Prouver les programmes : Les méthodes générales

Gérard Berry

Collège de France
Chaire Algorithmes, machines et langages
gerard.berry@college-de-france.fr

Cours 3, Paris, 11 mars 2015

Suivi du séminaire de Dominique Bolignano (Prove&Run)

COLLÈGE DE FRANCE
1530

Agenda

1. Les assertions et la terminaison
2. La réécriture
3. La sémantique dénotationnelle
4. L'interprétation abstraite
5. Logiques et assistants de preuve
   1. calcul des propositions et des prédicats
   2. assistants de preuve
   3. calcul des prédicats d'ordre supérieur
   4. calcul des constructions inductives

Cours du 18 mars

Agenda

1. Les assertions et la terminaison
2. La réécriture
3. La sémantique dénotationnelle
4. L'interprétation abstraite
5. Logiques et assistants de preuve
   1. calcul des propositions et des prédicats
   2. assistants de preuve
   3. calcul des prédicats d'ordre supérieur
   4. calcul des constructions inductives

Cours du 18 mars

Alan Turing, 1949*

In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.

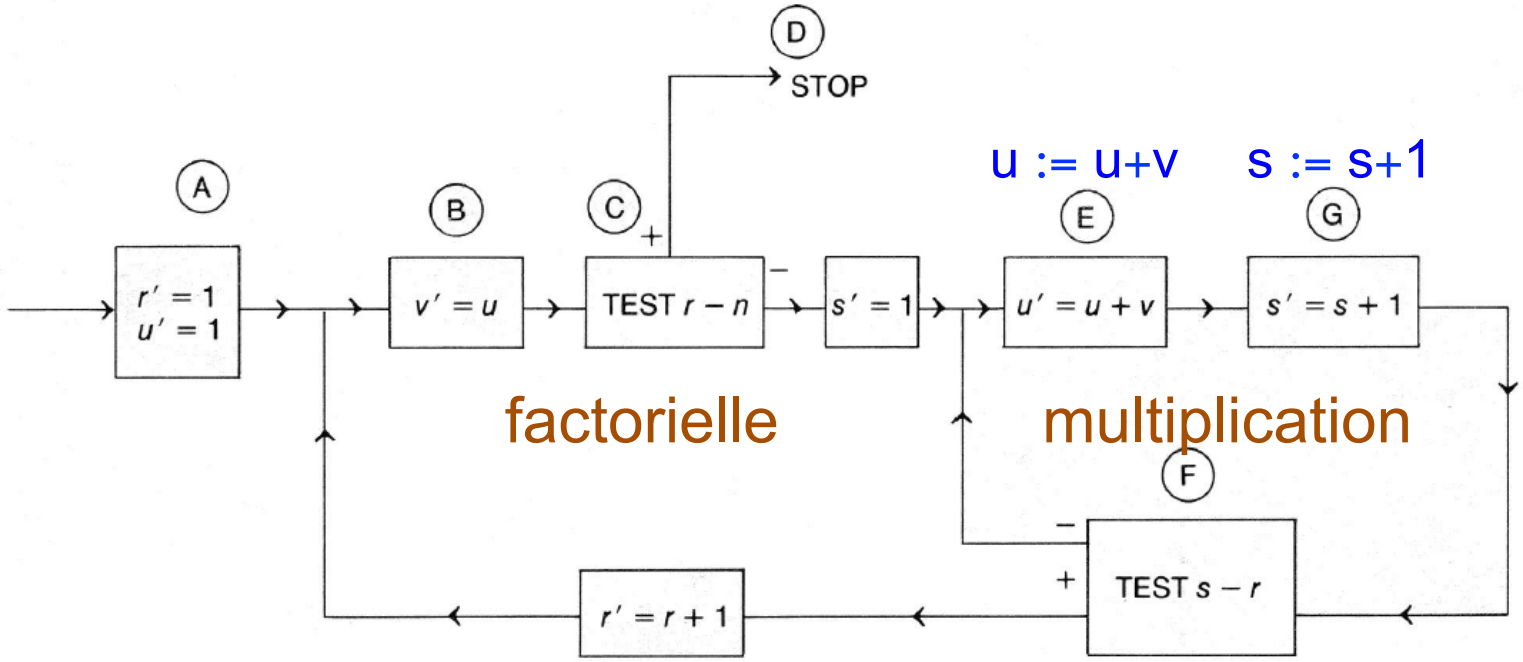


Table d'assertions / contrôle

STORAGE LOCATION	(INITIAL) (A) $k = 6$	(B) $k = 5$	(C) $k = 4$	(STOP) (D) $k = 0$	(E) $k = 3$	(F) $k = 1$	(G) $k = 2$
27					s	$s + 1$	s
28		r	r		r	r	r
29	n	n	n	n	n	n	n
30		$\lfloor r$	$\lfloor r$		$s \lfloor r$	$(s + 1) \lfloor r$	$(s + 1) \lfloor r$
31			$\lfloor r$	$\lfloor n$	$\lfloor r$	$\lfloor r$	$\lfloor r$
	TO (B) WITH $r' = 1$ $u' = 1$	TO (C)	TO (D) IF $r = n$ TO (E) IF $r < n$	↑ n!	TO (G)	TO (B) WITH $r' = r + 1$ IF $s \geq r$ TO (E) WITH $s' = s + 1$ IF $s < r$	TO (F)

Figure 2 (Redrawn from Turing's original)

Terminaison : décroissance d'un ordinal

Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops. To the pure mathematician it is natural to give an ordinal number. In this problem the ordinal might be $(n-r)\omega^2 + (r-s)\omega + k$. A less highbrow form of the same thing would be to give the integer $2^{80}(n-r) + 2^{40}(r-s) + k$. Taking the latter case and the step from B to C there would be a decrease from $2^{80}(n-r) + 2^{40}(r-s) + 5$ to $2^{80}(n-r) + 2^{40}(r-s) + 4$. In the step from F to B there is a decrease from $2^{80}(n-r) + 2^{40}(r-s) + 1$ to $2^{80}(n-r-1) + 2^{40}(r+1-s) + 5$.

On dit maintenant décroissance de l'ordinal lexicographique
< (*n* - *r*), (*r* - *s*), *k* > (ordre du dictionnaire)

Exemple : la fonction d'Ackermann

```
Ack (m,n) = if m=0 then n+1
                   else Ack (m-1, Ack (m, n-1))
```

- Intuition : Ack (m,n) = $F_m(n)$
  où F_m demande m récursions imbriquées:
  1 → addition, 2 → multiplication, 3 → exponentiation,
  4 → super-exponentiation, etc.

- Terminaison : ordre lexicographique bien fondé

  (m,n) < (m',n') ssi m < m' ou m = m' et n < n'

  pas de chaînes infinies descendantes (essayez!)

  alors (m,n-1) < (m,n), donc l'appel interne se termine,
  et (m-1, Ack (m, n-1)) < (m,n), donc l'appel externe se termine

