Directed Algebraic Topology and Concurrency

Emmanuel Haucourt

MPRI : Concurrency (2.3)

Monday, the 30^{th} of January 2012

Control Flow Graph (CFG)

-

```
input x;
while x<>1
  do
    if x mod 2 = 0
    then x:=x/2
    else x:=3*x+1
  done
```

Control Flow Graph (CFG)

```
input x;
while x<>1
   do
      if x mod 2 = 0
      then x:=x/2
      else x:=3*x+1
   done
```

Control Flow Graph (CFG)

```
input x;
while x<>1
  do
    if x mod 2 = 0
    then x:=x/2
    else x:=3*x+1
  done
```

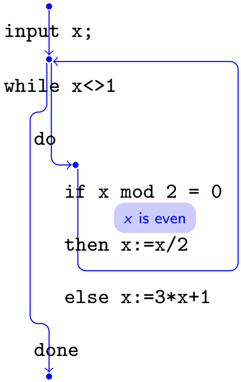
$x = 1$

Control Flow Graph (CFG)

```
input x;
while x<>1
  do
    if x mod 2 = 0
    then x:=x/2
    else x:=3*x+1
  done
```

$x \neq 1$

Control Flow Graph (CFG)

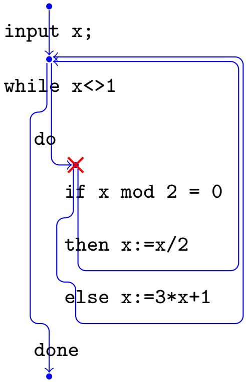


Control Flow Graph (CFG)

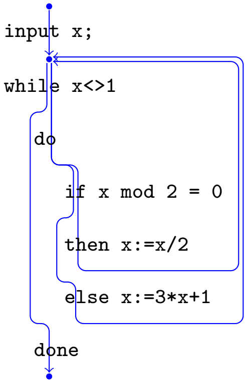
```
input x;
while x<>1
  do
    if x mod 2 = 0
    then x:=x/2
    else x:=3*x+1
  done
```

x is odd

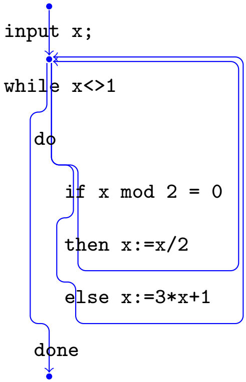
Control Flow Graph (CFG)

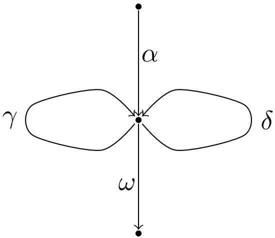


Control Flow Graph (CFG)



Control Flow Graph (CFG)



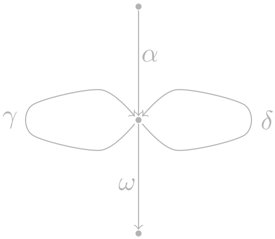


α stands for `input x` γ stands for `x:=x/2`

ω stands for "`exit`" δ stands for `x:=3*x+1`

An execution trace

```
input x;
while x<>1
  do
    if x mod 2 = 0
    then x:=x/2
    else x:=3*x+1
  done
```

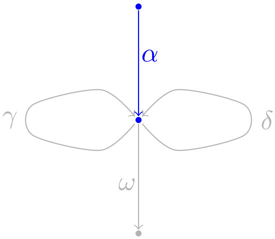


An execution trace

```
input x;
while x<>1
  do
    if x mod 2 = 0
    then x:=x/2
    else x:=3*x+1
  done
```

$x = 7$

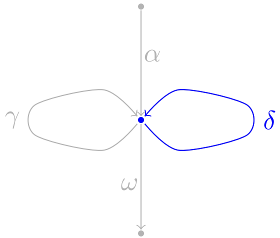
α



An execution trace

```
input x;
while x<>1
  do
    if x mod 2 = 0
    then x:=x/2
    else x:=3*x+1
  done
```

$x = 22$

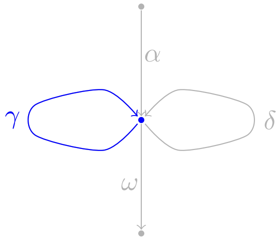


$\alpha\delta$

An execution trace

```
input x;
while x<>1
do
   if x mod 2 = 0
   then x:=x/2
   else x:=3*x+1
done
```

$x = 11$

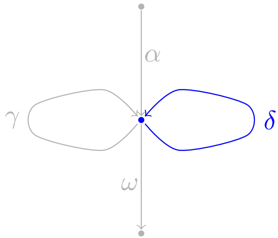


$\alpha\ \delta\ \gamma$

An execution trace

```
input x;
while x<>1
  do
    if x mod 2 = 0
    then x:=x/2
    else x:=3*x+1
  done
```

$x = 34$

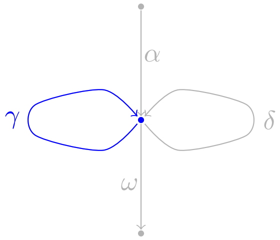


$\alpha\ \delta\ \gamma\ \delta$

An execution trace

```
input x;
while x<>1
  do
    if x mod 2 = 0
    then x:=x/2
    else x:=3*x+1
  done
```

$x = 17$

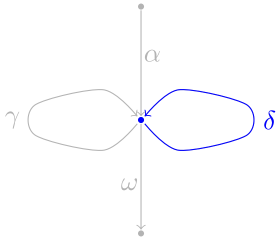


$\alpha\ \delta\ \gamma\ \delta\ \gamma$

An execution trace

```
input x;
while x<>1
  do
    if x mod 2 = 0
    then x:=x/2
    else x:=3*x+1
  done
```

