding forbidden points



the value of x is 2

Admissible path

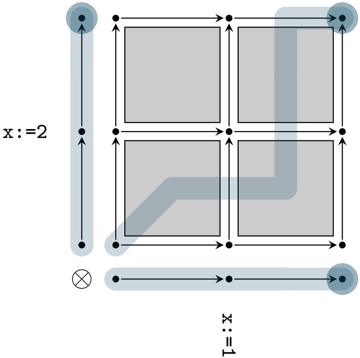
avoiding forbidden points



the value of x is 2

Admissible path

avoiding forbidden points



the value of `x` is 2

The replacement property

for admissible paths

Replacement:

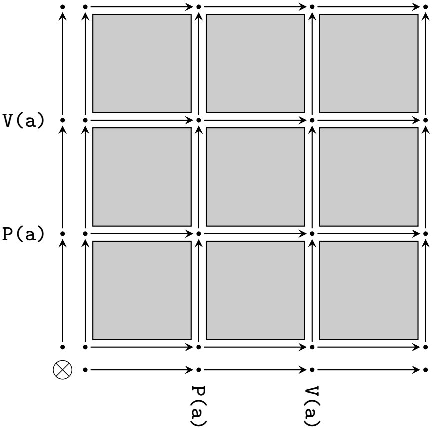
Any admissible path that meets a race condition is “equivalent” to an admissible path which avoids all of them.

One token for two processes

```
sem:  1 a
proc:
   p = P(a);V(a)
init:  2p
```

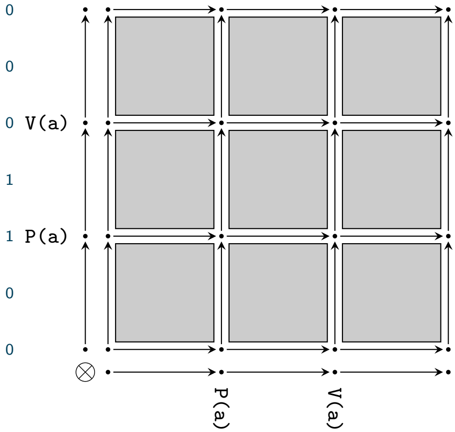
Discrete model

sem: 1 a



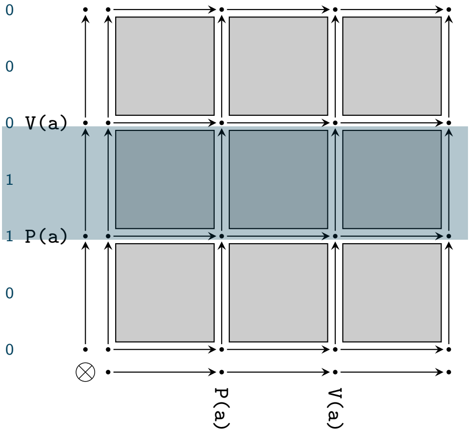
Discrete model

sem: 1 a



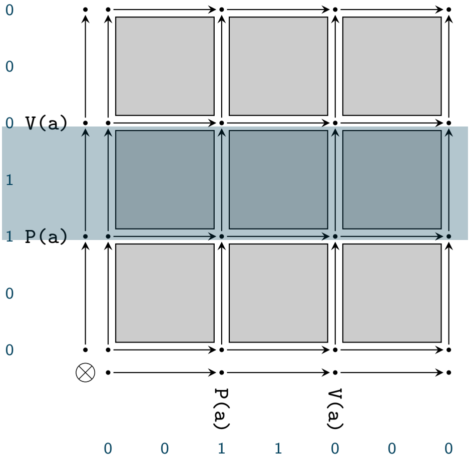
Discrete model

sem: 1 a



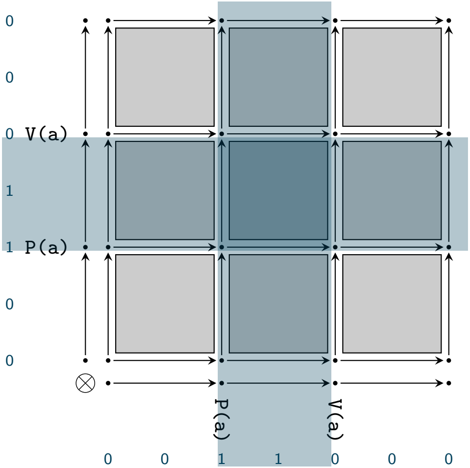
Discrete model

`sem: 1 a`



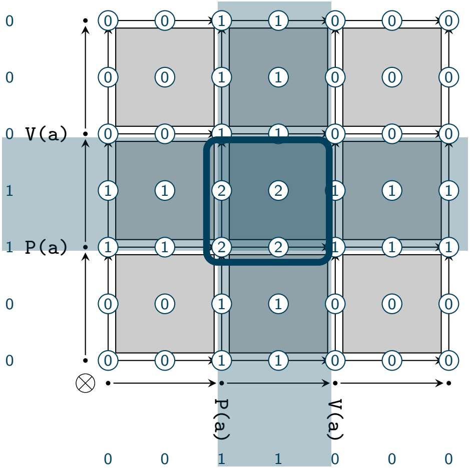
Discrete model

`sem: 1 a`



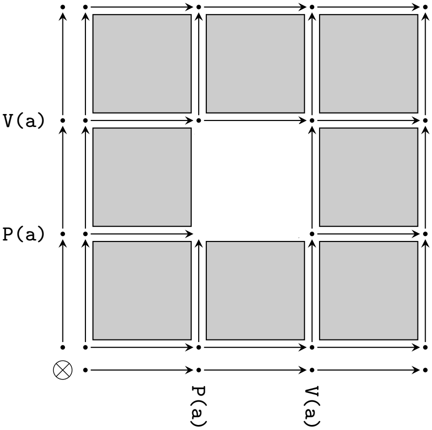
Discrete model

`sem: 1 a`



Discrete model

`sem: 1 a`



The potential functions of processes and programs

Conservative forces in physics
Fahrenberg, U., *Master's Thesis*, 2002

A process π is conservative when for all paths and all semaphores s, the amount of tokens of type s held by the process at the end of the execution trace only depends on its arrival point.

In that case the process π comes with a potential function F_π

$$F_\pi : \{\text{semaphores}\} \times \{\text{points}\} \to \mathbb{N}$$

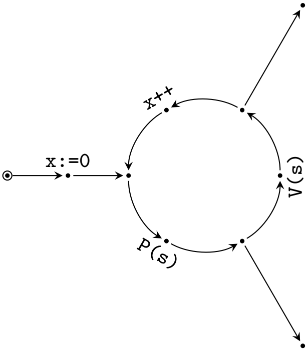
A program Π is conservative when so are its processes $\pi_1, \ldots, \pi_d$ and its potential function is given by

$$F_\Pi(s, (p_1, \ldots, p_d)) = \sum_{k=1}^{d} F_{\pi_k}(s, p_k)$$

If $F_\Pi(s, p) > \text{arity}(s)$ for some semaphore s, then p is forbidden.