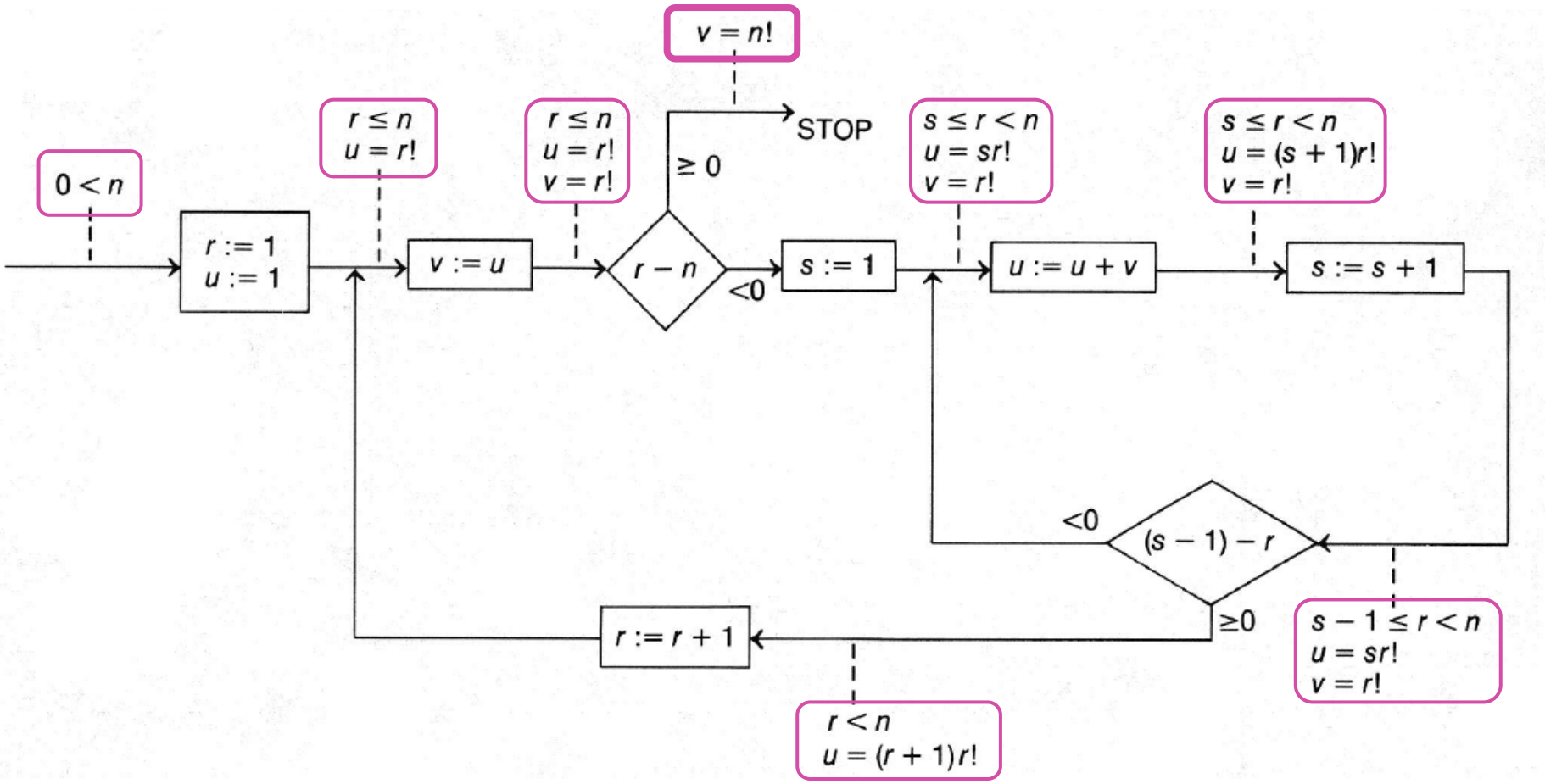
Que peut-on faire pour la terminaison?

- Elle est indécidable en général (Turing, 1936)
- Elle donne lieu à de jolis problèmes ouverts (Collatz)

  f(n) = si n =1 alors 1 sinon
  si pair(n) alors f(n/2) sinon f(3n+1) = 1 ?

- Mais il existe de bonnes restrictions assurant la terminaison (récursion primitive, types d'ordre supérieur, etc.)
- Et de bons logiciels pour trouver des ordres bien fondés
  Terminator / T2, B. Cook *et. al.* (Microsoft)
  application aux drivers de périphériques

La factorielle de Turing vue par Floyd (1967)*

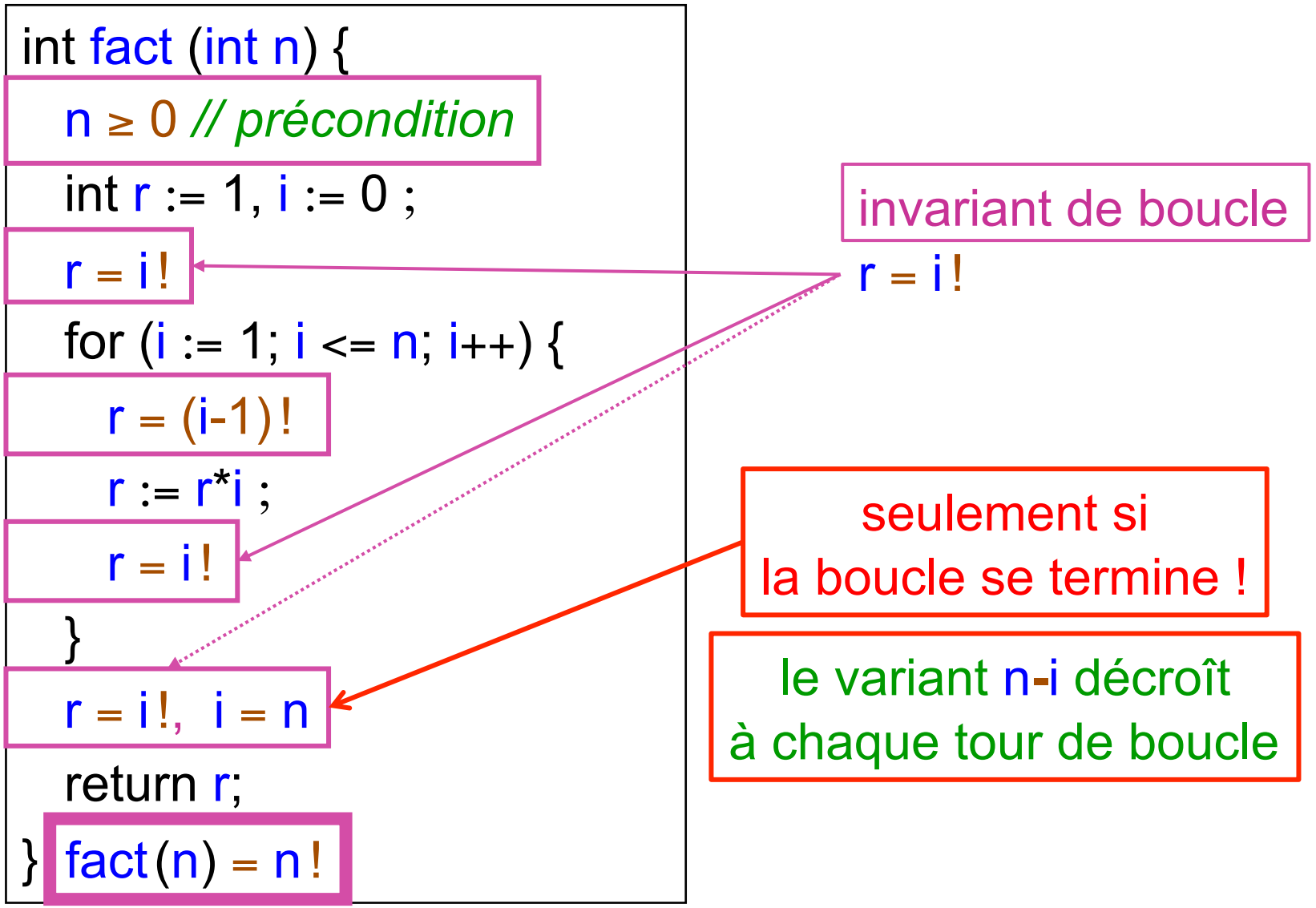


L'objectif n'est pas seulement la vérification. C'est aussi
la définition formelle du langage de programmation
à l'aide des assertions. L'un ne va pas sans l'autre !