$x = 52$

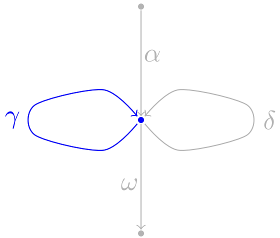


$\alpha\ \delta\ \gamma\ \delta\ \gamma\ \delta$

An execution trace

```
input x;
while x<>1
do
   if x mod 2 = 0
   then x:=x/2
   else x:=3*x+1
done
```

$x = 26$

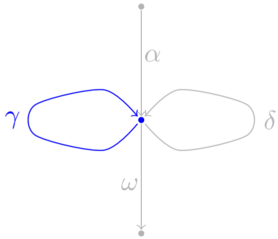


$\alpha\ \delta\ \gamma\ \delta\ \gamma\ \delta\ \gamma$

An execution trace

```
input x;
while x<>1
  do
    if x mod 2 = 0
    then x:=x/2
    else x:=3*x+1
  done
```

$x = 13$

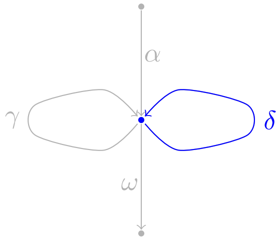


$\alpha\ \delta\ \gamma\ \delta\ \gamma\ \delta\ \gamma\ \gamma$

An execution trace

```
input x;
while x<>1
  do
    if x mod 2 = 0
    then x:=x/2
    else x:=3*x+1
  done
```

$x = 40$

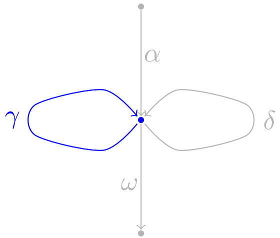


$\alpha\ \delta\ \gamma\ \delta\ \gamma\ \delta\ \gamma\ \gamma\ \delta$

An execution trace

```
input x;
while x<>1
  do
    if x mod 2 = 0
    then x:=x/2
    else x:=3*x+1
  done
```

$x = 20$

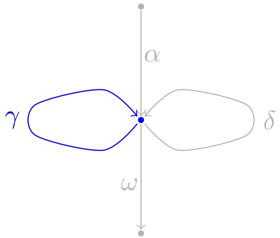


$\alpha\ \delta\ \gamma\ \delta\ \gamma\ \delta\ \gamma\ \gamma\ \delta\ \gamma$

An execution trace

```
input x;
while x<>1
  do
    if x mod 2 = 0
    then x:=x/2
    else x:=3*x+1
  done
```

$x = 10$

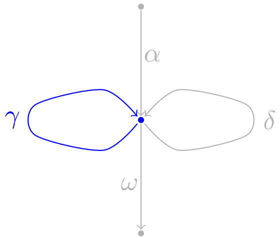


$\alpha\ \delta\ \gamma\ \delta\ \gamma\ \delta\ \gamma\ \gamma\ \delta\ \gamma\ \gamma$

An execution trace

```
input x;
while x<>1
  do
    if x mod 2 = 0
    then x:=x/2
    else x:=3*x+1
  done
```

$x = 5$

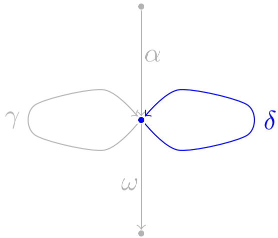


$$\alpha\ \delta\ \gamma\ \delta\ \gamma\ \delta\ \gamma\ \gamma\ \delta\ \gamma\ \gamma\ \gamma$$

An execution trace

```
input x;
while x<>1
  do
    if x mod 2 = 0
    then x:=x/2
    else x:=3*x+1
  done
```

$x = 16$

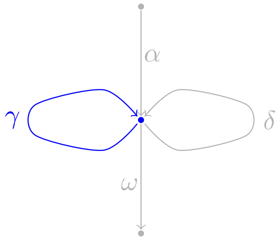


$\alpha\ \delta\ \gamma\ \delta\ \gamma\ \delta\ \gamma\ \gamma\ \delta\ \gamma\ \gamma\ \gamma\ \delta$

An execution trace

```
input x;
while x<>1
do
   if x mod 2 = 0
   then x:=x/2
   else x:=3*x+1
done
```

$x = 8$

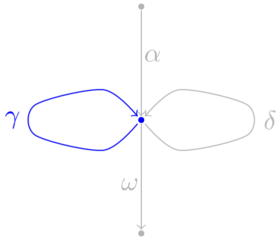


$\alpha\ \delta\ \gamma\ \delta\ \gamma\ \delta\ \gamma\ \gamma\ \delta\ \gamma\ \gamma\ \gamma\ \delta\ \gamma$

An execution trace

```
input x;
while x<>1
  do
    if x mod 2 = 0
    then x:=x/2
    else x:=3*x+1
  done
```

$x = 4$

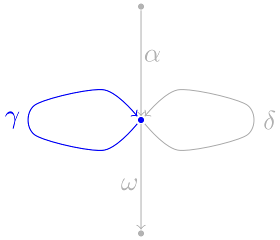


$$\alpha\ \delta\ \gamma\ \delta\ \gamma\ \delta\ \gamma\ \gamma\ \delta\ \gamma\ \gamma\ \gamma\ \delta\ \gamma\ \gamma$$

An execution trace

```
input x;
while x<>1
  do
    if x mod 2 = 0
    then x:=x/2
    else x:=3*x+1
  done
```

$x = 2$

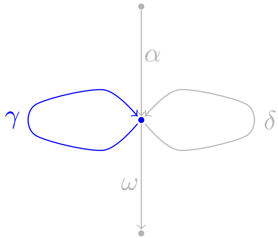


$\alpha\ \delta\ \gamma\ \delta\ \gamma\ \delta\ \gamma\ \gamma\ \delta\ \gamma\ \gamma\ \gamma\ \delta\ \gamma\ \gamma\ \gamma$

An execution trace

```
input x;
while x<>1
do
  if x mod 2 = 0
  then x:=x/2
  else x:=3*x+1
done
```

