# Graded Types Parametricity:
## Principles and Application to Abstract Interpretation

Flavien BREUVART

LIPN, Paris XIII

Chocola: 15 November 2016

# Project CoGITARe :

## Combining Graded and Intersection Types for the Analyses of Resources

ANR JCJC with funding

**Begin in March** **1 PhD and 1y posdoc**

### Treated Issue : Abstract Interpretation of Functional Programs

An old objective with no satisfactory solution

### (Naive ?) Solution : Performing the Analysis at Type Level

Tried several times since the 90's with mixed results

### Modern approach : Graded, Intersection and Refinement Types

We will revisit old issues with a modern point of view

Today : Brief overview and focus in a specific issue and associated research directions

# Overview

# Contents

# Abstract Interpretation for Dummies

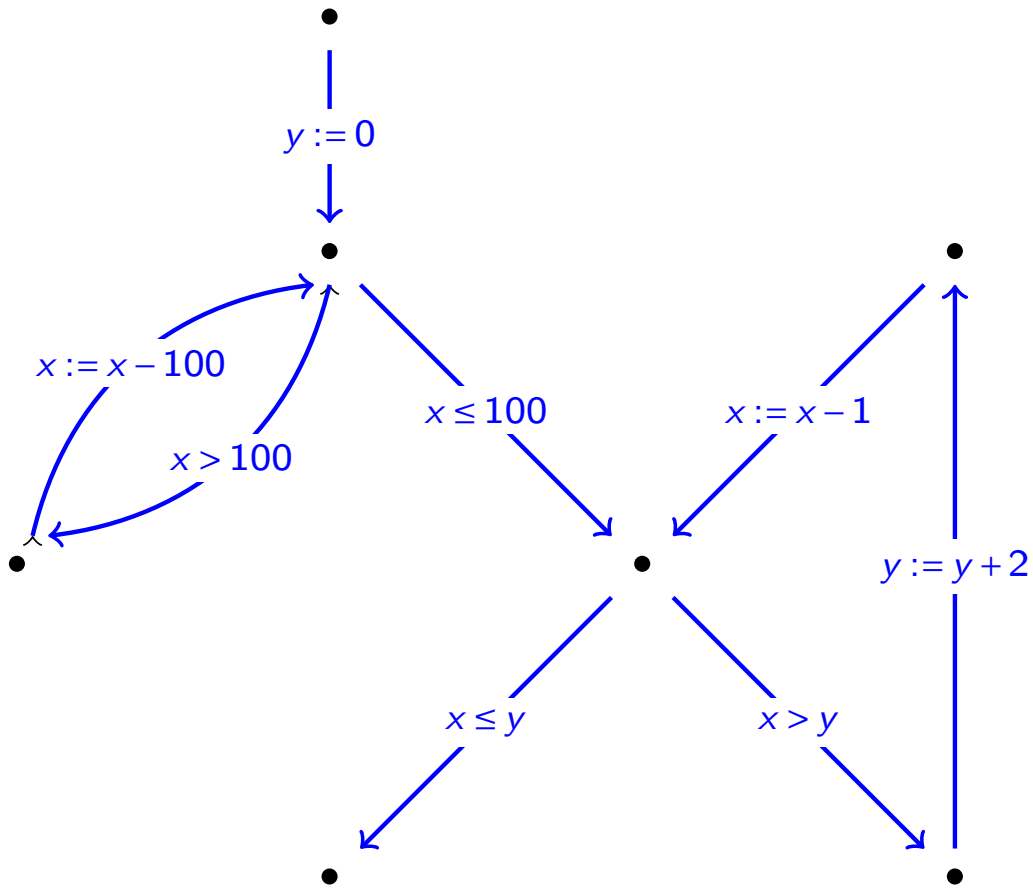
**Objective** : bound the set of reachable states

Both lower and uper bounds (Safety,liveness,quantitative)

# Abstract Interpretation for Dummies

**Idea 1** : Represent the program as an automata with memory (state)

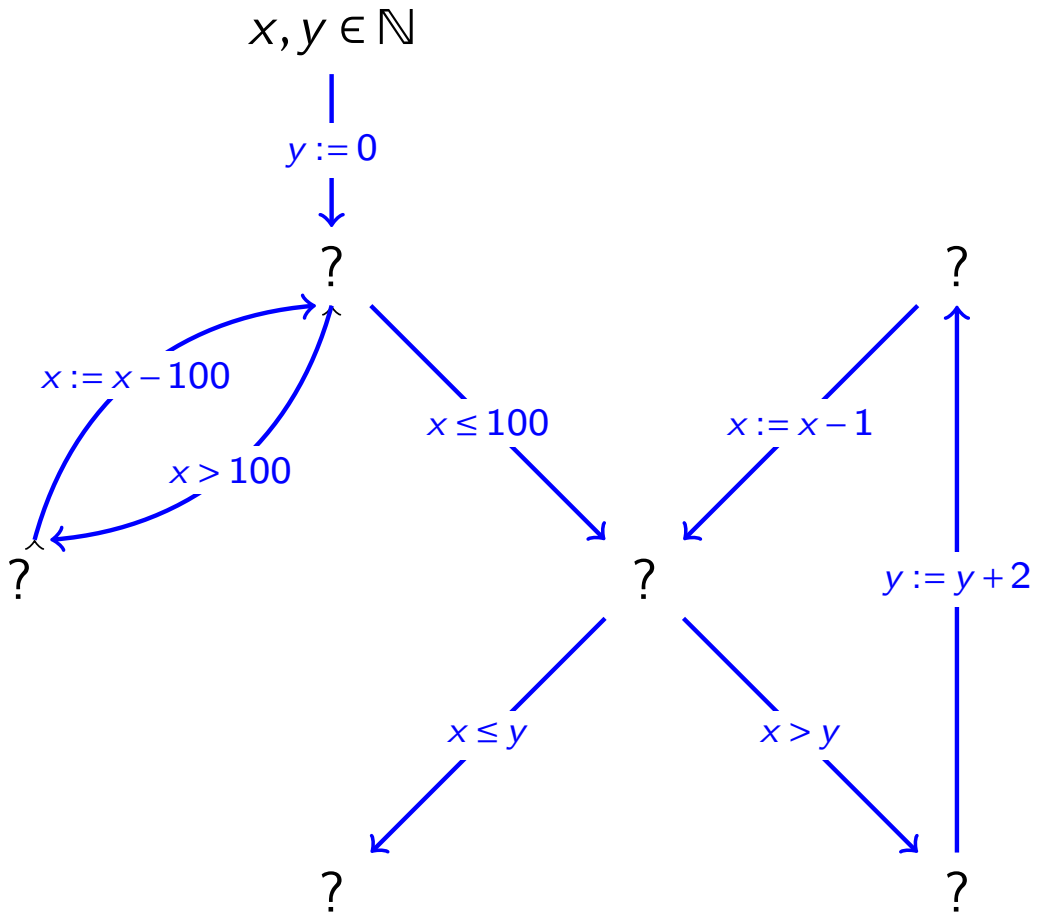
```
 y := 0 ;
while (x > 100) {
  x := x − 100 ;
}
while (x > y) {
  y := y + 2x ;
  x := x − 1 ;
}
```



# Abstract Interpretation for Dummies

**Idea 1** : Represent the program as an automata with memory (state)
**New goal** : Propagate possible states to the end