Les logiques de programmes

- Variables et fonctions : x, y, f(x, y), etc.
- Instructions : if x > 0 then x := x-1; y := y-1 end
- Prédicats élémentaires : $P(x) =_{def} (x = 1)$
- Ensembles (ou listes) de prédicats : P, x = 1, y > x-z
- Calcul propositionnel : $(x = 1) \vee \neg(x > 2 \wedge y \geq x)$
  $(x = 1 \wedge y > x\text{-}z) \Rightarrow (y > 5)$
- Calcul des prédicats : $\forall x > 2.\ \exists y, z.\ (x = f(y,z))$

Encore la factorielle (cours du 22/02/2008)



Utilisation des assertions locales

- Vérification des assertions lors de l'exécution
  → extrêmement utile aussi pour les tests !
- Vérification par transformation systématique des programmes en formules logiques
- Vérification formelle (statique) par moteurs automatiques ou assistants logiques + prouveurs de terminaison
- Compatibilité avec d'autres approches sémantiques, cf. interprétation abstraite plus loin

Une méthode de choix pour la programmation impérative séquentielle, pas encore pour la programmation parallèle (Separation Logic ?)

Tony Hoare 1969 : la logique des assertions

{P} S {Q} : si P est vrai avant S et si S se termine, alors Q est vrai après S

$$\frac{}{\{P\}\ \text{skip}\ \{P\}} \qquad \frac{}{\{P[E/x]\}\ x := E\ \{P\}}$$

$$\frac{\{P\}\ S\ \{Q\} \quad \{Q\}\ T\ \{R\}}{\{P\}\ S;T\ \{R\}} \qquad \frac{\{P \wedge B\}\ S\ \{Q\} \quad \{(P \wedge \neg B)\}\ T\ \{Q\}}{\{P\}\ \text{if } B \text{ then } S \text{ else } T\ \{Q\}}$$

$$\frac{\{P \wedge B\}\ S\ \{P\}}{\{P\}\ \text{while } B \text{ do } S\ \{P \wedge \neg B\}} \qquad \frac{P' \Rightarrow P \quad \{P\}\ S\ \{Q\} \quad Q \Rightarrow Q'}{\{P'\}\ S\ \{Q'\}}$$

Ce qui compte, ce n'est pas ce que la boucle change, c'est ce qu'elle laisse invariant

E. W. Dijkstra 1976 : commandes gardées

- Algorithme d'Euclide pour pgcd(X,Y)

```
x := X ; y := Y
do
   x > y → x := x-y
 | y > x → y := y-x
od
```

déterminisme car un seul choix possible

- Tri de 4 nombres Q1, Q2, Q3, Q4

```
q1 := Q1 ; q2 := Q2 ; q3 := Q3 ; q4 := Q4 ;
do
   q1 > q2 → q1, q2 := q2, q1
 | q2 > q3 → q2, q3 := q3, q2
 | q3 > q4 → q3, q4 := q4, q3
od
```

non-déterminisme

mais résultat déterministe

Spécifier le contrôle de façon minimale

E. W. Dijkstra 1976 : weakest precondition

wp(S, Q) = Q' ssi Q' est le prédicat le plus général tel que S vérifie Q s'il se termine

- Permet de réduire la preuve d'assertions à un problème standard de calcul des prédicats :

  $\{P\}\ S\ \{Q\} \Leftrightarrow (P \Rightarrow wp(S, Q))$

- Facile à définir :

  wp(skip, Q) = Q

  wp(abort, Q) = false // {P} abort {Q} ⇔ P = false

  wp(x := E, Q) = Q[E/x]

  exemple : wp(x := x+1, x<5) = x+1<5 ⇔ x<4

- $wp(S;T, Q) = wp(S, wp(T, Q))$
- $wp(\text{if } B \text{ then } S \text{ else } T \text{ end}, Q) =$
  $(B \Rightarrow wp(S, Q)) \wedge (\neg B \Rightarrow wp(T, Q))$
- pour IF = if $B_1 \rightarrow S_1$ ▯ $B_2 \rightarrow S_2$ ▯ ... ▯ $B_n \rightarrow S_n$ fi
  $wp(IF, Q) = \exists\, i \leq n.\ B_i \ \wedge\ (\forall\, i \leq n.\ B_i \Rightarrow wp(S_i, Q))$
- $wp(\text{while } B \text{ do } S, Q)_I =$ I
  $\wedge\ (B \wedge I \Rightarrow wp(S, I))$
  $\wedge\ (\neg B \wedge I \Rightarrow Q)$

invariant à choisir

Peut s'étendre à la terminaison en rajoutant un variant <,
i.e. une relation d'ordre bien fondé sur l'état
telle que l'état décroit pour < à chaque tour de boucle

Des assertions aux contrats

- Pourquoi se contenter de la vérification? Utilisons les assertions comme moyen majeur de spécifier, programmer et documenter plus sûrement !

- Contrat = assume / guarantee modulaire
  A toute fonction (objet, module) F on associe :
  - des hypothèses (assume) qui doivent être satisfaite par les entrées
  - des garanties (guarantee) alors satisfaite par les sorties
  - tellement mieux que les commentaires non vérifiables !

- La programmation ne peut plus se faire « à l'arrache »
- La vérification locale assume ⇒ guarantee modularise la vérification globale (lemmes)

Eiffel (B. Meyer), B/Event-B (J-R. Abrial), Spec# (R. Leino), etc.

Agenda

1. Les assertions et la terminaison
2. La réécriture
3. La sémantique dénotationnelle
4. L'interprétation abstraite
5. Logiques et assistants de preuve
   1. calcul des propositions et des prédicats
   2. assistants de preuves
   3. calcul des prédicats d'ordre supérieur
   4. calcul des constructions inductives

Cours du 18 mars

John McCarthy, 1963

A Basis For a Mathematical Theory of Computation
Computer Programming and Formal Systems, North-Holland (1963)

Computation is sure to become one of the most important of the sciences. This is because it is the science of how machines can be made to carry out intellectual processes. We know that any intellectual process that can be carried out mechanically can be performed by a general purpose computer.

Moreover, the limitations on what we have been able to make computers do so far come far more from our weakness as programmers than from the intrinsic limitations of the machines. We hope that these limitations can be greatly reduced by developing a mathematical theory of computation.

... We are not yet able to discuss formal proofs of convergence

Des définitions encore balbutiantes (des pages de définitions et d'exemples)

- Variables, fonctions, expressions, prédicats, conditionnelles, récursion simple ou multiple
  n! = (n=0 → 1, T → n · (n-1))
- Un peu de typage: f : U_1, U_2,..., U_n → V
- Evaluation par valeur, sauf pour la conditionnelle, arrêtée au premier test vrai
- Obligation de considérer les fonctions partielles (non-terminaison)
- Fonctionnelles (fonctions de fonctions), λ-notation (Church)

  label(fact, λ((n), (n=0 → 1, T → n · fact (n-1))))

  point fixe (à la Y) mentionné mais rejeté comme lourd
- Quantificateurs simples ou fonctionnels, ambiguïté, etc.

Recursion Induction : la première vraie règle de preuve

Langage : booléens T, F, variables, expressions,
conditionnelle $b_1 \rightarrow e_1$,..., $b_n \rightarrow e_n$

Entiers et Opérateurs : 0, m' (successeur), m^- (prédécesseur)

Règle de calcul : appel par valeur
conséquence : $f(b_1 \rightarrow e_1, ..., b_n \rightarrow e_n) = b_1 \rightarrow f(e_1), ..., b_n \rightarrow f(e_n)$

Définitions par réécriture :

$m + n = (n=0 \rightarrow m , T \rightarrow (m' + n^-))$

$m * n = (n=0 \rightarrow 0 , T \rightarrow (m + (m * n^-)))$

Si f se termine sur un ensemble A
et si g et h satisfont l'équation de f, alors g = h sur A

Théorème 1 : $(m + n)' = m' + n$

Preuve : Posons $f(m,n) = (n=0 \rightarrow m' , T \rightarrow f(m',n^-)')$

- Par récurrence sur n, f termine toujours pour $m,n \in N$.