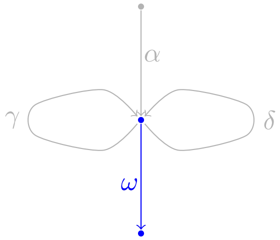
$x = 1$



$\alpha\ \delta\ \gamma\ \delta\ \gamma\ \delta\ \gamma\ \gamma\ \delta\ \gamma\ \gamma\ \gamma\ \delta\ \gamma\ \gamma\ \gamma\ \gamma$

An execution trace

```
input x;
while x<>1
  do
    if x mod 2 = 0
    then x:=x/2
    else x:=3*x+1
  done
```



$\alpha\ \delta\ \gamma\ \delta\ \gamma\ \delta\ \gamma\ \gamma\ \delta\ \gamma\ \gamma\ \gamma\ \delta\ \gamma\ \gamma\ \gamma\ \gamma\ \omega$

Execution traces of a program

as paths over its CFG

- Any execution trace induces a path
- Some paths do not come from an execution trace

Execution traces of a program

as paths over its CFG

- Any execution trace induces a path
- Some paths do not come from an execution trace

Therefore the collection of path provides a (strict) overapproximation of the collection of execution traces

Execution traces of a program
as paths over its CFG

- Any execution trace induces a path
- Some paths do not come from an execution trace

Therefore the collection of path provides a (strict) overapproximation of the collection of execution traces

The (infinite) collection of paths is entirely determined by the (finite) CFG

The overall idea
of static analysis

The model of a program should be the finite representation of an overapproximation of the collection of all its execution traces.

Category $\mathcal{C}$

Definition (the "underlying graph" part)

$\mathrm{Ob}(\mathcal{C})$: collection of objects

$\mathrm{Mo}(\mathcal{C})$: collection of morphisms

s, t : mappings source, target as follows

$$\mathrm{Mo}(\mathcal{C}) \overset{s}{\underset{t}{\rightrightarrows}} \mathrm{Ob}(\mathcal{C})$$

We define the homset $\mathcal{C}[x,y] := \left\{\gamma \in \mathrm{Mo}(\mathcal{C}) \;\middle|\; \mathrm{s}(\gamma) = x \text{ and } \mathrm{t}(\gamma) = y\right\}$

Category $\mathcal{C}$

Definition (the "underlying local monoid" part)

id : provides each object with an identity

$$\mathrm{Mo}(\mathcal{C}) \underset{t}{\overset{s}{\rightrightarrows}} \mathrm{Ob}(\mathcal{C}) \qquad \mathrm{Mo}(\mathcal{C}) \xleftarrow{\mathrm{id}} \mathrm{Ob}(\mathcal{C})$$

The (local) composition is a partially defined binary operation often denoted by $\circ$

$$\Big\{(\gamma,\delta) \;\Big|\; \gamma,\delta \text{ morphisms of } \mathcal{C} \text{ s.t. } \mathrm{s}(\gamma)=\mathrm{t}(\delta)\Big\} \xrightarrow{\text{composition}} \mathrm{Mo}(\mathcal{C})$$

Category $\mathcal{C}$

Definition (the axioms)

$$\begin{tikzcd} & \mathsf{t}(\delta) = \mathsf{s}(\gamma) \arrow[dr, "\gamma"] & \\ \mathsf{s}(\delta) \arrow[ur, "\delta"] \arrow[rr, "\gamma\circ\delta"'] & & \mathsf{t}(\gamma) \end{tikzcd}$$

The composition law is associative

For all morphisms γ one has $\mathsf{id}_{\mathsf{t}(\gamma)} \circ \gamma = \gamma = \gamma \circ \mathsf{id}_{\mathsf{s}(\gamma)}$

$$\begin{tikzcd} x \arrow[loop left, "\mathsf{id}_x"] \end{tikzcd}$$

For all objects x one has $\mathsf{s}(\mathsf{id}_x) = x = \mathsf{t}(\mathsf{id}_x)$

Category of paths (1)
freely generated by a graph

- I_n is the *finite linear order* with $n+1$ elements

$$\overset{0}{\bullet} \longrightarrow \overset{1}{\bullet} \longrightarrow \overset{2}{\bullet} \longrightarrow \overset{3}{\bullet} \longrightarrow \cdots \overset{n}{\bullet}$$

- A path γ on G is a *morphism* of graphs from I_n to G

- The *source* and the *target* of γ are $\gamma(0)$ and $\gamma(n)$

Category of paths (2)

freely generated by a graph

- Given two paths γ (over I_n) and δ (over I_m) such that $tgt(\delta) = src(\gamma)$ we can define the *concatenation* $\delta \cdot \gamma$ as the following path

$$\begin{array}{rcl} I_{n+m} & \longrightarrow & G \\ k & \longmapsto & \begin{cases} \delta(\overrightarrow{k}) & \text{if } 0 \leqslant k < n \\ \gamma(\overrightarrow{k-n}) & \text{if } n \leqslant k < n+m \end{cases} \end{array}$$

where $\overrightarrow{k}$ stands for the arrow $(k, k+1)$ of I_n

- The concatenation is *associative*

- If γ (resp. δ) is defined over I_0 then $\delta \cdot \gamma = \delta$ (resp. γ)

We defined F(G) also called the *Free Category over G*.

