Méthodes formelles

Sébastien Bardin

CEA-LIST, Laboratoire de Sûreté Logicielle

sebastien.bardin@cea.fr
http://sebastien.bardin.free.fr

Objectifs du cours

Compétences à aquérir

- connaître les principes généraux des méthodes formelles
- avoir un aperçu des tendances et de l'état de l'art académique

Pour quelles industries ?

- systèmes critiques
- systèmes "de qualité" (sécurité, etc.)
- beaucoup plus "amont" que le test

Plan Général

- Introduction aux méthodes formelles
- Sémantique
- Spécification formelle
- Vérification déductive
- Analyse dataflow, interprétation abstraite
- Logiques temporelles et model checking

Motivations (rappel)

Coût des bugs

- Coûts économique : 64 milliards $/an rien qu'aux US (2002)
- Coûts économique : 150 milliards EUR/an en Europe (2012)
- Coûts humains, environnementaux, etc.

Nécessité d'assurer la qualité des logiciels

Domains critiques

- atteindre le (très haut) niveau de qualité imposée par les lois/normes/assurances/... (ex : DO-178B pour aviation)

Autres domaines

- atteindre le rapport qualité/prix jugé optimal (c.f. attentes du client)

Motivations (2)

Validation et Vérification (V & V)

- Vérification : est-ce que le logiciel fonctionne correctement ?
  - *"are we building the product right ?"*
- Validation : est-ce que le logiciel fait ce que le client veut ?
  - *"are we building the right product ?"*

Quelles méthodes ?

- revues
- simulation/ animation
- tests méthode de loin la plus utilisée
- méthodes formelles encore très confidentielles, mais progresse !

Coût de la V & V

- 10 milliards $/an en tests rien qu'aux US
- plus de 50% du développement d'un logiciel critique (parfois $> 90\%$)
- en moyenne 30% du développement d'un logiciel standard

Un peu d'histoire

"Crise du logiciel" en 1969

The major cause of the software crisis is that the machines have become several orders of magnitude more powerful ! To put it quite bluntly : as long as there were no machines, programming was no problem at all ; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.

- Edsger Dijkstra, The Humble Programmer (EWD340)

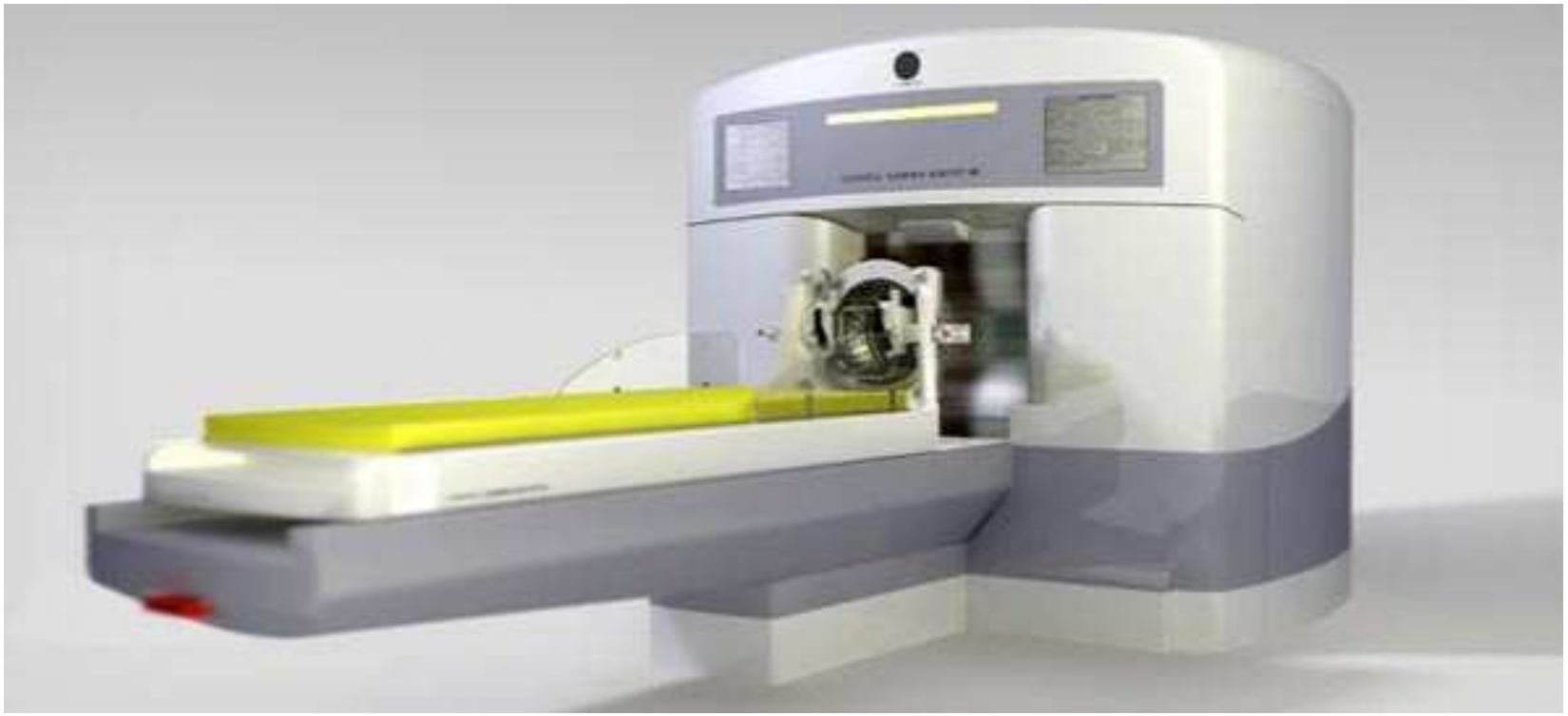
Ariane 5, 1996

Connaissez-vous l'histoire ?

Ariane 5, 1996

- en 1996, destruction de la fusée Ariane 5 durant son vol inaugural
- cause :
  - réutilisation d'un composant d'Ariane 4... sans respecter sa spécification (accélération d'Ariane 5 trop forte)
  - débordement lors de la conversion d'un nombre à virgule flottante de 64 bits vers un entier 16 bits
- conséquence :
  - mise hors service du composant (système de guidage inertiel principal)
  - mauvaise interprétation du pilote automatique
  - déviation rapide de trajectoire
  - arrachage des boosters
  - auto-destruction préventive de la fusée
- coût : $\approx$ 500 millions de dollars
- bogue le plus cher de l'histoire

Connaissez-vous l'histoire ?



Therac-25, 1985

- de 1985 à 1987, l'appareil de radiothérapie Therac-25 est responsable du décès de plusieurs personnes
- causes :
  - la cause première était la mauvaise architecture du système et de mauvaises pratiques de développements plutôt que les erreurs de programmation trouvées.
  - en particulier, il n'était pas possible de tester automatiquement et proprement le système
- conséquence :
  - l'utilisateur pouvait programmer un surdosage du patient (jusqu'à 100x) sans s'en rendre compte
  - aucun dispositif de contrôle
  - les patients se retrouvaient alors sévèrement irradiés
- bogue le plus grave de l'histoire

Et plein d'autres...

http ://en.wikipedia.org/wiki/List_of_software_bugs

Catégories

- exploration spatiale
- médical
- *tracking years*
- réseau électrique
- finance
- télécommunication
- militaire
- média
- jeux
- cryptographie
- automobile

Système embarqué critique

Système embarqué

Système électronique et informatique autonome, souvent temps-réel, spécialisé dans une tâche bien précise.

Système critique

Système dont une panne peut avoir des conséquences dramatiques.

Exemples de domaine critique

- transport (aérospatial, aéronautique, ferroviaire, automobile)
- énergie, en particulier nucléaire
- gestion des réseaux (telecom, électricité, eau)
- santé
- finance
- applications militaires

Évolution de 1960 à 1995

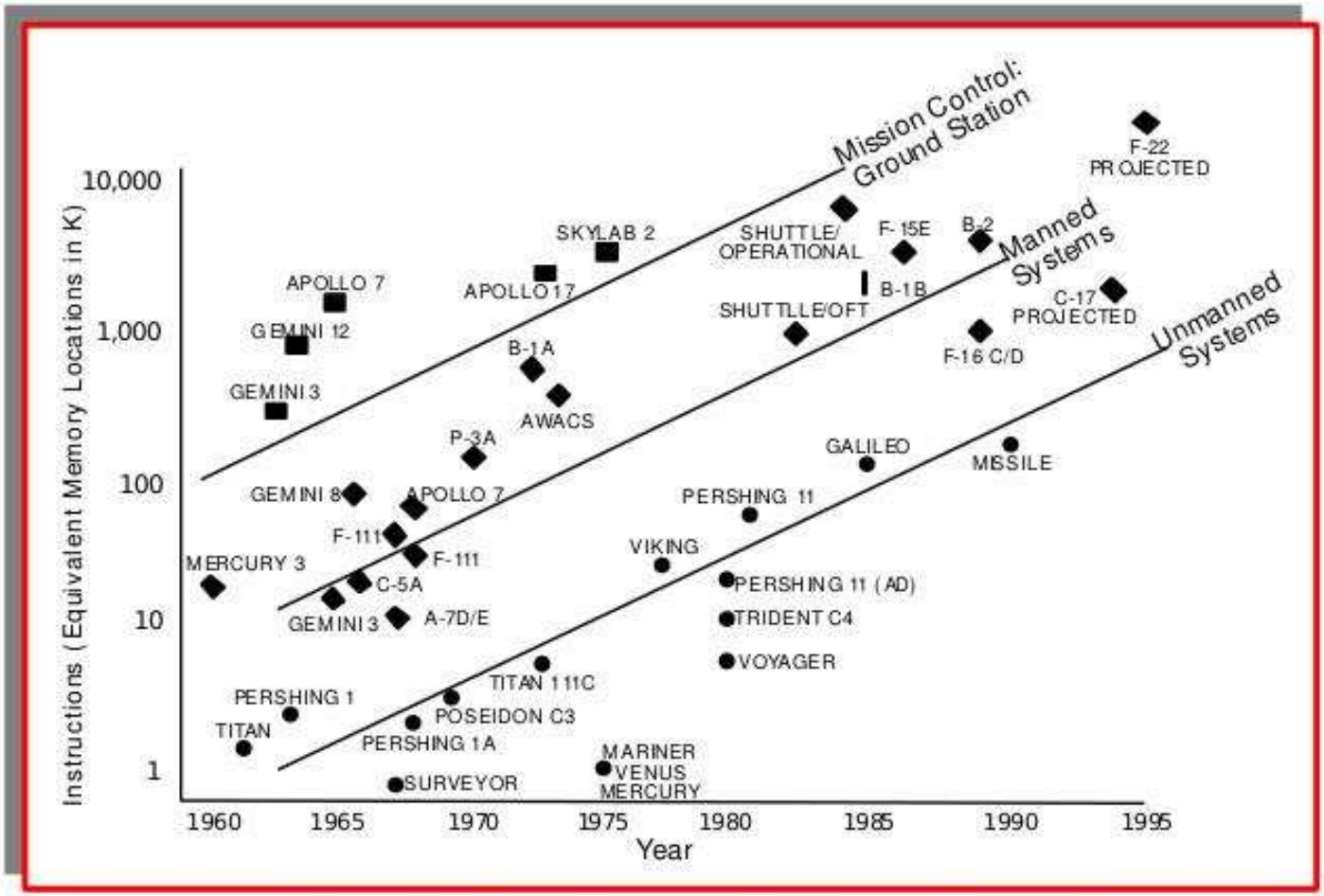
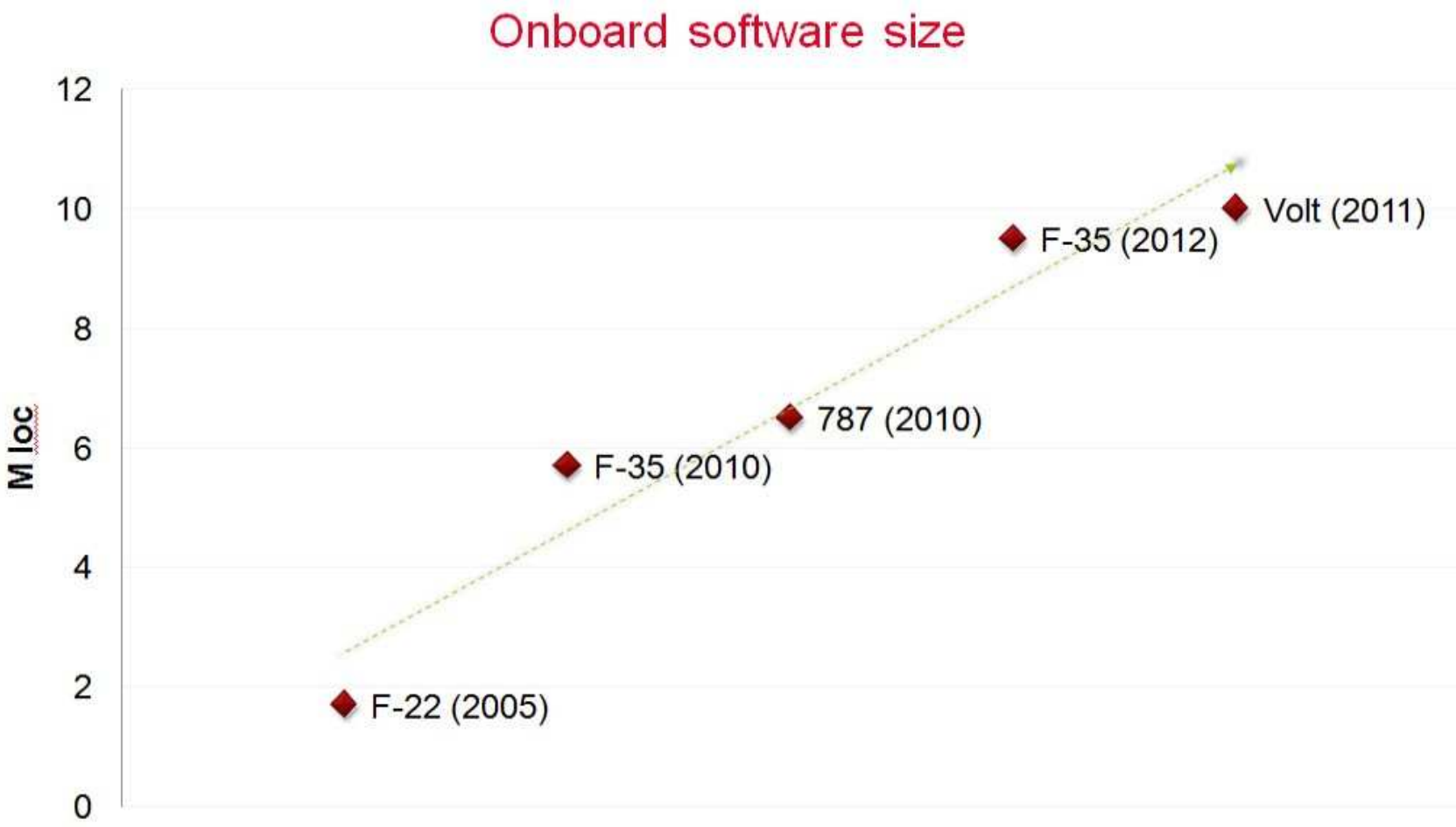


Figure 1-2. Software Systems Size Growth 1960 to 1995

Le nombre de bogues est notamment fonction de la taille du système...