- Posons $g(m,n) = (m + n)'$. Alors

$g(m,n) = (n=0 \rightarrow m , T \rightarrow (m' + n^-))'$ // définition de +

$= (n=0 \rightarrow m' , T \rightarrow (m' + n^-)')$ // appel par valeur

$= (n=0 \rightarrow m' , T \rightarrow g(m', n^-))$ // définition de g

donc g vérifie l'équation de f.

- Posons $h(m,n) = m' + n$. Alors

$h(m,n) = (n=0 \rightarrow m' , T \rightarrow (m'' + n^-))$ // définition de +

$= (n=0 \rightarrow m' , T \rightarrow h(m', n^-))$ // définition de h

donc h vérifie l'équation de f.

Donc $g = h$, CQFD

Le choix des axiomes

Théorème 2 : $(m + n) + p = (m + p) + n$
preuve : idem par *recursion induction*

Théorème 3 : $m + 0 = m$
pas montrable directement par *recursion induction*...

Axiomes supplémentaires (Peano moins la récurrence) :
$m' \neq 0$
$(m')^- = m$
$(m \neq 0) \Rightarrow (m^-)' = m$

Le seul mécanisme de preuve est la réécriture
Trouver les bons axiomes et les bonnes règles
est le problème principal

Les descendants de McCarthy

Vérification par calcul symbolique + numérique
et algorithmes de décision spécialisés

- Boyer & Moore (U. Texas) → ACL-2
  Maude (Goguen), REVE (Kirchner et. al.), PVS (SRI)
- Algorithmes d'unification, de Knuth-Bendix pour les formes normales, de Shostak pour le mélange de théories etc.
- Clauses de Horn, SLD-résolution (Robinson)
  Prolog (Colmerauer), Prolog 2
- Programmation par contraintes (Saraswat, Fages, etc.)
- Vérification de protocoles de sécurité : ProVerif (Blanchet)
- ...

Difficulté: difficile de savoir quand ça marche
et ce qu'il faut faire quand ça ne marche pas!

Agenda

1. Les assertions et la terminaison
2. La réécriture
3. La sémantique dénotationnelle
4. L'interprétation abstraite
5. Logiques et assistants de preuve
   1. calcul des propositions et des prédicats
   2. assistants de preuves
   3. calcul des prédicats d'ordre supérieur
   4. calcul des constructions inductives

Cours du 18 mars

La sémantique dénotationnelle

- P. Landin, 1966 : ISWIM = If You See What I Mean

  λ-calcul + fondations sémantiques

- D. Scott*, 1970 : D^{∞}, le premier modèles du λ-calcul pur
- D. Scott*, C. Strachey : sémantique dénotationnelle par le

  λ-calcul et la logique de Scott

- G. Plotkin, 1970 : PCF et ses modèles

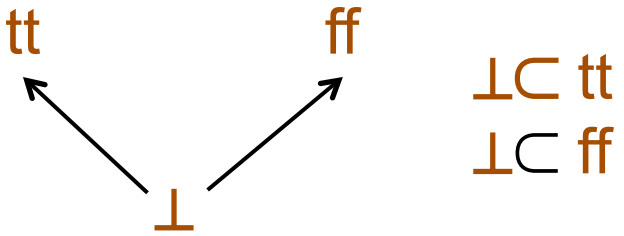
→ G. Berry, P-L. Curien, A. Stoughton, S. Abramsky, ... :

La théorie de la programmation séquentielle : C, ML, etc.

Cf. cours des 2 et décembre 2009

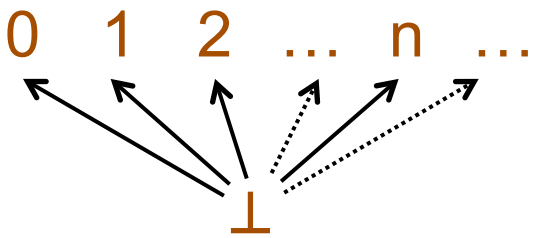
*Les domaines de Scott**

Ordre d'information : $x \subset y \leftrightarrow$ x a moins d'information que y



$O_\perp$: espace de Sierpinski

$B_\perp$: booléens de Scott



$\mathbf{N}_\perp$: entiers de Scott

Fonctions totales et croissantes

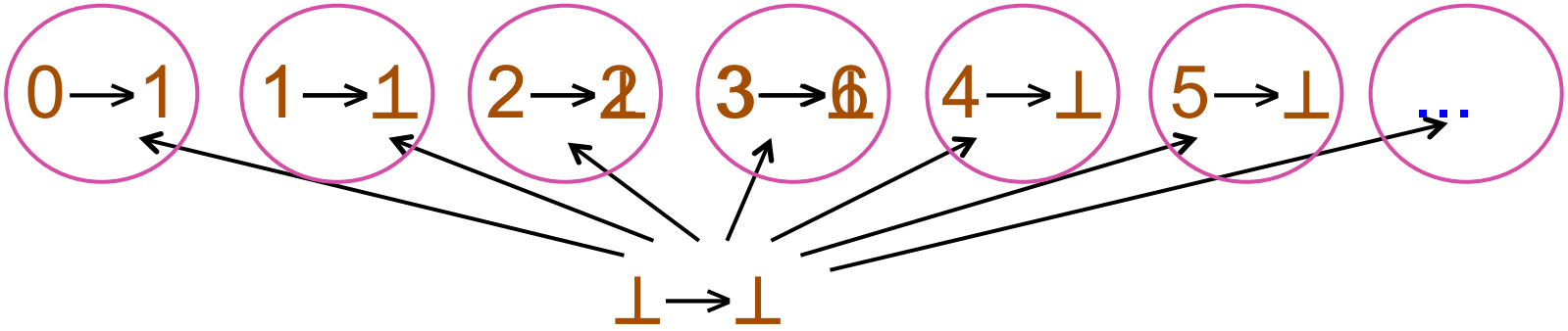
- Plus on en donne, plus on en récupère !

  f : <D,⊂,⊥> → <D',⊂',⊥'> croissante

  ssi ∀x, y. x ⊂ y ⇒ f(x) ⊂' f(y)

- Calcul itératif jusqu'à point fixe

  fact(n) = si n=0 alors 1 sinon n * fact(n-1)



Fixpoint et Fixpoint Induction

Théorème du point fixe : toute fonction $f : D \to D$ continue sur un domaine de Scott a un plus petit point fixe $Fix(f)$ t.q. $f(Fix(f)) = Fix(f)$ et $\forall x \in D.\ f(x) = x \Rightarrow Fix(f) \subset x$

Principe de récurrence de point fixe (*fixpoint induction*) :

Soit $f : D \to D$ continue. Soit $E \subset D$ complet. i.e. contenant $\bot$ et toutes les limites de ses suites croissantes.
Si $\forall x.\ x \in E \Rightarrow f(x) \in E$ alors $Fix(f) \in E$

Proposition : si $f(x) \subset x$ alors $Fix(f) \subset x$

Preuve : l'ensemble $\downarrow x = \{y \mid y \subset x\}$ est complet.
Soit $y \in \downarrow x$. Alors $f(y) \subset f(x) \subset x$.
Donc $Fix(f) \in \downarrow x$ par *fixpoint induction*, i.e. $Fix(f) \subset x$

Un résultat surprenant

Proposition 2 : si $f(\bot) = g(\bot)$ et $f \circ g \subset g \circ f$ alors $Fix(f) \subset Fix(g)$

preuve : l'ensemble $E = \{ x \mid f(x) \subset g(x) \}$ est complet.
Si $x \in E$ alors $f(x) \subset g(x) \Rightarrow f(g(x)) = g(f(x)) \subset g(g(x)) \Rightarrow g(x) \in E$
Donc $Fix(g) \in E$ par *fixpoint induction,* c'est à dire
$f(Fix(g)) \subset g(Fix(g)) = Fix(g)$, ce qui entraîne $Fix(f) \subset Fix(g)$
par la proposition 1