Model of a sequential program P

with G_P the control flow graph of P

The model of the program is defined as the *category of paths* over its control flow graph

$$[\![P]\!] := F(G_p)$$

Cartesian product
in $\mathcal{Set}$

$$A \times B := \{(a, b) \mid a \in A \text{ and } b \in B\}$$

There exist two mappings π_A and π_B

$$\begin{aligned} \pi_A :\ A \times B &\longrightarrow A \\ (a, b) &\longmapsto a \end{aligned} \qquad\qquad \begin{aligned} \pi_B :\ A \times B &\longrightarrow B \\ (a, b) &\longmapsto b \end{aligned}$$

such that for all sets X the following map is a bijection

$$\begin{aligned} \mathcal{Set}[X, A \times B] &\longrightarrow \mathcal{Set}[X, A] \times \mathcal{Set}[X, B] \\ h &\longmapsto (\ \pi_A \circ h\ ,\ \pi_B \circ h\ ) \end{aligned}$$

Cartesian product
in a category $\mathcal{C}$

The object c is the Cartesian product (in $\mathcal{C}$) of a and b when there exist two morphisms $\pi_a : c \to a$ and $\pi_b : c \to b$ such that for all objects x of $\mathcal{C}$ the following map is a bijection

$$\begin{array}{ccc} \mathcal{C}[x,c] & \longrightarrow & \mathcal{C}[x,a]\times\mathcal{C}[x,b] \\ h & \longmapsto & (\ \pi_a\circ h\ ,\ \pi_b\circ h\ ) \end{array}$$

When such an object c exists we write $c = a \times b$

Cartesian products

modelling Concurrency

A family $P_1, \ldots, P_n$ of programs is independent if

$$[\![P_1|\cdots|P_n]\!] \cong [\![P_1]\!] \times \cdots \times [\![P_n]\!]$$

Example

Cartesian product in the category of graphs ($\mathcal{Grph}$)

The elements of V are the vertices and those of A are the arrows
In particular A and V are sets

Objects | Morphisms | Composition

$$\begin{tikzcd} A \arrow[d, shift right, "s"'] \arrow[d, shift left, "t"] \\ V \end{tikzcd}
\qquad
\begin{tikzcd} A \arrow[r, "\phi_1"] \arrow[d, shift right, "s"'] \arrow[d, shift left, "t"] & A' \arrow[d, shift right, "s'"'] \arrow[d, shift left, "t'"] \\ V \arrow[r, "\phi_0"'] & V' \end{tikzcd}
\qquad
\begin{tikzcd} A \arrow[r, "\phi_1"] \arrow[d, shift right, "s"'] \arrow[d, shift left, "t"] & A' \arrow[r, "\psi_1"] \arrow[d, shift right, "s'"'] \arrow[d, shift left, "t'"] & A'' \arrow[d, shift right, "s''"'] \arrow[d, shift left, "t''"] \\ V \arrow[r, "\phi_0"'] & V' \arrow[r, "\psi_0"'] & V'' \end{tikzcd}$$

with $\mathrm{s}'(\phi_1(\alpha)) = \phi_0(\mathrm{s}(\alpha))$ and $\mathrm{t}'(\phi_1(\alpha)) = \phi_0(\mathrm{t}(\alpha))$

Example

Cartesian product in the category of graphs ($\mathcal{Grph}$)

The elements of V are the vertices and those of A are the arrows
In particular A and V are sets

Objects — Morphisms — Composition

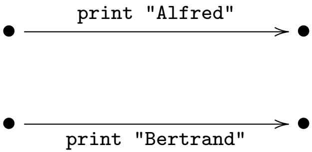
$$
\begin{tikzcd}
A \arrow[d, shift right, "s"'] \arrow[d, shift left, "t"] \\
V
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, "\phi_1"] \arrow[d, shift right, "s"'] \arrow[d, shift left, "t"] & A' \arrow[d, shift right, "s'"'] \arrow[d, shift left, "t'"] \\
V \arrow[r, "\phi_0"'] & V'
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, "\phi_1"] \arrow[d, shift right, "s"'] \arrow[d, shift left, "t"] & A' \arrow[r, "\psi_1"] \arrow[d, shift right, "s'"'] \arrow[d, shift left, "t'"] & A'' \arrow[d, shift right, "s''"'] \arrow[d, shift left, "t''"] \\
V \arrow[r, "\phi_0"'] & V' \arrow[r, "\psi_0"'] & V''
\end{tikzcd}
$$

with $\mathrm{s}'(\phi_1(\alpha)) = \phi_0(\mathrm{s}(\alpha))$ and $\mathrm{t}'(\phi_1(\alpha)) = \phi_0(\mathrm{t}(\alpha))$

$$
\left(\begin{tikzcd}
A \arrow[d, shift right, "\mathrm{t}"'] \arrow[d, shift left, "\mathrm{s}"] \\
V
\end{tikzcd}\right)
\times
\left(\begin{tikzcd}
A' \arrow[d, shift right, "\mathrm{t}'"'] \arrow[d, shift left, "\mathrm{s}'"] \\
V'
\end{tikzcd}\right)
\cong
\left(\begin{tikzcd}
A \times A' \arrow[d, shift right, "\mathrm{t}\times\mathrm{t}'"'] \arrow[d, shift left, "\mathrm{s}\times\mathrm{s}'"] \\
V \times V'
\end{tikzcd}\right)
$$

The Cartesian product in $\mathcal{Grph}$ is deduced form the Cartesian product in $\mathcal{Set}$

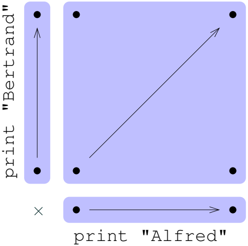
Two simple sequential programs

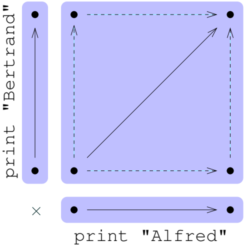


Two simple sequential processes running concurrently

What goes wrong with the graphs

What we have product in *Grph*	What we expect product in *Cat*



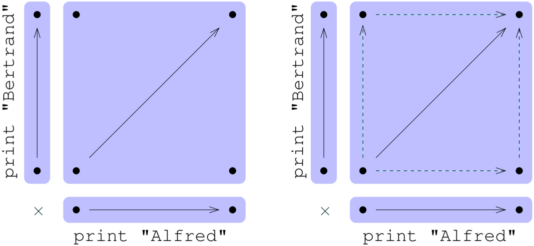


Two simple sequential processes running concurrently

What goes wrong with the graphs

What we have
product in $\mathcal{Grph}$

What we expect
product in $\mathcal{Cat}$



Given two graphs G and G' we have

$$F(G \times G') \not\cong F(G) \times F(G')$$

Topological spaces
reminder

A topological space is a set X and a collection $\Omega_X \subseteq \mathcal{P}(X)$ s.t.