Évolution récente



Le nombre de bogues est notamment fonction de la taille du système...

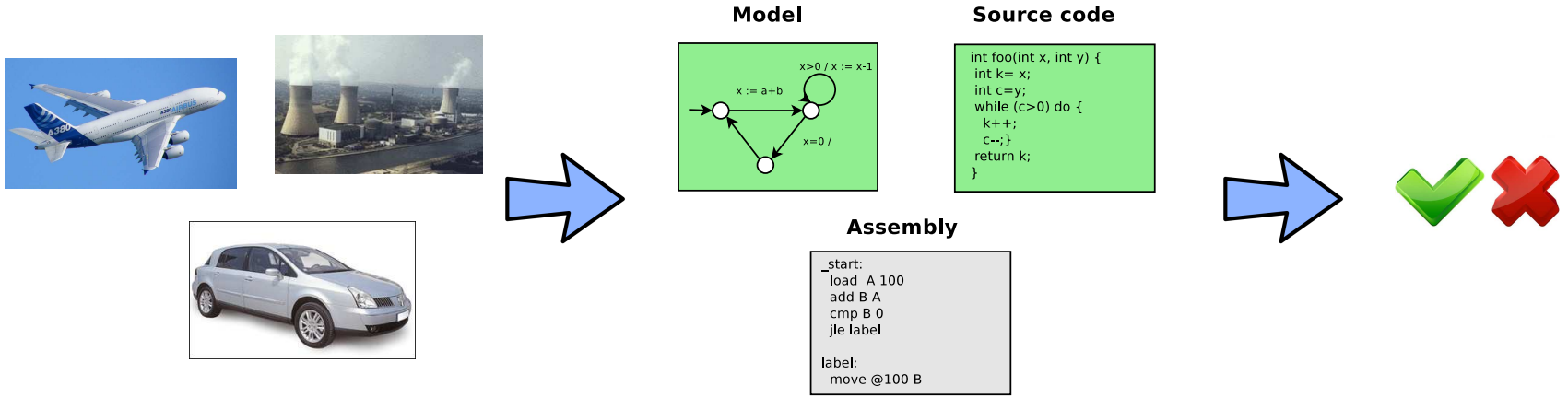
Comment assurer le niveau de qualité voulu ?

Testing can only reveal the presence of errors but never their absence.

- E. W. Dijkstra (Notes on Structured Programming, 1972)

Un rêve de logicien

- Between Software Engineering and Theoretical Computer Science
- Goal = proves correctness in a mathematical way



Key concepts : $M \models \varphi$

- M : semantic of the program
- φ : property to be checked
- $\models$: algorithmic check

Kind of properties

- absence of runtime error
- pre/post-conditions
- temporal properties

Eléments clés

Système M

- besoin d'une sémantique : décrire précisément ce qui se passe
- notion d'état mémoire s ($\approx$ associe à chaque variable une valeur)
- définition de la relation de transition $s \xrightarrow{t} s'$: effet de l'instruction t sur l'état s
  - . typiquement, modifie des valeurs associées à des variables dans s
  - . $\{a \mapsto 5, b \mapsto 10\} \xrightarrow{b:=a+b} \{a \mapsto 5, b \mapsto 15\}$
  - . sémantique = interpréteur
- notion d'états accessibles, et de traces

Propriété φ

- implicites (runtime error, mauvais typage, non terminaison)
- ou spécifiable (logique, automates)
- rmq : langage naturel ou semi-formel (UML) trop ambigu
- définition de $\models$: tous les états / traces / ... de M satisfont φ
  - . états$(M) \subseteq$ états(φ), traces$(M) \subseteq$ traces(φ), etc.

Algorithme : attention, indécidable bien souvent. Et cher sinon

Exemple simple

. Imaginons un programme avec seulement des entrées booléennes, et la propriété P : "vérifier que l'on a toujours (si a alors b)"

    . le nombre d'états est fini : on peut tous les énumérer

    . sur chaque état s, on peut facilement vérifier P

. Algorithme de calcul des états ! !

Exemple simple

. Imaginons un programme avec seulement des entrées booléennes, et la propriété P : "vérifier que l'on a toujours (si a alors b)"

. le nombre d'états est fini : on peut tous les énumérer

. sur chaque état s, on peut facilement vérifier P

. Algorithme de calcul des états ! !

Mais en pratique :

- espace des états trop grand ou infini
- propriétés sur un nombre infini de traces potentiellement infinies
- propriétés atomiques complexes (quantificateurs)

Du rêve à la réalité

Industrial reality in some area, especially safety-critical domains (but not only)

- hardware [intel, cadence, etc.], aeronautics [airbus], railroad [metro 14 and many others], smartcards, drivers [SDV, Windows], certified compiler [CompCert], safe DSL for optimizing compilers [alive, in llvm], certified OS [Sel4], memory safety of Linux kernel [with Frama-C], memory safety of parts of Windows [prefast], polar SSL [with Frama-C], many other [Facebook, Amazon, Google, etc.]

Du rêve à la réalité

Industrial reality in some area, especially safety-critical domains (but not only)

- hardware [intel, cadence, etc.], aeronautics [airbus], railroad [metro 14 and many others], smartcards, drivers [SDV, Windows], certified compiler [CompCert], safe DSL for optimizing compilers [alive, in llvm], certified OS [Sel4], memory safety of Linux kernel [with Frama-C], memory safety of parts of Windows [prefast], polar SSL [with Frama-C], many other [Facebook, Amazon, Google, etc.]

Ex : Airbus

Verification of

- runtime errors [Astrée]
- functional correctness [Frama-C]
- numerical precision [Fluctuat]
- source-binary conformance [CompCert]
- ressource usage [Absint]

Du rêve à la réalité

Industrial reality in some area, especially safety-critical domains (but not only)

- hardware [intel, cadence, etc.], aeronautics [airbus], railroad [metro 14 and many others], smartcards, drivers [SDV, Windows], certified compiler [CompCert], safe DSL for optimizing compilers [alive, in llvm], certified OS [Sel4], memory safety of Linux kernel [with Frama-C], memory safety of parts of Windows [prefast], polar SSL [with Frama-C], many other [Facebook, Amazon, Google, etc.]

Ex : Microsoft

Verification of drivers [SDV]

- conformance to MS driver policy
- home developers
- and third-party developers

Du rêve à la réalité

Industrial reality in some area, especially safety-critical domains (but not only)

- hardware [intel, cadence, etc.], aeronautics [airbus], railroad [metro 14 and many others], smartcards, drivers [SDV, Windows], certified compiler [CompCert], safe DSL for optimizing compilers [alive, in llvm], certified OS [Sel4], memory safety of Linux kernel [with Frama-C], memory safety of parts of Windows [prefast], polar SSL [with Frama-C], many other [Facebook, Amazon, Google, etc.]

Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability.

- Bill Gates (2002)

Taxonomie des approaches (1)

Place dans le processus de développement

- a priori
- a posteriori
- approche par raffinement

Objet de la vérification

- modèle vs source (vs binaire)

Niveau d'automatisation

- complètement automatique [enfin, presque]
- manuel
- intermédiaire [annotations]

Propriétés

- prédicats simples ou complexes (logique propositionnelle, 1er ordre QF, 1er ordre, ordres supérieurs)
- aspect temporel (invariance, sûreté, vivacité)
- propriétés implicites ou langage de description

Taxonomie des approaches (2)

Variantes du problème initial

- inférence : $M \mapsto \psi$
- vérif paramétrée : $(M_k, \varphi) \mapsto \{k | M_k \models \varphi\}$
- synthèse : $\psi \mapsto M$

Taxonomie des approches (3)

	system	properties		algorithm		automation
		temporal	atomic pred.	answer	termination	
MC	finite	liveness	boolean	yes/no (+cex)	ok	ok
AI	infinite	invariance *	QF-FOL	yes/ ?	ok	ok
WP	infinite	invariance * termination	FOL	yes/ ?	ok	X

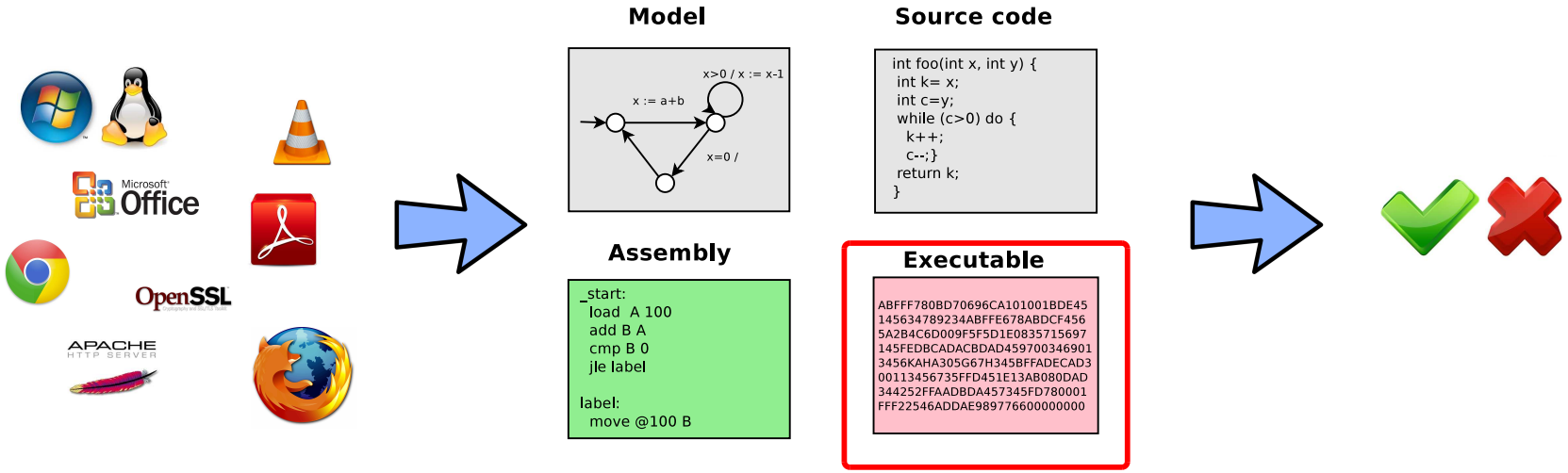
Historique

- 1970's : quelles propriétés, quels modèles ? (sémantique)
- 1980's : premiers algos, classification des problèmes (complexité)
- 1990's : premières études de cas réalistes (protocoles, hardware)
- 2000's : début du software verification

quelques dates clés :
. logique temporelles (Pnueli, 1977),
. lcalcul de WP (Hoare, 197xx),
. linterprétation abstraite (Cousot, 1977),
. lmodel checking (Clarke et al., Sifakis et al., 1981),

Prochain défis

- Apply formal methods to less-critical software
- Very different context : no formal spec, less developer involvement, etc.



Some difficulties

- robustness [w.r.t. software constructs]
- no place for false alarms
- scale
- no model, sometimes no source
- quantitative verification

Guidelines & trends

- find sweetspots [drivers]
- manage abstractions
- reduction to logic

Plan Général

- Introduction aux méthodes formelles
- Sémantique
- Spécification formelle
- Vérification déductive
- Analyse dataflow, interprétation abstraite
- Logiques temporelles et model checking

Petit langage impératif à valeurs entières

Expressions *Expr*

$e ::=$	x	variable, $x \in Var$
$\mid$	z	constante entière, $z \in \mathbb{Z}$
$\mid$	$e + e \mid - e \mid e * e \mid e / e$	expressions arithmétiques
$\mid$	$e = e \mid e <= e \mid ! e \mid (e)$	expressions booléennes

Instructions *Instr*

$i ::=$	x := e	affectation
$\mid$	if e then i else i fi	conditionnelle
$\mid$	while e do i done	boucle
$\mid$	$i ; i$	séquence
$\mid$	skip	sans effet

- la syntaxe décrit seulement l'ensemble des programmes qu'on peut d'écrire
- elle ne précise pas le sens de chaque entité syntaxique (ici les expressions et les instructions)
- c'est le rôle de la sémantique
- en particulier, elle ne précise pas les constructions qui n'ont "pas de sens", comme 1 / 0.
- un programme est une instruction particulière
- pas de booléens : les expressions booléennes sont à valeurs entières
- pas de déclarations de variable
- pas de fonctions/procédures

Sémantique d'un programme

Cas simple : juste des variables entières a, b, c, d, ...

Sémantique opérationnelle

- une configuration du système :
  . c : Var $\mapsto \mathbb{N}$
- une affectation modifie l'état mémoire :
  . ex : z := a+b
  . c $\xrightarrow{z:=a+b}$ c' ssi c' = c[z $\leftarrow$ c[a] + c[b]]
- un branchement teste l'état mémoire

Exo

- écrire la sémantique du langage While

Plan Général

- Introduction aux méthodes formelles
- Sémantique
- Spécification formelle
- Vérification déductive
- Analyse dataflow, interprétation abstraite
- Logiques temporelles et model checking

But

- les spécifications permettent de :
  - mieux comprendre le code
  - clarifier les interfaces (modules/fonctions)
  - faciliter la maintenance
  - tester le logiciel en boite noire
- les spécifications formelles permettent de :
  - raisonner formellement sur le programme
  - vérifier que le code correspondant les satisfait
  - utiliser des outils

Éléments clés

Langages de contrat :

- précondition, postcondition [appelant, appelé]
- frame / assigns
- prédicats avancés : built-in (\valid), $\forall$, $\exists$, etc.
- boucles : invariants et variants
- assert intermédiaires [hints]

Aspects temporels : cf model checking

Bibliographie : langages de spécification

- Eiffel : Analysis, Design and Programming Language 2nd edition. 2006.
  http://www.ecma-international.org/publications/standarc
- Java Modeling Language (JML). 2000.
  http://www.eecs.ucf.edu/~leavens/JML/index.shtml
- Spec♯. 2004.
  http://research.microsoft.com/en-us/projects/specsharp
- ANSI C Specification Language (ACSL). 2008.
  http://frama-c.cea.fr/acsl.html
- Executable ANSI C Specification Language (E-ACSL). 2012.
  http://frama-c.cea.fr/eacsl.html
- Spark-2014. 2013.
  http://www.spark-2014.org

Plan Général

- Introduction aux méthodes formelles
- Sémantique
- Spécification formelle
- Vérification déductive
- Analyse dataflow, interprétation abstraite
- Logiques temporelles et model checking

Principe

Procédure

- raisonner par déduction sur le programme de manière automatique
- prend en compte les spécifications formelles

Obligation de preuve

- extraire les parties à vérifier sous forme de propriétés appelées obligations de preuve (OP)
- la vérification des OP est laissée à la charge de l'utilisateur
- l'utilisation de prouveurs automatiques et/ou semi-automatiques est indispensable

Garantie

sous l'hypothèse que les OP sont toutes vérifiées, les méthodes déductives garantissent la correction du programme vis-à-vis de sa spécification

Triplet de Hoare

- Un triplet de Hoare est un triplet noté $\{P\}i\{Q\}$ où P et Q sont des propriétés (formules logiques) et i est une instruction du programme.
- P est appelé la pré-condition de i
- Q est appelé la post-condition de i
- Informellement, un triplet de Hoare est valide si la propriété suivante est vraie :

  si la pré-condition P est valide avant d'exécuter l'instruction i, alors la post-condition Q sera valide après l'exécution de i

Exemples

Les triplets de Hoare suivants sont-ils valides ?

