927 Exemples de preuves d'algorithmes : correction et terminaison

Def 1 : La spécification consiste à définir exactement les entrées sur lesquels l'algorithme doit tourner et quelles sorties il doit rendre.

I Terminaison.

Def 2 : On dit qu'un algorithme termine si son exécution s'arrête après un nombre fini d'étapes pour toute entrée conforme à sa spécification.

Ex 3 : Si l'algorithme A termine alors

```
for i=0 to n do
    A
done
```

termine aussi

```
while true do:
    skip
```

ne termine pas.

Ex 4 :

```
while n≠0
    n := n-1
```

termine ssi $n \geq 0$

Thm 5 : Le problème de l'arrêt

ARRET : { entrée : un algorithme A ; sortie : oui si A termine, non sinon }

est indécidable.

1) Ensemble bien fondé

Def 6 : $(E, \leq)$ est dit bien fondé s'il n'existe pas de suite infinie d'éléments de E strictement décroissante.

Ex 7 : $(\mathbb{N}, \leq)$ est bien fondé, $(\mathbb{Z}, \leq)$ ne l'est pas.

Prop 8 : $\mathbb{N}^2$ muni de l'ordre lexicographique est bien fondé

Prop 9 : Un ensemble $(E, \leq)$ est bien fondé SSi toute partie non vide de E admet un élément minimal

Thm 10 : Soit A un algo récursif, X l'ensemble de ses entrées, $\varphi : X \to E$ avec $(E, \leq)$ bien fondé.

Soit $Y = \{x \in X, \varphi(x) = \min \varphi(X)\} \subset X$

Si A(y) termine pour tout $y \in Y$ et que dans l'exécution de A(x) n'apparaissent, en nombre fini, que des appels à A(z) avec $\varphi(z) < \varphi(x)$, alors l'exécution de A(x) termine pour tout $x \in X$.

Ex 11 :

```
pgcd(a,b):
    if b=0:
        return a
    else
        return pgcd(b, a mod b)
```

{ entrée : $0 \leq b < a$; sortie : pgcd(a,b) termine par l'ordre naturel sur le premier argument }

Ex 12 :

```
ackermann(m,n):
    if m=0:
        return n+1
    elif n=0:
        return ackermann(m-1, 1)
    else:
        return ackermann(m-1, ackermann(m, n-1))
```

{ entrée : $m, n \in \mathbb{N}$; sortie : un entier }

termine en utilisant l'ordre lexicographique

2) Variant de boucle

Def 13 : Un variant de boucle est une quantité à valeur dans un ensemble $(E, \leq)$ bien fondé qui décroît strictement à chaque tour de boucle.

Thm 14 : Si une boucle (dont le corps termine) admet un variant de boucle, alors elle termine.

Ex 15 :

```
divis_euclid(a,b):
    q=0
    r=a
    while r≥b do
        r := r-b
        q := q+1
    done
    return (q,r)
```

{ entrée : $(a,b) \in \mathbb{N} \times \mathbb{N}^*$; sortie : $q, r \in \mathbb{N}$ tq $\begin{cases} a = bq + r \\ 0 \leq r < b \end{cases}$ }

Variant : $r \in \mathbb{N}$.

II Correction

Def 16 : Un algorithme est correct s'il est conforme à sa spécification.
On parle de correction totale si l'algorithme termine, partielle sinon.

Ex 17 :

```
syracuse(n):
    if n=1:
        return 1
    elif n pair:
        return syracuse(n/2)
    else:
        return syracuse(3×n+1)
```

{entrée : $n \in \mathbb{N}^*$
sortie : 1

partiellement correct

DEV 1 : Algorithme du tri topologique et correction

1) Correction des algorithmes récursifs

Prop 18 : Soit $(E, \leq)$ un ensemble bien fondé et p un prédicat sur les éléments de E. Si p est vérifié sur tous les éléments minimaux de E et si $\forall x \in E \left([\forall y < x, p(y)] \Rightarrow p(x) \right)$ alors $\forall x \in E \; p(x)$.