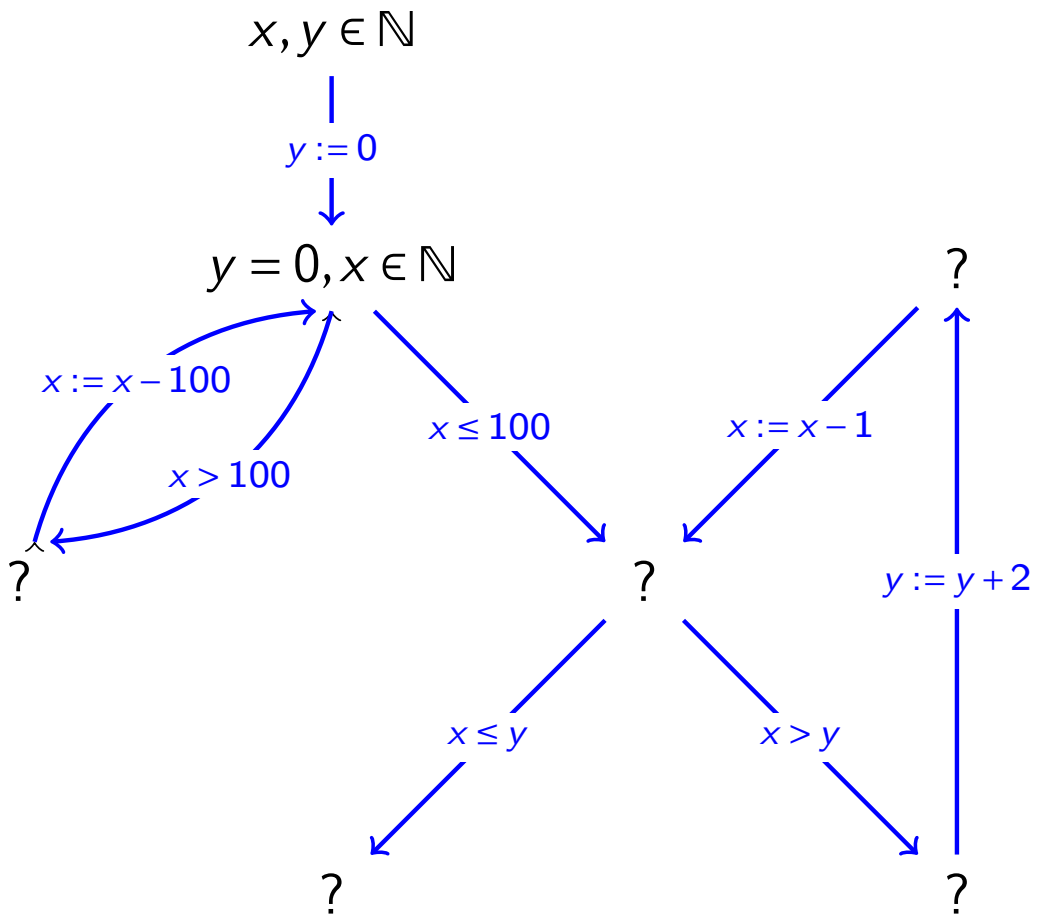
```
 y := 0 ;
while (x > 100) {
  x := x − 100 ;
}
while (x > y) {
  y := y + 2x ;
  x := x − 1 ;
}
```



# Abstract Interpretation for Dummies

**Idea 1** : Represent the program as an automata with memory (state)
**New goal** : Propagate possible states to the end

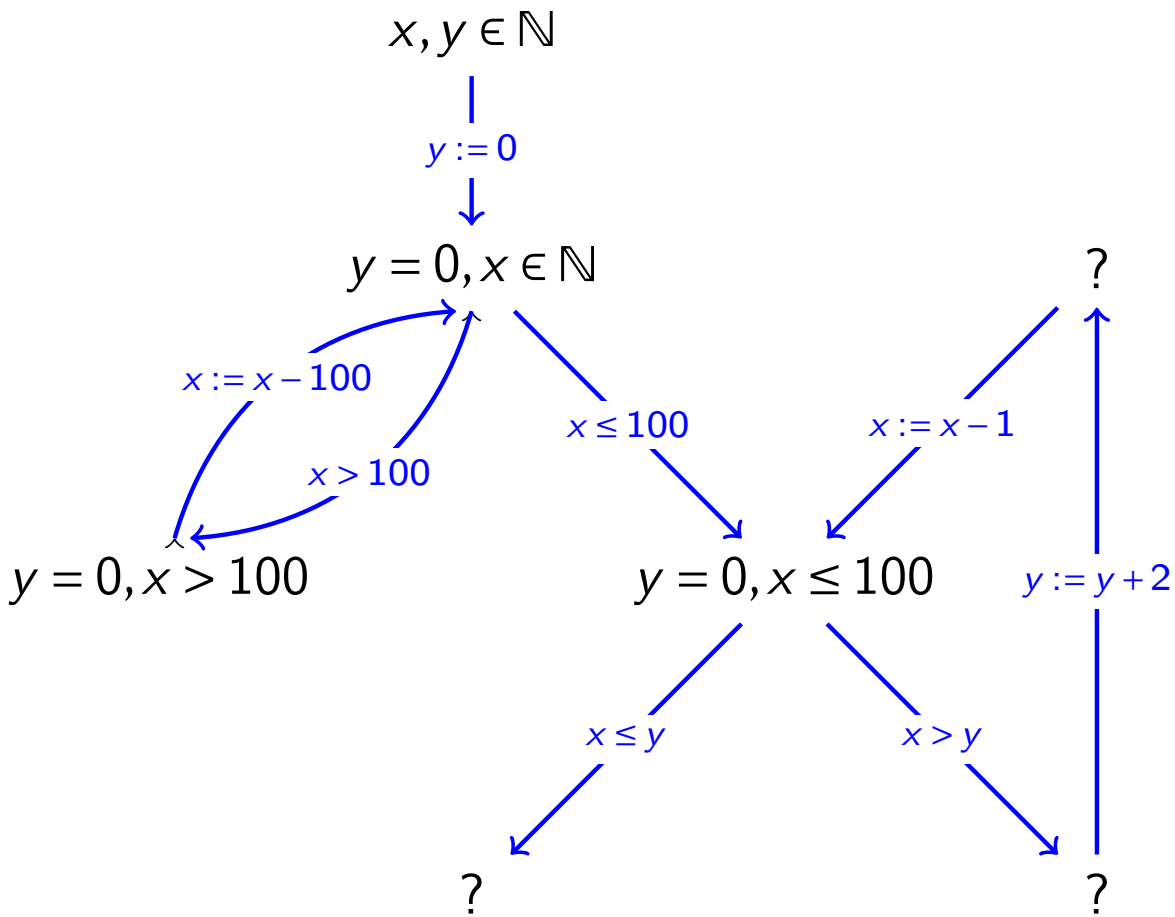
```
 y := 0 ;
while (x > 100) {
  x := x − 100 ;
}
while (x > y) {
  y := y + 2x ;
  x := x − 1 ;
}
```



# Abstract Interpretation for Dummies

**Idea 1** : Represent the program as an automata with memory (state)
**New goal** : Propagate possible states to the end

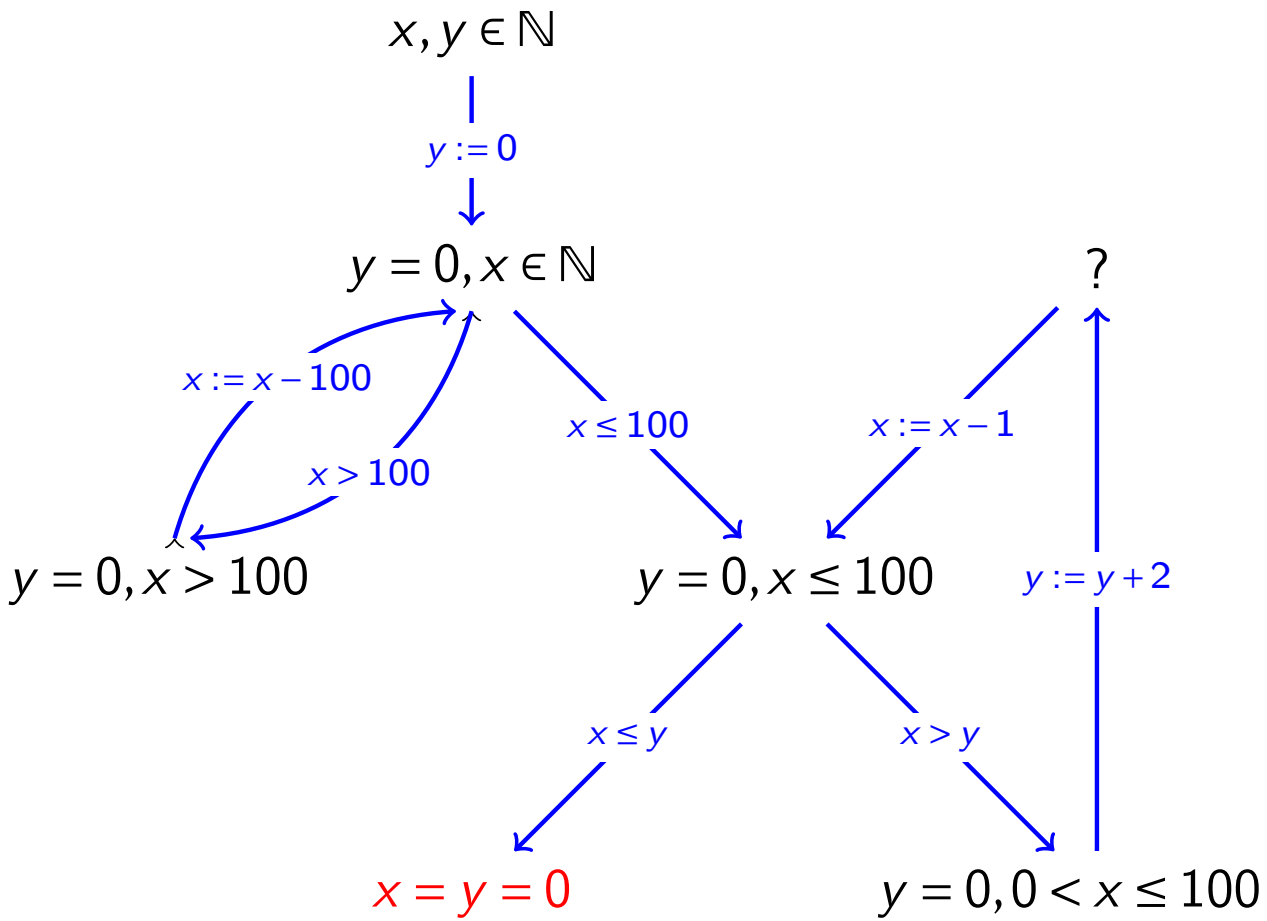
```
 y := 0 ;
while (x > 100) {
  x := x − 100 ;
}
while (x > y) {
  y := y + 2x ;
  x := x − 1 ;
}
```