- $\{x = 10\}x := x+1\{x = 11\}$? oui
- $\{c = 0\}$if c then $x = c$ else $x = 1$ fi$\{x = 0 \wedge c = 0\}$? non
- $\{c = 2\}$if c then $x = c$ else $x = 1$ fi$\{x = 2 \wedge c = 2\}$? oui
- $\{faux\}x = 0; y = 1\{x = 0 \wedge y = 2\}$? oui
- $\{i = 1\}$while i <= 10 do $i := i + 1$ done$\{i = 10\}$? non
- $\{i = 1\}$while ! (i <= 0) do $i := i + 1$ done$\{i = 0\}$? oui

Exemples

Les triplets de Hoare suivants sont-ils valides ?

- $\{x = 10\}x := x+1\{x = 11\}$? oui
- $\{c = 0\}$if c then x = c else x = 1 fi$\{x = 0 \wedge c = 0\}$? non
- $\{c = 2\}$if c then x = c else x = 1 fi$\{x = 2 \wedge c = 2\}$? oui
- $\{faux\}x = 0; y = 1\{x = 0 \wedge y = 2\}$? oui
- $\{i = 1\}$while i <= 10 do i := i + 1 done$\{i = 10\}$? non
- $\{i = 1\}$while ! (i <= 0) do i := i + 1 done$\{i = 0\}$? oui

Exemples

Les triplets de Hoare suivants sont-ils valides ?

- $\{x = 10\}x := x+1\{x = 11\}$? oui
- $\{c = 0\}$if c then x = c else x = 1 fi$\{x = 0 \wedge c = 0\}$? non
- $\{c = 2\}$if c then x = c else x = 1 fi$\{x = 2 \wedge c = 2\}$? oui
- $\{faux\}x = 0; y = 1\{x = 0 \wedge y = 2\}$? oui
- $\{i = 1\}$while i <= 10 do i := i + 1 done$\{i = 10\}$? non
- $\{i = 1\}$while ! (i <= 0) do i := i + 1 done$\{i = 0\}$? oui

Exemples

Les triplets de Hoare suivants sont-ils valides ?

- $\{x = 10\}x := x+1\{x = 11\}$? oui
- $\{c = 0\}$if c then $x = c$ else $x = 1$ fi$\{x = 0 \wedge c = 0\}$? non
- $\{c = 2\}$if c then $x = c$ else $x = 1$ fi$\{x = 2 \wedge c = 2\}$? oui
- $\{\text{faux}\}x = 0; y = 1\{x = 0 \wedge y = 2\}$? oui
- $\{i = 1\}$while i <= 10 do $i := i + 1$ done$\{i = 10\}$? non
- $\{i = 1\}$while ! (i <= 0) do $i := i + 1$ done$\{i = 0\}$? oui

Exemples

Les triplets de Hoare suivants sont-ils valides ?

- $\{x = 10\}x := x+1\{x = 11\}$? oui
- $\{c = 0\}$if c then $x = c$ else $x = 1$ fi$\{x = 0 \wedge c = 0\}$? non
- $\{c = 2\}$if c then $x = c$ else $x = 1$ fi$\{x = 2 \wedge c = 2\}$? oui
- $\{faux\}x = 0; y = 1\{x = 0 \wedge y = 2\}$? oui
- $\{i = 1\}$while i <= 10 do $i := i + 1$ done$\{i = 10\}$? non
- $\{i = 1\}$while ! (i <= 0) do $i := i + 1$ done$\{i = 0\}$? oui

Exemples

Les triplets de Hoare suivants sont-ils valides ?

- $\{x = 10\}x := x+1\{x = 11\}$? oui
- $\{c = 0\}$if c then $x = c$ else $x = 1$ fi$\{x = 0 \wedge c = 0\}$? non
- $\{c = 2\}$if c then $x = c$ else $x = 1$ fi$\{x = 2 \wedge c = 2\}$? oui
- $\{faux\}x = 0; y = 1\{x = 0 \wedge y = 2\}$? oui
- $\{i = 1\}$while i <= 10 do $i := i + 1$ done$\{i = 10\}$? non
- $\{i = 1\}$while ! (i <= 0) do $i := i + 1$ done$\{i = 0\}$? oui

Exemples

Les triplets de Hoare suivants sont-ils valides ?

- $\{x = 10\}x := x+1\{x = 11\}$? oui
- $\{c = 0\}$if c then $x = c$ else $x = 1$ fi$\{x = 0 \wedge c = 0\}$? non
- $\{c = 2\}$if c then $x = c$ else $x = 1$ fi$\{x = 2 \wedge c = 2\}$? oui
- $\{faux\}x = 0; y = 1\{x = 0 \wedge y = 2\}$? oui
- $\{i = 1\}$while i <= 10 do $i := i + 1$ done$\{i = 10\}$? non
- $\{i = 1\}$while ! (i <= 0) do $i := i + 1$ done$\{i = 0\}$? oui

Sémantique axiomatique

- La sémantique axiomatique d'une instruction i peut être définie par l'ensemble des triplets de Hoare valides portant sur i.
- avec une définition informelle, il peut être néanmoins difficile de savoir si un triplet de Hoare est valide ou non, même sur le langage While
- la sémantique axiomatique doit être définie formellement pour éviter toute ambigüité.
- on utilise un système de règles de déduction dans ce but
- ce système est la logique de (Floyd-)Hoare

Quelques définitions et notations

- Une règle d'inférence $\dfrac{H_1 \quad \cdots \quad H_n}{G}$ signifie sous les hypothèses $H_1, \ldots, H_n$, on peut déduire le but G

- La propriété $P[x \leftarrow e]$ signifie la propriété P dans laquelle la variable x a été substituée par l'expression e.

Sémantique axiomatique du langage While

$$\frac{}{\{P\}\text{skip}\{P\}} \qquad \frac{\{P\}i_1\{R\} \qquad \{R\}i_2\{Q\}}{\{P\}i_1; i_2\{Q\}}$$

$$\frac{}{\{P[x \leftarrow e]\}x := e\{P\}}$$

$$\frac{\{P \wedge e \neq 0\}i_1\{Q\} \qquad \{P \wedge e = 0\}i_2\{Q\}}{\{P\}\text{if } e \text{ then } i_1 \text{ else } i_2 \text{ fi}\{Q\}}$$

$$\frac{\{I \wedge e \neq 0\}i\{I\}}{\{I\}\text{while } e \text{ do } i \text{ done}\{I \wedge e = 0\}}$$

$$\frac{P \implies P' \qquad \{P'\}i\{Q'\} \qquad Q' \implies Q}{\{P\}i\{Q\}}$$

- ces règles d'inférence définissent une sémantique, pas un algorithme
- séquence : comment établir le prédicat R intermédiaire ?
- boucle : comment établir l'invariant de boucle I ?
- règle d'affaiblissement (la dernière) : quand l'appliquer ?
- erreurs à l'exécution : comment sont-elles prises en compte ?
- affectation : nature "arrière" de cet axiome, le secret de la réussite du calcul de plus-faible précondition qui va suivre

$$\frac{A \qquad B}{\{\}\texttt{c} := 0; \{0 \leq c \leq 101\} \qquad \{0 \leq c \leq 101\}\texttt{while ... do ... done}\{c = 101\}}$$

$$\{\}\texttt{c} := 0; \texttt{while c <= 100 do c} := \texttt{c + 1 done}\{c = 101\}$$

$$\frac{\texttt{true} \implies 0 = 0 \qquad \dfrac{}{\{0 = 0\}\texttt{c} := 0\{c = 0\}} \qquad c = 0 \implies 0 \leq c \leq 101}{A}$$

$$\frac{\dfrac{\dfrac{}{\{0 \leq c + 1 \leq 101\}\texttt{c} := \texttt{c + 1}\{0 \leq c \leq 101\}} \quad 0 \leq c \leq 101 \wedge \texttt{c <= 100} \neq 0 \implies 0 \leq c + 1 \leq 101}{\{0 \leq c \leq 101 \wedge \texttt{c <= 100} \neq 0\}\texttt{c} := \texttt{c + 1}\{0 \leq c \leq 101\}} \quad 0 \leq c \leq 101 \implies \texttt{true}}{\{0 \leq c \leq 101\}\texttt{while ... do ... done}\{0 \leq c \leq 101 \wedge \texttt{c <= 100} = 0\}}$$

$$B$$

Démontrer la validité des triplets de Hoare suivants.

1. $\{\}$
   if c then $i := 0; x := i$ else $x := c$ fi
   $\{x = 0\}$

2. $\{\}$
   $N := 10; sum := 0; i := 1;$
   while i <= N do $sum := sum + i; i := i + 1$ done
   $\{sum = 55\}$

   rappel : $\sum_{i=1}^{n} i = n \times (n+1)/2$

Calcul de plus-faible précondition (WP)

But

Étant donné, une pré-condition Pre, une post-condition $Post$ et un programme p, $\{Pre\}p\{Post\}$ est-il un triplet de Hoare valide ?

Comment ?

- Le programme est vu comme un transformeur de prédicat
- On part de la post-condition $Post$ à la fin du programme et on raisonne en effectuant les étapes de calcul "à l'envers"
- à chaque étape de calcul, on connaît la post-condition Q à vérifier et l'instruction i concernée et on déduit la propriété P minimale (la plus faible) telle que $\{P\}i\{Q\}$ soit un triplet de Hoare valide
- Lorsque le début du programme est atteint, on doit prouver $Pre \implies P$

Calcul de WP

Définition par induction sur la structure du programme

$$WP(\text{skip}, Q) = Q$$

$$WP(i_1; i_2, Q) = WP(i_1, WP(i_2, Q))$$

$$WP(x := e, Q) = Q[x \leftarrow e] \qquad \text{si } e \text{ est évaluée sans erreur}$$

$$WP(\text{if } e \text{ then } i_1 \text{ else } i_2 \text{ fi}, Q) = e \neq 0 \implies WP(i_1, Q) \wedge e = 0 \implies WP(i_2, Q)$$

$$WP(\text{while } e \text{ do } i \text{ done}, Q) = \begin{cases} I & \text{à propager} \\ \left[\begin{array}{l} I \implies \\ (e \neq 0 \implies WP(i, I) \wedge e = 0 \implies Q) \end{array} \right. \end{cases}$$

Calcul de WP : exemple

$$
\begin{aligned}
e &= \mathtt{c <= 100} \\
B &= \text{while } e \text{ do } \mathtt{c} := \mathtt{c} + 1 \text{ done} \\
P &= \mathtt{c} := 0; B \\
Q &= c = 101 \\
I &= 0 \leq c \leq 101
\end{aligned}
$$

$$
\begin{aligned}
WP(P, Q) &= WP(\mathtt{c} := 0, WP(B, Q)) \\
&= WP(B, Q)[c \leftarrow 0] \\
&= 0 \leq 0 \leq 101 \wedge (I \implies \\
&\qquad (e \neq 0 \implies WP(\mathtt{c} := \mathtt{c} + 1, I) \wedge e = 0 \implies Q)) \\
&= I \implies (e \neq 0 \implies 0 \leq c + 1 \leq 101 \wedge e = 0 \implies Q) \\
&= 0 \leq c \leq 101 \implies \\
&\qquad (c \mathtt{<= 100} \neq 0 \implies 0 \leq c + 1 \leq 101 \wedge c \mathtt{<= 100} = 0 \implies Q) \\
&= 0 \leq c \leq 100 \implies 0 \leq c + 1 \leq 101 \wedge 100 < c \leq 101 \implies Q \\
&= \mathtt{true}
\end{aligned}
$$

Calcul de WP : exercices

À l'aide d'un calcul de plus faible précondition, indiquer si les propriétés Q sont valides ou non après l'exécution des programmes suivants.

1. $Q : x = 0$

   programme :

   if c then $i := 0; x := i$ else $x := c$ fi

2. $Q : sum = 55$

   programme :

   $N := 10; sum := 0; i := 1;$
   while $i <= N$ do $sum := sum + i; i := i + 1$ done

   rappel : $\sum_{i=1}^{n} i = n \times (n+1)/2$

Calcul de WP : discussion

- obligation de preuves pour la gestion des erreurs
- boucle : comment établir l'invariant I ?
  - indiqué manuellement par l'utilisateur :-(
  - idéalement, déduit automatiquement (par exemple, par interprétation abstraite)
- *WP* définit un algorithme, modulo le problème des invariants de boucle

Corrections partielle et totale

Correction partielle

garantie que si le programme p termine, alors p satisfait la spécification S

Correction totale

garantie que le programme p termine et p satisfait la spécification S

Terminaison

prouver la terminaison nécessite la donnée d'une mesure décroissante pour chaque boucle (un variant), le plus souvent indiquée manuellement par l'utilisateur

Choix

Dépend du contexte : la correction partielle peut être suffisante.

Quand et pourquoi sont-elles générées ?

- pour garantir la sûreté du programme, dès qu'une opération peut entraîner une erreur à l'exécution
  - divisions par zéro
  - déréférencement des pointeurs
  - overflow des opérations arithmétiques
- pour vérifier les spécifications utilisateurs
  - post-conditions
  - assertions
  - invariants et variants de boucles
  - ...

Comment effectuer une preuve ?

Prouver ? Oui, mais comment ?