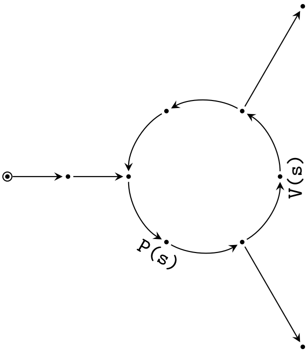
Conservative process

example



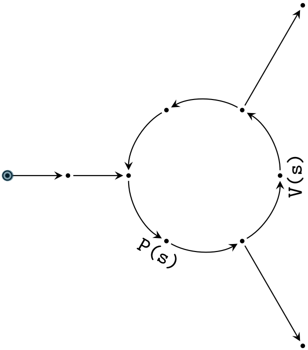
Conservative process

example



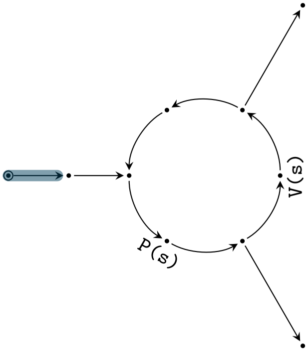
Conservative process

example



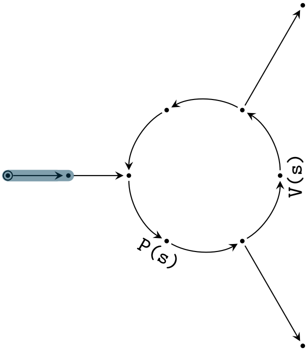
Conservative process

example



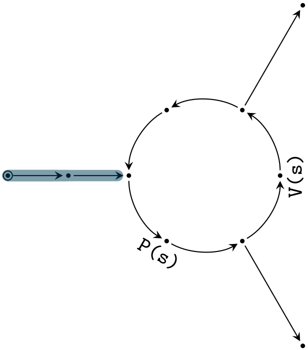
Conservative process

example



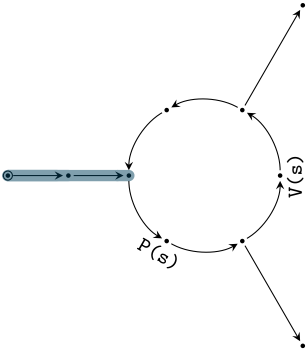
Conservative process

example



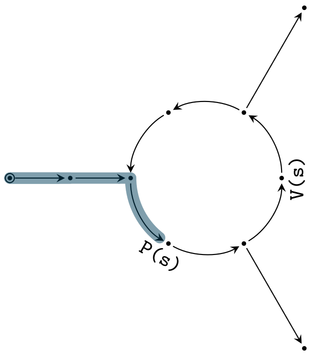
Conservative process

example



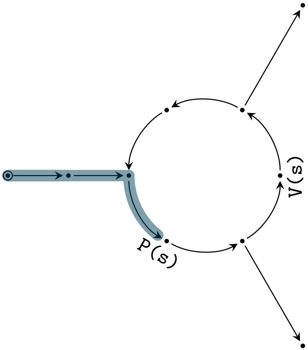
Conservative process

example



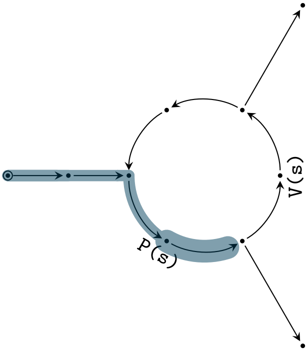
Conservative process

example



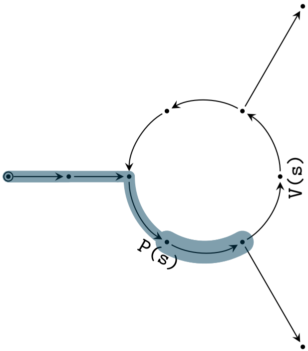
Conservative process

example



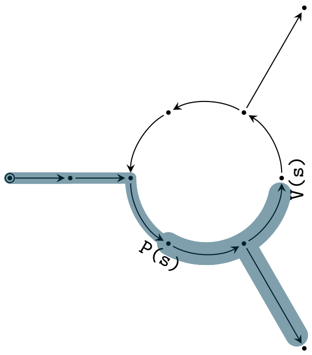
Conservative process

example



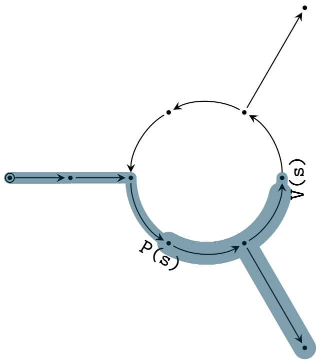
Conservative process

example



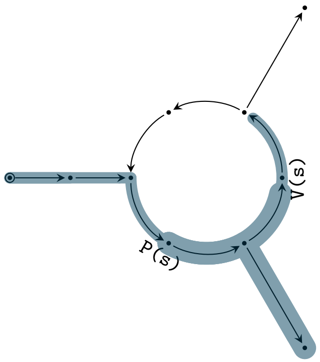
Conservative process

example



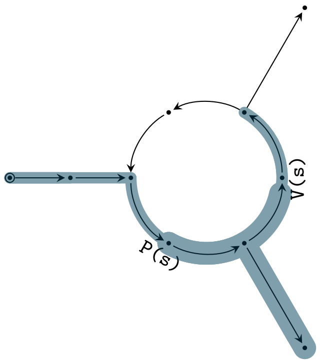
Conservative process

example



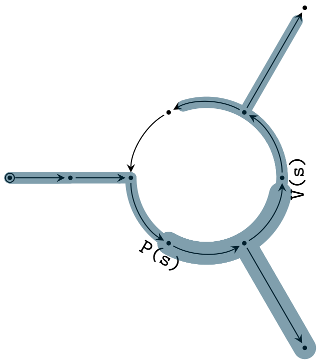
Conservative process

example



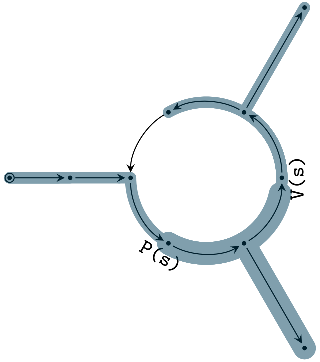
Conservative process

example



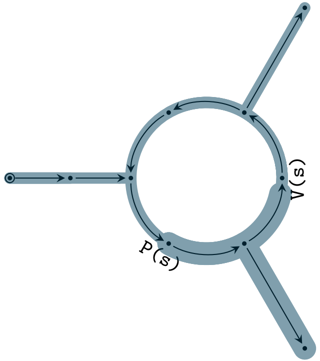
Conservative process

example



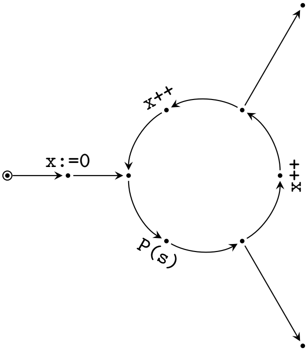
Conservative process

example



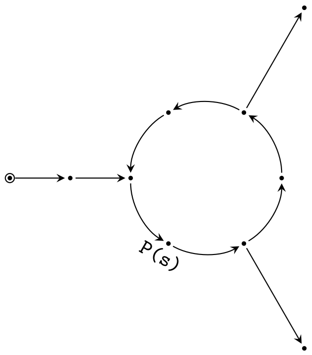
Not conservative process

example



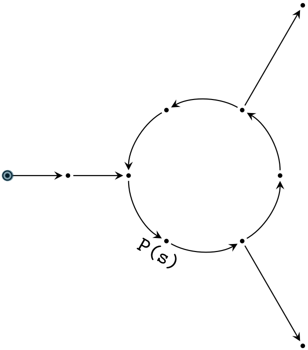
Not conservative process

example



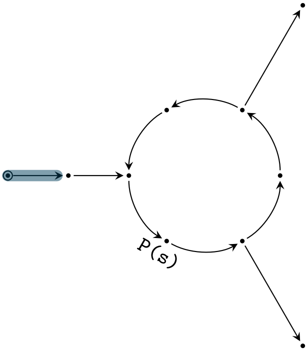
Not conservative process

example



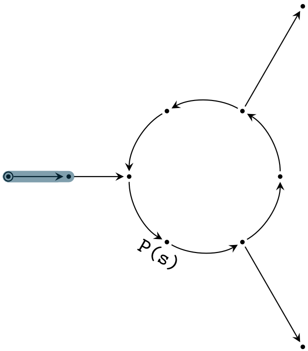
Not conservative process

example



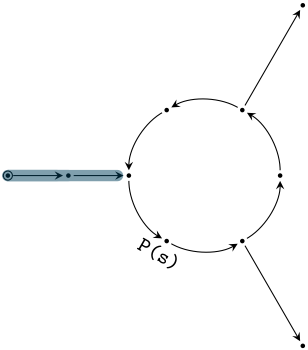
Not conservative process

example



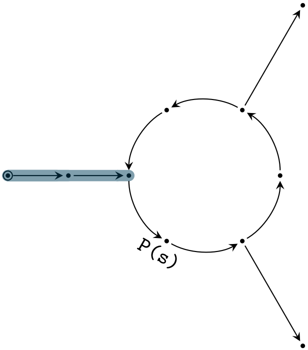
Not conservative process

example



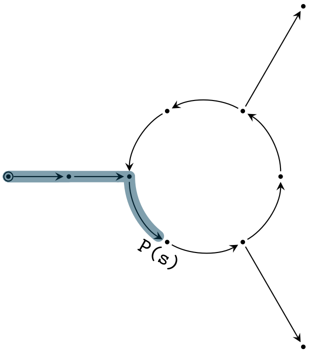
Not conservative process

example



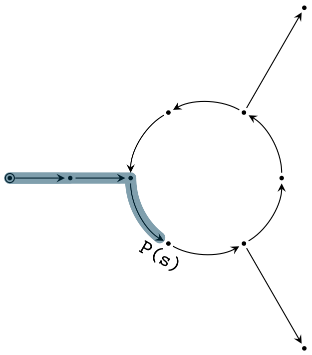
Not conservative process

example



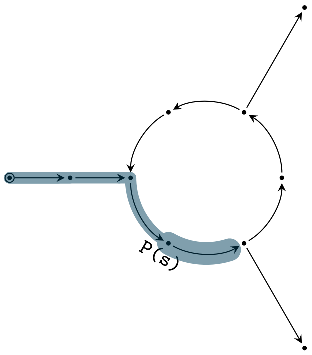
Not conservative process

example



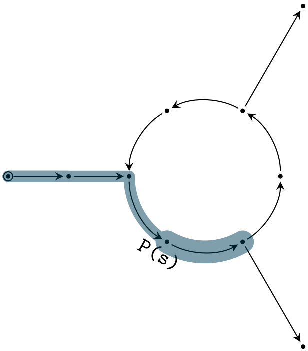
Not conservative process

example



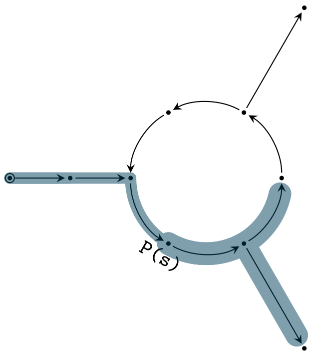
Not conservative process

example



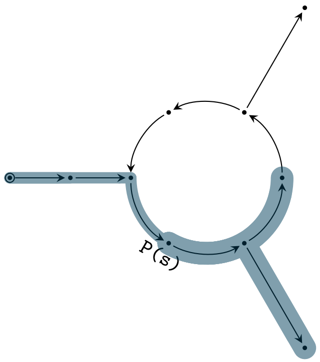
Not conservative process

example



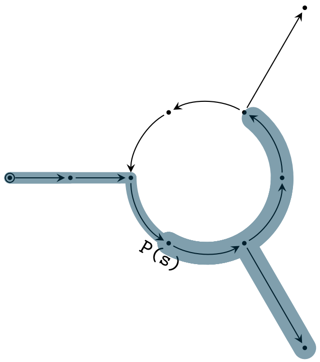
Not conservative process

example



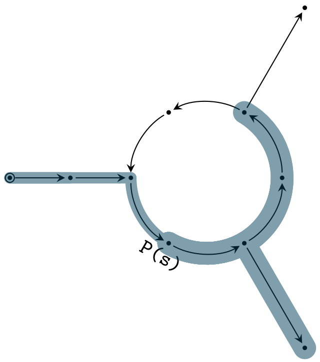
Not conservative process

example



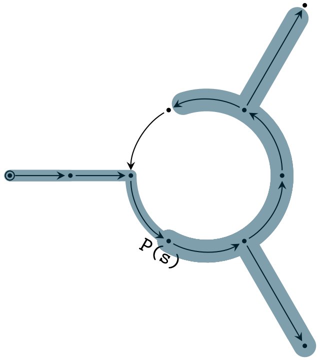
Not conservative process

example



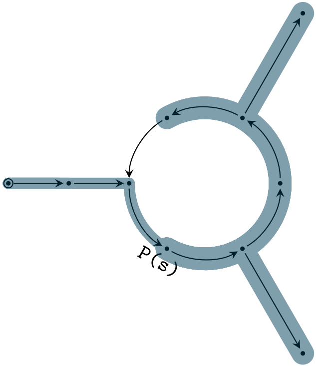
Not conservative process

example



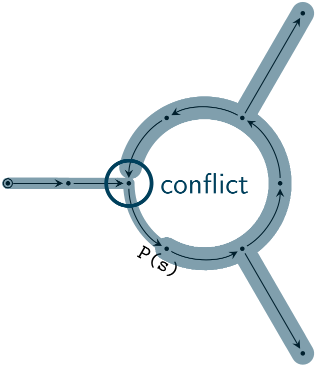
Not conservative process

example



Not conservative process

example

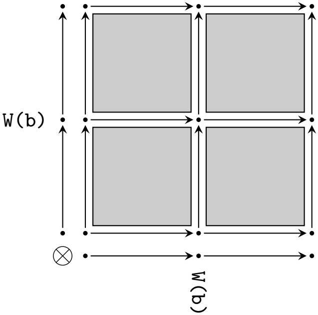


A synchronization barrier

```
sync:  1 b
proc:
    p = W(b)
init:  2p
```

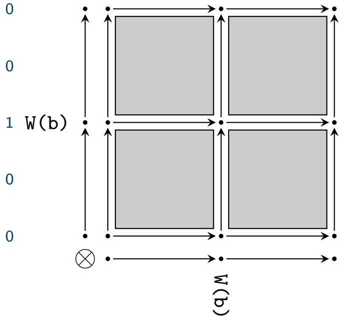
Discrete Model

`sync: 1 b`



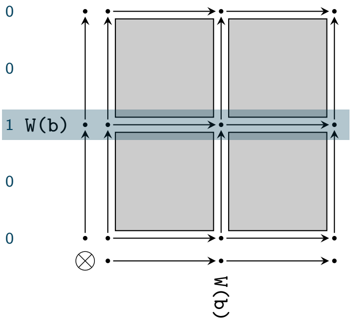
Discrete Model

`sync:  1 b`



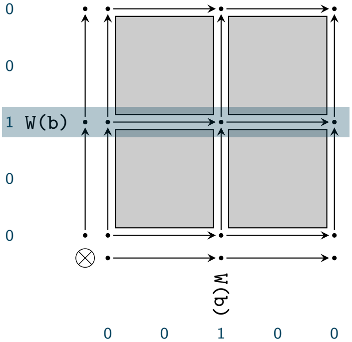
Discrete Model

`sync: 1 b`



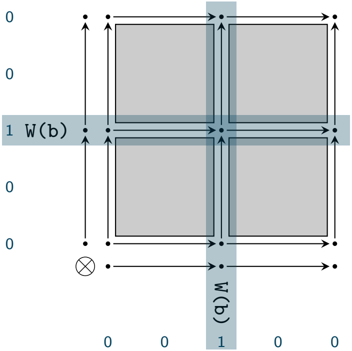
Discrete Model

`sync: 1 b`



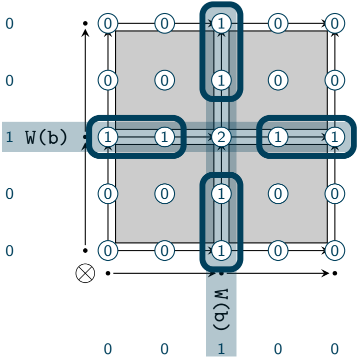
Discrete Model

sync: 1 b



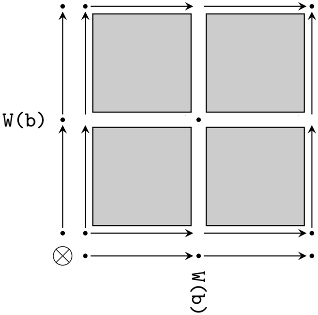
Discrete Model

`sync:  1 b`



Discrete Model

sync: 1 b

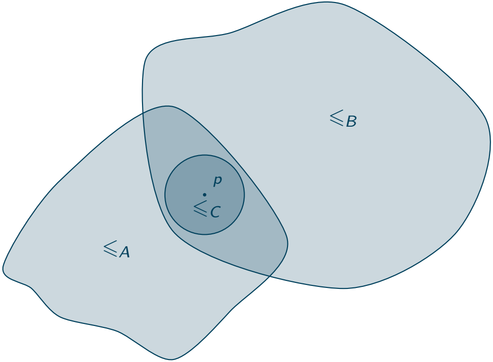


4. Providing Models with Local Pospace Structure

Locally ordered spaces

L. Fajstrup, É. Goubault, and M. Raussen, *Algebraic Topology and Concurrency*, 1998

Directed atlas $\mathcal{U}$ For all points p, for all directed neighborhoods A and B of p, there exists a directed neighborhood C of p such that $C \subseteq A \cap B$ and $\leqslant_A |_C = \leqslant_C = \leqslant_B |_C$.