Corollaire (!) : si $f(\bot) = g(\bot)$ et $f \circ g = g \circ f$ alors $Fix(f) = Fix(g)$
preuve : par la proposition, $Fix(f) \subset Fix(g)$ et $Fix(g) \subset Fix(f)$

Agenda

1. Les assertions et la terminaison
2. La réécriture
3. La sémantique dénotationnelle
4. L'interprétation abstraite
5. Logiques et assistants de preuve
   1. calcul des propositions et des prédicats
   2. assistants de preuves
   3. calcul des prédicats d'ordre supérieur
   4. calcul des constructions inductives

Cours du 18 mars

Un bug spatial

- Explosion d'Ariane 501, 4 juin 1996
  - overflow non protégé dans une conversion fp32→ int16
  - les 2 gyrolasers se déclarent en panne → plus de contrôle
  - ce code ne servait à rien
  - tout fonctionnait normalement!
  - Ariane 5 avait été simulée sur la trajectoire d'Ariane 4, ...

L'interprétation abstraite : assertions + domaines de Scott + abstraction

- Idée initiale de M. Sintzoff en 1972 pour les signes
  - abstraction des nombres par leur signe +, -, ⊤
  - utiliser les lois de Brahmagupta (~630) :
    addition : (+,+)→+ (-,-)→- (+,-)→⊤ (-,+)→⊤
    multiplication : (+,+)→+ (-,-)→+ (+,-)→- (-, +)→-
    division : idem sauf si le dénominateur est nul
  - voir aussi la preuve par 9

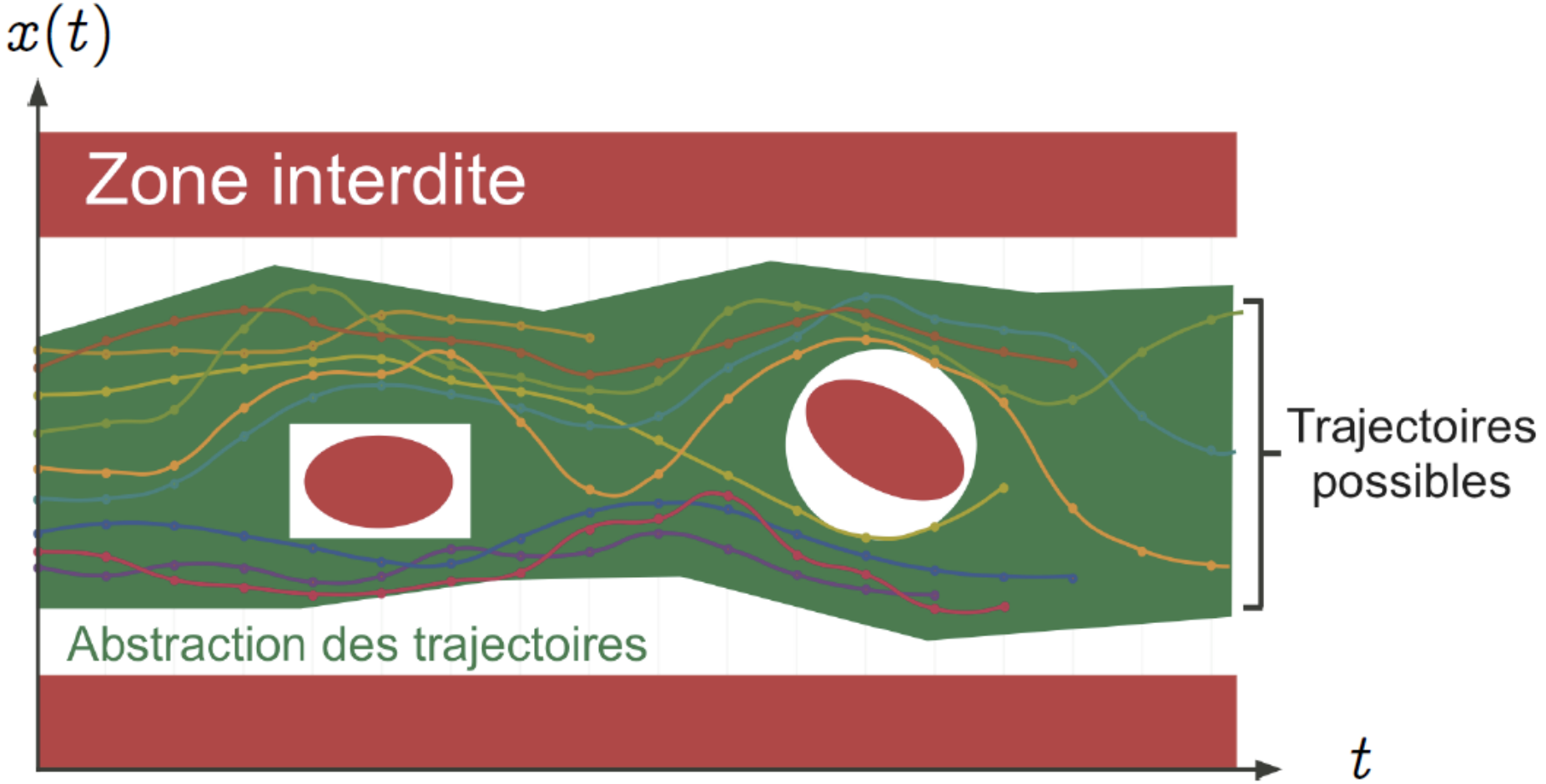
- P. et R. Cousot, 1977 : interprétation abstraite
  - théorie générale d'approximations de données, fondée sur Scott
  - théorie générale de la composition d'approximations
  - algorithmes efficaces de calcul des approximations finies

Cf. séminaire de Patrick Cousot dans mon cours 2007-2008,
http://www.college-de-france.fr/site/gerard-berry/seminar-2008-02-22-11h30.htm

Construction de domaines abstraits (cf séminaire de P. Cousot, 22/02/2008)

- Intervalles : $x \in [a,b]$
  pas d'erreur run-time dans y/x si a et b sont du même signe
  ordre : inclusion inverse $[a,b] \sqsubset [a',b']$ ssi $a \leq a'$ et $b' \leq b$
- Octogones : $x-y \leq a$, $x+y \leq b$
- Polyèdres : $ax+by \leq b$
- Ellipsoïdes : $(x-a)^2 + (y-b)^2 \leq c$
- Exponentielles : $a^x \leq y$
- Ordres de Scott sur les domaines, points fixes
- Théorie générale de la composition de domaines
  connexions de Galois
- Théorie et pratique de l'accélération de points fixes