- prouver à la main (comme n'importe quel théorème de maths)
  - impossible en pratique
- prouver à l'aide d'un assistant de preuves : outil pour assister un humain dans l'activité de preuves en le guidant, en résolvant les parties triviales ou calculatoires, et en vérifiant leur validité
  - Coq
  - Isabelle
  - PVS
- prouver avec des prouveurs automatiques : outil "presse-bouton" effectuant les preuves de manière automatique
  - Alt-Ergo
  - Simplify
  - Z3

Gestion des obligations de preuve

Contexte de la preuve de programmes (de taille réelle)

- les obligations de preuve sont générées automatiquement par des outils
- plusieurs milliers d'obligations de preuve (voire plus)
- plusieurs centaines d'hypothèses par obligation de preuve (voire plus)

Faisabilité des preuves

- infaisable par un humain
- reste très lourd avec un assistant de preuves
- emploi de prouveurs automatiques indispensables
- les prouveurs automatiques ne savent pas tout faire

Guider les preuves

Vérifier les obligations de preuve en pratique

- utilisation de différents prouveurs automatiques en parallèle
- spécifications ad-hoc pour aider les prouveurs
- assistant de preuves pour les quelques obligations de preuve restantes

Spécifications additionnelles

- annotations de boucle : clauses `loop invariant` et `loop variant`

```
int a = 0, b = 10;
/*@ loop invariant 0 <= a && b <= 10;
  @ loop variant b - a; */
while (a <= b) { a++; b--; }
```

- insertion d'assertions (joue le rôle de lemme intermédiaire)
- insertion de lemmes additionnels

Modèle mémoire

Représentation de la mémoire

- En pratique, un programme (en particulier un programme C) contient des opérations sur la mémoire de nature complexe (manipulations de pointeurs, de tableaux, structures, ...)
- Le calcul de WP doit en tenir compte

Modèle mémoire et abstraction

Différents modèles mémoires

- modèle mémoire bas-niveau : la mémoire est vue comme un grand tableau de bits ou d'octets
  - simple, modélise bien la réalité
  - possibilité de vérifier plein de propriétés (y compris très bas niveaux)
  - obligations de preuve très compliquées à vérifier
- modèle mémoire haut-niveau : vue plus abstraite de la mémoire
  - obligations de preuve plus faciles à prouver
  - plus le modèle est de haut-niveau, moins les propriétés de bas-niveaux peuvent être prouvées
- exemple : modèle de Burstall-Bornat pour gérer les structures. Chaque champ est considéré comme séparé des autres.
- plus généralement, on peut séparer la mémoire en différentes régions pour faciliter la gestion des pointeurs et des tableaux.
- limitations : par exemple non gestion des casts de pointeurs

Calcul de WP : résumé

Avantages

- modulaire : grâce aux contrats, l'analyse est faite fonction par fonction
- correcte (sous réserve de prouver toutes les obligations de preuve, dans un modèle logique cohérent)
- complète (sous réserve de fournir les bons invariants et variants)
- possible de vérifier des spécifications très compliquées

Inconvénients

- requiert d'insérer manuellement des annotations
- 100 % des obligations de preuve prouvées automatiquement est illusoire sur un programme réel avec des spécifications complexes
- peut nécessiter un travail de preuve manuelle complexe

Calcul de WP : résumé

"Beware of bugs in the above code :
I have only proved it correct, not tried it."

(Donald Knuth 1977)

Plan Général

- Introduction aux méthodes formelles
- Sémantique
- Spécification formelle
- Vérification déductive
- Analyse dataflow, interprétation abstraite
- Logiques temporelles et model checking

Analyse flot de données (data flow)

Vue informelle

- on propage des informations le long du graphe de contrôle (fonction de transition)
- lorsqu'un sommet admet plusieurs prédécesseurs (sommet de jonction), on considère la réunion des informations propagés
- on termine lorsque la propagation des informations ne fait plus "augmenter" le résultat

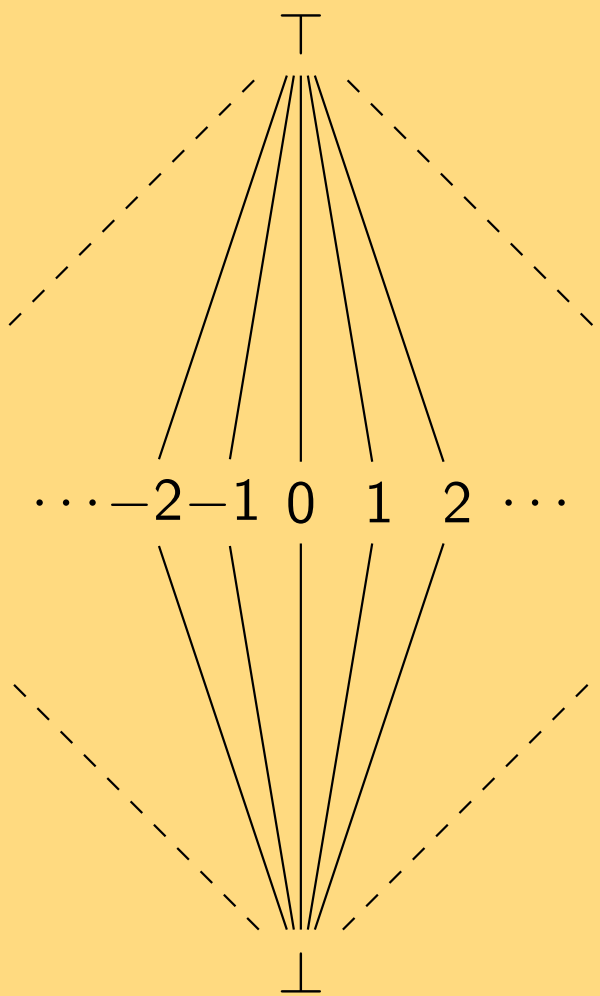
Questions

- ce procédé termine-t-il toujours ?
- ce procédé est-il correct ?

Propagation de constantes

Treillis de base

On part du treillis $\mathcal{C}$ suivant :



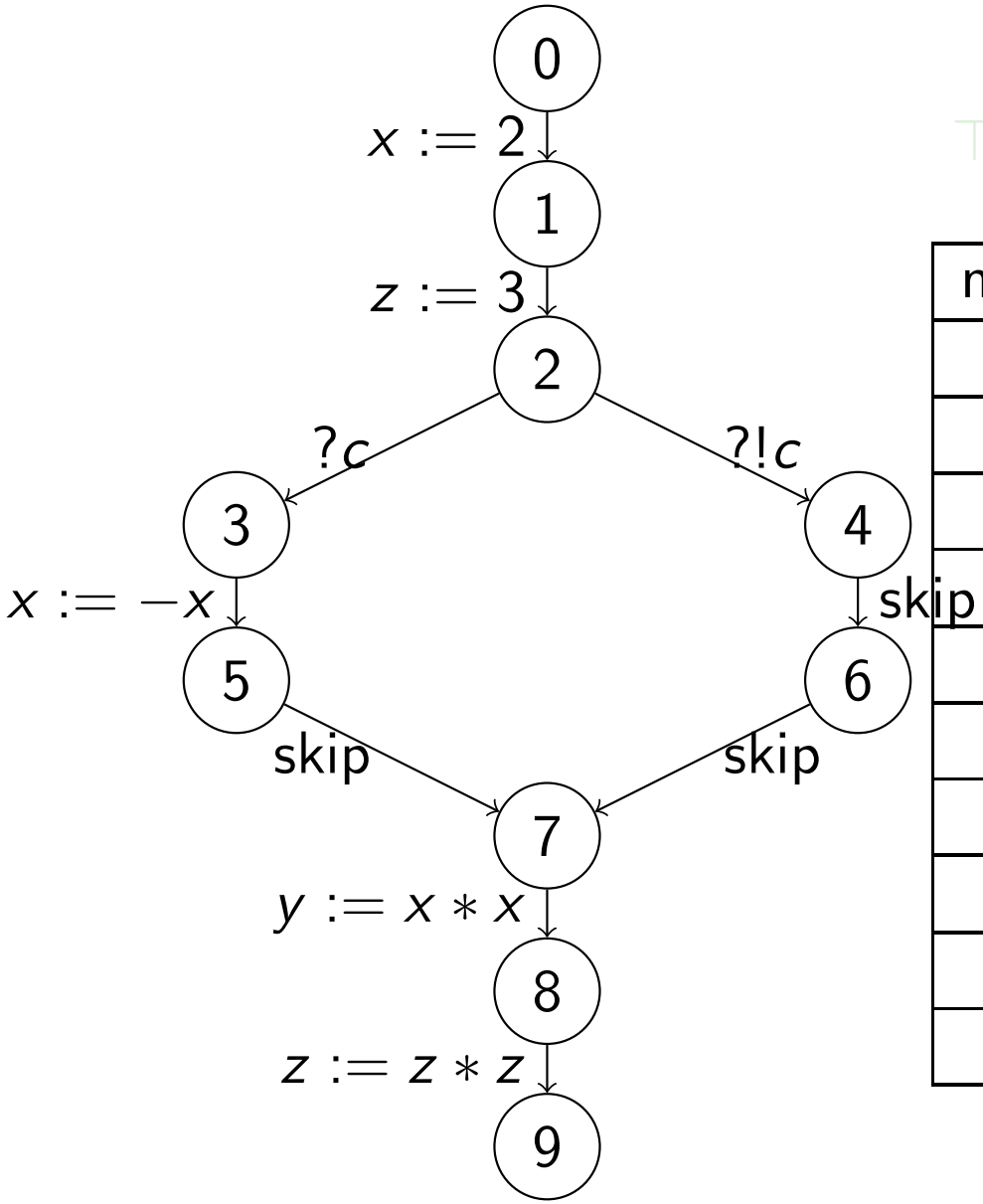
Cadre d'analyse

- $E = Var \rightharpoonup \mathcal{C}$
- $i = \lambda x.\top$
- $f_{x:=e}(\sigma^\sharp) = \sigma^\sharp[x \leftarrow [\![e]\!]\,\sigma^\sharp]$
- $f_{\text{skip}}(\sigma^\sharp) = \sigma^\sharp$
- $f_{?e}(\sigma^\sharp) = \begin{cases} \bot & \text{si } [\![e]\!]\,\sigma^\sharp = 0 \\ \sigma^\sharp & \text{sinon} \end{cases}$

Propriétés

- Correction : Si à la fin de l'analyse, on a ${\sigma^\sharp}_i(x) = \mathtt{v}$, toute trace passant par i associe $\mathtt{v}$ à x.
- Terminaison : L'analyse termine toujours

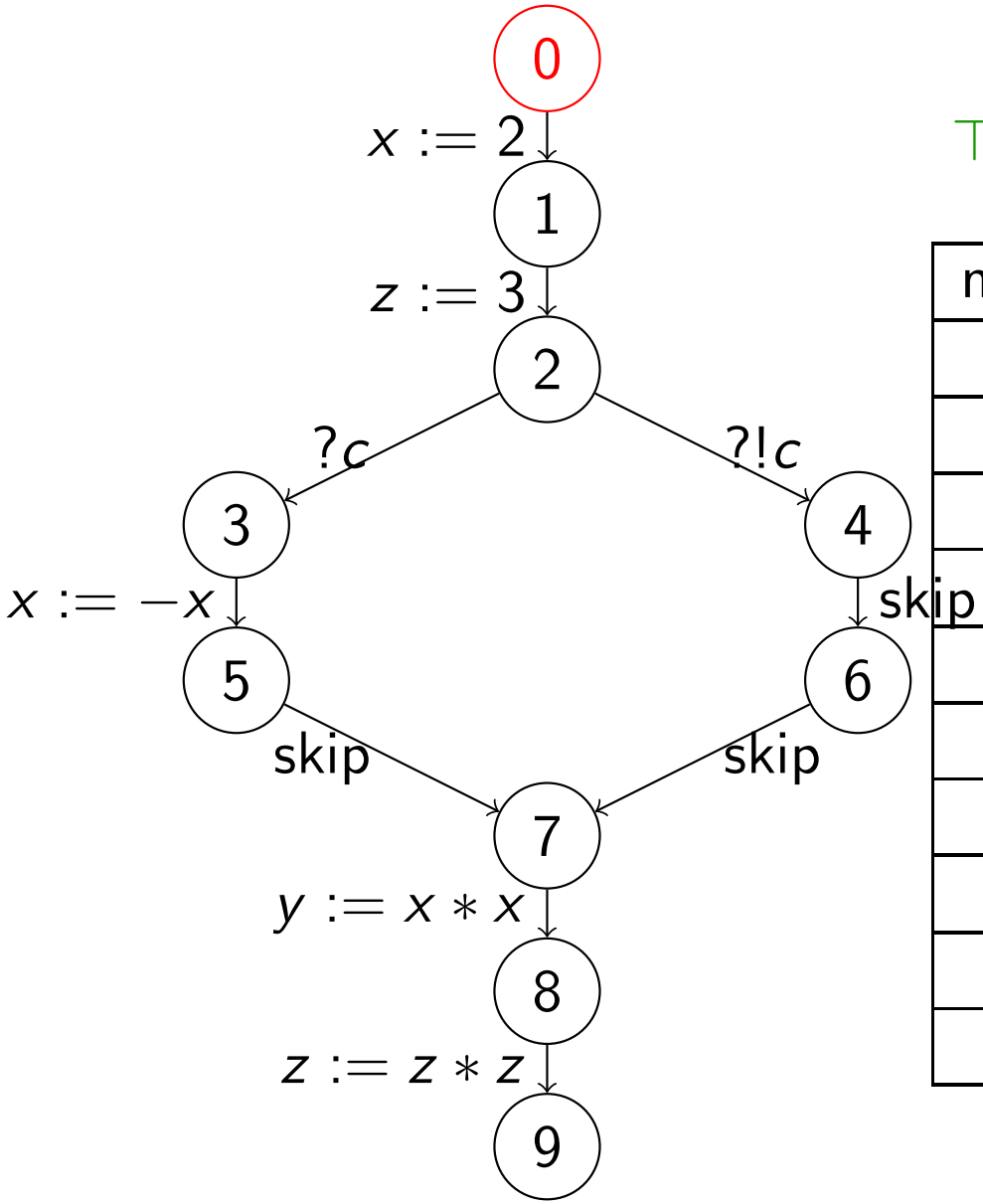
Exemple de propagation



$\top$ = "pas d'information"

nœud	c	x	y	z
0	$\top$	$\top$	$\top$	$\top$
1	$\top$	2	$\top$	$\top$
2	$\top$	2	$\top$	3
3	$\top$	2	$\top$	3
4	0	2	$\top$	3
5	$\top$	-2	$\top$	3
6	0	2	$\top$	3
7	$\top$	$\top$	$\top$	3
8	$\top$	$\top$	$\top$	3
9	$\top$	$\top$	$\top$	9