Topological spaces

reminder

A topological space is a set X and a collection $\Omega_X \subseteq \mathcal{P}(X)$ s.t.

1) $\emptyset \in \Omega_X$ and $X \in \Omega_X$

Topological spaces
reminder

A topological space is a set X and a collection $\Omega_X \subseteq \mathcal{P}(X)$ s.t.

1) $\emptyset \in \Omega_X$ and $X \in \Omega_X$
2) Ω_X is stable under union

Topological spaces
reminder

A topological space is a set X and a collection $\Omega_X \subseteq \mathcal{P}(X)$ s.t.

1) $\emptyset \in \Omega_X$ and $X \in \Omega_X$
2) Ω_X is stable under union
3) Ω_X is stable under finite intersection

Topological spaces

reminder

A topological space is a set X and a collection $\Omega_X \subseteq \mathcal{P}(X)$ s.t.

1) $\emptyset \in \Omega_X$ and $X \in \Omega_X$
2) Ω_X is stable under union
3) Ω_X is stable under finite intersection

A continuous map $f : (X, \Omega_X) \to (Y, \Omega_Y)$ is a map $f : X \to Y$ s.t.

$$\forall U \in \Omega_Y \ f^{-1}(U) \in \Omega_X$$

Topological spaces
reminder

A topological space is a set X and a collection $\Omega_X \subseteq \mathcal{P}(X)$ s.t.

1) $\emptyset \in \Omega_X$ and $X \in \Omega_X$
2) Ω_X is stable under union
3) Ω_X is stable under finite intersection

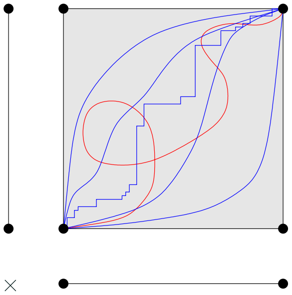
A continuous map $f : (X, \Omega_X) \to (Y, \Omega_Y)$ is a map $f : X \to Y$ s.t.

$$\forall U \in \Omega_Y \ f^{-1}(U) \in \Omega_X$$

Topological spaces and continuous maps form the category $\mathcal{Top}$

Two simple sequential processes running concurrently
The topological model

Working in $\mathcal{Top}$ instead of $\mathcal{Grph}$ we have



Functors f from $\mathcal{C}$ to $\mathcal{D}$

Definition (preserving the "underlying graph")

A functor $f : \mathcal{C} \to \mathcal{D}$ is defined by two "mappings" $\mathrm{Ob}(f)$ and $\mathrm{Mo}(f)$ such that

$$\begin{tikzcd}
\mathrm{Mo}(\mathcal{C}) \arrow[r, shift left, "\mathrm{s}"] \arrow[r, shift right, "\mathrm{t}"'] \arrow[d, "\mathrm{Mo}(f)"'] & \mathrm{Ob}(\mathcal{C}) \arrow[d, "\mathrm{Ob}(f)"] \\
\mathrm{Mo}(\mathcal{D}) \arrow[r, shift left, "\mathrm{s'}"] \arrow[r, shift right, "\mathrm{t'}"'] & \mathrm{Ob}(\mathcal{D})
\end{tikzcd}$$

with $\mathrm{s}'(\mathrm{Mo}(f)(\alpha)) = \mathrm{Ob}(f)(\mathrm{s}(\alpha))$ and $\mathrm{t}'(\mathrm{Mo}(f)(\alpha)) = \mathrm{Ob}(f)(\mathrm{t}(\alpha))$

Hence it is in particular a morphism of graphs.

Functors f from $\mathcal{C}$ to $\mathcal{D}$

Definition (preserving the "underlying local monoid")

The "mappings" Ob(f) and Mo(f) also make the following diagram commute

$$\begin{tikzcd}
\mathsf{Mo}(\mathcal{C}) \arrow[d, "\mathsf{Mo}(f)"'] & \mathsf{Ob}(\mathcal{C}) \arrow[l, "\mathrm{id}"'] \arrow[d, "\mathsf{Ob}(f)"] \\
\mathsf{Mo}(\mathcal{D}) & \mathsf{Ob}(\mathcal{D}) \arrow[l, "\mathrm{id}'"]
\end{tikzcd}$$

and satisfies $\mathsf{Mo}(f)(\gamma\circ\delta) = \mathsf{Mo}(f)(\gamma)\circ\mathsf{Mo}(f)(\delta)$

$$\begin{tikzcd}
x \arrow[r, "\delta"'] \arrow[rr, bend left, "\gamma\circ\delta"] & y \arrow[r, "\gamma"'] & z
\end{tikzcd}
\qquad
\begin{tikzcd}
f(x) \arrow[r, "f(\delta)"'] \arrow[rr, bend left, "f(\gamma\circ\delta)"] & f(y) \arrow[r, "f(\gamma)"'] & f(z)
\end{tikzcd}$$