# Abstract Interpretation for Dummies

**Idea 1** : Represent the program as an automata with memory (state)
**New goal** : Propagate possible states to the end

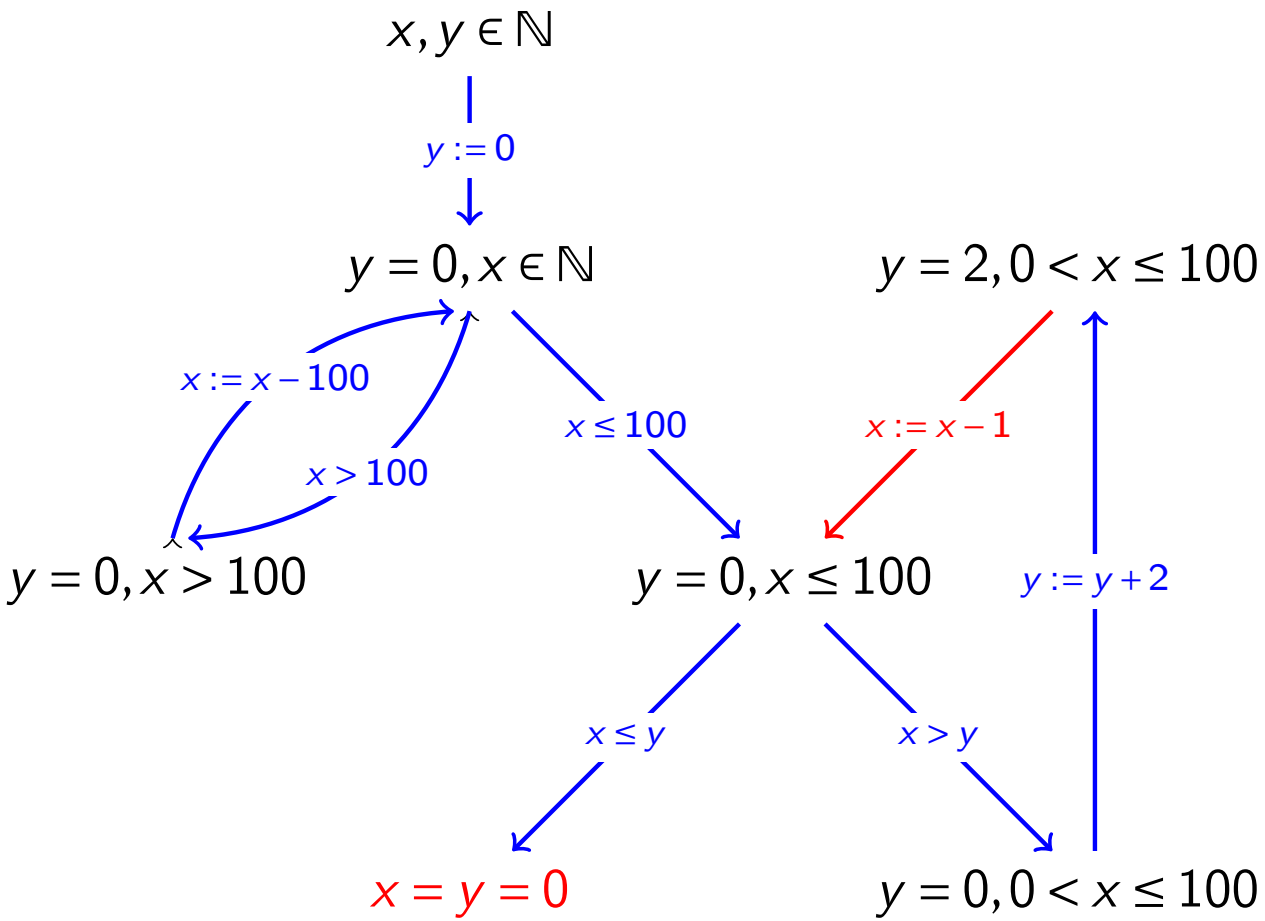
```
y := 0 ;
while (x > 100) {
  x := x − 100 ;
}
while (x > y) {
  y := y + 2x ;
  x := x − 1 ;
}
```



# Abstract Interpretation for Dummies

**Idea 1** : Represent the program as an automata with memory (state)
**New goal** : Propagate possible states to the end
**Issue** : Loops are fixpoints

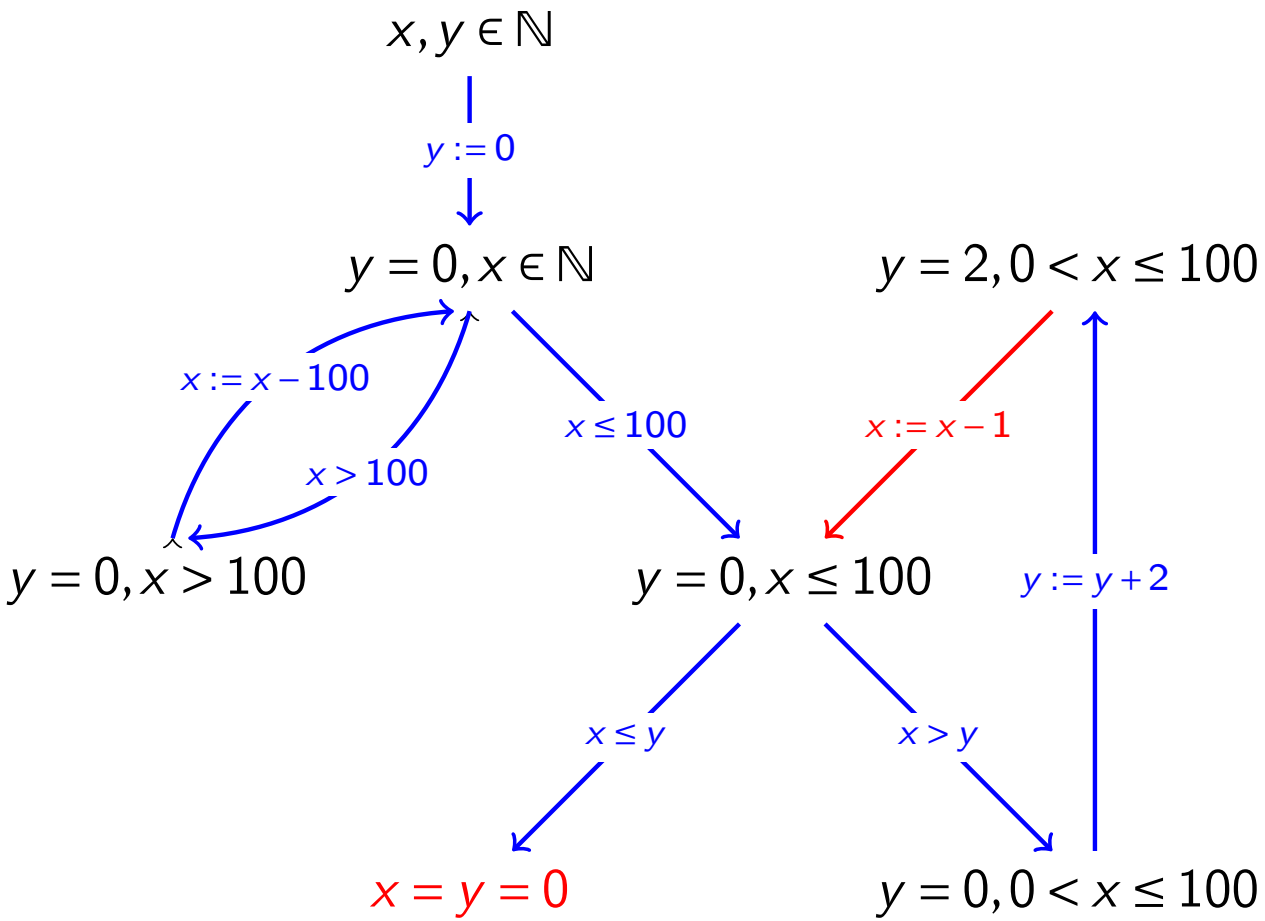
```
y := 0 ;
while (x > 100) {
  x := x − 100 ;
}
while (x > y) {
  y := y + 2x ;
  x := x − 1 ;
}
```