Thm 19 : Soit A un algorithme récursif, X l'ensemble de ses entrées.
$\varphi : X \to E$ avec $(E, \leq)$ bien fondé.
Soit $Y = \{x \in X, \varphi(x) = \min \varphi(X)\} \subset X$ et p un prédicat sur les valeurs calculés par A.
Si $p(y)$ est vérifié pour tout $y \in Y$ et si
$\forall z$ argument d'appel de A dans l'exécution de $A(x)$,
$$\varphi(z) < \varphi(x)$$
et $[\forall z \; p(z)] \Rightarrow p(x)$
Alors p est vérifié pour tout $x \in X$.

Ex 20 : L'algorithme pgcd (Ex 11) est correct.

Ex 21 :

```
tri_inser(L):
    if L==[] then return []
    else return insere(tete(L), tri_inser(queue(L)))
```

{Entrée : Une liste L.
Sortie : Une liste triée contenant les éléments de L.

```
insere(x, L):
    if L==[]:
        return [x]
    elif tete(L) ≥ x:
        return x::L
    else:
        return tete(L)::insere(x, queue(L))
```

{Entrée : Une liste L triée d'éléments de $(E, \leq)$, $x \in E$.
Sortie : Une liste triée contenant les éléments de L et x.

L'algorithme tri_inser est correct

2) Correction des algorithmes itératifs

Def 22 : Un invariant de boucle est une propriété vraie au début de la première itération de la boucle et qui vérifie : si elle est vraie au début d'une itération de la boucle, alors elle est vraie à la fin de l'itération.

TH 23 : Si une boucle admet un invariant I, alors I est vrai à la sortie de la boucle.

Ex 24 :

```
dijkstra(G, w, v):
    d = [∞, ..., ∞] de taille |V|
    d[v] = 0
    E = ∅
    F = V
    while F ≠ ∅:
        u = Extraire_min(F)
        E = E ∪ {u}
        for s ∈ Adj(u)
            relacher(u, s, w)
    return d
```

{Entrée : Un graphe $G=(V,E)$, $v \in E$, w un poids sur G
Sortie : Un tableau d tel que $\forall u \in V, d[u] = d(v,u)$

Invariant de la boucle while : "$\forall u \in E, d[u] = d(v,u)$".
↳ L'algorithme dijkstra est correct.

III Logique de Hoare

1) Le langage IMP

Def 25 : On définit les expressions arithmétiques :

$a ::= n \in \mathbb{N} \mid x \in Var \mid a_1 + a_2 \mid a_1 * a_2 \mid a_1 - a_2$

On définit les expressions booléennes :

$b ::= true \mid false \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b_1 \mid b_1 \wedge b_2$

On définit un programmes :

$S ::= skip \mid x := a \mid S_1 ; S_2 \mid if\ b\ then\ S_1\ else\ S_2 \mid while\ b\ do\ S$

Ex 26 : Un programme de factorielle :

$i := 1 ; f := 1 ; while\ i \leq n\ do\ (f := f * i ; i := i+1)$

2) Sémantique naturelle

L'état de la mémoire est représenté par une fonction $\delta : Var \to \mathbb{Z}$. On note $State = \mathbb{Z}^{Var}$.

Def 27 : On définit les règles de dérivation syntaxique ci-dessous.

On note $\langle S, \delta \rangle \to \delta'$ s'il existe un arbre de dérivation pour $\langle S, \delta \rangle \to \delta'$