From discrete models to continuous ones

$$G : A \underset{\partial^-}{\overset{\partial^+}{\rightrightarrows}} V \qquad\qquad \upharpoonleft G \downharpoonleft \;=\; V \;\sqcup\; A \times \,]0,1[$$

$$\upharpoonleft G_1 \downharpoonleft \times \cdots \times \upharpoonleft G_N \downharpoonleft \;=\; \bigsqcup_{\substack{\text{points } p \text{ of} \\ G_1, \ldots, G_N}} \{p\} \times]0,1[^{\,\dim(p_1,\ldots,p_N)}$$

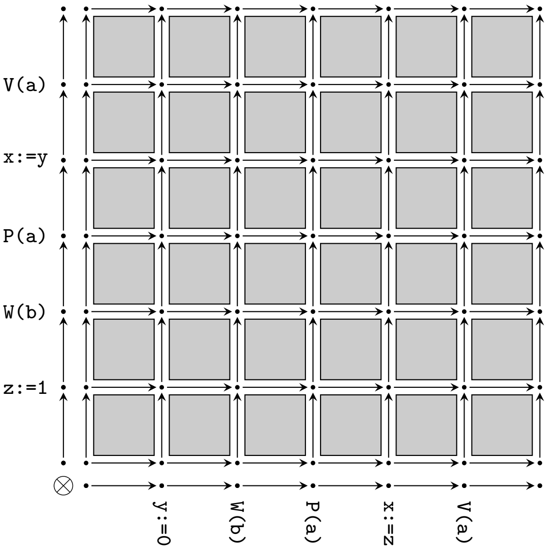
where $p = (p_1, \ldots, p_N)$ and $\dim p = \#\{n \in \{1, \ldots, N\} \mid p_n \in A_n\}$

The directed topological model is then

$$\bigsqcup_{\substack{\text{not forbidden} \\ \text{points } p \text{ of} \\ G_1, \ldots, G_N}} \{p\} \times]0,1[^{\,\dim(p_1,\ldots,p_N)}$$

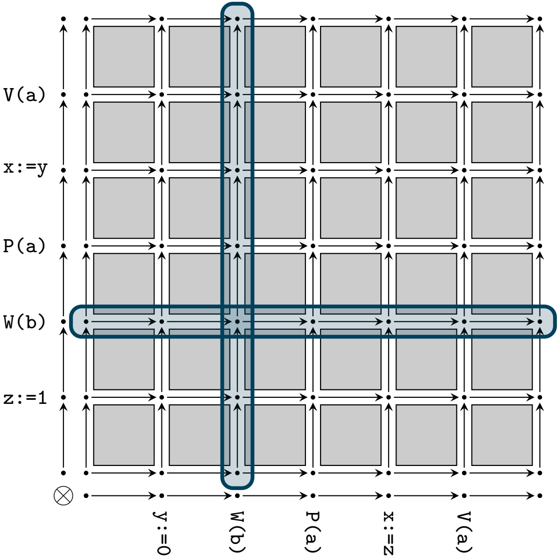
From discrete to continuous

sem: 1 a sync: 1 b



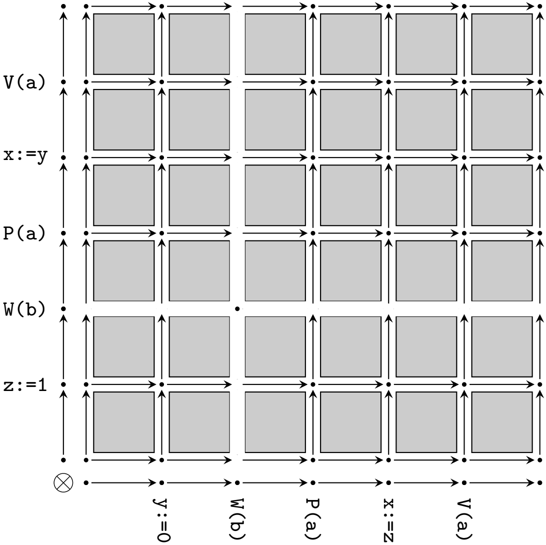
From discrete to continuous

sem: 1 a sync: 1 b



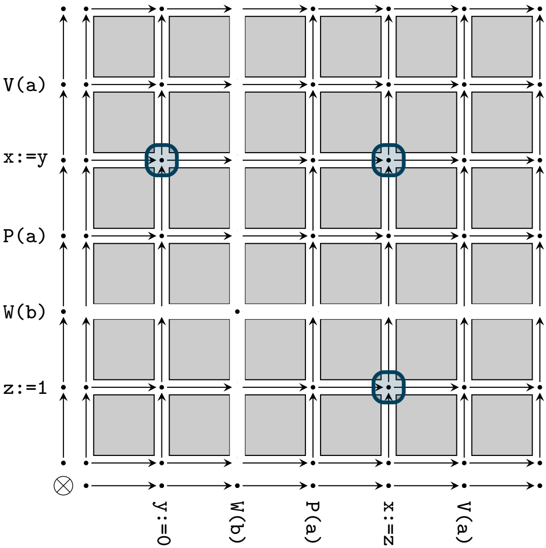
From discrete to continuous

`sem:  1 a        sync:  1 b`



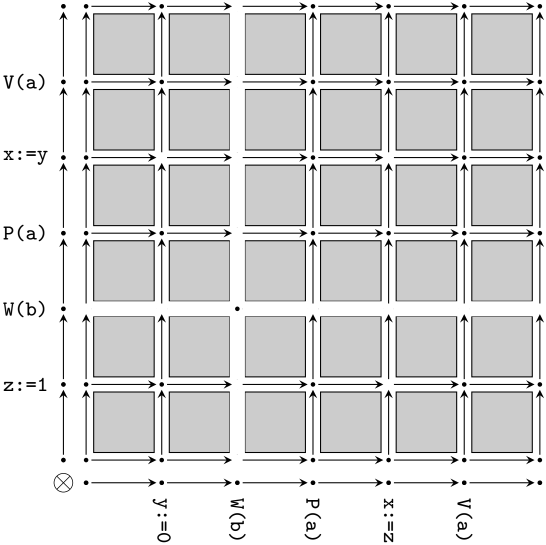
From discrete to continuous

sem: 1 a sync: 1 b



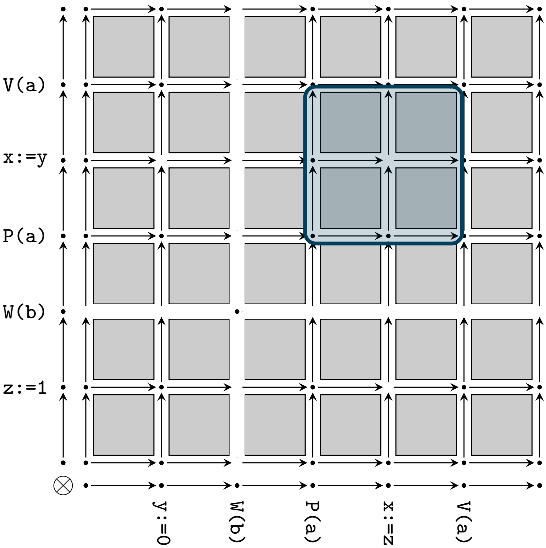
From discrete to continuous

`sem: 1 a` `sync: 1 b`



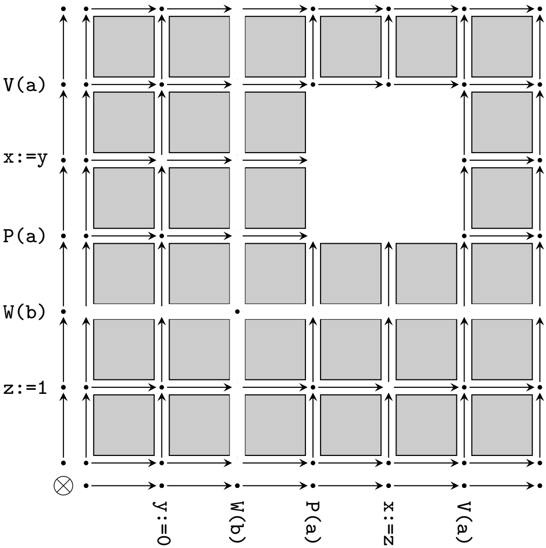
From discrete to continuous

`sem: 1 a` `sync: 1 b`



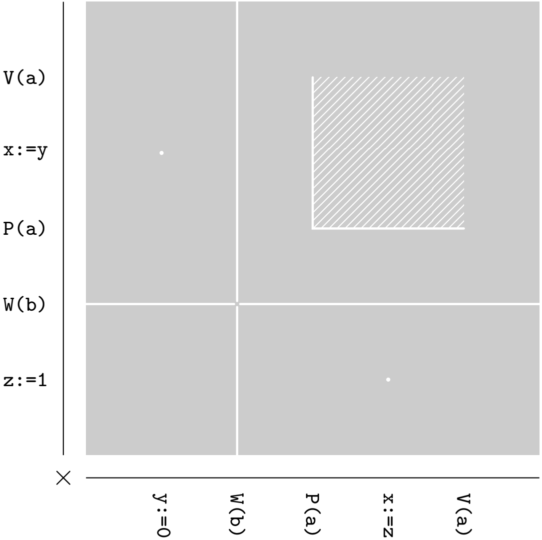
From discrete to continuous

`sem: 1 a` `sync: 1 b`



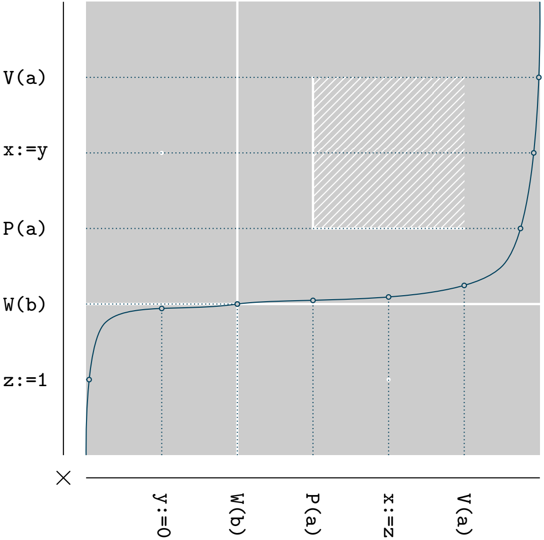
From discrete to continuous

`sem: 1 a` `sync: 1 b`



From discrete to continuous

sem: 1 a sync: 1 b



Directed homotopy of directed paths

L. Fajstrup, É. Goubault, and M. Raussen, *Algebraic Topology and Concurrency*, 1998
M. Grandis, *Directed Homotopy Theory, I. The Fundamental Category*, 2001

Weakly directed homotopy: A homotopy of paths whose intermediate paths are directed.

Strongly directed homotopy: A morphism of local pospace whose underlying map is a homotopy of paths.

The dihomotopy classes of a local pospace X are the morphisms of its fundamental category usually denoted by $\overrightarrow{\pi_1}X$.

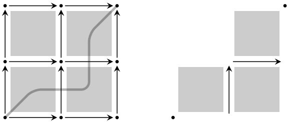
Adequacy

Theorem

Adequacy

Theorem

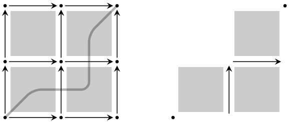
1. Each directed path on a continuous model gives rise to an admissible path on the corresponding discrete model. Hence directed paths act on valuations.