# Abstract Interpretation for Dummies

**Idea 2** : The best set of states is the least fixpoint

```
 y := 0 ;
while (x > 100) {
  x := x − 100 ;
}
while (x > y) {
  y := y + 2x ;
  x := x − 1 ;
}
```

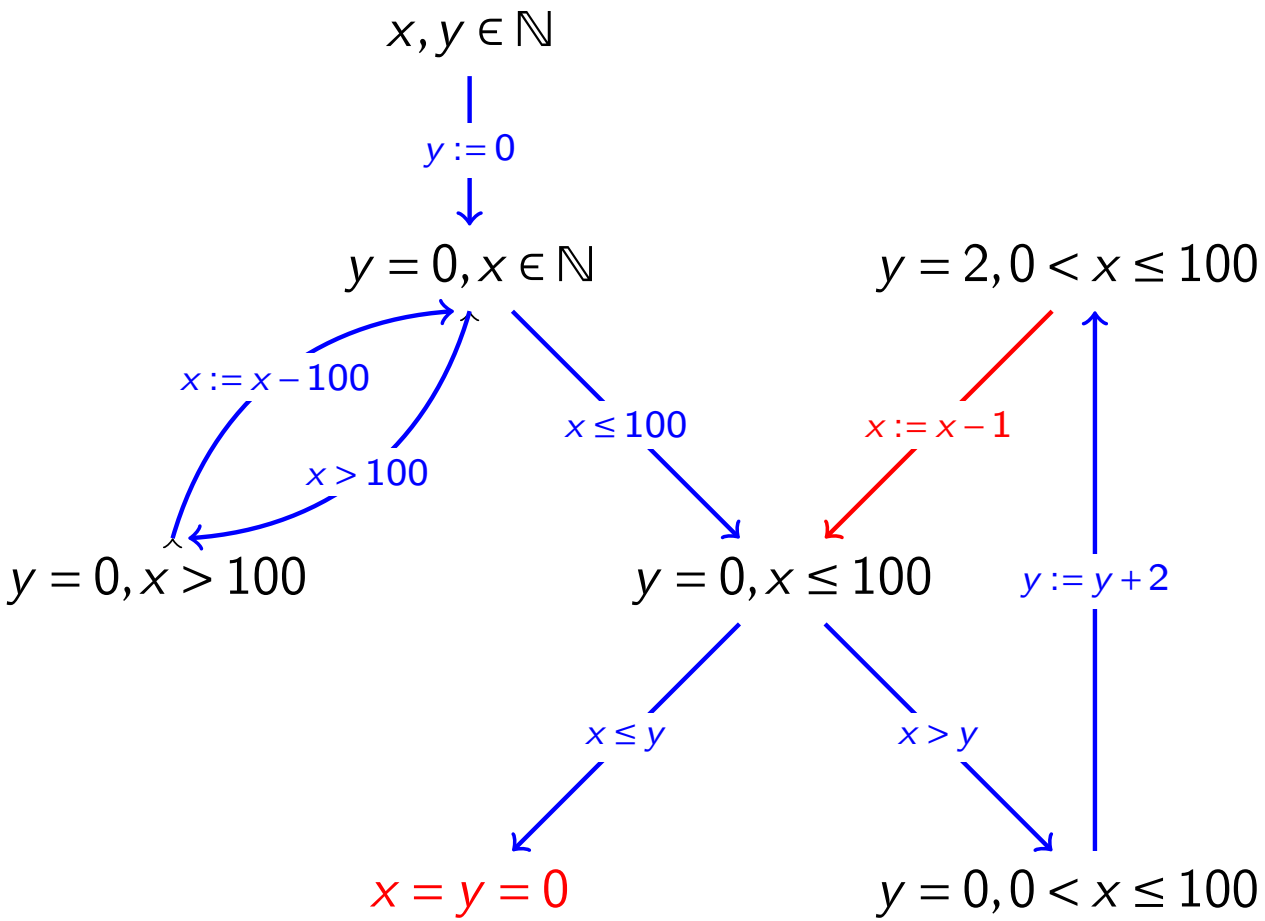


# Abstract Interpretation for Dummies

**Idea 2** : The best set of states is the least fixpoint

**New goal** : Find the least fixpoint

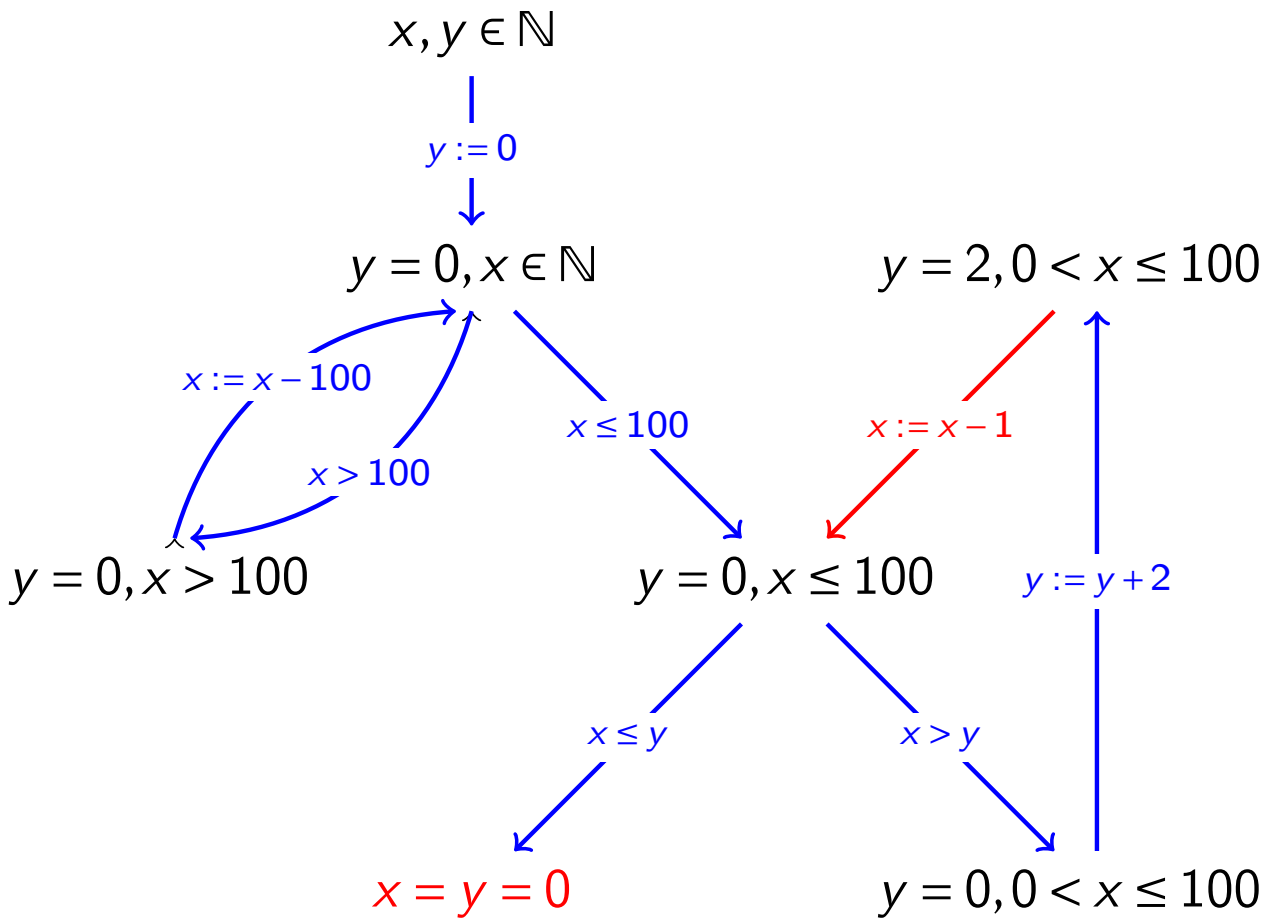
```
 y := 0 ;
while (x > 100) {
  x := x − 100 ;
}
while (x > y) {
  y := y + 2x ;
  x := x − 1 ;
}
```



# Abstract Interpretation for Dummies

**Idea 2** : The best set of states is the least fixpoint
**New goal** : Find the least fixpoint
**Issue** : Too long ...