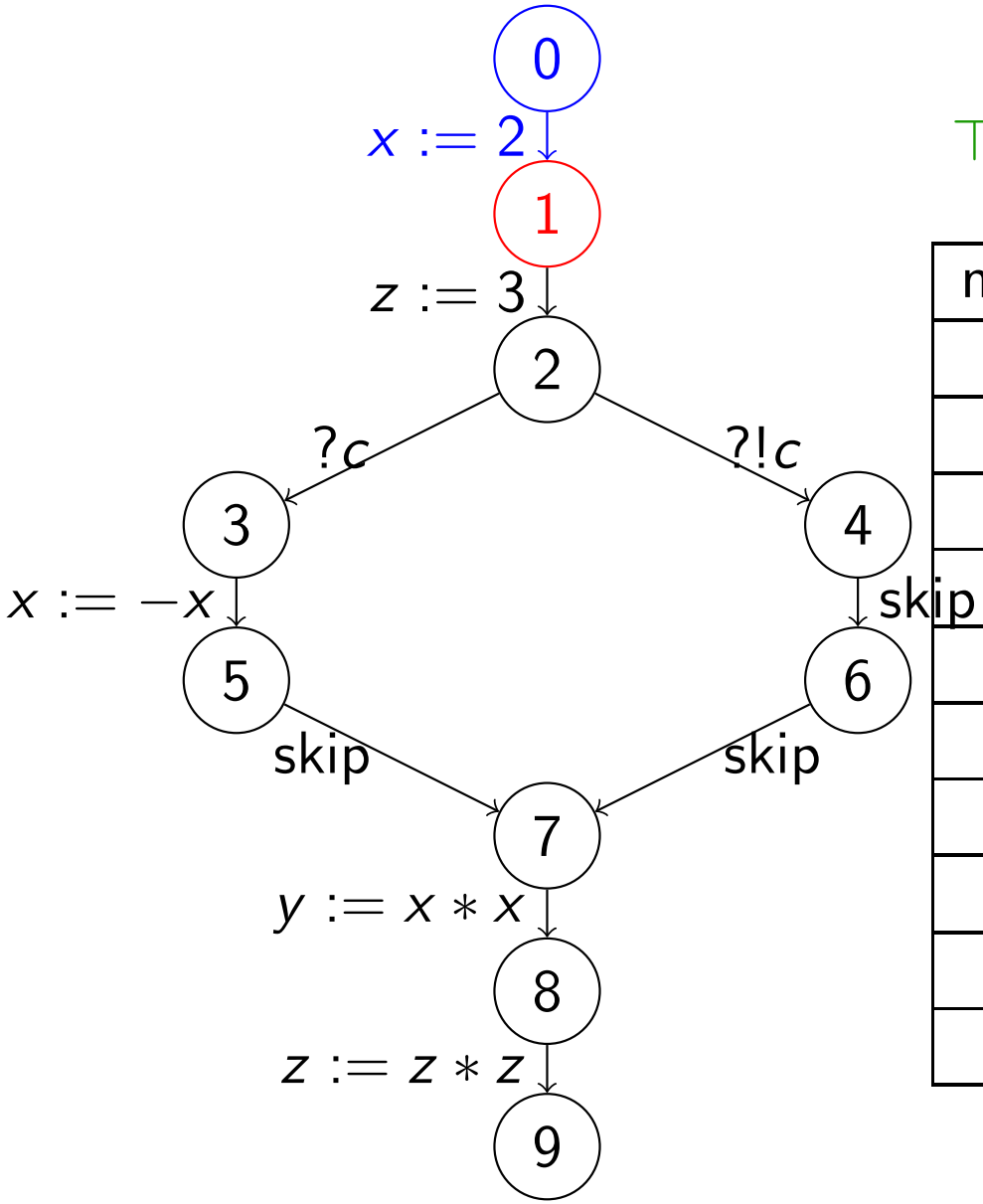
Exemple de propagation



$\top$ = "pas d'information"

nœud	c	x	y	z
0	$\top$	$\top$	$\top$	$\top$
1	$\top$	2	$\top$	$\top$
2	$\top$	2	$\top$	3
3	$\top$	2	$\top$	3
4	0	2	$\top$	3
5	$\top$	−2	$\top$	3
6	0	2	$\top$	3
7	$\top$	$\top$ 2	$\top$	3
8	$\top$	$\top$	$\top$	3
9	$\top$	$\top$	$\top$	9

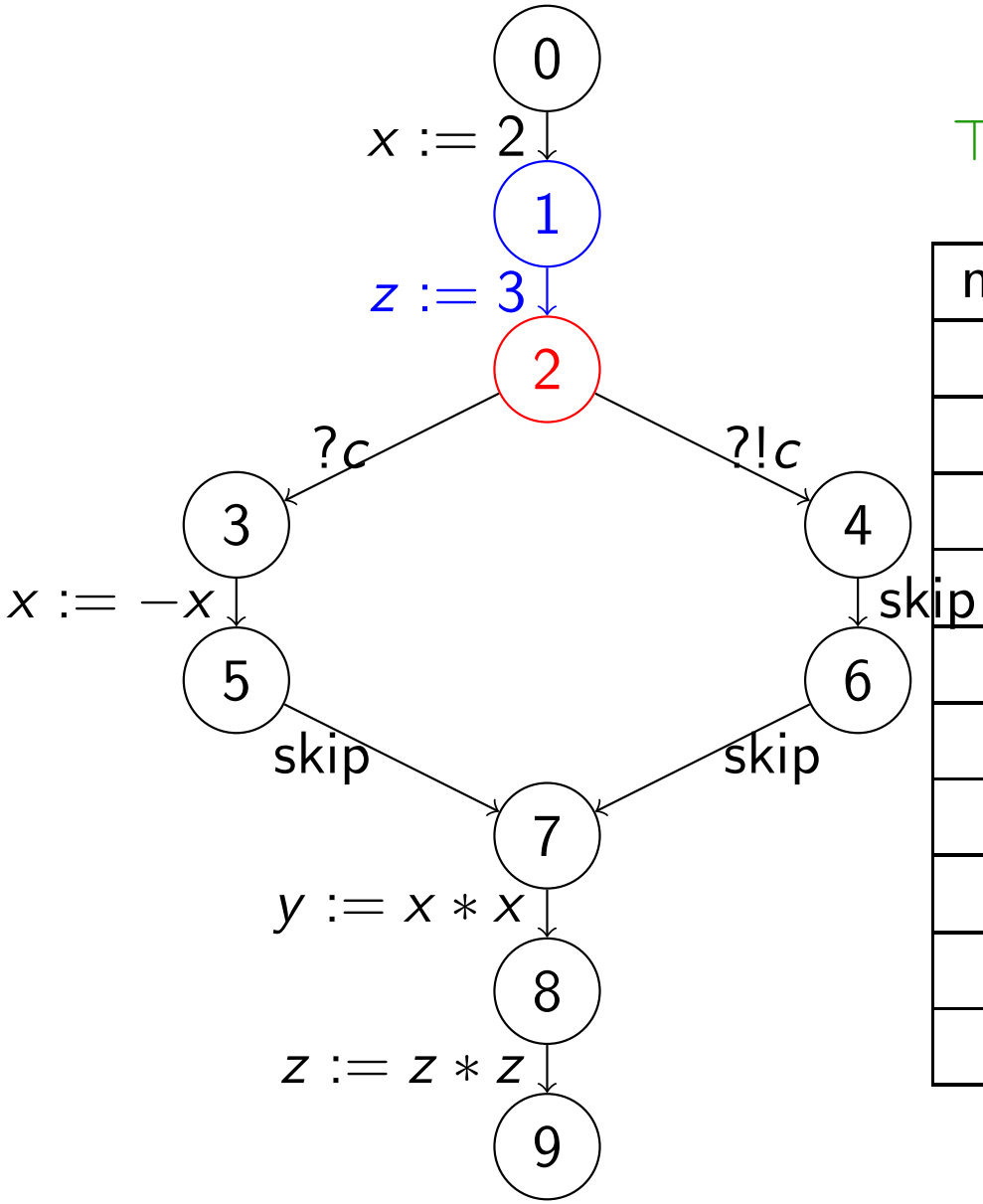
Exemple de propagation



$\top$ = "pas d'information"

nœud	c	x	y	z
0	$\top$	$\top$	$\top$	$\top$
1	$\top$	2	$\top$	$\top$
2	$\top$	2	$\top$	3
3	$\top$	2	$\top$	3
4	0	2	$\top$	3
5	$\top$	-2	$\top$	3
6	0	2	$\top$	3
7	$\top$	$\top$	$\top$	3
8	$\top$	$\top$	$\top$	3
9	$\top$	$\top$	$\top$	9

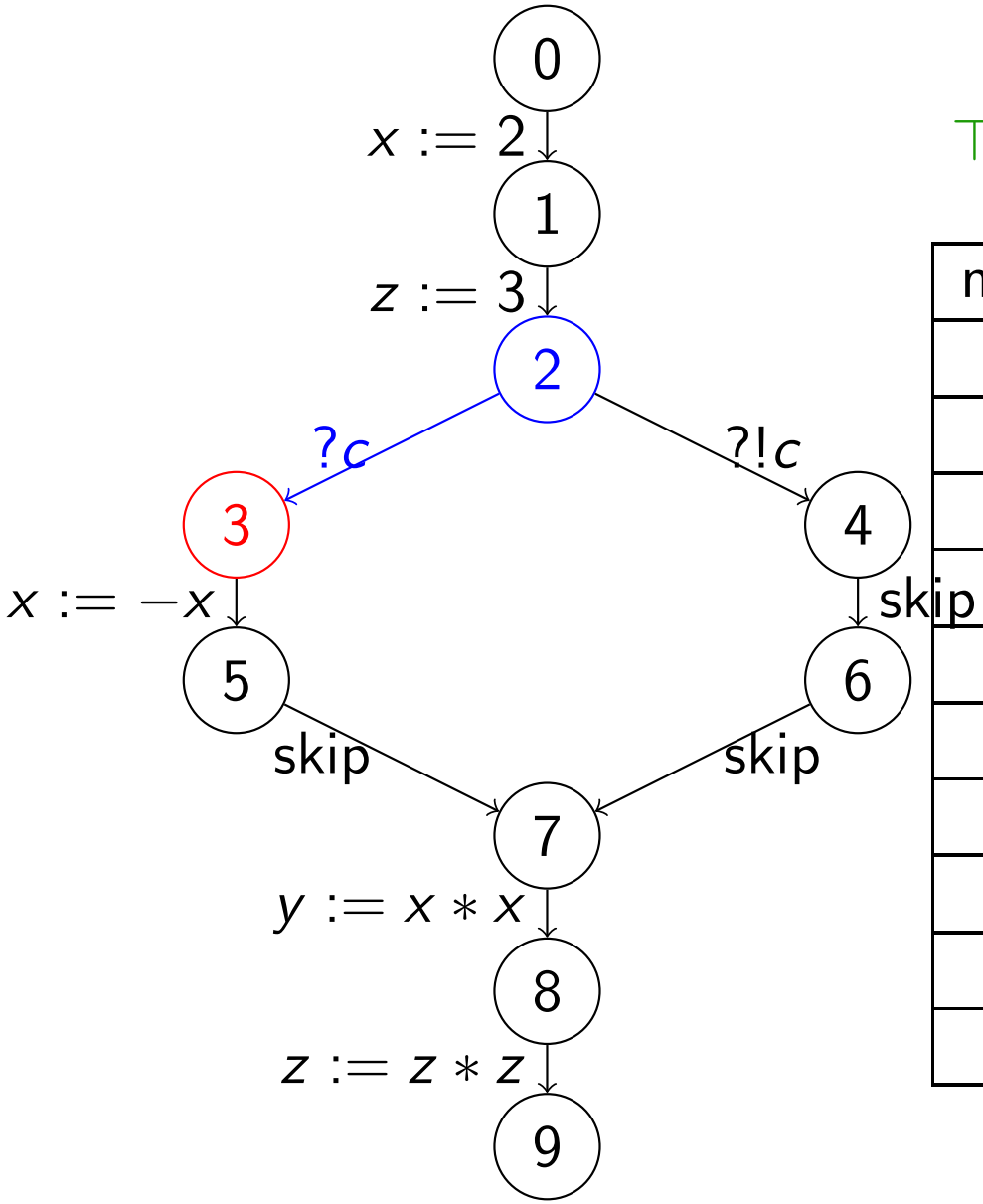
Exemple de propagation



$\top$ = "pas d'information"

nœud	c	x	y	z
0	$\top$	$\top$	$\top$	$\top$
1	$\top$	2	$\top$	$\top$
2	$\top$	2	$\top$	3
3	$\top$	2	$\top$	3
4	0	2	$\top$	3
5	$\top$	-2	$\top$	3
6	0	2	$\top$	3
7	$\top$	$\top$	$\top$	3
8	$\top$	$\top$	$\top$	3
9	$\top$	$\top$	$\top$	9

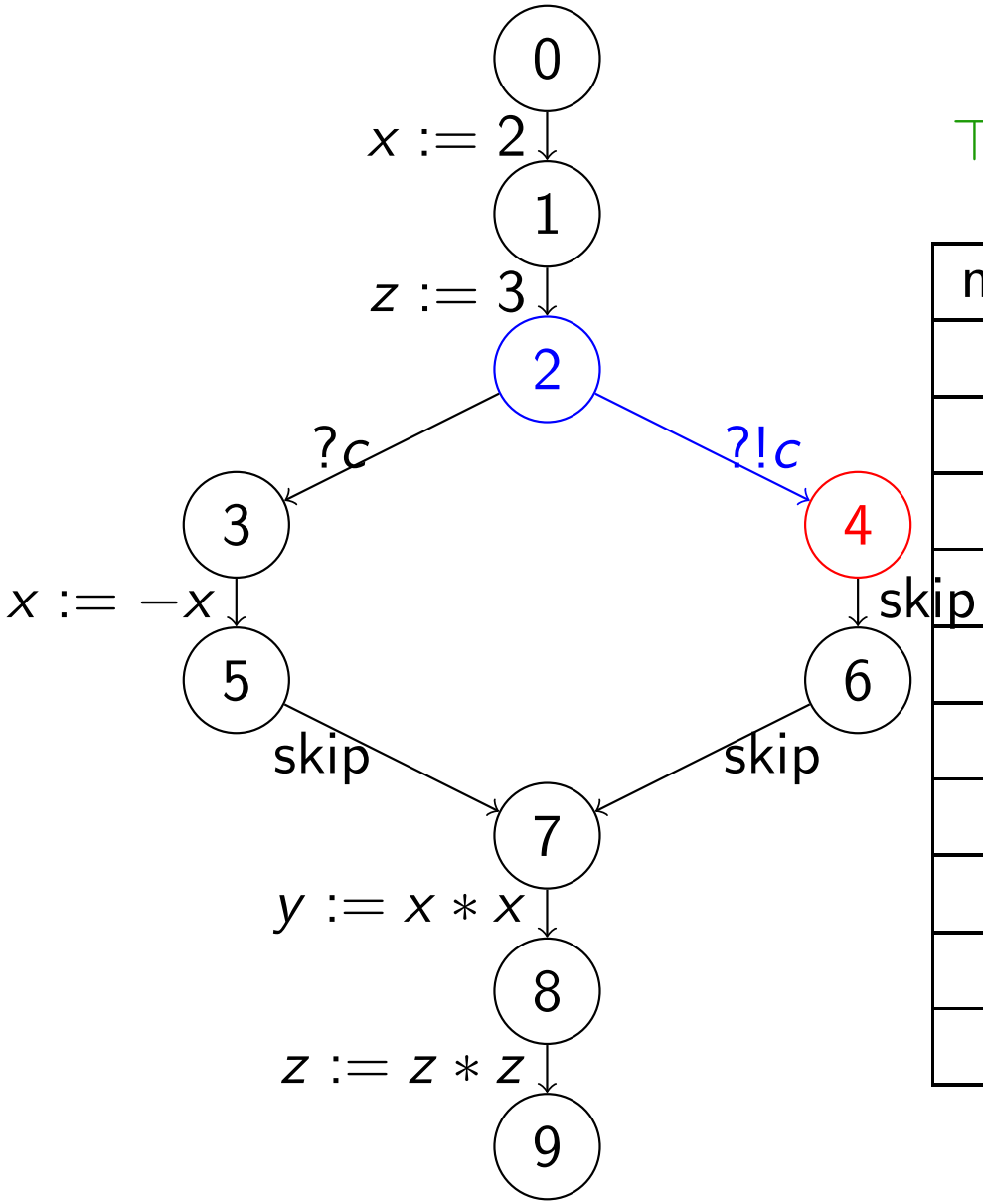
Exemple de propagation



$\top$ = "pas d'information"

nœud	c	x	y	z
0	$\top$	$\top$	$\top$	$\top$
1	$\top$	2	$\top$	$\top$
2	$\top$	2	$\top$	3
3	$\top$	2	$\top$	3
4	0	2	$\top$	3
5	$\top$	-2	$\top$	3
6	0	2	$\top$	3
7	$\top$	$\top 2$	$\top$	3
8	$\top$	$\top$	$\top$	3
9	$\top$	$\top$	$\top$	9