Adequacy

Theorem

1. Each directed path on a continuous model gives rise to an admissible path on the corresponding discrete model. Hence directed paths act on valuations.

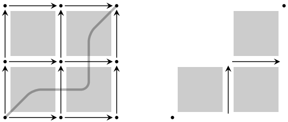


2. The output valuations of weakly dihomotopic directed paths are the same. Hence weak dihomotopy classes (i.e. the fundamental category of the model) act on valuations.

Adequacy

Theorem

1. Each directed path on a continuous model gives rise to an admissible path on the corresponding discrete model. Hence directed paths act on valuations.

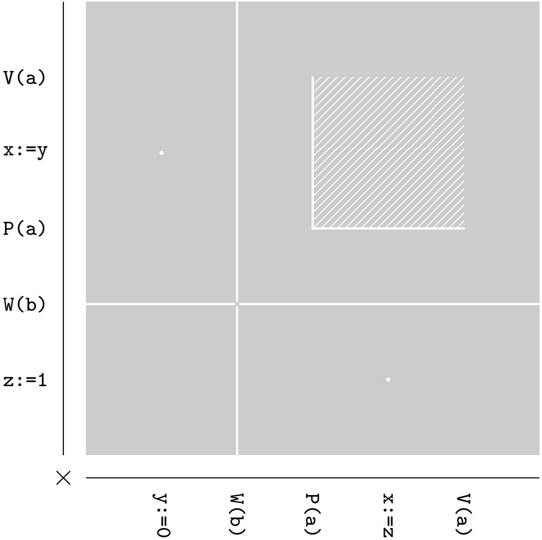


2. The output valuations of weakly dihomotopic directed paths are the same. Hence weak dihomotopy classes (i.e. the fundamental category of the model) act on valuations.