* $\dfrac{}{\langle skip, \delta \rangle \to \delta}\ (skip)$
* $\dfrac{}{\langle x := a, \delta \rangle \to \delta[x \mapsto a]}\ (affect)$
* $\dfrac{\langle S_1, \delta \rangle \to \delta'' \quad \langle S_2, \delta'' \rangle \to \delta'}{\langle S, \delta \rangle \to \delta'}\ (seq)$
* $\dfrac{\langle S_1, \delta \rangle \to \delta'}{\langle if\ b\ then\ S_1\ else\ S_2, \delta \rangle \to \delta'}$ si $[\![b]\!] = true$ (if_t)
* $\dfrac{\langle S_2, \delta \rangle \to \delta'}{\langle if\ b\ then\ S_1\ else\ S_2, \delta \rangle \to \delta'}$ si $[\![b]\!] = false$ (if_f)
* $\dfrac{\langle S, \delta \rangle \to \delta' \quad \langle while\ b\ do\ S, \delta' \rangle \to \delta''}{\langle while\ b\ do\ S, \delta \rangle \to \delta''}$ si $[\![b]\!] = true$
* $\langle while\ b\ do\ S, \delta \rangle \to \delta$ si $[\![b]\!] = true$

TR 28 : IMP est déterministe : $\forall S, \forall \delta, \begin{cases} \langle S, \delta \rangle \to \delta_1 \\ \langle S, \delta \rangle \to \delta_2 \end{cases} \Rightarrow \delta_1 = \delta_2$

Def 29 : On définit $\mathcal{S}_{ns}[\![S]\!]_\delta = \begin{cases} \delta' \text{ si } \langle S, \delta \rangle \to \delta \\ \text{undef} \end{cases}$

Ex 30 : L'arbre sémantique de "while true do skip" est infini, donc $\mathcal{S}_{ns}[\![while\ true\ do\ skip]\!] = \text{undef}$

3) Triplets de Hoare

Def 31 : On appelle triplet de Hoare un triplet de la forme $\{P\} S \{Q\}$ où P et Q sont des prédicats et S est un programme.

Def 32 : On définit les règles de dérivation suivantes :

* $\dfrac{}{\{P\}\ skip\ \{P\}}\ (skip_H)$
* $\dfrac{}{\{P[x \mapsto a]\}\ x := a\ \{P\}}\ (Aff_H)$
* $\dfrac{\{\ \}\ S_1\ \{R\} \quad \{Q\}\ S_1\ \{R\}}{\{P\}\ S_1 ; S_2\ \{R\}}\ (seq_H)$
* $\dfrac{\{[\![b]\!]_1 \wedge P\}\ S_1\ \{Q\} \quad \{[\![\neg b]\!]_1 \wedge P\}\ S_2\ \{Q\}}{\{P\}\ if\ b\ then\ S_1\ else\ S_2\ \{Q\}}\ (if_H)$
* $\dfrac{\{[\![b]\!]_1 \wedge P\}\ S\ \{P\}}{\{P\}\ while\ b\ do S\ \{\neg [\![b]\!]_1 \wedge P\}}\ (while_H)$
* $\dfrac{\{P'\}\ S\ \{Q'\}}{\{P\}\ S\ \{Q\}}$ si $\begin{cases} P \Rightarrow P' \\ Q' \Rightarrow Q \end{cases}$ (CSQ)

Def 33 : On note $\vdash \{P\} S \{Q\}$ s'il existe un arbre de dérivation pour $\{P\} S \{Q\}$

On note $\models \{P\} S \{Q\}$ si $(\forall \delta \in State$ tel que $\delta \models P, \mathcal{S}_{ns}[\![S]\!]_\delta \models Q)$

(par convension, $\underline{undef} \models Q$ pour tout Q)

TR 34 : $(\vdash \{P\} S \{Q\}) \Rightarrow (\models \{P\} S \{Q\})$

DEV 2 : $(\models \{P\} S \{Q\}) \Rightarrow (\vdash \{P\} S \{Q\})$

Ex 35 : On a $\vdash \{x = m\}\ y := 1 ; while\ \neg(x = 1)\ do\ (y := y * x ; x := x - 1)\ \{y = m!\}$

Références :

* Cormen, Leiserson, Rivest, Stein, Introduction à l'algorithmique
* Albert, Cours et exercices d'informatique
* Froidevaux, Gaudel, Soria, Types de données et algorithmes
* Nielson, Nielson, Semantics with applications
* Winskel, The formal semantics of programming languages