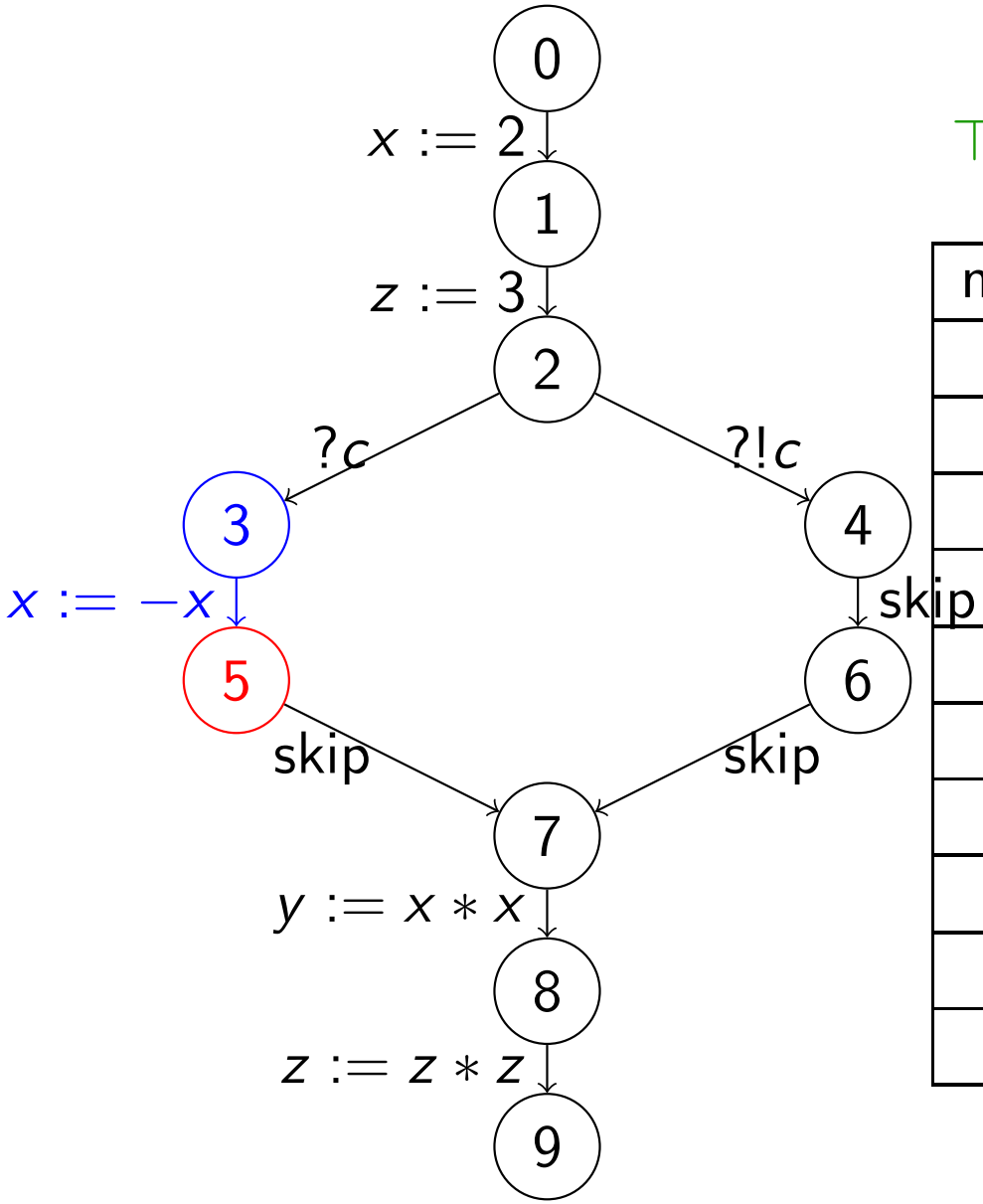
Exemple de propagation



$\top$ = "pas d'information"

nœud	c	x	y	z
0	$\top$	$\top$	$\top$	$\top$
1	$\top$	2	$\top$	$\top$
2	$\top$	2	$\top$	3
3	$\top$	2	$\top$	3
4	0	2	$\top$	3
5	$\top$	-2	$\top$	3
6	0	2	$\top$	3
7	$\top$	$\top \neq 2$	$\top$	3
8	$\top$	$\top$	$\top$	3
9	$\top$	$\top$	$\top$	9

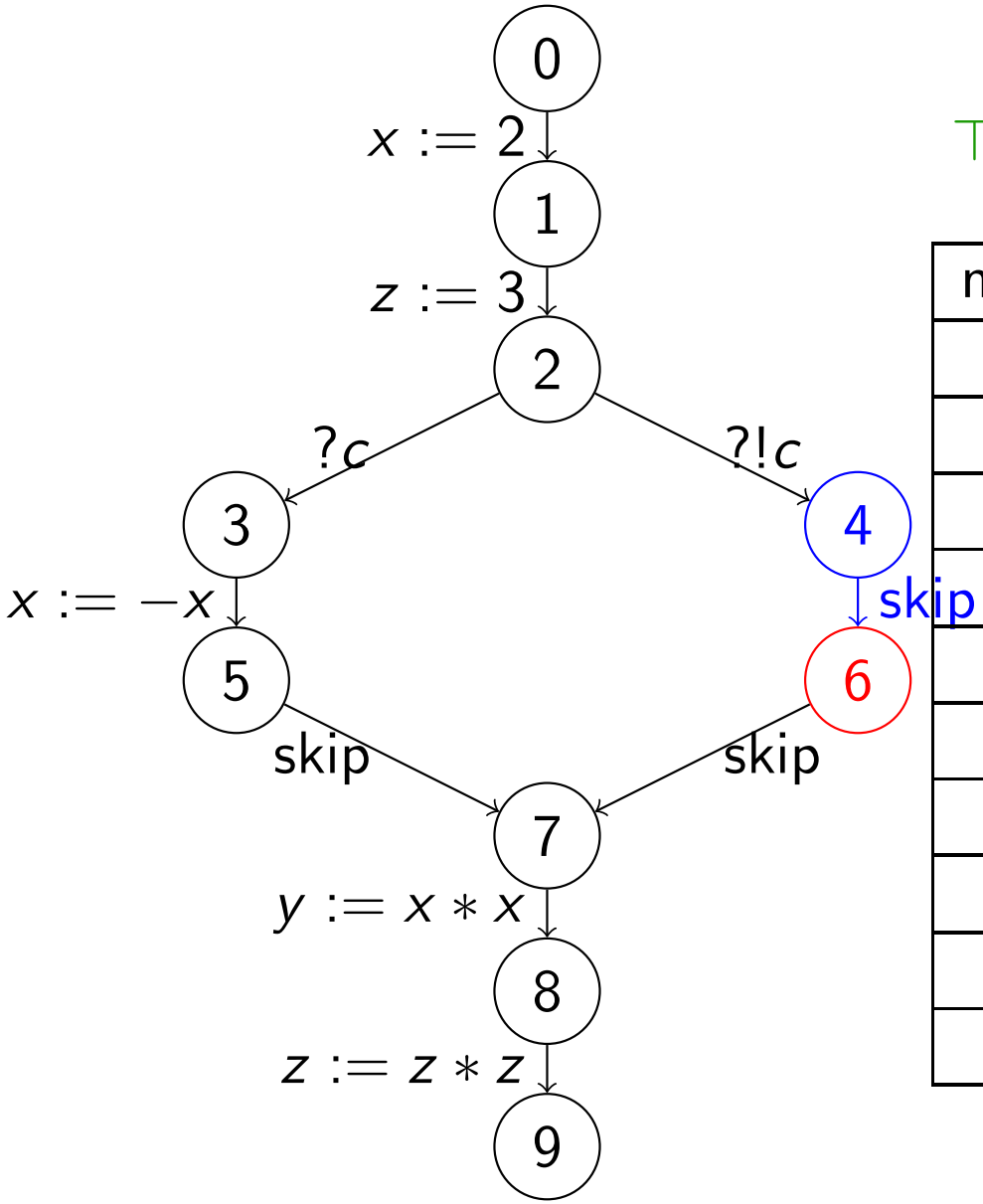
Exemple de propagation



$\top$ = "pas d'information"

nœud	c	x	y	z
0	$\top$	$\top$	$\top$	$\top$
1	$\top$	2	$\top$	$\top$
2	$\top$	2	$\top$	3
3	$\top$	2	$\top$	3
4	0	2	$\top$	3
5	$\top$	-2	$\top$	3
6	0	2	$\top$	3
7	$\top$	$\top 2$	$\top$	3
8	$\top$	$\top$	$\top$	3
9	$\top$	$\top$	$\top$	9

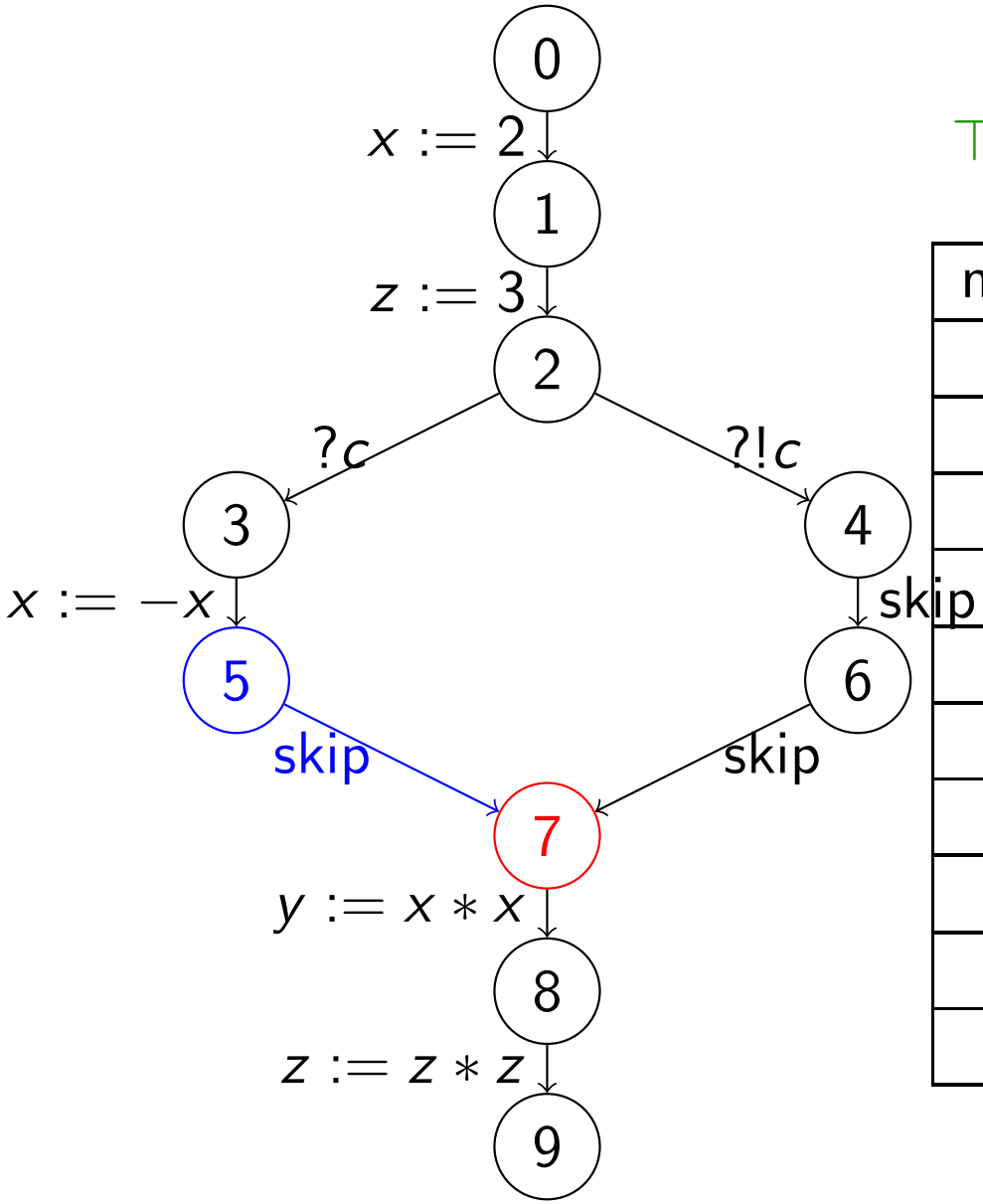
Exemple de propagation



$\top$ = "pas d'information"

nœud	c	x	y	z
0	$\top$	$\top$	$\top$	$\top$
1	$\top$	2	$\top$	$\top$
2	$\top$	2	$\top$	3
3	$\top$	2	$\top$	3
4	0	2	$\top$	3
5	$\top$	-2	$\top$	3
6	0	2	$\top$	3
7	$\top$	$\top 2$	$\top$	3
8	$\top$	$\top$	$\top$	3
9	$\top$	$\top$	$\top$	9