Functors compose as morphisms of graphs do

$$
\begin{tikzcd}
\mathsf{Mo}(\mathcal{C}) \arrow[r, "\mathsf{Mo}(f)"] \arrow[rr, bend left, "\mathsf{Mo}(g \circ f)"] \arrow[d, shift right, "\mathsf{t}"'] \arrow[d, shift left, "\mathsf{s}"] & \mathsf{Mo}(\mathcal{D}) \arrow[r, "\mathsf{Mo}(g)"] \arrow[d, shift right, "\mathsf{t}'"'] \arrow[d, shift left, "\mathsf{s}'"] & \mathsf{Mo}(\mathcal{E}) \arrow[d, shift right, "\mathsf{t}''"'] \arrow[d, shift left, "\mathsf{s}''"] \\
\mathsf{Ob}(\mathcal{C}) \arrow[r, "\mathsf{Ob}(f)"'] \arrow[rr, bend right, "\mathsf{Ob}(g \circ f)"'] & \mathsf{Ob}(\mathcal{D}) \arrow[r, "\mathsf{Ob}(g)"'] & \mathsf{Ob}(\mathcal{E})
\end{tikzcd}
$$

Hence the functors should be thought of as the morphisms of categories

The small categories and their funtors form a (large) category denoted by $\mathcal{C}at$

Isomorphisms
of a category $\mathcal{C}$

A morphism $\gamma \in \mathcal{C}[x, y]$ is an isomorphism when there exists $\delta \in \mathcal{C}[y, x]$ s.t.

$$\gamma \circ \delta = \mathrm{id}_{t(\gamma)} \text{ and } \delta \circ \gamma = \mathrm{id}_{s(\gamma)}$$

In this case δ is unique and we write $\delta = \gamma^{-1}$

$$x \underset{\delta=\gamma^{-1}}{\overset{\gamma=\delta^{-1}}{\rightleftarrows}} y$$

We also say that x and y are isomorphic which is denoted by $x \cong y$
A category in which every morphism is an isomorphism is called a groupoid

The overall idea of Algebraic Topology

Any functor preserve the isomorphisms

Problem: prove the topological spaces X and Y are *not* the same
Strategy: find a functor F defined over $\mathcal{T}op$ such that $F(X) \ncong F(Y)$

In this case, if $X = [\![P]\!]$ and $Y = [\![Q]\!]$ then the programs P and Q do not have the same behaviour.

The connected component functor
from $\mathcal{T}op$ to $\mathcal{S}et$

1) A topologcial space X is the disjoint sum of its *connected* components
2) Any *connected* subset of X is contained in a *connected* component of X
3) Any continuous direct image of a *connected* subset of X is *connected*

$$\mathcal{T}op \xrightarrow{\pi_0} \mathcal{S}et$$

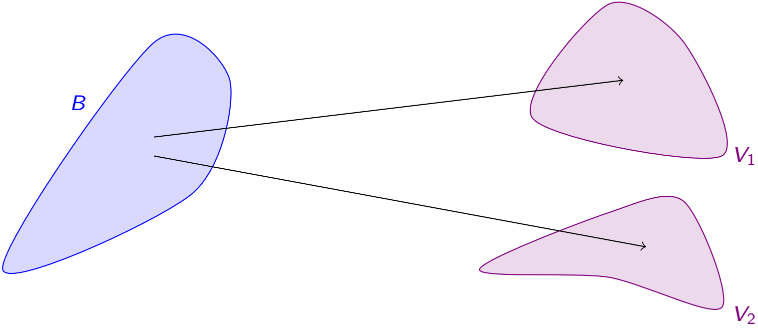
$$\begin{array}{ccc} X & & \pi_0(X) \\ f \downarrow & \longmapsto & \downarrow \pi_0(f) \\ Y & & \pi_0(Y) \end{array}$$

Moreover we have $\pi_0(g \circ f) = \pi_0(g) \circ \pi_0(f)$

An application involving basic (algebraic) topology

The continuous image of a connected space is connected

The image of the space B is entirely contained in a *connected component* of the space V.



The set of connected components
is a functorial construction

This situation is abstracted by classifying continuous maps from B to V according to which connected component (V_1 or V_2) the single connected components of B (namely B itself) is sent to. There are exactly two set theoretic maps from the singleton $\{B\}$ to the pair $\{V_1, V_2\}$ hence there is at most (in fact exactly) two kinds of continuous maps from B to V.

$$\{B\} \rightrightarrows \{V_1, V_2\}$$

In particular B and V are *not homeomorphic*.

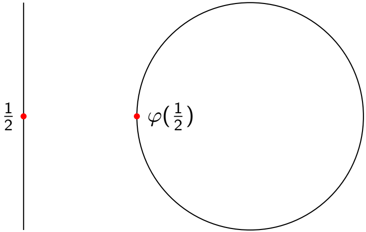
Application

The compact interval and the circle are not homeomorphic

Let $\mathbb{S}^1 := \left\{ z \in \mathbb{C} \;\middle|\; |z| = 1 \right\}$ be the Euclidean circle.
Suppose $\varphi : [0,1] \to \mathbb{S}^1$ is a homeomorphism. Then φ induces a homeomorphism

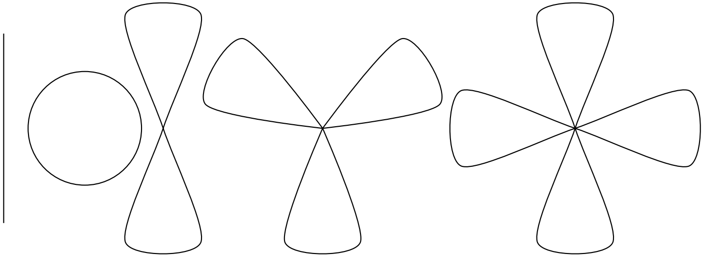
$$[0, \tfrac{1}{2}[\ \cup\ ]\tfrac{1}{2}, 1] \ \to\ \mathbb{S}^1 \backslash \{\varphi(\tfrac{1}{2})\}$$

which does not exist!



Generalization
Bouquets of circles

These topological spaces are pairwise not homeomorphic. Why ?