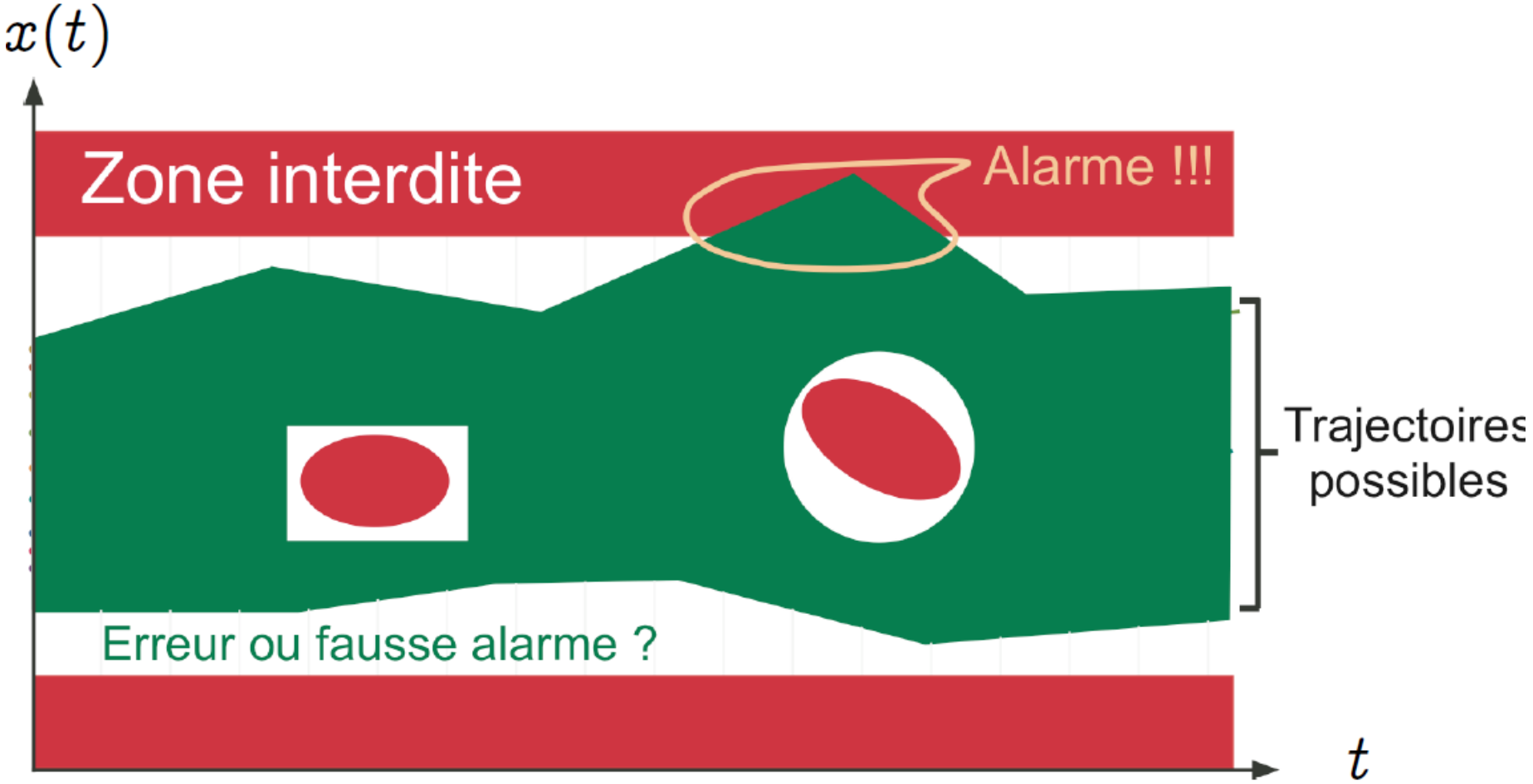
L'interprétation abstraite est correcte



$$\text{Sémantique}[\![P]\!] \subseteq \text{Abstraction}(\text{Sémantique}[\![P]\!])$$