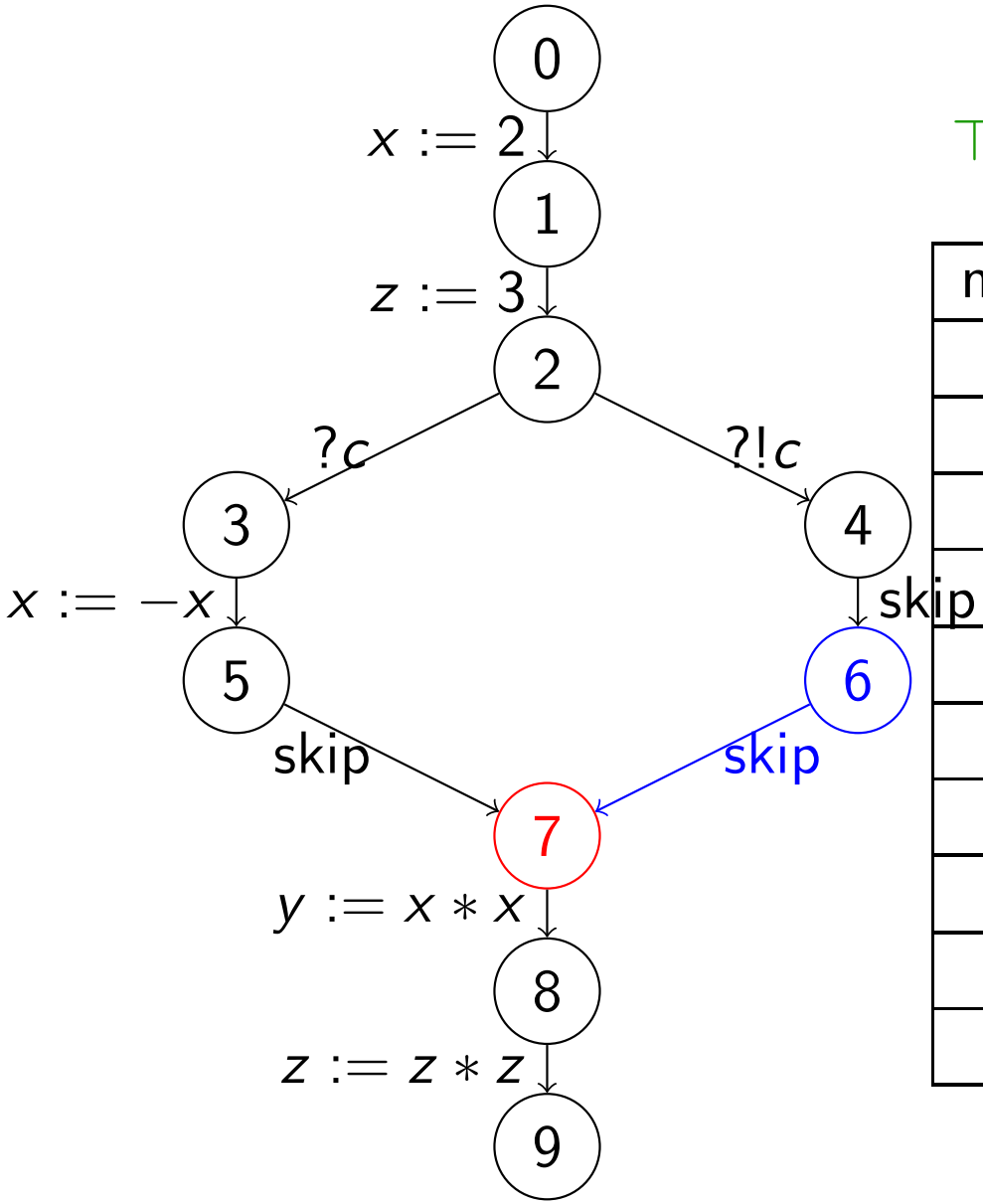
Exemple de propagation



$\top$ = "pas d'information"

nœud	c	x	y	z
0	$\top$	$\top$	$\top$	$\top$
1	$\top$	2	$\top$	$\top$
2	$\top$	2	$\top$	3
3	$\top$	2	$\top$	3
4	0	2	$\top$	3
5	$\top$	-2	$\top$	3
6	0	2	$\top$	3
7	$\top$	-2	$\top$	3
8	$\top$	$\top$	$\top$	3
9	$\top$	$\top$	$\top$	9

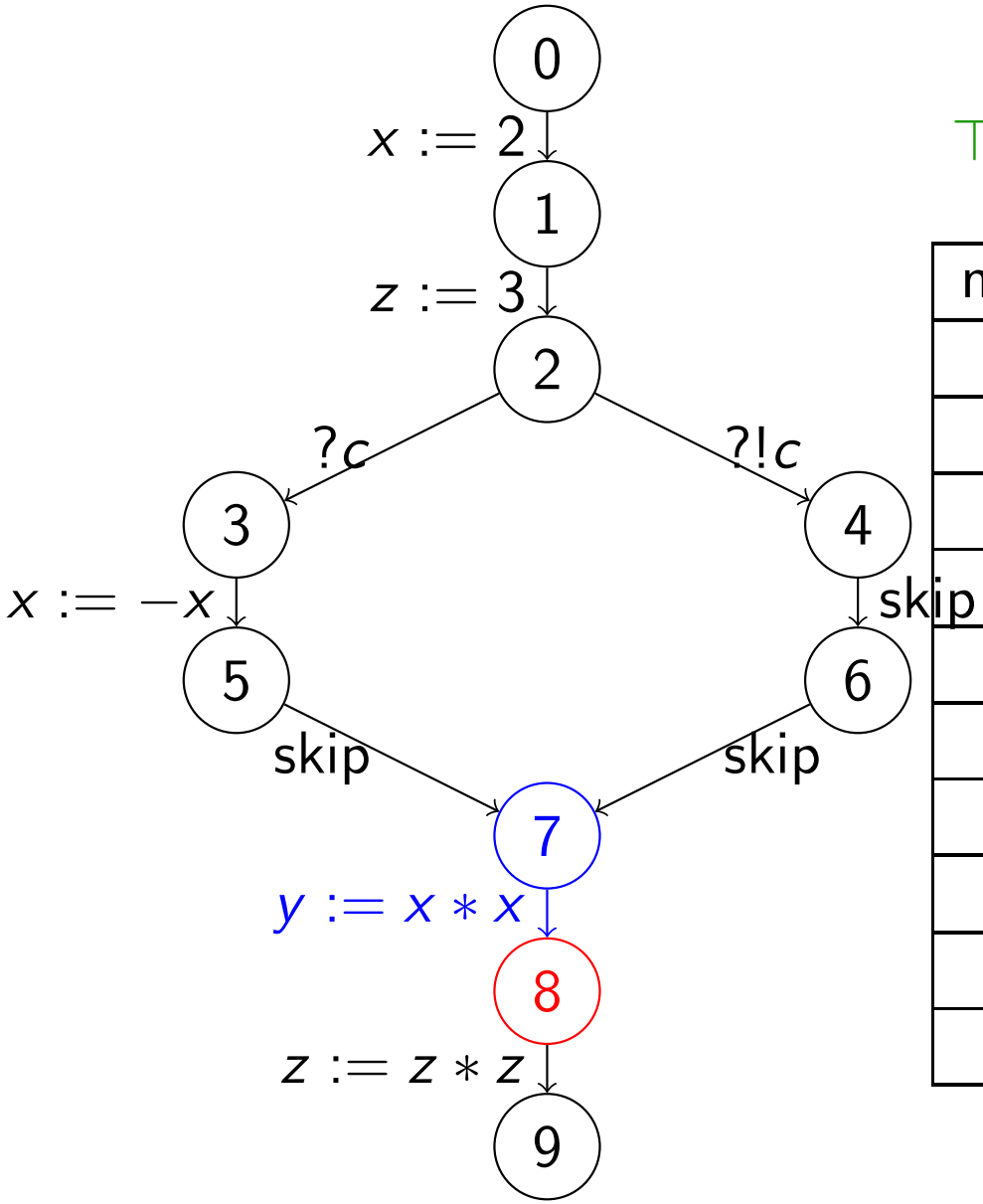
Exemple de propagation



$\top$ = "pas d'information"

nœud	c	x	y	z
0	$\top$	$\top$	$\top$	$\top$
1	$\top$	2	$\top$	$\top$
2	$\top$	2	$\top$	3
3	$\top$	2	$\top$	3
4	0	2	$\top$	3
5	$\top$	-2	$\top$	3
6	0	2	$\top$	3
7	$\top$	$\top 2$	$\top$	3
8	$\top$	$\top$	$\top$	3
9	$\top$	$\top$	$\top$	9

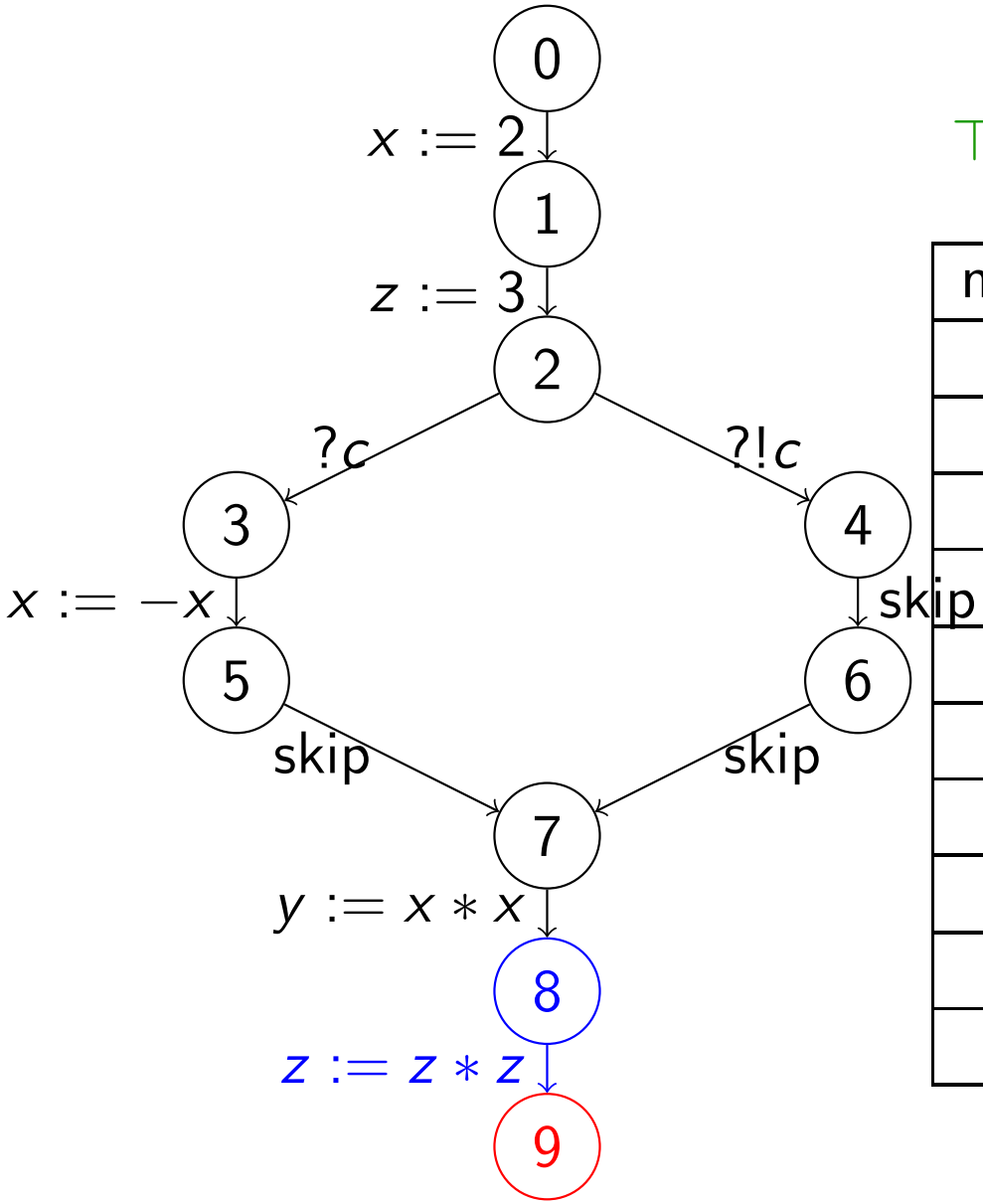
Exemple de propagation



$\top$ = "pas d'information"

nœud	c	x	y	z
0	$\top$	$\top$	$\top$	$\top$
1	$\top$	2	$\top$	$\top$
2	$\top$	2	$\top$	3
3	$\top$	2	$\top$	3
4	0	2	$\top$	3
5	$\top$	-2	$\top$	3
6	0	2	$\top$	3
7	$\top$	$\top 2$	$\top$	3
8	$\top$	$\top$	$\top$	3
9	$\top$	$\top$	$\top$	9

Exemple de propagation



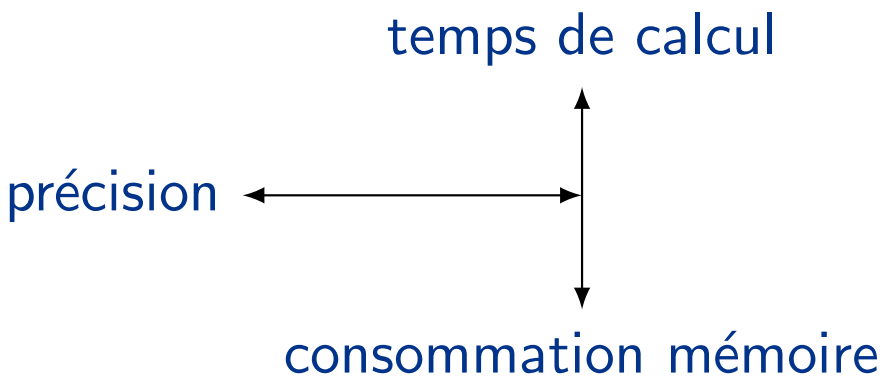
$\top$ = "pas d'information"

nœud	c	x	y	z
0	$\top$	$\top$	$\top$	$\top$
1	$\top$	2	$\top$	$\top$
2	$\top$	2	$\top$	3
3	$\top$	2	$\top$	3
4	0	2	$\top$	3
5	$\top$	-2	$\top$	3
6	0	2	$\top$	3
7	$\top$	$\top 2$	$\top$	3
8	$\top$	$\top$	$\top$	3
9	$\top$	$\top$	$\top$	9

Une question de choix

En pratique, bien choisir son domaine abstrait est fondamental

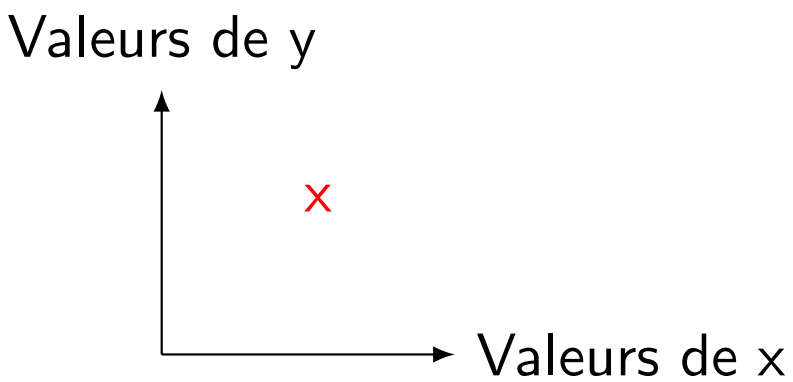
- doit être suffisament précis
- en particulier, doit permettre d'énoncer la propriété souhaitée
- doit être calculable pour un coût temps/mémoire raisonnable
  (heures de calcul, Go de RAM pour des programmes réels)



- domaines non relationels : aucune relation conservée entre les éléments du domaine $\Rightarrow$ peu précis mais pas cher
- domaines relationnels : relations entre éléments du domaine $\Rightarrow$ plus précis mais plus cher