3. The weak dihomotopy class of an execution path only contains execution paths.

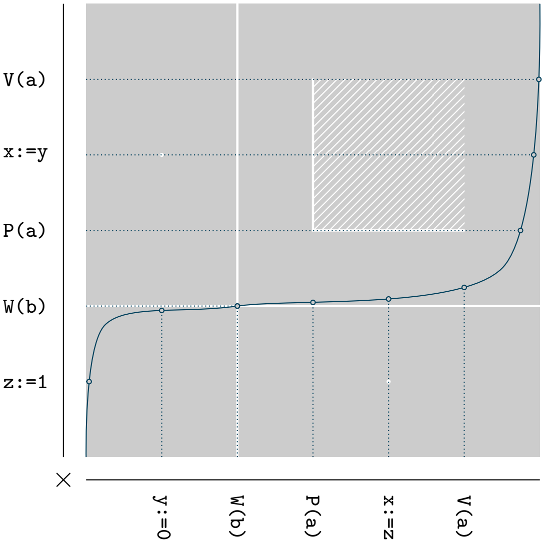
Directed homotopy

sem: 1 a sync: 1 b



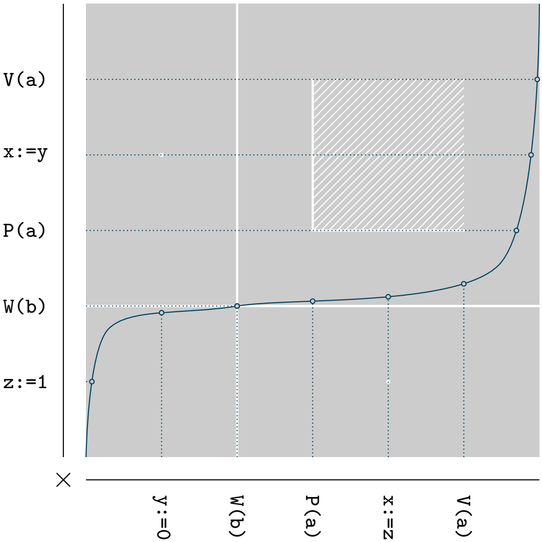
Directed homotopy

`sem: 1 a` `sync: 1 b`



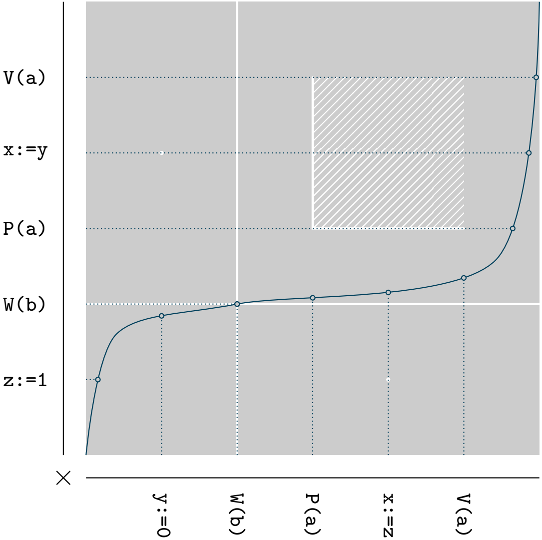
Directed homotopy

`sem:  1 a      sync:  1 b`



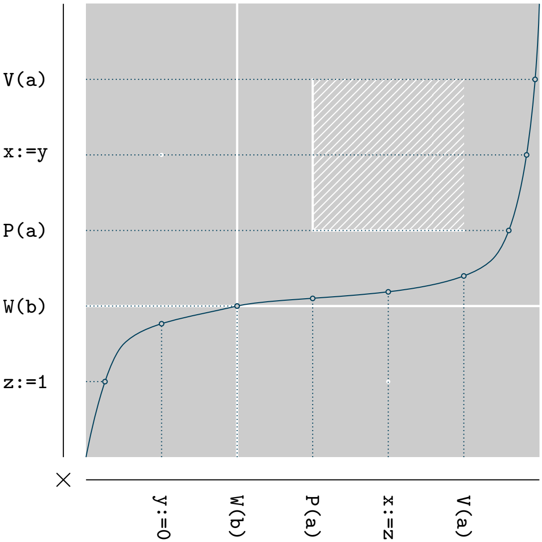
Directed homotopy

`sem: 1 a` `sync: 1 b`



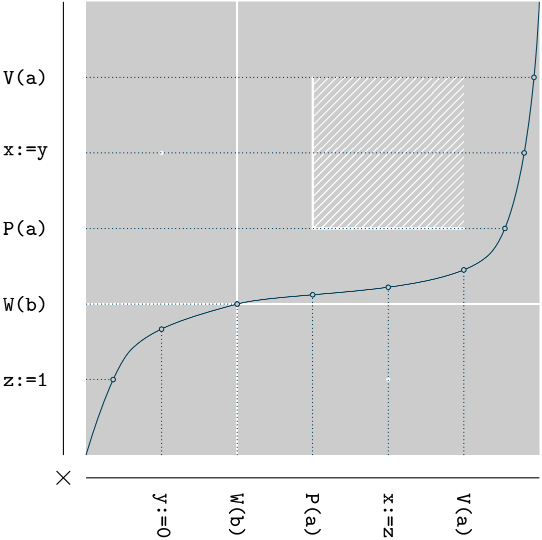
Directed homotopy

`sem: 1 a` `sync: 1 b`



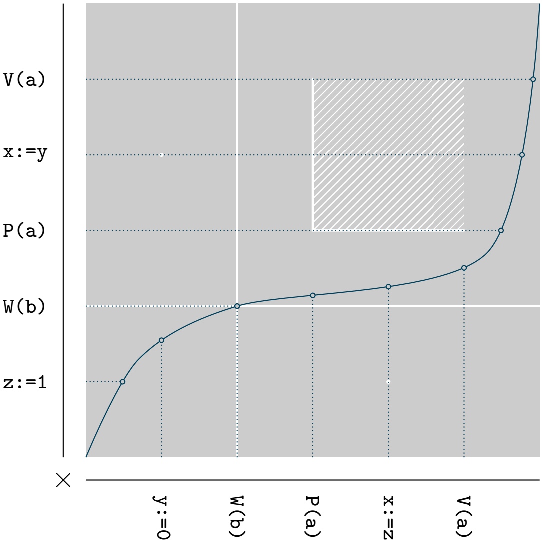
Directed homotopy

sem: 1 a sync: 1 b



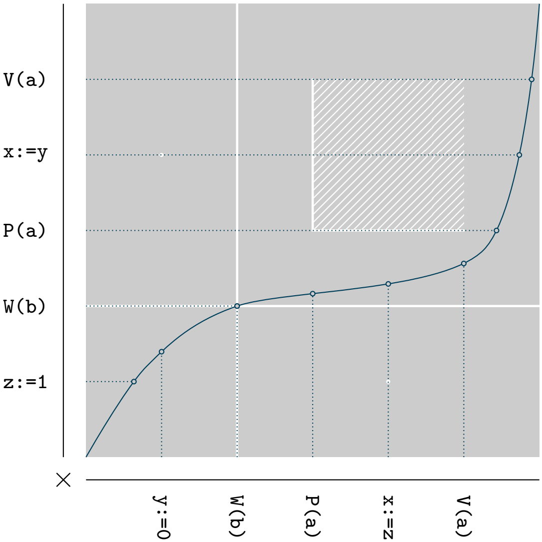
Directed homotopy

sem: 1 a   sync: 1 b



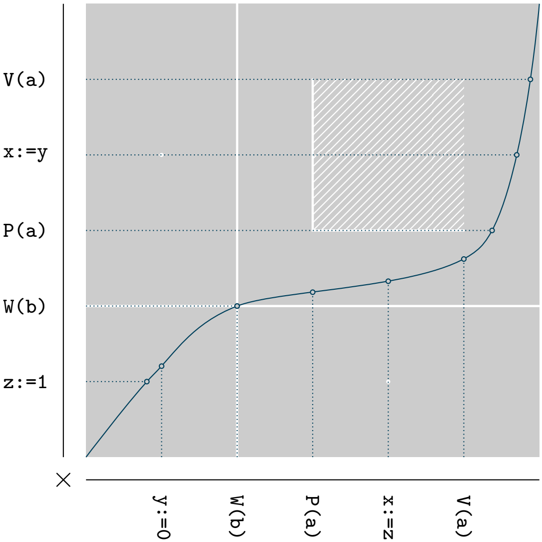
Directed homotopy

sem: 1 a sync: 1 b



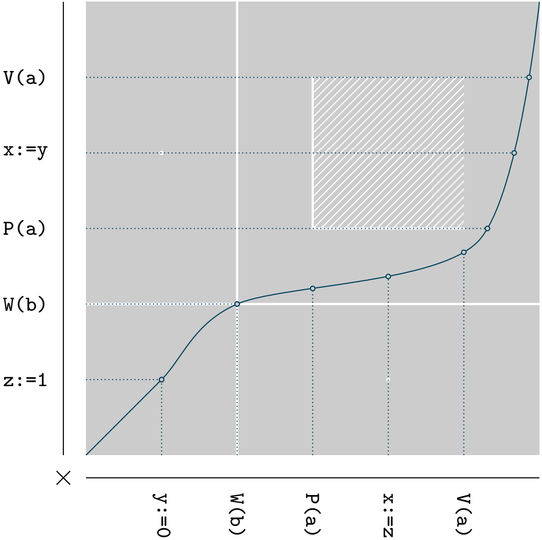
Directed homotopy

`sem:  1 a      sync:  1 b`



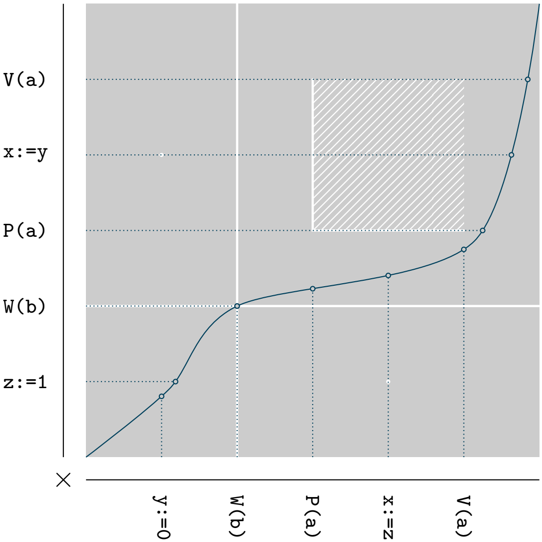
Directed homotopy

`sem: 1 a        sync: 1 b`



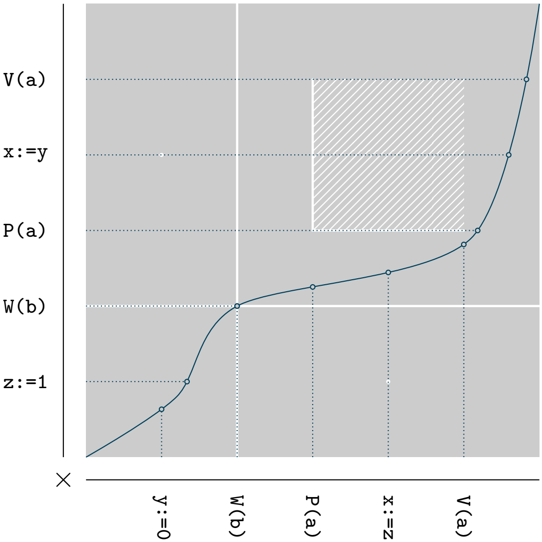
Directed homotopy

sem: 1 a   sync: 1 b



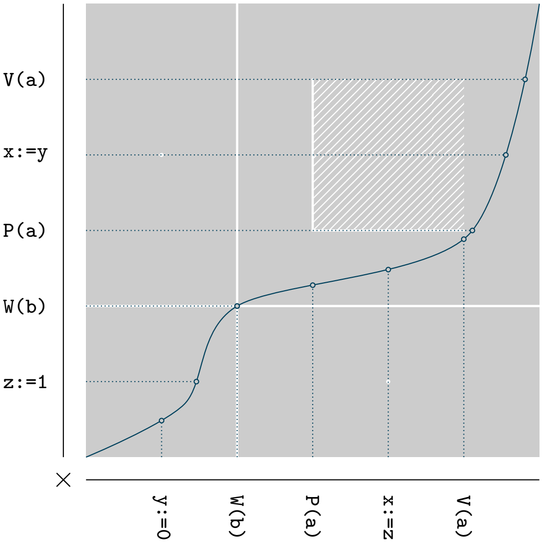
Directed homotopy

sem: 1 a sync: 1 b



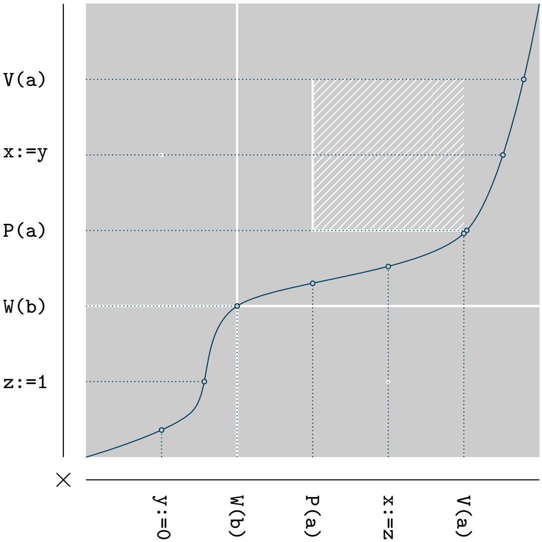
Directed homotopy

`sem:  1 a      sync:  1 b`



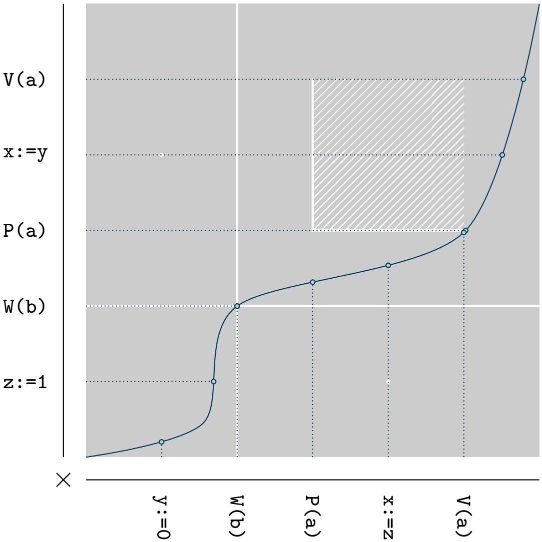
Directed homotopy

`sem:  1 a      sync:  1 b`



Directed homotopy

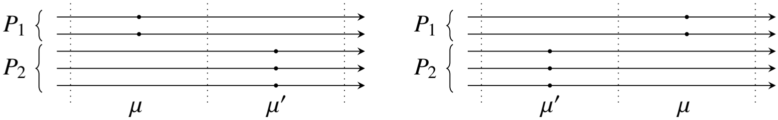
`sem: 1 a` `sync: 1 b`



Independence of programs

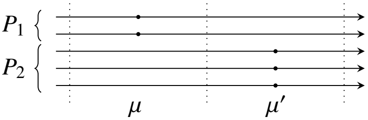
Independence of programs

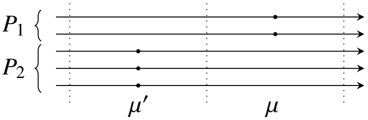
Observational independence



Independence of programs

Observational independence



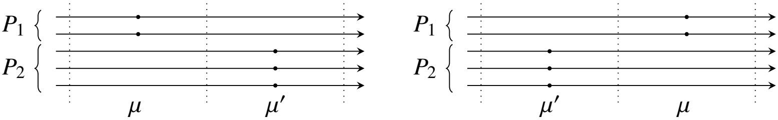


Model independence

$$[\![\ P_1 \mid P_2\ ]\!] = [\![\ P_1\ ]\!] \times [\![\ P_2\ ]\!]$$

Independence of programs

Observational independence



Model independence

$$[\![\ P_1 \mid P_2\ ]\!] = [\![\ P_1\ ]\!] \times [\![\ P_2\ ]\!]$$

Theorem [Haucourt - not published yet]: The following chain of implications is strict.

syntactic independence
$\Downarrow$
model independence
$\Downarrow$
observational independence

Parallelizing a program

```
sem:  1 a
sem:  2 c
```

```
proc:
  p = P(a);P(c);V(c);V(a)

  q = P(c);V(c)
```

```
init:  2p q
```

Parallelizing a program

`sem: 1 a` `sem: 2 c`	`sem: 1 a` `sem: 2 c`
`proc:` `p = P(a);P(c);V(c);V(a)`	`proc:` `q = P(c);V(c)`
`init: 2p`	`init: q`

Parallelizing a program

`sem: 1 a`	
`proc:` `p = P(a);V(a)`	`proc:` `q = ()`
`init: 2p`	`init: q`

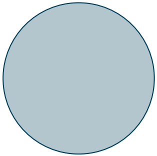
5. Handling Continuous Models

Almost finite graphs

A graph G is said to be linear when $|G|$ is an interval of $\mathbb{R}$.

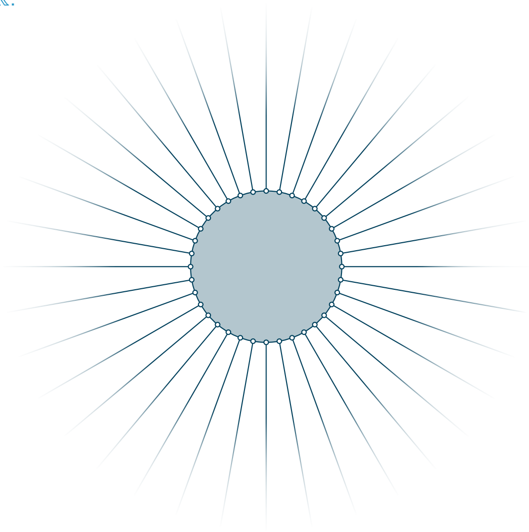
Almost finite graphs

A graph G is said to be linear when $|G|$ is an interval of $\mathbb{R}$.



Almost finite graphs

A graph G is said to be linear when $|G|$ is an interval of $\mathbb{R}$.



Characterizing almost finite graphs

Characterizing almost finite graphs

$$\mathcal{R}_G = \{\text{finite union of connected subsets of } |G|\}$$

Characterizing almost finite graphs

$$\mathcal{R}_G = \{\text{finite union of connected subsets of } |G|\}$$

Theorem [Haucourt - Ninin not published yet]

Given a graph G, the following are equivalent:

- G is almost finite,
- The collection $\mathcal{R}_G$ forms a Boolean subalgebra of $2^{|G|}$
- The following sum is finite

$$\sum_{v \text{ vertex}} |\deg(v) - 2| + \#\{\text{connected components}\} \quad < \quad \infty$$

- The Freudenthal extension of $|G|$ is homeomorphic with the geometric realization of some finite graph.

When the preceding statements are satisfied, the number of ends of $|G|$ is the number of infinite conneected components of L.

Isothetic regions (1)

Isothetic regions (1)

block: $B_1 \times \cdots \times B_N$ with $B_n \neq \emptyset$ and $B_n \in \mathcal{R}_{G_n}$ for $1 \leqslant n \leqslant N$.

Isothetic regions (1)

block: $B_1 \times \cdots \times B_N$ with $B_n \neq \emptyset$ and $B_n \in \mathcal{R}_{G_n}$ for $1 \leqslant n \leqslant N$.

Theorem [Haucourt - not published yet]

The (nonempty) graphs $G_1, \ldots, G_N$ are almost finite, iff

$$\mathcal{R}_{G_1,\ldots,G_N} = \{\text{finite unions of blocks}\}$$

is a Boolean subalgebra of $2^{|G_1|\times\cdots\times|G_N|}$.

In that case $\mathcal{R}_{G_1,\ldots,G_N}$ is stable under interior, closure, forward and backward operators, and its elements are the subsets of $|G_1| \times \cdots \times |G_N|$ with finitely many maximal subblocks. They are called isothetic regions.

$$\text{frw}(A, B) = \bigcup\{\text{img}(\delta) \mid \delta \text{ a dipath of } A \cup B \text{ starting in } A\}$$

$$\text{bck}(A, B) = \bigcup\{\text{img}(\delta) \mid \delta \text{ a dipath of } A \cup B \text{ ending in } B\}$$

Isothetic regions (2)

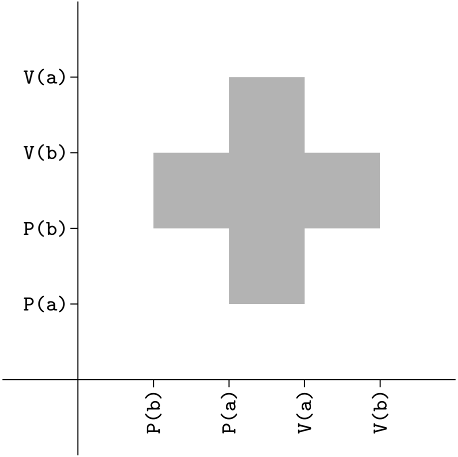
Main motivation

The continuous model of a program is an isothetic region.