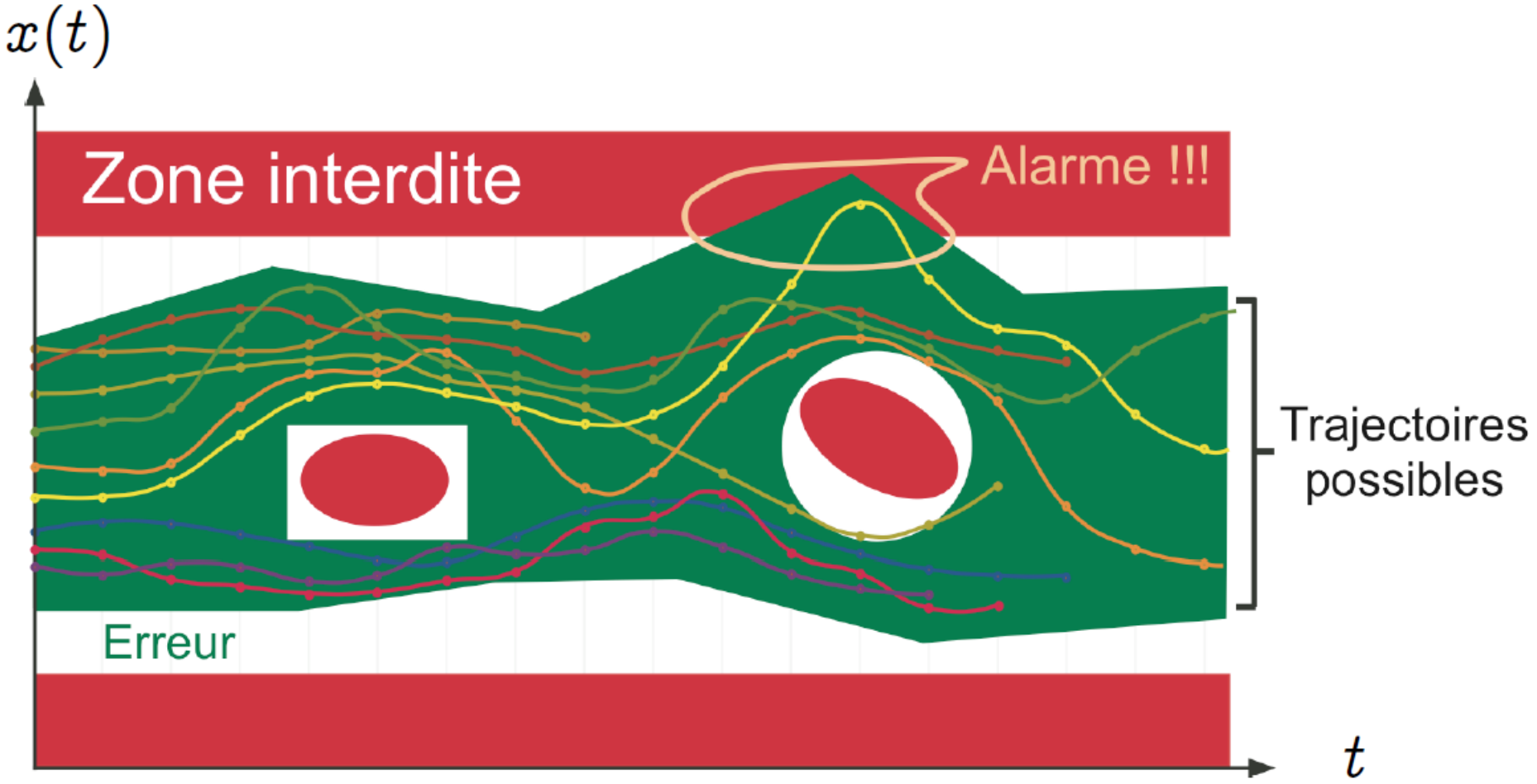
séminaire de P. Cousot, 22/02/2008

Alarme



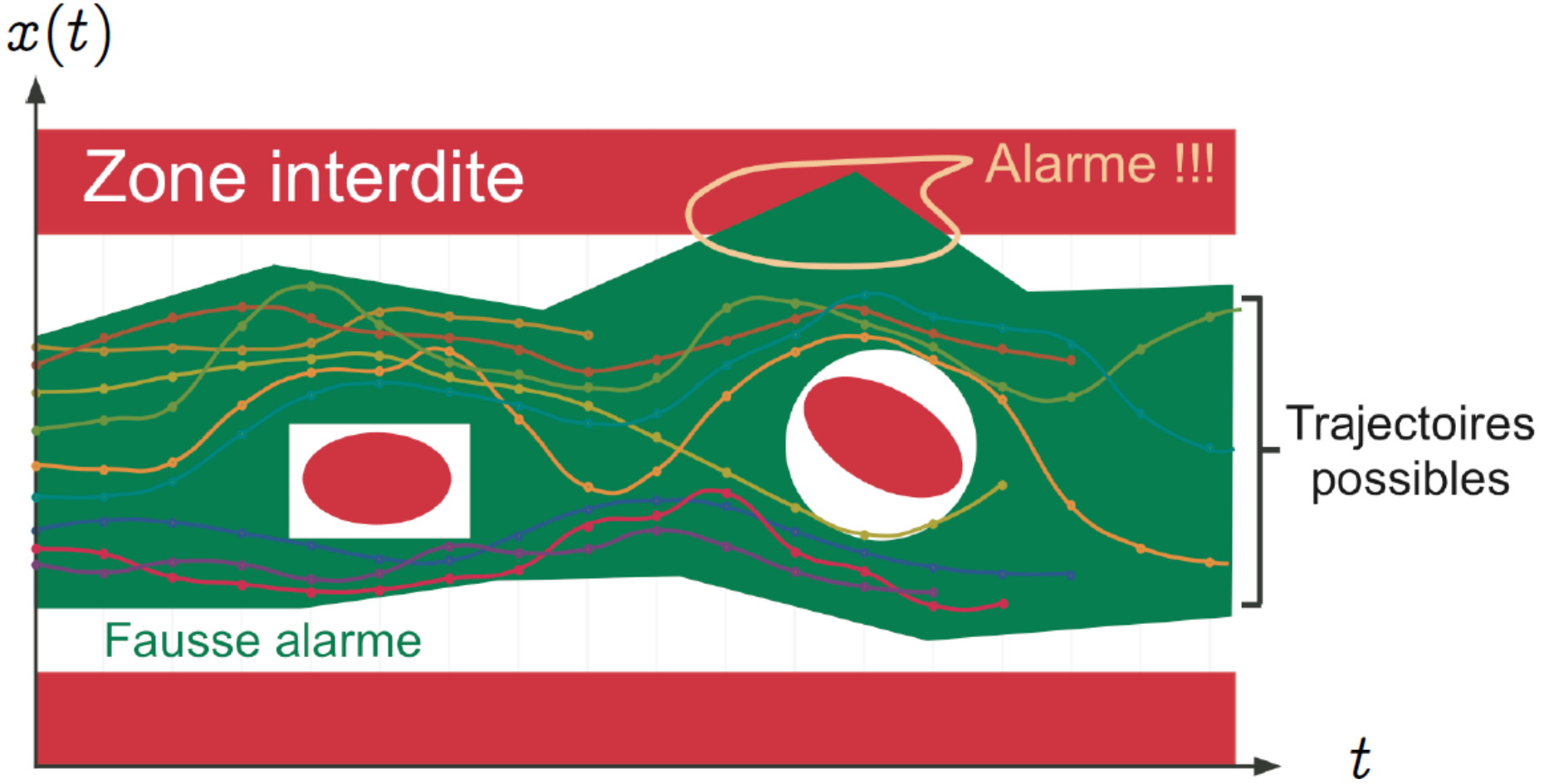
séminaire de P. Cousot, 22/02/2008

Une alarme peut correspondre à une erreur



séminaire de P. Cousot, 22/02/2008

Une alarme peut correspondre à une approximation



séminaire de P. Cousot, 22/02/2008

L'interprétation abstraite : devenue industrielle, applications majeures

- Syntox : F. Bourdoncle, sur Pascal
- Ariane 501 → Polyspace : A. Deutsch, D. Pilaud, 1999
  → The Mathworks, 2007 : automobile etc.
- Absint : R. Wilhelm, C. Ferdinand (Sarrebrücken)
  → WCET (Worst Case Execution Time)
  abstraction d'états des caches et pipelines HW
  → taille maximale de pile à l'exécution
  Airbus, avionique, automobile, etc.
- Astrée (ENS), P. et R. Cousot, J. Feret, A. Miné, X. Rival *et. al.*
  code de vol de l'Airbus A380 → AbsInt
- Spec# (Microsoft) : analyses fines à la compilation
- Ailleurs : analyses statiques un peu partout

Agenda

1. Les assertions et la terminaison
2. La réécriture
3. La sémantique dénotationnelle
4. L'interprétation abstraite
5. Logiques et assistants de preuve
   1. calcul des propositions et des prédicats
   2. assistants de preuves
   3. calcul des prédicats d'ordre supérieur
   4. calcul des constructions inductives

Cours du 18 mars

Le calcul des propositions

- Propositions élémentaires : A, B, C, vrai (⊤) , faux (⊥), ...
- Formules P construites avec des connecteurs ∧, ∨, ¬, ⇒, ⇔
- Modèle booléen :
  - une proposition est vraie (V) ou fausse (F)
  - sens des connecteurs = fonctions définies par des tables de vérités

⊤	V
⊥	F

∧	V	F
V	V	F
F	F	F

∨	V	F
V	V	V
F	V	F

¬	
V	F
F	V

⇒	V	F
V	V	F
F	V	V

⇔	V	F
V	V	F
F	F	V

- satisfiabilité (SAT) : F peut-elle être rendue vraie ?
- Tautologie : vraie pour toutes valeurs des propositions, ⊨ F

  exemples : ((P ⇒ Q) ∧ (Q ⇒ R)) ⇒ (P ⇒ R)

  (P ⇒ Q) ⇒ ((Q ⇒ R) ⇒ (P ⇒ R))

Le calcul des propositions : utilisations

- Fondamental partout !
  - base de tout raisonnement logique
  - base directe des circuits électroniques
  - au cœur de la plupart des systèmes de model-checking
  - SAT est NP-complet : permet de coder tous les problèmes NP
    pas d'algorithme polynomial connu

- Mais encore bien mystérieux (alchimique ?)...
  - décidable, mais longtemps pensé impossible à grande échelle
  - mais BDDs de R. Bryant en 1986 (cf cours 6, 1e avril 2015)
  - puis algorithmes SAT de plus en plus performants, cf. cours 2016.

Question ouverte : quand les BDD / SAT marchent-ils bien ?

Le calcul des prédicats

- constantes, opérations, variables et fonctions dans les formules élémentaires

  x+1 f(x)+g(h(x))

- quantificateurs ∀ et ∃ sur les variables

  ∀x, y. ∃z. (x+z = y)

- théories axiomatiques sur les objets

  entiers → axiomes de Peano

  ensembles → axiomes de Zermelo-Fraenkel

  etc.

- indécidable en général → assistants interactif + heuristiques

Base de B et Event-B (Abrial), TLA+ (Lamport), etc.

Une session d'assistance (en Coq)

```
Variables P Q R : Prop.
Theorem Imp_trans : (P→Q) → ((Q→R) → (P→R)).
```

Convention : → associe à droite

```
Theorem Imp_trans : (P→Q) → (Q→R) → P → R.
```

Une session d'assistance (en Coq)

Variables P Q R : Prop.
Theorem Imp_trans : (P→Q) → (Q→R) → P → R.

Proof.

1 subgoal

(P→Q) → (Q→R) → P → R

H1 H2 H3

Une session d'assistance (en Coq)

```
Variables P Q R : Prop.
Theorem Imp_trans : (P→Q) → (Q→R) → P → R.

Proof.
intros H1 H2 H3.
```

1 subgoal

H1 : P→Q
H2 : Q→R
H3 : P

R

H1 H2 H3

Une session d'assistance (en Coq)

```
Variables P Q R : Prop.
Theorem Imp_trans : (P→Q) → (Q→R) → P → R.

Proof.
intros H1 H2 H3.
apply H2.
```

1 subgoal

H1 : P→Q
H2 : Q→R
H3 : P

Q

Une session d'assistance (en Coq)

Variables P Q R : Prop.
Theorem Imp_trans : (P→Q) → (Q→R) → P → R.