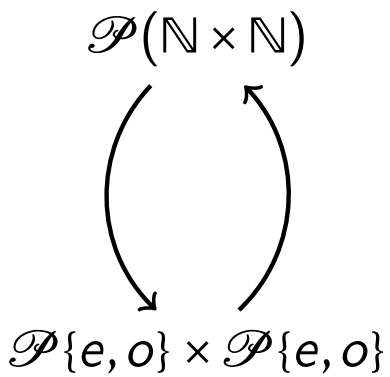
```
y := 0 ;
while (x > 100) {
  x := x − 100 ;
}
while (x > y) {
  y := y + 2x ;
  x := x − 1 ;
}
```

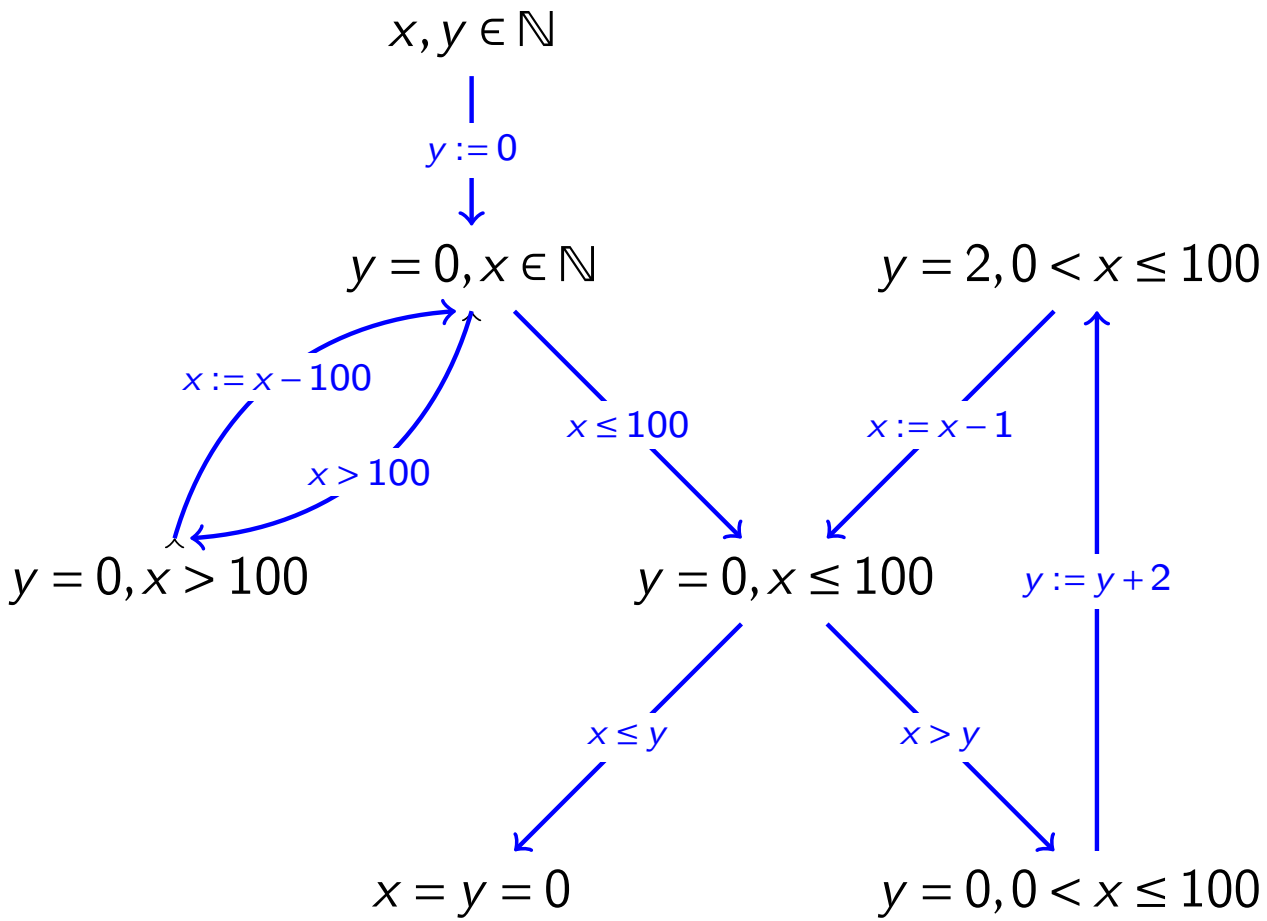


# Abstract Interpretation for Dummies

**Idea 3** : Abstract the set of states while preserving fixpoints

Abstraction:

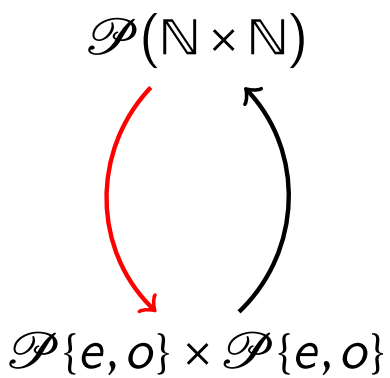


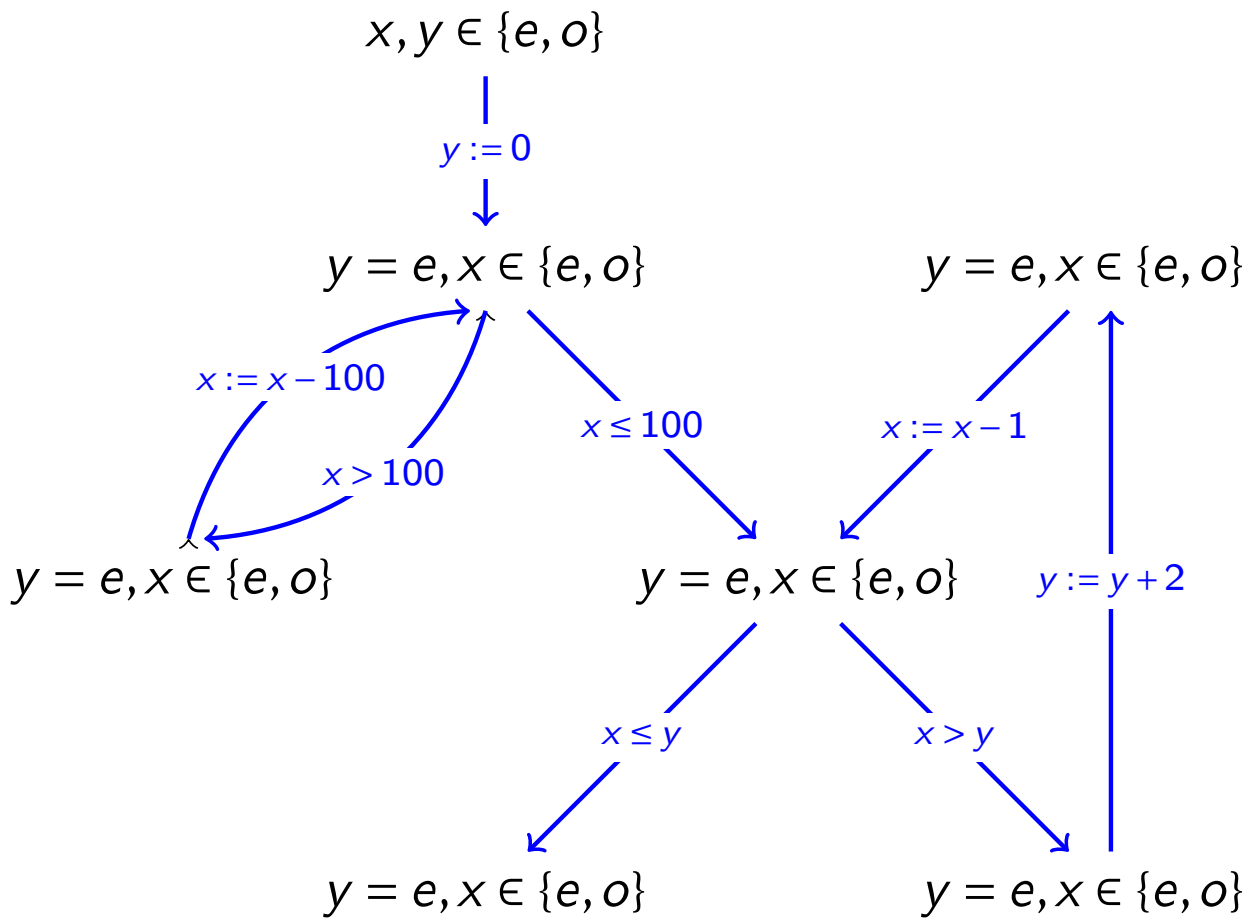



# Abstract Interpretation for Dummies

**Idea 3** : Abstract the set of states while preserving fixpoints
**New goal** : Find the least abstract fixpoint

Abstraction:

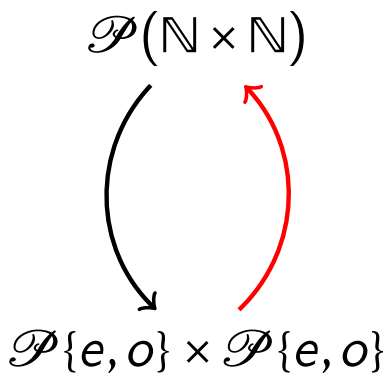


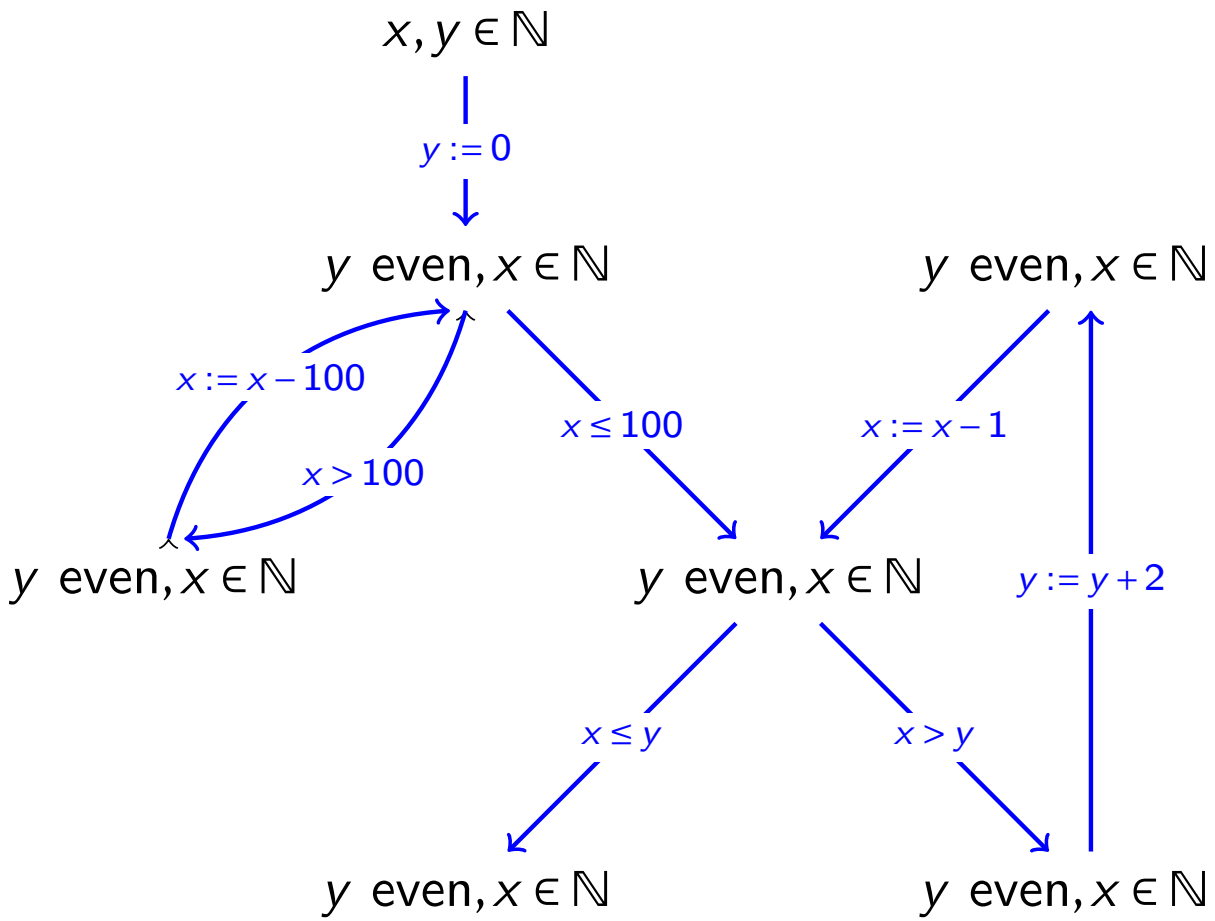



# Abstract Interpretation for Dummies

**Idea 3** : Abstract the set of states while preserving fixpoints
**New goal** : Find the least abstract fixpoint
**Result** : You get a concrete fixpoint

Abstraction:



# From Theory to Practice

## Theoretical Foundations

### Based on domain theory

We work with dcpos in order to get a notion of fixpoint

### Galois connection/adjunction

The abstraction is performed by a Galois connection (an adjunction)

## Practical Foundations

### Compile into a state machine

Refine locations and variables into more uniform ones

inlining, static single assignment (SSA)...

### Choose an abstract domain

Choose a more precise/simple/specific abstract domain

segments, polygons, arithmetic...

### Methods to find fixpoints

Use fast/powerful methods to search for fixpoints

widening, narrowing, finite domains...

# AI for Functional Programming: Works in Theory but Not in Practice

**Domain and categorical theories for FP have been understood and refined for 40 years !**

Dozens of full abstraction results or powerful types systems...

In the 90's, figures like Cousot and Abramsky thought that it was time to adapt AI to FP, but they ultimately failed.

**Methods to find fixpoints do not work on functional spaces**

| **Finite spaces** | **Widening and narrowing** |
|---|---|
| tower blow up on the (monomorphic) type order | are not inherited by functional spaces. |

**Other issues**

Complex flow of execution — Abstracting ADT

# Type Order of Functional Programs is Practically Bounded by 3

## False in the absolute

any CPS, closure, etc, will easily lift the order of your programmes to 5 or 6...

## True up-to polymorphism

Polymorphic function of order 2 are instantiated as monomorphic functions of order 5.

**What is an abstract domain for a polymorphic function ?**

## [Abramsky1992]

Model for polymorphism and polymorphic treatment of totality

## Types Systems

A type system with polymorphism is an abstraction of a kind.

# Graded Types and Polymorphic Abstract Interpretation

## Polymorphic abstractions can only abstract resource usage and preservation

A term $t : \forall A, (A \to A) \to A \to A$ is somehow characterised by its number of use of its first argument.

## Graded types systems seem to capture them

Most* polymorphic abstractions in the literature can be embedded into a kind of graded type system

* all those I know

## Example

$(A^1 \to A)^4 \to A^1 \to A$ has a unique inhabitant in nf : $\lambda fx.f(f(f(fx)))$

# Contents

# Multiple Type Systems with Similar Ideas

## Type systems for verification fall under[1] one of those:

encode a whole external proof (ex: coq)

use intersection types (ex: model checking of HO schemes)

attache quantitative information to ground-types[2] (ex: refinement types)

attache to any arrow a information on resources (ex: Hoar types)

1: at least 2: and ADTs

## Kind of information carried by these annotations

- postcondition / raised effect / resource modification
- precondition / required resource
- usage of the arguments

# Graded Structures

## Syntactical intuition

let $F$ a functor with nice properties (exponential, monad)
Suppose that $FA = \exists i{:}I, F_i A$, what are the conditions on $I$ ?
Natural transformation $\mapsto$ operators  diagrams $\mapsto$ equations