Swiss Flag

Maximal subblocks of the state space

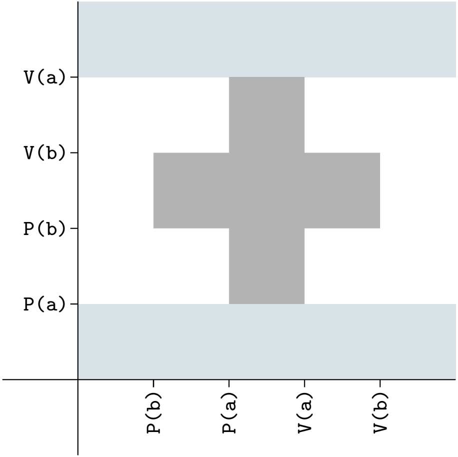
```
#mtx a b
proc:
p = P(a).P(b).V(b).V(a)
q = P(b).P(a).V(a).V(b)
init:  p q
```



Swiss Flag

Maximal subblocks of the state space

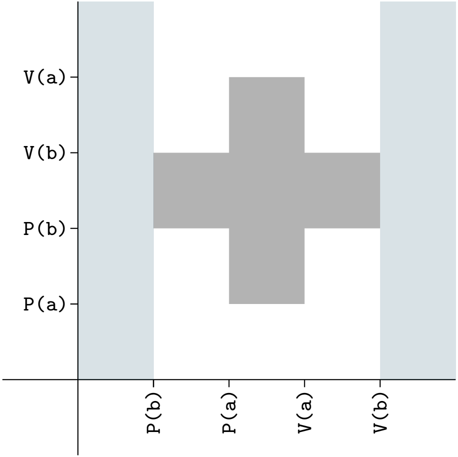
```
#mtx a b
proc:
p = P(a).P(b).V(b).V(a)
q = P(b).P(a).V(a).V(b)
init:  p q
```



Swiss Flag

Maximal subblocks of the state space

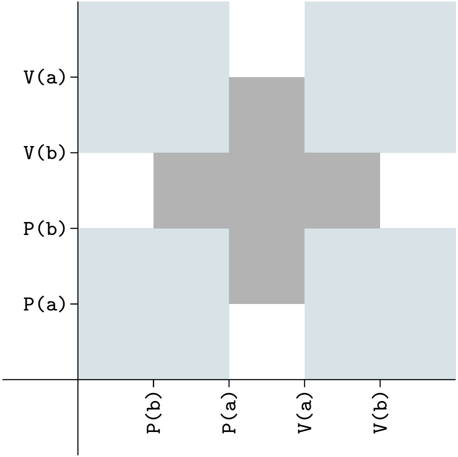
```
#mtx a b
proc:
p = P(a).P(b).V(b).V(a)
q = P(b).P(a).V(a).V(b)
init:  p q
```



Swiss Flag

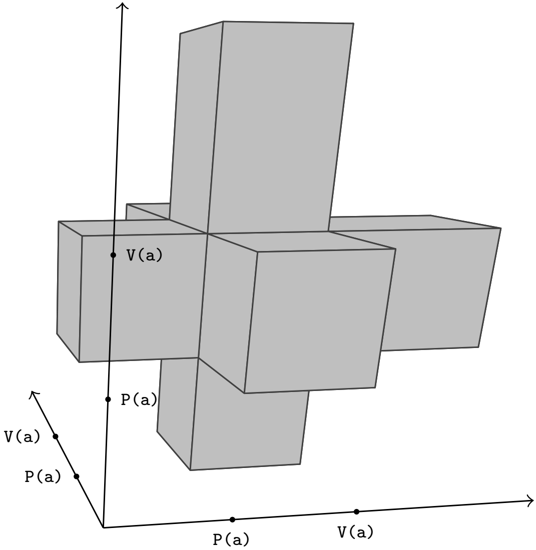
Maximal subblocks of the state space

```
#mtx a b
proc:
p = P(a).P(b).V(b).V(a)
q = P(b).P(a).V(a).V(b)
init:  p q
```



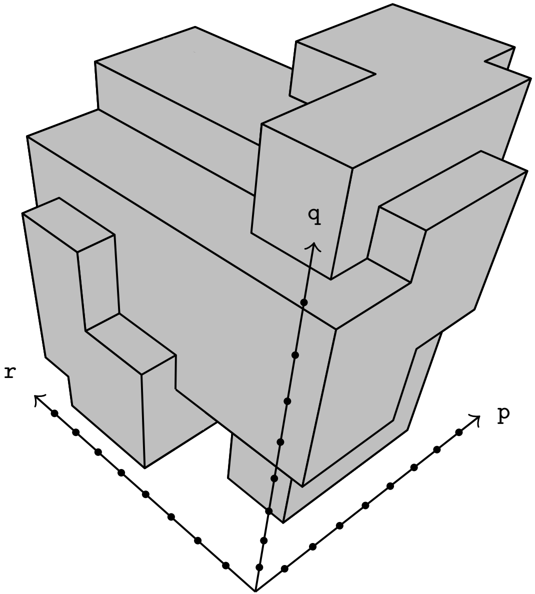
3D Swiss cross

Tetrahemihexacron



The Lipski algorithm

No deadlock



Applications

Past attractor and deadlocks

$$\Omega = \upharpoonleft G_1 \downharpoonright \times \cdots \times \upharpoonleft G_N \downharpoonright$$

$$\mathsf{cone}^{\mathsf{p}} A = \{p \in \Omega \text{ from which } A \text{ can be reached}\} = \mathsf{bck}(A, \Omega)$$

$$\mathsf{escape}^{\mathsf{f}} A = \{p \in \Omega \text{ from which } A \text{ cannot be reached}\}$$

$$\mathsf{escape}^{\mathsf{f}} A = (\mathsf{cone}^{\mathsf{p}} A)^c$$

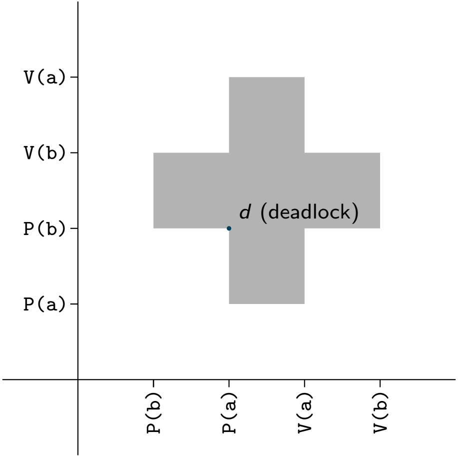
$$\mathsf{att}^{\mathsf{p}} A = \{p \in \Omega \text{ from which } A \text{ cannot be avoided}\}$$

$$\mathsf{att}^{\mathsf{p}} A = \mathsf{escape}^{\mathsf{f}}(\mathsf{escape}^{\mathsf{f}} A)$$

Swiss Flag

Past attractor of a deadlock point

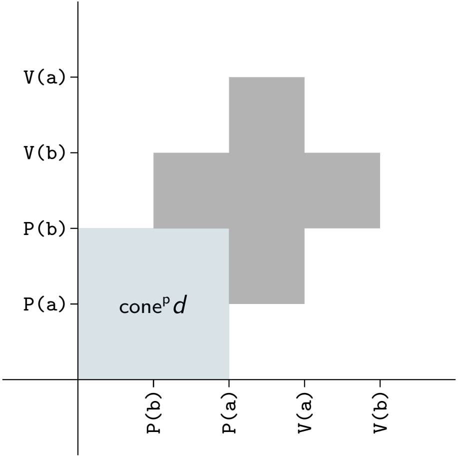
```
#mtx a b
proc:
p = P(a).P(b).V(b).V(a)
q = P(b).P(a).V(a).V(b)
init:  p q
```



Swiss Flag

Past attractor of a deadlock point

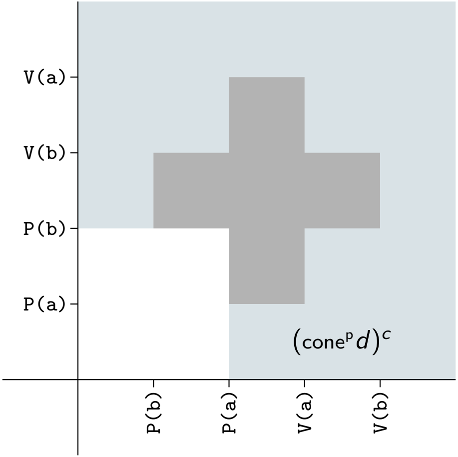
```
#mtx a b
proc:
p = P(a).P(b).V(b).V(a)
q = P(b).P(a).V(a).V(b)
init:  p q
```



Swiss Flag

Past attractor of a deadlock point

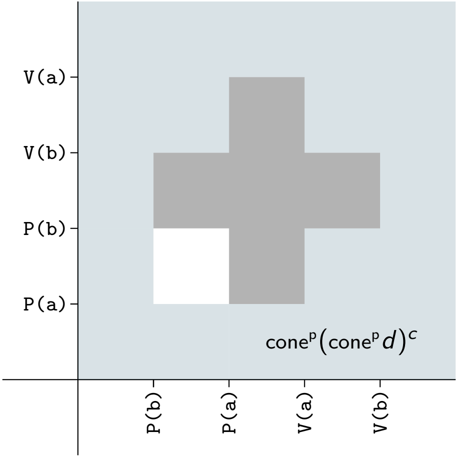
```
#mtx a b
proc:
p = P(a).P(b).V(b).V(a)
q = P(b).P(a).V(a).V(b)
init:  p q
```



Swiss Flag

Past attractor of a deadlock point

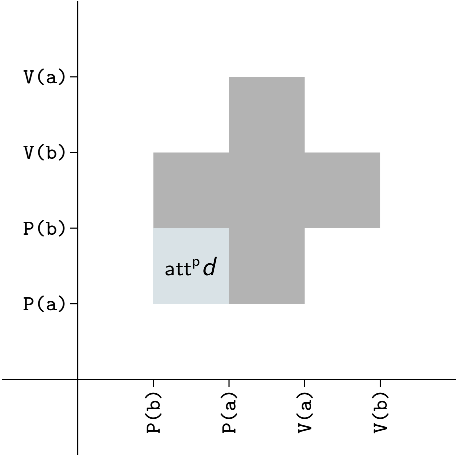
```
#mtx a b
proc:
p = P(a).P(b).V(b).V(a)
q = P(b).P(a).V(a).V(b)
init:  p q
```



Swiss Flag

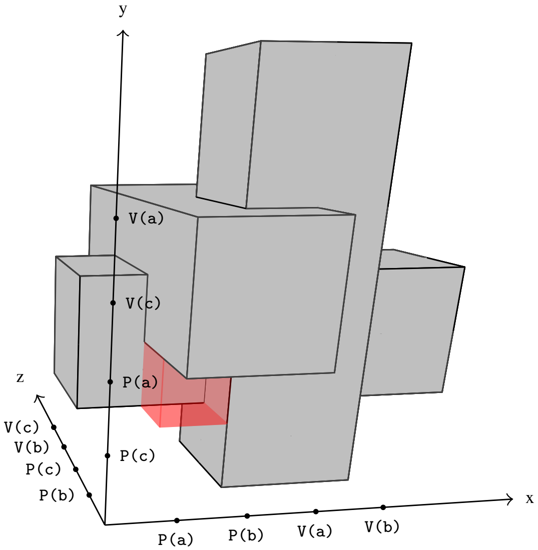
Past attractor of a deadlock point

```
#mtx a b
proc:
p = P(a).P(b).V(b).V(a)
q = P(b).P(a).V(a).V(b)
init:  p q
```



Three dining philosophers

Deadlock



Tensor product of Boolean algebras

Blocks are pure tensors

Tensor product of Boolean algebras

Blocks are pure tensors

For $\Omega \in \mathcal{R}_{G_1,\dots,G_n}$ define

$$\mathcal{R}_\Omega \quad = \quad \left\{X \in \mathcal{R}_{G_1,\dots,G_n} \;\middle|\; A \subseteq \Omega\right\}$$

For all elements $A \in \mathcal{R}_{\Omega_1}$, and $B, C \in \mathcal{R}_{\Omega_2}$:

$$\begin{aligned}(A \times B) \cup (A \times C) \quad &= \quad A \times (B \cup C) \\ (A \times B) \cap (A \times C) \quad &= \quad A \times (B \cap C) \\ A \times \emptyset \quad &= \quad \emptyset\end{aligned}$$

but

$$A \times \Omega_2 \quad \neq \quad \Omega_1 \times \Omega_2$$

Tensor product of Boolean algebras

Semilattices and some other algebraic theories

Structure	Signature	Axioms	Category
semilattice	$\vee$	commutative idempotent semigroup	**SLat**
semilattice with zero	$\vee, 0$	commutative idempotent monoid	$\mathbf{SLat_0}$
lattice	$\vee, \wedge$	two semilattices with $\sqsubseteq_\wedge = \sqsubseteq_\vee^{op}$	**Lat**
distributive lattice	$\vee, \wedge$	lattice in which $\wedge$ distributes over $\vee$	**DLat**
distributive lattice with zero	$\vee, 0, \wedge$	distributive lattice in which $\vee$ has a neutral element	$\mathbf{DLat_0}$
distributive lattice with difference	$\vee, 0, \wedge, \backslash$	distributive lattice with zero s.t. $(x \backslash y) \vee (x \wedge y) = x$ $(x \backslash y) \wedge y = 0$	$\mathbf{DLat_d}$
bounded distributive lattice	$\vee, 0, \wedge, 1$	lattice in which both $\vee$ and $\wedge$ have a neutral element	$\mathbf{DLat_b}$
Boolean algebra	$\vee, 0, \wedge, 1, \_^c$	bounded distributive lattice s.t. $x^c \wedge x = 0$ and $x^c \vee x = 1$	**BoolAlg**
	$\vee, 0, \wedge, 1, \backslash$	bounded distributive lattice with difference	