Proof.
intros H1 H2 H3.
apply H2.
apply H1.

1 subgoal

H1 : P→Q
H2 : Q→R
H3 : P

P

Une session d'assistance (en Coq)

```
Variables P Q R : Prop.
Theorem Imp_trans : (P→Q) → (Q→R) → P → R.

Proof.
intros H1 H2 H3.
apply H2.
apply H1.
assumption.
```

1 subgoal

H1 : P→Q

H2 : Q→R

H3 : P

P

Une session d'assistance (en Coq)

Variables P Q R : Prop.
Theorem Imp_trans : (P→Q) → (Q→R) → P → R.

Proof.
intros H1 H2 H3.
apply H2.
apply H1.
assumption.
Qed.

no more subgoals

H1 : P→Q
H2 : Q→R
H3 : P

P

Déduction : système de Gentzen

Γ = Q, R, ... contexte = ensemble (ou liste) de formules

séquent « $\Gamma \vdash P$ » : P se démontre sous les hypothèses Γ

$$\frac{}{\Gamma \vdash \top} \qquad \frac{P \in \Gamma}{\Gamma \vdash P}$$

introduction	élimination
$\dfrac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q}$	$\dfrac{P \wedge Q \in \Gamma \quad \Gamma, P, Q \vdash R}{\Gamma \vdash R}$
$\dfrac{\Gamma \vdash P}{\Gamma \vdash P \vee Q} \qquad \dfrac{\Gamma \vdash Q}{\Gamma \vdash P \vee Q}$	$\dfrac{P \vee Q \in \Gamma \quad \Gamma, P \vdash R \quad \Gamma, Q \vdash R}{\Gamma \vdash R}$
$\dfrac{\Gamma, P \vdash Q}{\Gamma \vdash P \Rightarrow Q}$	$\dfrac{P \Rightarrow Q \in \Gamma \quad \Gamma \vdash P \quad \Gamma, Q \vdash R}{\Gamma \vdash R}$

Le faux et la négation

$$\frac{\bot \in \Gamma}{\Gamma \vdash P}$$ *tout se prouve à partir du faux*

logique intuitionniste, cf cours du 26/03/2014

$$\frac{\Gamma, P \vdash \bot}{\Gamma \vdash \neg P}$$ *négation intuitionniste* (*alternative* $\neg P =_{def} P \Rightarrow \bot$)

Attention : de $\neg(P \wedge Q)$ on ne peut pas déduire $\neg P \vee \neg Q$

$$\frac{\Gamma, \neg P \vdash \bot}{\Gamma \vdash P}$$ *preuve par l'absurde*

$$\Leftrightarrow \quad \frac{}{\Gamma \vdash P \vee \neg P}$$ *tiers exclu*

logique classique

La quantification

$$\frac{\Gamma \vdash P(x) \quad x \text{ non libre dans } \Gamma}{\Gamma \vdash \forall x.\, P(x)} \qquad \frac{\forall x.\, P(x) \in \Gamma \quad \Gamma,\, P(t/x) \vdash Q}{\Gamma \vdash Q}$$

$$\frac{\Gamma \vdash P(t/x)}{\Gamma \vdash \exists x.\, P(x)} \qquad \frac{\exists x.\, P(x) \in \Gamma \quad P(x) \vdash Q \quad x \text{ non libre dans } \Gamma, Q}{\Gamma \vdash Q}$$

- Classique : $\exists x.\, P(x) \Leftrightarrow \neg\, (\forall x.\, \neg\, P(x))$
  = raisonnement par l'absurde

- Intuitionniste : je n'accepte pas ce raisonnement !
  Pour $\exists x.\, P(x)$, je veux voir un témoin t vérifiant $P(t)$
  De $\exists x.\, P(x)$ on peut déduire $\neg(\forall x.\, \neg\, P(x))$
  mais de $\neg(\forall x.\, \neg\, P(x))$ on ne peut pas déduire $\exists x.\, P(x)$

Agenda

1. Les assertions et la terminaison
2. La réécriture
3. La sémantique dénotationnelle
4. L'interprétation abstraite
5. Logiques et assistants de preuve
   1. calcul des propositions et des prédicats
   2. assistants de preuves
   3. calcul des prédicats d'ordre supérieur
   4. calcul des constructions inductives

Cours du 18 mars

LCF = Logic for Computable Functions

- Premier assistant de preuve, Milner, Weyhrauch et Newey, 1972, au Stanford AI Lab, écrit en LISP
- Directement fondé sur la logique de Scott
- Preuves dirigées par un langage d'assemblage :
  - substitution, instanciation d'axiomes et de règles logiques, etc.
- Mais encore possible d'introduire des déductions fausses...

Et mon vrai début de carrière en décembre 1972 :
preuve de BubbleSort en LCF sur un terminal vidéo*
avec Robin Milner comme tuteur personnel ☺

* muni d'un bouton TV pour regarder le football américain pendant que le PDP 10 calcule...

Milner : le manifeste des assistants de preuve

Our present point of view is that neither a straightforward proof-checker (laborious and repetitive to use) nor an automatic theorem-prover (inefficient because of general search) is satisfactory. What is required is a framework in which a user can both design his own partial proof strategies (where he can find them) and execute single steps of proof (where he needs to) .

A Metalanguage for Interactive Proof in LCF
M. Gordon, R. Milner, L. Morris, M. Newey, C.P. Wadsworth
Proc. 5th POPL Conf. on Principles of Programming Languages (1979)

Edinburgh LCF: A Mechanized Logic of Computation
M. Gordon, R. Milner, C.P. Wadsworth
Lecture Notes in Computer Science 79, Springer (1979)

De LCF à ML

As a first approximation, a tactic should take as argument a goal and produce as result a list of subgoals . We shall here assume that a goal is a sequent, that is

goal = form list × form

But now we can see a deficiency in our first approximation to a tactic. Suppose that a tactic T, applied to goal g , has generated the subgoal list $[g_1; \ldots ;g_n]$ and that somehow theorems th_i achieving g_i $(1 \leq i \leq n)$ have been found. Who is to deduce from $[th_1,...,th_n]$ a theorem th achieving g ? Our answer is that it is the job of T to provide a way of performing this deduction; to this end we define :

proof = thm list → thm
tactic = goal → (goal list × proof)

and we call the proof component of a tactic's result a validation. When a composite tactic has somehow generated an empty goal list, the validations of the components can be composed to yield a theorem, and this composite validation (a function) can be generated automatically as part of the business of composing tactics.

Tacticals are functions on tactics, building simple tactics into composite ones. Three obvious examples are: binary tacticals THEN (apply a second tactic to all subgoals produced by a first) and ORELSE (try one tactic, or if it fails try another), and a unary one REPEAT (iterate a tactic until failure). Defining these, and many others, in ML is a straightforward exercise in functional programming with lists. Moreover, it is easy to show that THEN, ORELSE and REPEAT preserve the validity - even strong validity - of tactics.

(It is worth noting that this tactics- and tacticals style could be adopted in general for problem solving; all that is involved is a type goal, a type for proposed solutions - which might be called shot - and an achievement relation between shots and goals.)

or with proof trees ? or better ?

L'évolution des systèmes de types

- Types en LISP / Scheme : (plus chou carotte) → run-time error
- Types en C : éviter de mélanger des choux et des carottes
  (mais il faut parfois le faire en programmation système)
- Types en ML, Haskell etc. : un chasseur de bugs fondamental
  une aide à l'architecture

proof = thm list → thm
tactic = goal → (goal list × proof)

- Théorie des types de P. Martin-Löf, SystemF de J.Y. Girard :
  une nouvelle approche du calcul
- Calcul des constructions inductives, G. Huet, T. Coquand et C. Paulin, puis Coq : l'unification de calcul, logique et preuves

A suivre