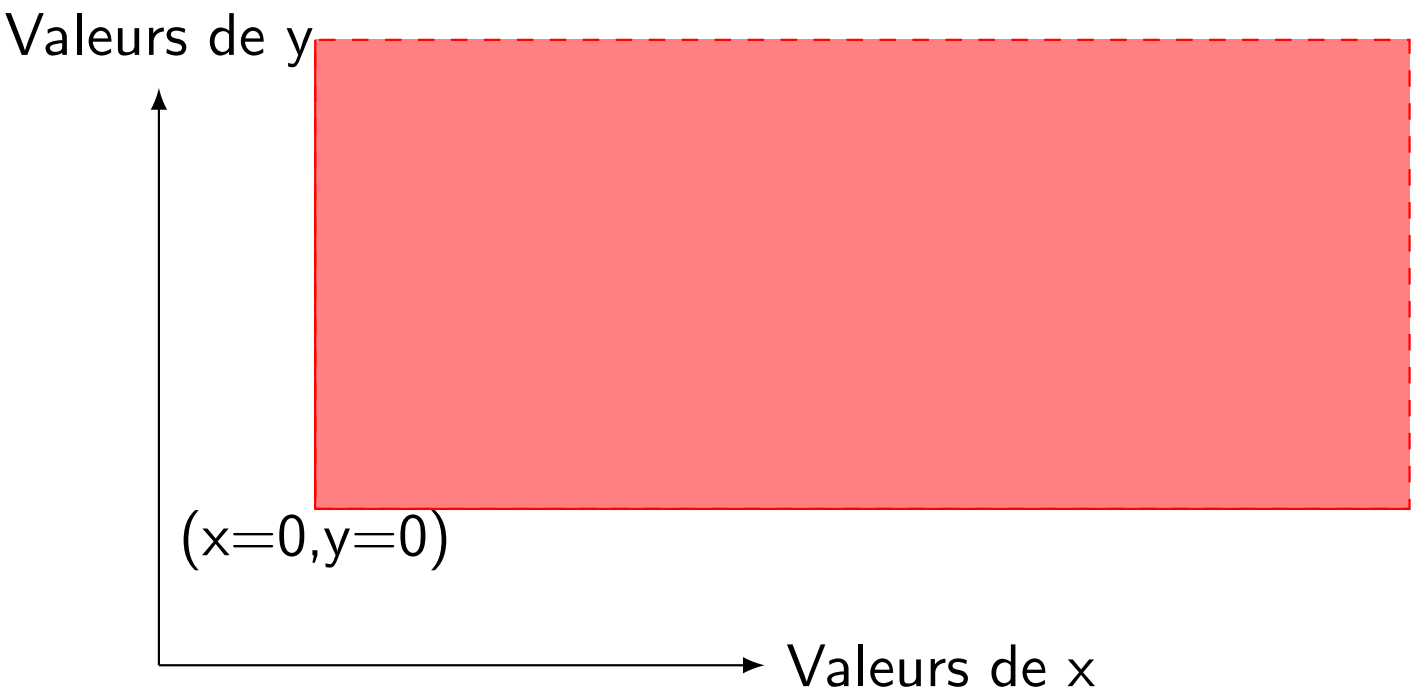
Domaines des constantes

- $x = z$ $(z \in \mathbb{Z})$
- domaine non relationnel
- si la valeur exacte n'est pas connue, perte de toute information



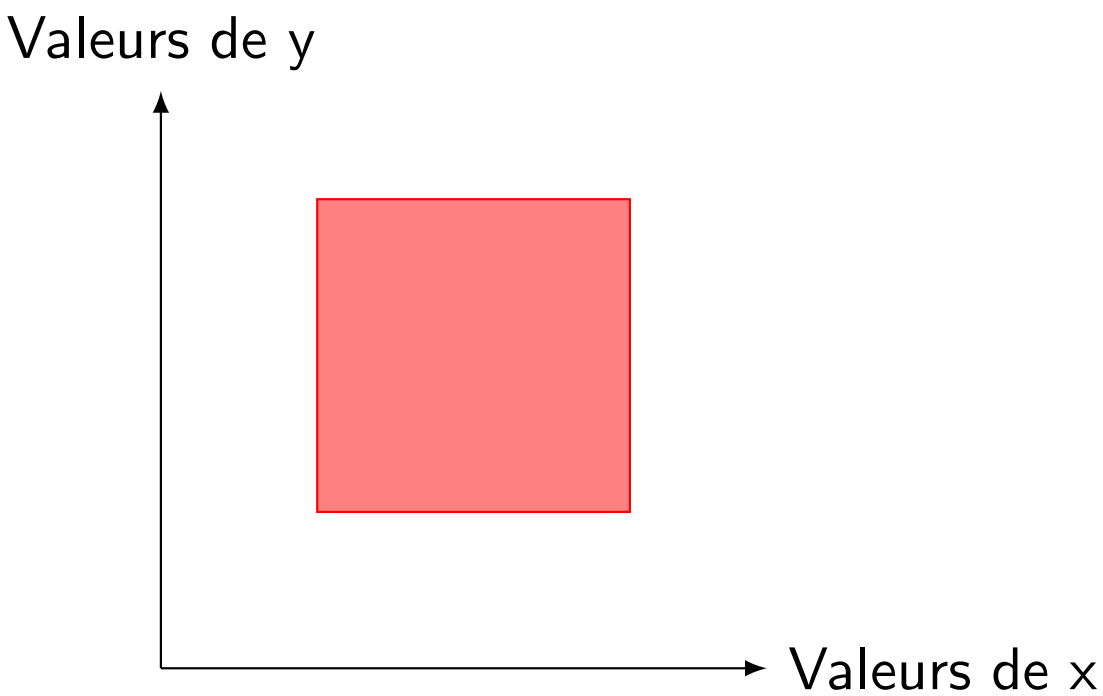
Domaine des signes

- $x \,\mathrm{op}\, 0$ $\mathrm{op} \in \{\geq, >, \leq, <, =, \neq\}$
- domaine non relationnel
- conservation de la polarité des valeurs possibles



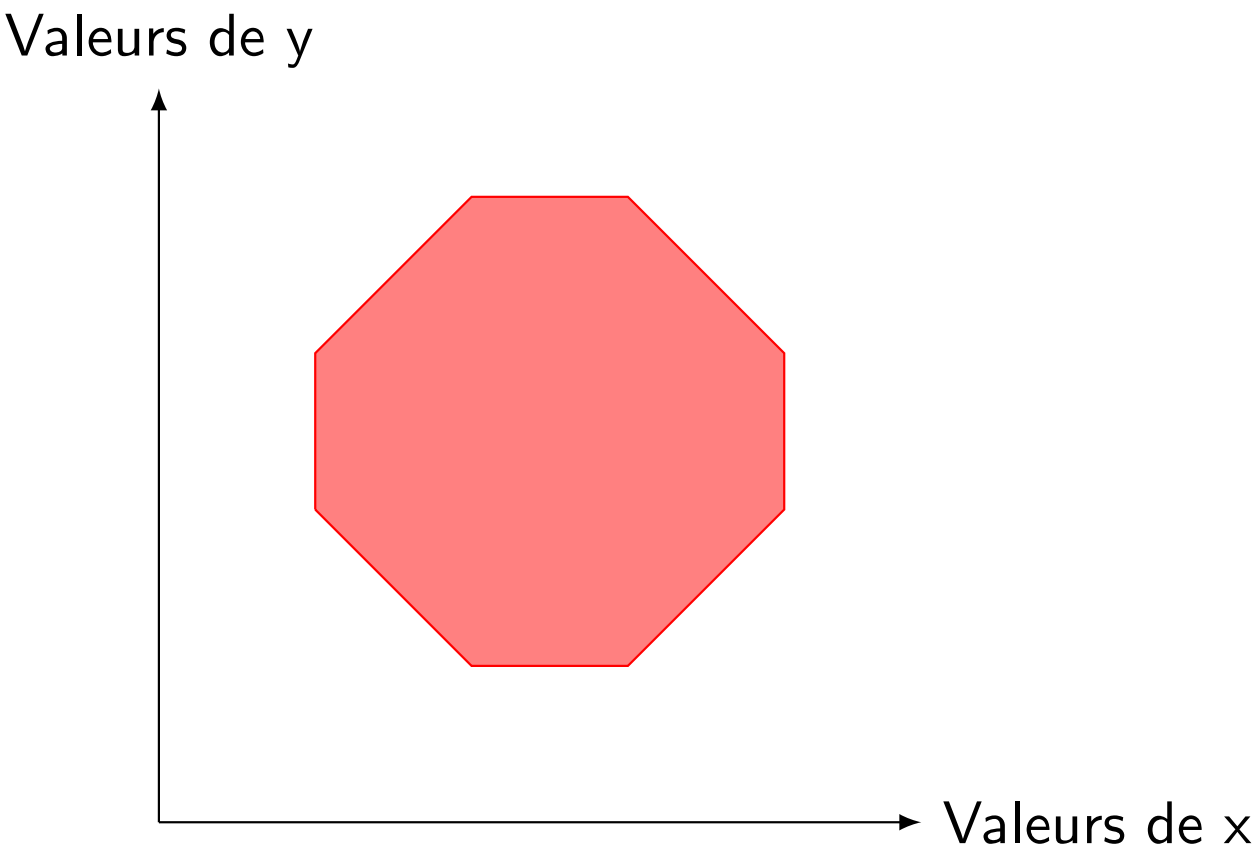
Domaine des intervalles

- $x \in [i_0, i_1]$
- domaine non relationnel
- conservation d'un intervalle regroupant toutes les valeurs possibles



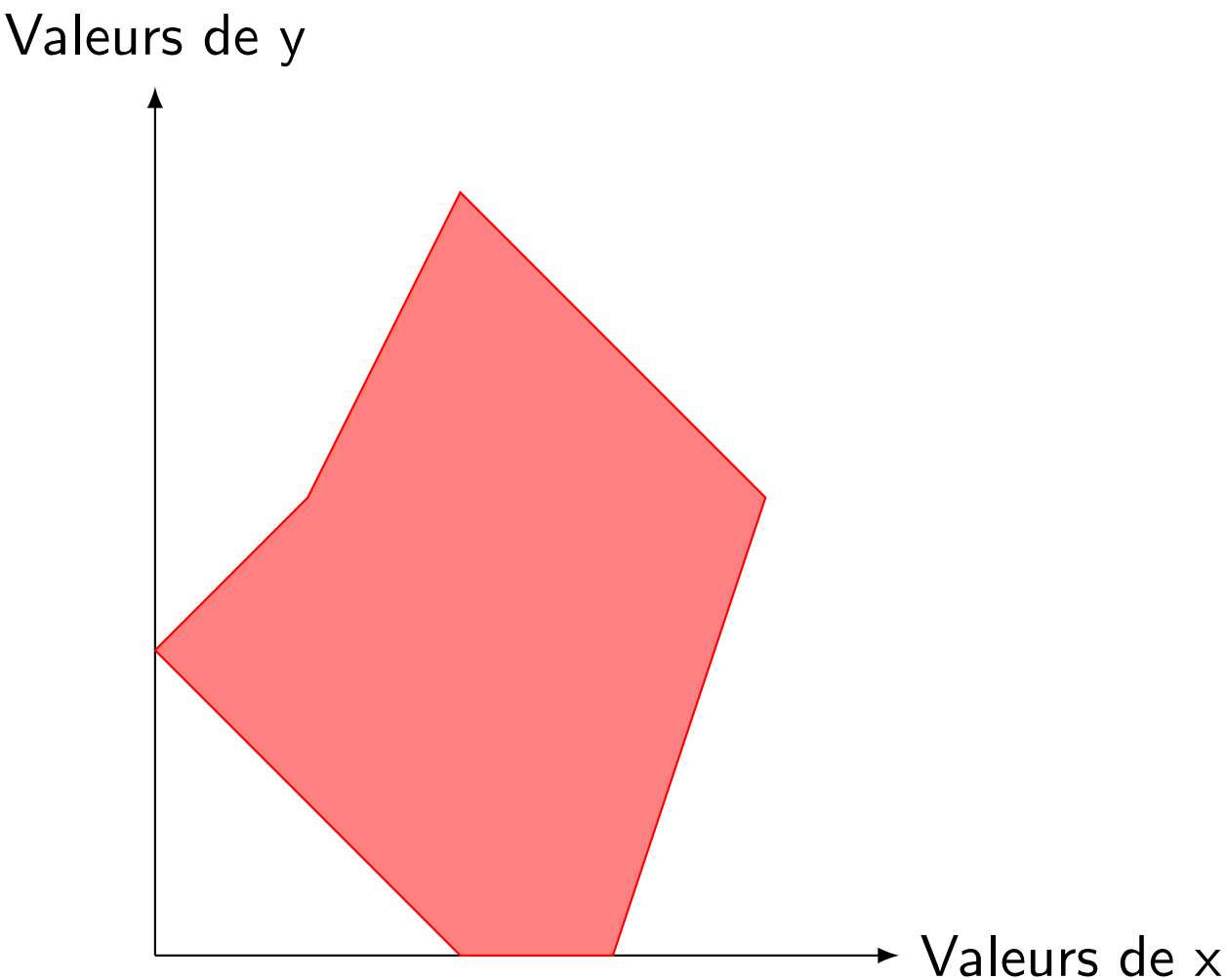
Domaine des octogones

- $\pm x \pm y \leq c$
- domaine relationnel
- conservation de relations linéaires simples entre élements



Domaine des polyèdres

- $kx + ly \leq c$
- domaine relationnel
- relations linéaires complexes entre deux éléments



Calcul de point fixe abstrait

Ingrédients

- domaine de valeurs symboliques D représentant des *ensembles* de configuration
  . $[\![]\!] : D \mapsto 2^Q$
- calculs symboliques sur D copiant les calculs ensemblistes sur 2^Q, en permettant une surapproximation
  . $\bot$ et $\top$ tels que $[\![\bot]\!] = \emptyset$ et $[\![\top]\!] = 2^Q$
  . $[\![d \sqcup d']\!] \supseteq [\![d]\!] \cup [\![d']\!]$
  . si $d \sqsubseteq d'$ alors $[\![d]\!] \subseteq [\![d']\!]$
  . si empty(d) alors $[\![d]\!] = \emptyset$
- calcul de post symbolique
  . $[\![post^{\#}(d)]\!] \supseteq post([\![d]\!])$

Avec ça : on fait le calcul d'accessibilité, version ensembliste

- pb1 : surapprox, ne peut conclure que yes/ ?
- pb2 : terminaison a priori pas garantie si chemins non bornés [mais possible]

Calcul de point fixe abstrait (2)

Calcul symbolique d'accessibilité (d_I tq $[\![d_I]\!]$ = états initiaux)

$d := d_I$
Tant que $post^{\#}(d) \not\sqsubseteq d$ **faire**
  $d := post^{\#}(d) \sqcup d$
Fin Tant que
retourner d

Pour aller plus loin

- correct si les opérations abstraites sont surapprochées
- interprétation abstraite : une connexion de Galois assure l'existence d'une meilleure approximation
- termine si le treillis n'a pas de chaine ascendante infinie
- widening pour assurer la terminaison
- industrie : ASTREE, SDV, Clousot, Absint, Polyspace, Frama-C, Fluctuat, etc.