Tensor product of Boolean algebras

Fraser, G. A., *The semilattice tensor product of distributive lattices*, 1976

Tensor product of Boolean algebras

Fraser, G. A., *The semilattice tensor product of distributive lattices*, 1976

$$\mathbf{BoolAlg} \longrightarrow \mathbf{DLat_d} \longrightarrow \mathbf{DLat_0} \longrightarrow \mathbf{SLat_0} \longrightarrow \mathbf{SLat}$$

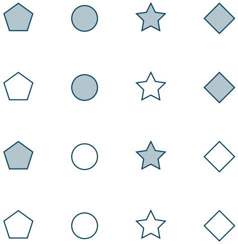
Theorem [Haucourt - Ninin (2014)]

The universal tensor products of (finitely many) Boolean algebras in $\mathbf{SLat_0}$, $\mathbf{DLat_0}$, and $\mathbf{DLat_d}$ are isomorphic Boolean algebras. Moreover

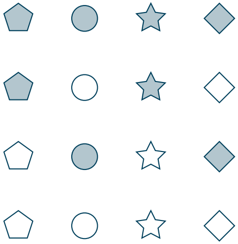
$$\mathcal{R}_{\Omega_1 \times \cdots \times \Omega_N} \quad \cong \quad \mathcal{R}_{\Omega_1} \otimes \cdots \otimes \mathcal{R}_{\Omega_N}$$

6. Factoring

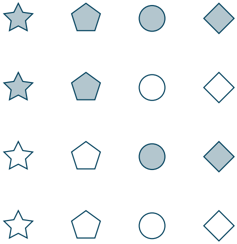
Example of factorization



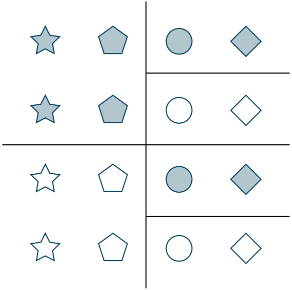
Example of factorization



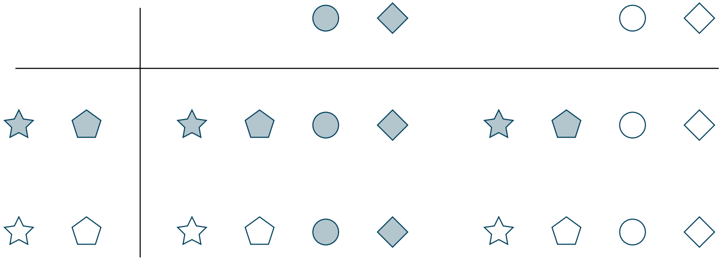
Example of factorization



Example of factorization



Example of factorization



Homogeneous languages

Main results

Theorem [Balabonki - Haucourt (2010)]

The collection $\mathcal{H}(\mathbb{A})$ forms a free commutative monoid under the product induced by word concatenation and the zero language.

Theorem [Balabonki - Haucourt (2010)]

The collection of isothetic regions

$$\bigcup_{n\in\mathbb{N}} \mathcal{R}_{\underbrace{G,\ldots,G}_{n\text{ times}}}$$

forms a free commutative monoid that is isomorphic to $\mathcal{H}(\mathbb{A})$ with $\mathbb{A}$ the collection of connected subsets of $|G|$.

Application to program factoring

If X is the model of a program P, a factorization of X induces a family of model independent programs whose parallel compound is P.

Parallelizing a program

```
sem:  1 a b
sem:  2 c
```

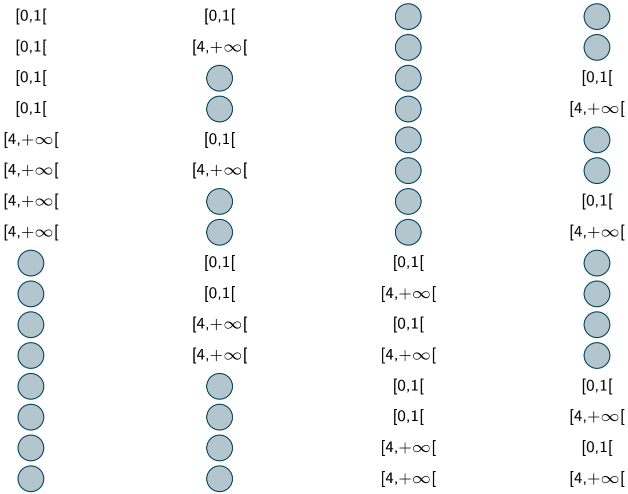
```
proc:
  p = P(a);P(c);V(c);V(a)

  q = P(b);P(c);V(c);V(b)
```

```
init:  p q p q
```

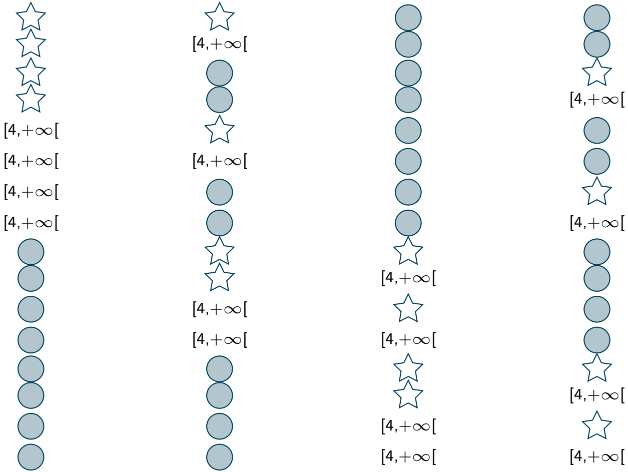
Factoring the space of states

[0,1[	[0,1[	[0,+∞[	[0,+∞[
[0,1[	[4,+∞[	[0,+∞[	[0,+∞[
[0,1[	[0,+∞[	[0,+∞[	[0,1[
[0,1[	[0,+∞[	[0,+∞[	[4,+∞[
[4,+∞[	[0,1[	[0,+∞[	[0,+∞[
[4,+∞[	[4,+∞[	[0,+∞[	[0,+∞[
[4,+∞[	[0,+∞[	[0,+∞[	[0,1[
[4,+∞[	[0,+∞[	[0,+∞[	[4,+∞[
[0,+∞[	[0,1[	[0,1[	[0,+∞[
[0,+∞[	[0,1[	[4,+∞[	[0,+∞[
[0,+∞[	[4,+∞[	[0,1[	[0,+∞[
[0,+∞[	[4,+∞[	[4,+∞[	[0,+∞[
[0,+∞[	[0,+∞[	[0,1[	[0,1[
[0,+∞[	[0,+∞[	[0,1[	[4,+∞[
[0,+∞[	[0,+∞[	[4,+∞[	[0,1[
[0,+∞[	[0,+∞[	[4,+∞[	[4,+∞[