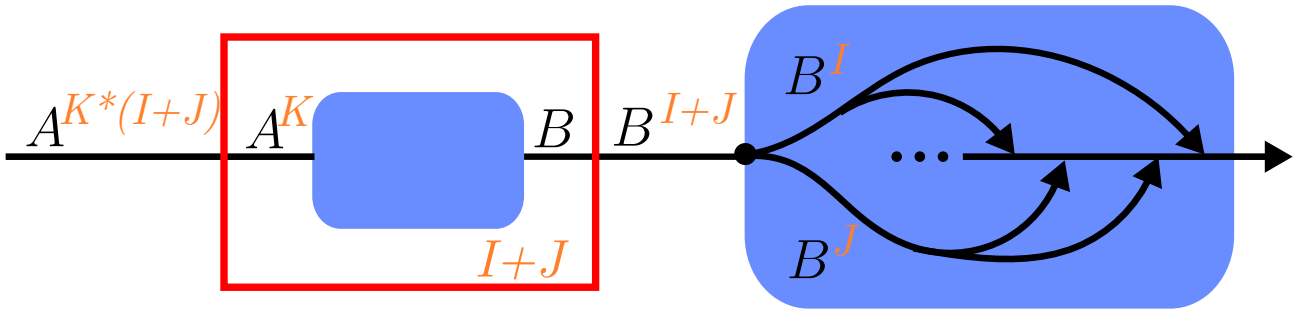
## For a monad $M = \exists i{:}I, i \bullet A$

$(I, 1, *)$ is a monoid with $\eta_{i,A} : A \to i \bullet A$  $\mu_{i,j,A} : i \bullet (j \bullet A) \to (i * j) \bullet A$

## Categorically formalised [Fuji,Katsumata,Mellies]

Preservation of lax algebraic structures by adjoin functors

# Linear Logic with Graded Exponential




## Logic of a graded exponential (me$_{\mathscr{S}}$ll)

$$\frac{\Gamma \vdash B}{\Gamma, A^0 \vdash B}\ \texttt{Weak} \qquad \frac{\Gamma, A \vdash B}{\Gamma, A^1 \vdash B}\ \texttt{Der} \qquad \frac{\Gamma, A^i, A^j \vdash B}{\Gamma, A^{i+j} \vdash B}\ \texttt{Contr}$$

$$\frac{\Gamma, A^i \vdash B \qquad i \leq j}{\Gamma, A^j \vdash B}\ \texttt{Der} \qquad \frac{A_1^{i_1}, \cdots, A_n^{j_n} \vdash B}{A_1^{i_1 * j}, \cdots A_n^{i_n * j} \vdash B^j}\ \texttt{Prom}$$

## Semiring Galois connection

If $\mathscr{S} \perp \mathscr{R}$ then any proof (i.e. any term) in me$_{\mathscr{R}}$ll is a proof in me$_{\mathscr{S}}$ll

# Issues To Be Dealt With in CoGITARe

## No ADTs

## Too many systems

- graded monads and exponentials
- any other graded structures (arrows ?)
- a lot of morally graded system do not enter those frameworks (BLL, DFuzz, indexed LL...)

## A lack of expressive power

We can compose of non-matching gradation :

$$\frac{A_1^{i_1}, \cdots, A_n^{j_n} \vdash B^{j} \multimap C \qquad A_1^{k_1}, \cdots, A_n^{k_n} \vdash B}{A_1^{i_1 + j * k_1}, \cdots, A_n^{i_n + j * k_n} \vdash C}\ \texttt{App}$$

but the $A_i$ have to be the sames (no identification of deep gradations)

$\rightsquigarrow$ lack of expressivity : $\lambda f.f(\lambda xy.x); f(\lambda xy.y)$ untypable in me$_{\mathbb{N}}$ll.

# Resource Polymorphism for More Expressivity

$\lambda fx.fx : \forall i.(A^i \to B) \to (A^i \to B)$

## Resource polymorphism is not type polymorphism

- trivial monomorphic resource $\not\Longrightarrow$ trivial polymorphic one
  $\forall \alpha, \beta.\alpha A \to \beta B \to \beta\alpha\beta C$
- substitution and binders may be "algebraised"

**Two Different Kinds of Dependency**

## Algebraisation of the syntactical polymorphism

Polymorphic labels are resources and substitution is an algebraic operation.

## Oidification of the logic me$\mathscr{S}$ll

Resources have source and target and the bind the resources inside the term they are applied to

# Contents

# Schedules Types for Weak Head-Reduction

## We use Weak-head reduction

Trad-off between algebraic and coalgebraic behaviour observations

## Definition: Schedules

An ordered monoid $(\mathscr{S}, \epsilon, \cdot, \leq)$ containing labels $\alpha, \beta, \gamma \cdots \in \mathscr{S}$ with

- a substitution $S[\alpha \mapsto T] \in \mathscr{S}$ th:
$$\alpha[\alpha \mapsto T] = T \qquad \beta[\alpha \mapsto T] = \beta$$
- behaving well...

## Definition : Schedules Types

$$\text{(Types)} \quad \mathscr{T} \quad A, B, C \quad := \quad \forall \alpha.B \quad | \quad S \cdot A \to T \cdot B \quad | \quad \cdots$$

## Reynolds-style local-variable block and rational schedules

$$\text{new} : \delta(\alpha\, int \to \beta(\forall \gamma.\gamma\, int \to \gamma\, com) \to \beta(\alpha + \beta)^* \cdot com) \to \delta \cdot com$$

# Extending Graded Monads and Exponentials

## Graded monads using polymorphic-free schedules

If $\alpha = \epsilon$ and $S[\alpha \mapsto T] = S$ always, then we are recovering the graded monads for weak-head reduction.

Scheduled types are polymorphic graded monads in the first place...

## Graded exponentials using vectors schedules

Let $\mathscr{R}$ a semiring.
If $\mathscr{S} = \bigoplus_{\alpha_i} \mathscr{R}$, is set of $\mathscr{R}$-valued monomials over the labels, so that

$S * T := S + T$ is the sum of monomials,

$S[\alpha \mapsto T]$ is the substitution of $\alpha$ by $T$ in the monomial $S$.

Then we are recovering a graded exponential for weak-head reduction.

## 4th church numeral for $\bigoplus_{\alpha_i} Nat$

$$(A^1 \to A)^4 \to A^1 \to A \quad \equiv \quad \alpha(\forall \beta.\beta A \to \beta A) \to \gamma A \to (4\alpha+\gamma)A$$

Few differences due to graded exponential not being well defined in WHR

# Inference Tools

In additions to usual AI tools (finite domains, widening...), we can restrict the scopes of the foralls:

## Local-polymorphism

Only arrows of the following form for $\alpha \notin dom(A)$ such that...

$$\forall \alpha.(\alpha \cdot A \to T \cdot B)$$

$\rightsquigarrow$ deterministic application rule

$$\frac{\Gamma \vdash_S \forall \alpha.\alpha A \to T B \qquad \Gamma \vdash_U A}{\Gamma \vdash_{S \cdot T[\alpha \mapsto U]} B[\alpha \mapsto U]}$$

## Let-polymorphism

Let-sequents of the following form with monomorphic $B_j$ and $T$ and...

$$(\alpha_i \cdot A_i) \vdash_S \beta_1 B_1 \to \cdots \beta_k B_k \to T\,\mathbb{1}$$

$\rightsquigarrow$ ML let-polymorphism inference ?

# Higher-Order Resource Polymorphism ?

## Future work: need for 2nd order

We still do not have that

$$\lambda fx.fx : \forall i.(A^i \to B) \to (A^i \to B)$$

Indeed, the exponent is already a 1st order parametric information, we need something like:

$$\lambda fx.fx : \forall h.(\forall \alpha.A \to h(\alpha).B) \to (\forall \alpha.A \to h(\alpha).B)$$

Would be also useful to understand strong head reduction or unification algorithms.

Question 1: is such a 2nd order variable a function ?
Question 2: Where do we stop ?

# Absence of Model

For the moment we failed at providing
either categorical or denotational models...

Some leads using game semantics, but only for strong head reduction

# Contents

# Oidification of the semiring

**Oidification:** Generalising an algebraic structure by associating to elements a source and a target

$\rightsquigarrow$ algebras becomes categories

$\rightsquigarrow$ the original structures are the one-point categories

Monoids $\mapsto$ Categories  Groups $\mapsto$ Groupoids

## Idea : Grading the exponentials by morphisms

We get resource environments and exponentials are binders.

## Similar examples in the literature

LL(I):

$$\frac{\sigma \mid \Gamma \vdash B \qquad f : \sigma \to \rho}{\rho \mid \Gamma^f \vdash B^f}\ \textbf{Prom}$$

BLL:

$$\frac{\rho, x \mid \Gamma \vdash B \qquad f : \mathbb{N}^\rho \to \mathbb{N}}{\rho \mid \Gamma^{x<f} \vdash B^{x<f}}\ \textbf{Prom}$$

LL(I) results from an attempt at understanding logical relations [Bucc,Ehrh.00]
BLL is a system characterising polynomial algorithms [Gir,Sce,Sco.92]

# Substitution

## Dereliction

In such a system, the dereliction is performing an action :

$$\frac{\Gamma, f_*(A) \vdash_\sigma B}{\Gamma, A^f \vdash_\sigma B}\ \text{Der}$$

Such action is only available on specific $f$ (playing the previous role of $1$) which are basically projections.