Examples of large categories
used in (directed) algebraic topology

$\mathcal{Set}$: sets and mappings
$\mathcal{Top}$: topological spaces and continuous maps
$\mathcal{Pre}$: preordered sets and preorder preserving maps
$\mathcal{Pos}$: partially ordered sets and order preserving maps
$\mathcal{Mon}$: Monoids and their morphisms
$\mathcal{Cmon}$: Commutative monoids and their morphisms
$\mathcal{Gr}$: Groups and their morphisms

$\mathcal{Ab}$: Abelian groups and their morphisms

Example of small categories

$(\mathbb{N}, \leqslant)$: set of objects $\mathbb{N}$
(guess the remaining)

$(\mathbb{N}, +, 0)$: set of morphisms $\mathbb{N}$
(guess the remaining)

The same way, any poset or monoid
can be seen as a small category

F(G): The category freely generated by the graph G

The *Prolaag-Verhogen* language
Edsger Wybe Dijkstra (1968)

$\mathcal{S}$: set of semaphores and $\alpha : \mathcal{S} \to \mathbb{N} \backslash \{0, 1\}$ associates each semaphore $\mathtt{s}$ with its arity $\alpha_{\mathtt{s}} \geq 2$.
Hypothesis : For all $\alpha \geq 2$, there exist infinitely many semaphores whose arity is α.
`P(s)` and `V(s)` are the only instructions (where $\mathtt{s} \in \mathcal{S}$) of the language.
A processes P is a finite sequence of instructions, $P(j)$ the j^{th} instruction with $j \geqslant 1$.

`P(a).V(a)` and `P(a).P(b).V(a).V(b)`

A PV program is a finite sequence of processes

`P(a).V(a) | P(a).V(a)`

`P(a).P(b).V(a).V(b) | P(b).P(a).V(b).V(a)`

Therefore a PV program can be seen as a matrix of instructions each line of which being a process. The operator . bounds tighter that the operator |

PV Programs as heterogeneous matrices of instructions

PV program = vector of processes $\overrightarrow{P}$
$\overrightarrow{P}_i$ = i^{th} process of the program
$\overrightarrow{P}_i(j)$ = j^{th} instruction of the i^{th} process.
l_i = number of instructions of the process $\overrightarrow{P}_i$ (indexed from 1 to l_i)

$$\mathrm{dom}(\overrightarrow{P}) := \{0, \ldots, l_1\} \times \cdots \times \{0, \ldots, l_n\}$$

One has intentionally included 0

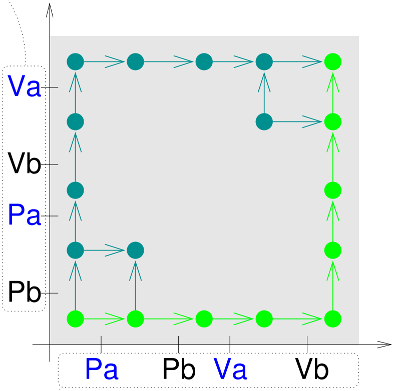
Intuition

- `P` stands for "prolaag" (short for "probeer te verlagen" i.e. "try to reduce" in Dutch) and `P(s)` means: take an occurence of the semaphore `s` from the pool of resources, but wait if none is available.

- `V` stands for "verhogen" ("increase" in Dutch) and `V(s)` means: release an occurence of the semaphore `s`, if the process trying to perform this action does not hold any occurence of `s` then the instruction is just ignored and the process keeps on running.

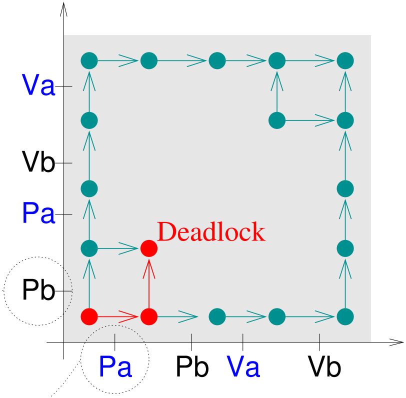
An example of trace

T_1	T_2
Pa	–
Pb	–
Va	–
Vb	–
–	Pb
–	Pa
–	Vb
–	Va



Another example of trace

T_1	T_2
Pa	—
—	Pb



Semaphore held by a process

The real positive half-line is $\mathbb{R}_+ = [0, +\infty[$
For each process P, each semaphore $\mathtt{s}$ and each point $x \in \mathbb{R}_+$, we define

$$a_x := \max \left\{ k \in \mathbb{N} \;\middle|\; k \leq x \text{ et } P(k) = \mathtt{P(s)} \right\}$$

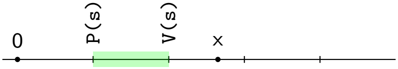
and

$$b_x := \min \left\{ k \in \mathbb{N} \;\middle|\; a_x \leq k \text{ et } P(k) = \mathtt{V(s)} \right\}$$

with the (unusual) convention that $\max \emptyset = \min \emptyset = \infty$.
The occupied/held part of $\mathtt{s}$ is

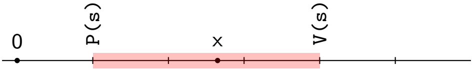
$$B_{\mathtt{s}}(P) := \left\{ x \in \mathbb{R}_+ \;\middle|\; x \in [a_x, b_x[\right\}$$

Example



x not occupied

$$a_x = 1 \text{ and } b_x = 2$$



x occupied

$$a_x = 1 \text{ and } b_x = 4$$

The forbidden area generated by a semaphore s of a PV program $\overrightarrow{P} = P_1 | \ldots | P_n$

The indicator function of the set $B_s(P)$

$$\begin{array}{rcl} \chi_P^s : \mathbb{R}_+ & \longrightarrow & \{0,1\} \\ x & \longmapsto & \begin{cases} 1 & \text{if } x \in B_s(P) \\ 0 & \text{otherwise} \end{cases} \end{array}$$