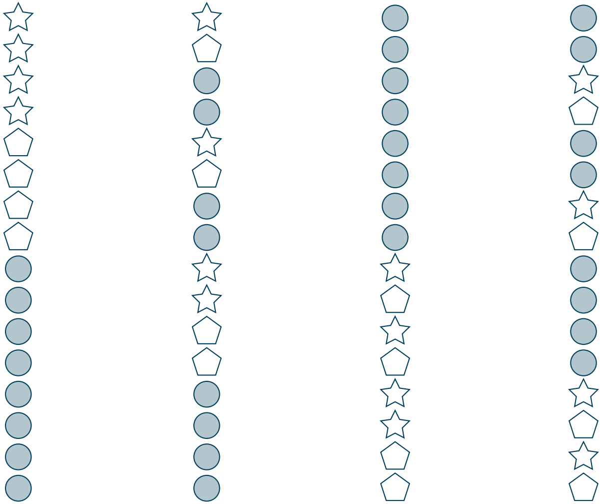
Factoring the space of states



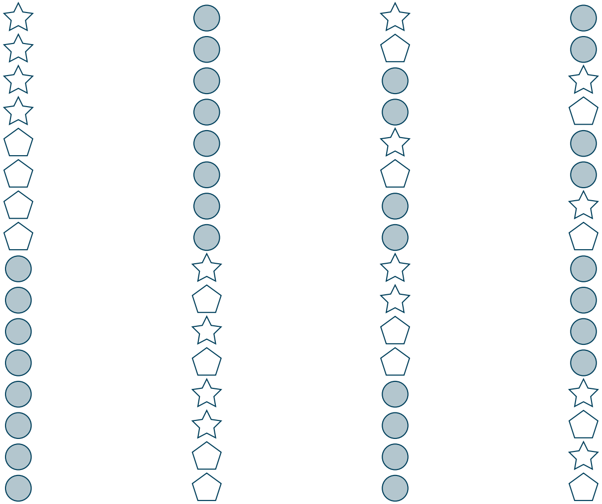
Factoring the space of states



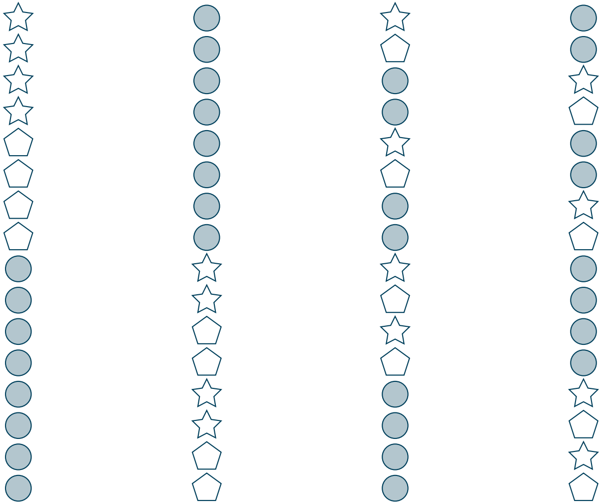
Factoring the space of states



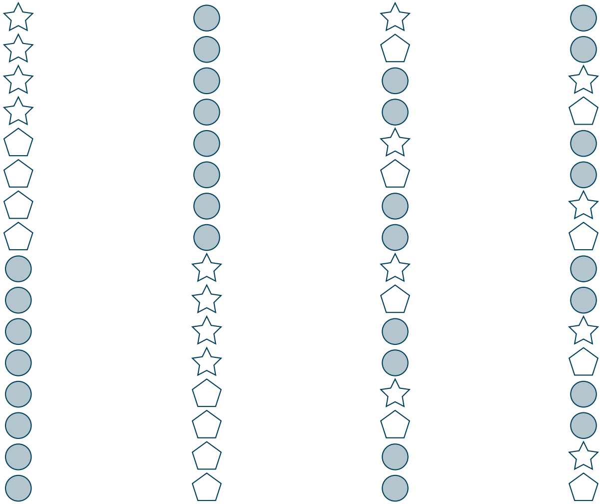
Factoring the space of states



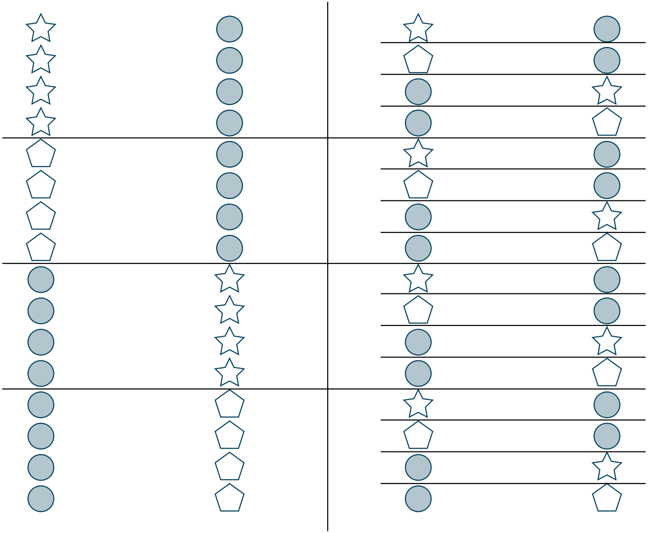
Factoring the space of states



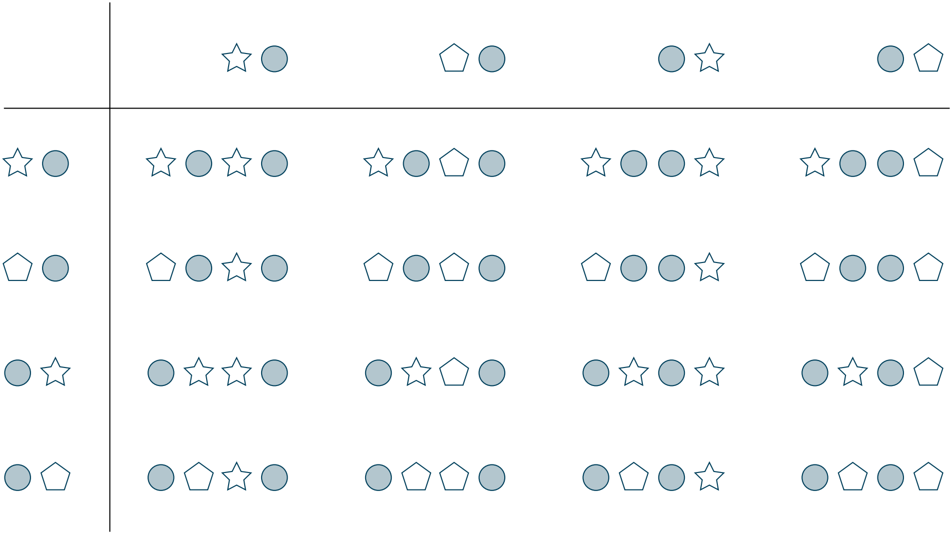
Factoring the space of states



Factoring the space of states



Factoring the space of states



Parallelizing a program

`sem: 1 a b` `sem: 2 c`	`sem: 1 a b` `sem: 2 c`
`proc:` `p = P(a);P(c);V(c);V(a)`	`proc:` `q = P(b);P(c);V(c);V(b)`
`init: 2p`	`init: 2q`

Parallelizing a program

`sem: 1 a`	`sem: 1 b`
`proc:` `p = P(a);V(a)`	`proc:` `q = P(b);V(b)`
`init: 2p`	`init: 2q`

Unique decomposition conjectures

Boolean algebras

1. Is the following decomposition unique ?

$$\mathcal{R}_{\Omega_1 \times \cdots \times \Omega_N} \quad \cong \quad \mathcal{R}_{\Omega_1} \otimes \cdots \otimes \mathcal{R}_{\Omega_N}$$

2. In that case, does the Boolean algebra decomposition matches that of isothetic regions ?

R. S. Pierce (*Tensor Product of Boolean Algebras*, 1983) proved that for all $n \in \mathbb{N}$ there exists a countable Boolean algebra b such that the tensor products $b, b^2, \ldots, b^n$ are distinct though $b^n = b^{n+1}$. Yet, Boolean algebras of the form $\mathcal{R}_\Omega$ are very specific.

Unique decomposition conjectures

Metrics

Any isothetic region can be turned into a finite affine rank length metric space in a natural way.

T. Foertsch and A. Lytchak (*The De Rham Decomposition Theorem for Metric Spaces*, 2008) proved a unique decomposition property for finite affine rank geodesic metric spaces.

1. Does the result extend to length metrics ?

2. In that case, does the metric decomposition matches that of isothetic regions ?

Unique decomposition conjectures

Categories of components *vs* Isothetic regions

category of components $\overrightarrow{\pi_0}(C)$: a generalized notion of skeleton that fits with categories C with no isomorphisms but identities. Well-defined for all loop-free categories.

E.g.: $\overrightarrow{\pi_1}(X)$ for some isothetic region X.

Property (Haucourt 2006): For C loop-free, $\overrightarrow{\pi_0}(C) = 1$ iff C is a lattice.

Property: $\overrightarrow{\pi_1}$ and $\overrightarrow{\pi_0}$ preserves products.

Theorem (Balabonski 2006, unpublished): the collection of (isomorphism classes of) nonempty connected finite loop-free categories with Cartesian product form a free commutative monoid M.

Problem: relate the decomposition of an isothetic region to that of its category of components.

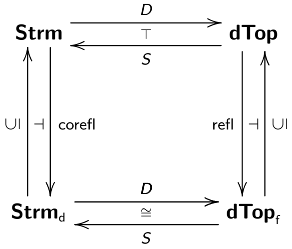
E.g.: $\overrightarrow{\pi_0}(\overrightarrow{\pi_1}([0,1])) = 1$ though $[0,1]$ is not the neutral isothetic region.

7. Directed Topology

Beyond locally ordered spaces

M. Grandis, *Directed Homotopy Theory, I. The Fundamental Category*, 2003
S. Krishnan, *Convenient Category of Locally Preordered Spaces*, 2009

Theorem (Haucourt 2012)



All the categories on the diagram are complete and cocomplete.

Realization of (pre)cubical sets

Glabbeek (van), R.J., *Bisimulations for Higher Dimensional Automata*, 1991
Pratt, V., *Modeling Concurrency with Geometry*, 1991

Face maps:

$$xx01 \equiv (2, x_1 x_2 01) : (a, b) \in [0,1]^2 \mapsto (a, b, 0, 1) \in [0,1]^4$$

Realization of (pre)cubical sets

Glabbeek (van), R.J., *Bisimulations for Higher Dimensional Automata*, 1991
Pratt, V., *Modeling Concurrency with Geometry*, 1991

Face maps:

$$xx01 \equiv (2, x_1x_201) : (a, b) \in [0,1]^2 \mapsto (a, b, 0, 1) \in [0,1]^4$$

Degeneracy maps:

$$(4, x_1x_3) : (a, b, c, d) \in [0,1]^4 \mapsto (a, c) \in [0,1]^2$$

Realizations in streams and d-spaces

and their fundamental categories

Theorem (Haucourt 2012)

For any cubical set K, the fundamental categories of the following objects are isomorphic: $D(\rceil K\lfloor_{\mathbf{Strm}})$, $\rceil K\lfloor_{\mathbf{Strm}}$, $\rceil K\lfloor_{\mathbf{Strm_d}}$, $S(\rceil K\lfloor_{\mathbf{dTop_f}})$, $\rceil K\lfloor_{\mathbf{dTop_f}}$.

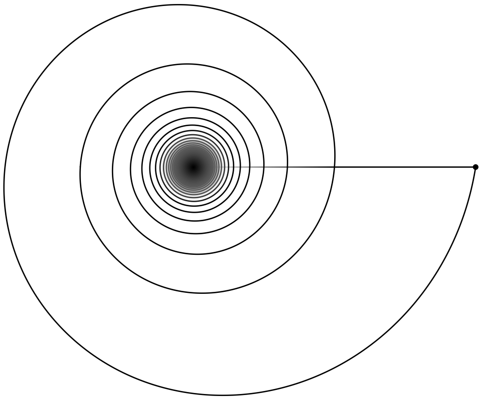
But they may differ from the fundamental category of $\rceil K\lfloor_{\mathbf{dTop}}$.

Conjecture

If K is a precubical set the preceding pathology vanishes.

The downward spiral

A directed path on the directed complex plane



Fundamental category *vs* fundamental groupoid

$\overrightarrow{\pi_1}$ and Π_1 are the fundamental category and the fundamental groupoid functors.

$G : \mathbf{Cat} \to \mathbf{Grd}$ is the enveloping groupoid functor (i.e. left adjoint to $\mathbf{Cat} \hookrightarrow \mathbf{Grd}$)

U is the forgetful functor to **Top**.

There is a natural transformation $g : G \circ \overrightarrow{\pi_1} \to \Pi_1 \circ U$

Fundamental category *vs* fundamental groupoid

$\overrightarrow{\pi_1}$ and Π_1 are the fundamental category and the fundamental groupoid functors.

$G : \mathbf{Cat} \to \mathbf{Grd}$ is the enveloping groupoid functor (i.e. left adjoint to $\mathbf{Cat} \hookrightarrow \mathbf{Grd}$)

U is the forgetful functor to **Top**.

There is a natural transformation $g : G \circ \overrightarrow{\pi_1} \to \Pi_1 \circ U$

Conjecture: The groupoid morphism g_X is an isomorphism when X is:

- the directed realization of a precubical set
- an isothetic region

Direction generated by vector fields on a manifold

Given some tuple of vector fields $f_1, \ldots, f_k$ over a manifold $\mathcal{M}$, the forward cone of $\mathcal{M}$ at x is the set

$$F_x \quad := \quad \left\{ \quad \sum_{i=1}^{k} \lambda_i \cdot f_i(x) \quad \middle| \quad \lambda_i \geqslant 0 \text{ for } i = 1, \ldots, k \quad \right\}$$

A curve γ is said to be forward (with respect to $f_1, \ldots, f_k$) when its derivative at time t belongs to $F_{\gamma(t)}$ for all $t \in \operatorname{dom} \gamma$:

$$\frac{\partial \gamma}{\partial t}(t) \quad \in \quad F_{\gamma(t)}$$

Example: $\mathbb{R}^n$ with the constant vector fields $f_k(x) = (\ldots, 0, 1, 0, \ldots)$

Parallelizable manifolds

A parallelization of a manifold $\mathcal{M}$ of dimension n is an tuple of vector fields $(f_1, \ldots, f_n)$ s.t. for all $x \in \mathcal{M}$, $(f_1(x), \ldots, f_n(x))$ is a vector basis of the tangent space of $\mathcal{M}$ at x namely $T_x\mathcal{M}$.

A manifold $\mathcal{M}$ is said to be parallelizable when it admits a parallelization. All parallelizations of a given manifold are "isomorphic" (the frame manifold acts transitively on the set of parallelizations).

Conjecture: Any parallelization induces a local pospace structure on its underlying manifold. That local pospace structure does not depend on the parallelization.

Conjecture: Given a manifold $\mathcal{M}$ equipped with the local order induced by some parallelization, there exists a precubical set K whose local pospace realization is the local pospace $\mathcal{M}$.

Example: Every Lie group is parallelizable.

Example: It works for the circle! What about the spheres S^3 and S^7 ?

8. Conclusion

1. Connect a value analysis to the backend of the static analyzer ALCOOL
2. Prove that all precubical sets can be realized in the category of local pospaces
3. Extend the notion of category of components to realization of precubical sets and isothetic regions
4. Directed version of the Gelfand-Naimark-Segal theorem

1. Connect a value analysis to the backend of the static analyzer ALCOOL
2. Prove that all precubical sets can be realized in the category of local pospaces
3. Extend the notion of category of components to realization of precubical sets and isothetic regions
4. Directed version of the Gelfand-Naimark-Segal theorem

1. Connect a value analysis to the backend of the static analyzer ALCOOL
2. Prove that all precubical sets can be realized in the category of local pospaces
3. Extend the notion of category of components to realization of precubical sets and isothetic regions
4. Directed version of the Gelfand-Naimark-Segal theorem

1. Connect a value analysis to the backend of the static analyzer ALCOOL
2. Prove that all precubical sets can be realized in the category of local pospaces
3. Extend the notion of category of components to realization of precubical sets and isothetic regions
4. Directed version of the Gelfand-Naimark-Segal theorem

1. Connect a value analysis to the backend of the static analyzer ALCOOL
2. Prove that all precubical sets can be realized in the category of local pospaces
3. Extend the notion of category of components to realization of precubical sets and isothetic regions
4. Directed version of the Gelfand-Naimark-Segal theorem