## From an operational point of view, they are substitutions

ex. in BLL:

$$\frac{\Gamma, A[x := 0] \vdash_\sigma B}{\rho \mid \Gamma, A^{x<1} \vdash_\sigma B}\ \text{Der}$$

## Categorically we need "substitutions" $s : \rho \to \sigma$

they correspond to specific resources and that commutes with them :

$$A[s]^f \simeq A^{s.f} \quad \text{and} \quad A^f[s] \simeq A^{f.s}$$

# An oidified semiring inside linear categories

$\mathscr{L}$: A linear category $\quad \mathtt{m}_{\mathbb{1}}, \mathtt{m}_{\rho,\rho}$: monoidality of !

## The environments $\mathscr{E}$; monoids

The $\otimes$-monoids $(\rho, \mu_\rho, \eta_\rho)$ over $\mathscr{L}$.

## The resource category $\mathscr{C}$; $f : !\rho \to \tau$

The Kleisli category of monoid morphisms for the monad $(\rho, \mu, \eta) \mapsto (!\rho, (\mathtt{m}_{\rho,\rho}; \mu), (\mathtt{m}_\perp; \eta))$.
Spelling out, the morphisms the $f : !\rho \to \tau$ commuting:

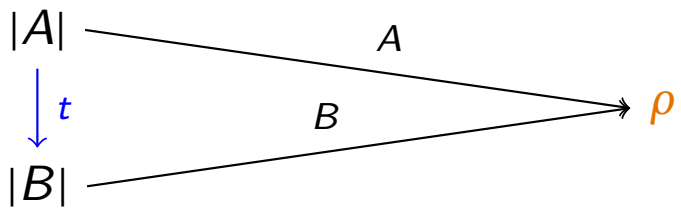
$$\begin{array}{ccc} !\mathbb{1} & \xrightarrow{!\eta_\rho} & !\rho \\ {\scriptstyle \mathtt{m}_\perp}\uparrow & & \downarrow{\scriptstyle f} \\ \mathbb{1} & \xrightarrow{\eta_\tau} & \tau \end{array} \qquad \begin{array}{ccccc} !(\rho \otimes \rho) & & \xrightarrow{!\mu_\rho} & & !\rho \\ {\scriptstyle \mathtt{m}_{\rho,\rho}}\uparrow & & & & \downarrow{\scriptstyle f} \\ !\rho \otimes !\rho & \xrightarrow{f \otimes f} & \tau \otimes \tau & \xrightarrow{\mu_\tau} & \tau \end{array}$$

## The rewriting/substitution category $\mathscr{R}$; $\iota : \rho \to \tau$

The category of $\otimes$-monoid morphisms.

# Slicing linear categories into an hyperdoctrine

- Formula $A$ of domain $\rho$ are coslices, i.e., morphisms $A : |A| \to \rho$
- Programs (or derivations) $x : A \vdash_\rho t : B$ of domain $\rho$ are morphisms $t : |A| \to |B|$ such that:




- Exponentials and substitutions are the non-linear and linear coextensions of contexts :

$$A^f := \ !|A| \xrightarrow{!A} !\rho \xrightarrow{f} \tau \qquad\qquad A[s] := \ |A| \xrightarrow{A} \rho \xrightarrow{s} \tau$$

- Tensor product is defined by:

$$A \widetilde{\otimes} B := \ |A| \otimes |B| \xrightarrow{A \otimes B} \rho \otimes \rho \xrightarrow{\mu_\rho} \rho$$

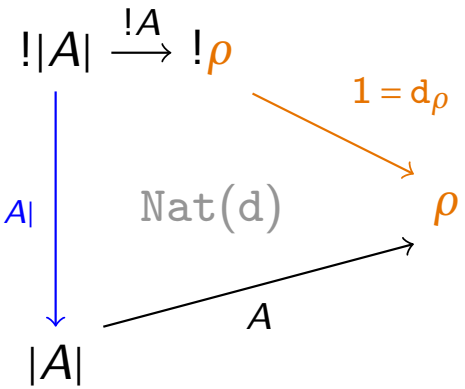
- Cartesian coproduct and product are defined by:

$$A \widetilde{\oplus} B := \ |A| \oplus |B| \xrightarrow{\{A,B\}} \rho \qquad\qquad A \,\&\, B := |A| \,\&\, |B| \xrightarrow{A \,\&\, B} \rho \,\&\, \tau =: \rho + \tau$$

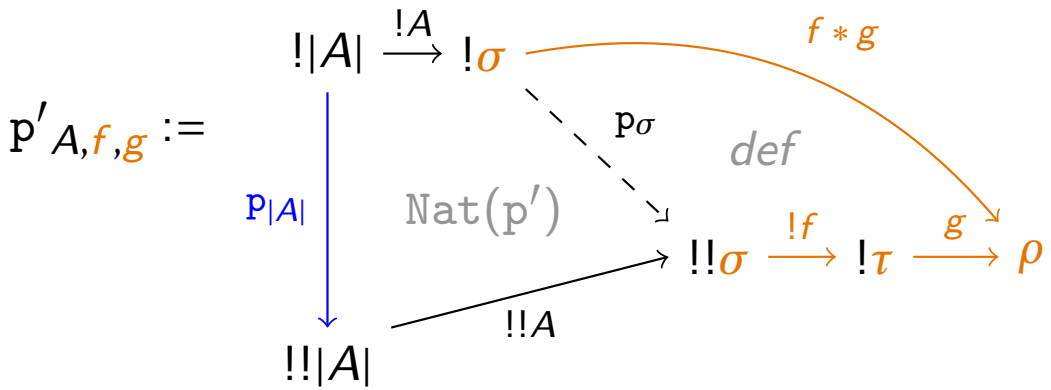
However, the monoidal closure has to be required...

# Exponential formuli

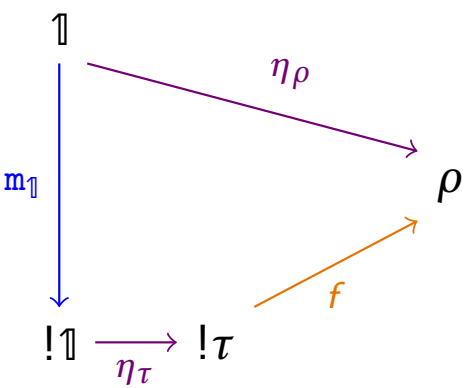
$\mathrm{d}_A :=$



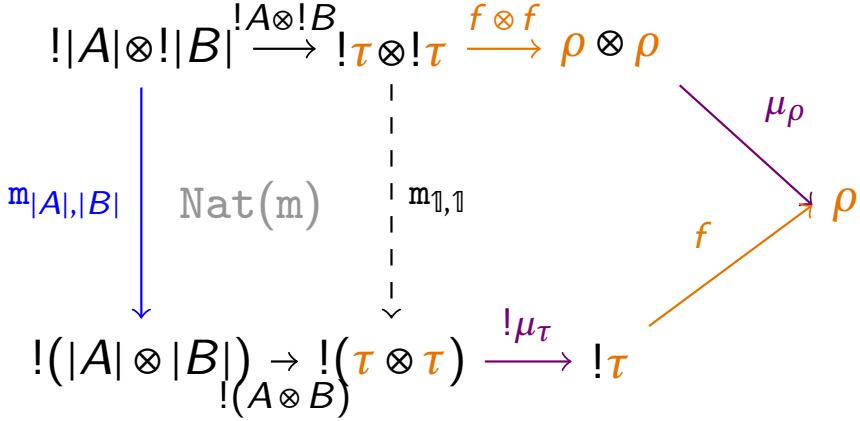

$\mathrm{p}'_{A,f,g} :=$




$\mathrm{m}_f :=$




$\mathrm{m}_{A,B,f} :=$



# Conclusion

I hope to have convinces you that :

- abstract interpretation for functional programming is an open and interesting field of study, even for theorists,
- graded types are the future ! (or just that they may be used for real thing in years to come...)
- resources' dependency/polymorphism is really different from terms', but still quite fun.

## Project CoGITARe

**begin in March**
**for 42 months**

**1 funded PhD (+internship)**
**funding for 1y posdoc**

Contact: breuvart@lipn.fr