For $\overrightarrow{f} := (f_1, \ldots, f_n)$ n-uple of functions $\mathbb{R}_+ \to \mathbb{R}$ and $\overrightarrow{x} := (x_1, \ldots, x_n) \in \mathbb{R}_+^n$

$$\overrightarrow{f} \cdot \overrightarrow{x} := \sum_{i=1}^n f_i(x_i)$$

Let $\overrightarrow{\chi}$ be the n-uple $(\chi_{P_1}^s, \ldots, \chi_{P_n}^s)$ of indicators of the sets $B_s(P_1), \ldots, B_s(P_n)$

The forbidden region generated by s of arity α is

$$F_s := \{\overrightarrow{x} \in \mathbb{R}_+^n \mid \overrightarrow{\chi} \cdot \overrightarrow{x} \geq \alpha\}$$

Forbidden area and Model

of a PV program $\overrightarrow{P} = P_1 \,|\, \ldots \,|\, P_n$

The forbidden area of the program $\overrightarrow{P}$ is

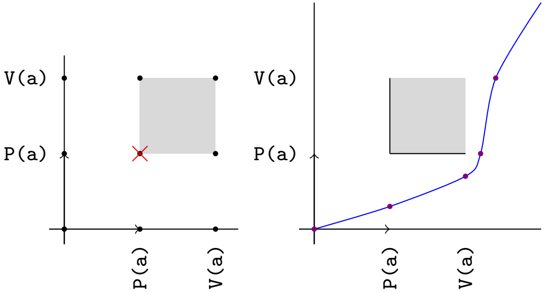
$$F := \bigcup_{\mathrm{s} \in \mathcal{S}} F_{\mathrm{s}}$$

The model is the set theoretic complement (relatively to $\mathbb{R}_+^n$) of its forbidden area.

$$[\![P_1 \,|\, \ldots \,|\, P_n]\!] := \mathbb{R}_+^n \backslash F$$

Example

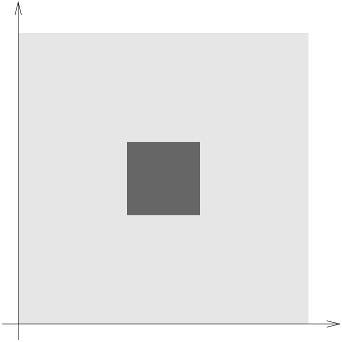
P(a).V(a) | P(a).V(a)

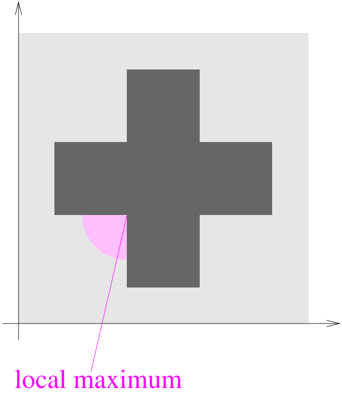


If we were working in $\mathbb{R}_+ \times \mathbb{R}_+$ then all the points of the grey square should be removed.

Taking direction into account

The following spaces are homeomorphic though the first one has no local maximum

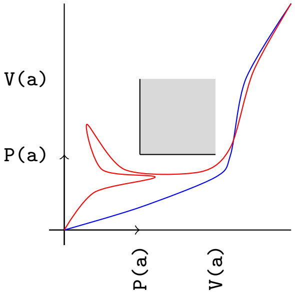




Topological spaces

are not enough

The homset $\mathcal{Top}\big[[0,1], [\![\overrightarrow{P}]\!]\big]$ still contains elements which do not correspond to any execution trace



Partially Ordered Spaces *or* Pospaces

Leopoldo Nachbin (1968)

A pospace is a topological space X and a partial order $\sqsubseteq$ over the underling set of X s.t.

$\{(x, y) \in X \times X \mid x \sqsubseteq y\}$ is closed in $X \times X$

The morphisms of pospaces are the continuous order preserving maps

The pospaces and their morphisms form the category $\mathcal{Po}$

The real line $\mathbb{R}$ (with its standard topology and order) provides a pospace

The products $\mathbb{R}^n$ with the product topology and product order are pospaces

Any subset of a pospace inherits a pospace structure

Models of PV program $\overrightarrow{P}$

using pospaces

The previous topological model $[\![\overrightarrow{P}]\!]$ inherits a pospace structures as a subset of $\mathbb{R}^n$

The paths on a pospace X are the elements of the homset

$$\mathcal{Po}\big[[0,1],X\big]$$

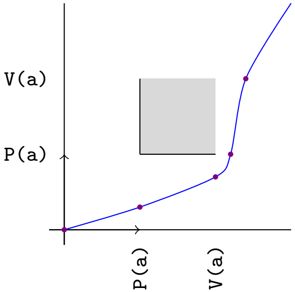
$$\forall\gamma\in\mathcal{Po}\big[[0,1],X\big],\ \gamma \text{ is constant iff } \gamma(0)=\gamma(1)$$

$$\forall\gamma\in\mathcal{Po}\big[[0,1],[0,1]\big],\ \gamma \text{ is onto (surjective) iff } \gamma(0)=0 \text{ and } \gamma(1)=1$$

Paths and Execution traces

Theorem

Given a PV program $\overrightarrow{P}$, any element of $\mathcal{P}o\left[[0,1],[\![\overrightarrow{P}]\!]\right]$ *induces an execution trace of $\overrightarrow{P}$ and conversely, any execution trace of $\overrightarrow{P}$ is induced by some element of* $\mathcal{P}o\left[[0,1],[\![\overrightarrow{P}]\!]\right]$



Direct image of paths on a pospaces

Characterization

The direct image $\mathrm{im}(f)$ of any $f \in \mathcal{P}o[X, Y]$ inherits a pospace struture still denoted by $\mathrm{im}(f)$

Theorem

Given a pospace X and $\gamma \in \mathcal{P}o\big[[0,1], X\big]$, we have $\mathrm{im}(\gamma) \cong \{*\}$ *or* $\mathrm{im}(\gamma) \cong [0,1]$