

Machines de Turing et Algorithmes

Notion informelle d'algorithme

Processus qui permet d'obtenir une solution à un problème général en un nombre fini d'étapes.

Exemples

Recettes de cuisine, multiplication "à la main" de deux nombres, schéma de tricot, etc.

Propriétés attendues

Un algorithme doit

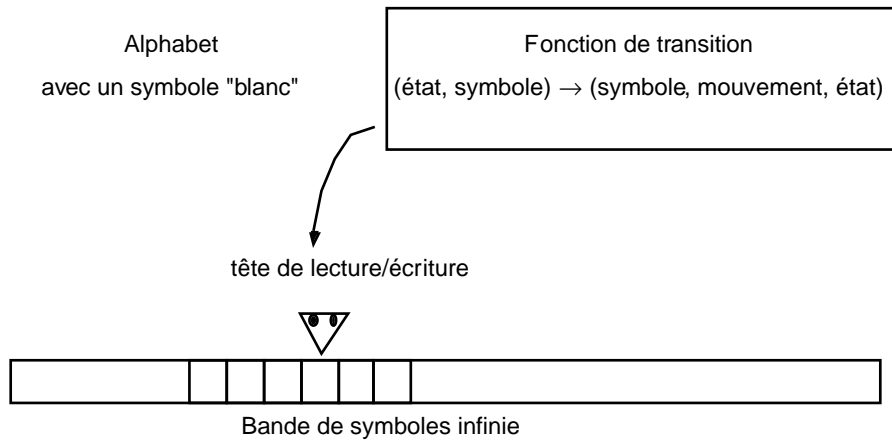
- être **correct**, c-à-d mener à tout coup à la solution
- se **terminer**, c-à-d mener à la solution en un nombre fini d'étapes
- être général, c-à-d résoudre toutes les instances d'une **classe de problèmes**

Machine de Turing

Formaliser le travail d'un humain (mathématicien)

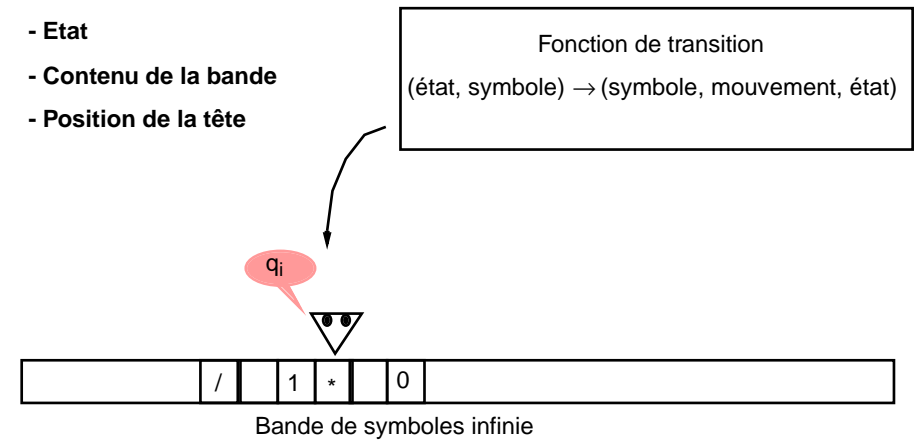
- utilise une feuille de papier,
- ne voit qu'un symbole à la fois
- peut lire et/ou remplacer le symbole courant
- se trouve à tout moment dans un "état d'esprit"
- peut déplacer son attention vers le symbole suivant ou précédent
- prend ses décisions en fonction de son état d'esprit et du symbole courant

Structure de la machine



Etat instantané de la machine

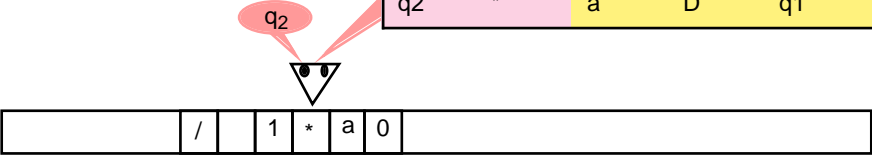
- Etat
- Contenu de la bande
- Position de la tête



Principe

Tableau écriture, mouvement, transition

état	sym	écrit	mvt	suivant
q1	a	0	G	q2
q1	0	0	D	q1
...
q2	*	a	D	q1

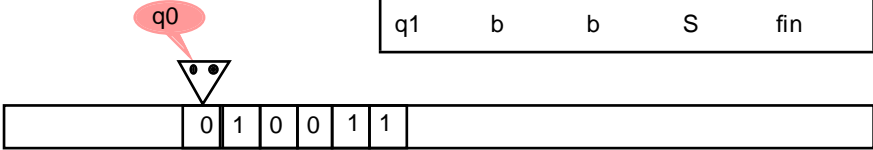


Bande de symboles infinie

Exemple

Remplacer les 0 par des 1 et revenir à gauche des symboles

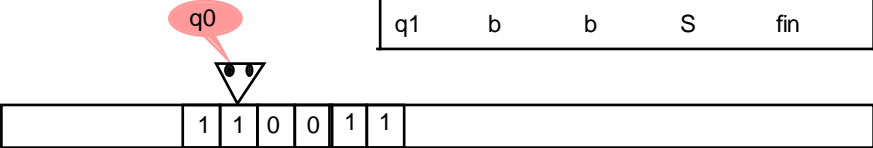
état	sym	écrit	mvt	→état
q0	0	1	D	q0
q0	1	1	D	q0
q0	b	b	G	q1
q1	0	0	G	q1
q1	1	1	G	q1
q1	b	b	S	fin



Exemple

Remplacer les 0 par des 1 et revenir à gauche des symboles

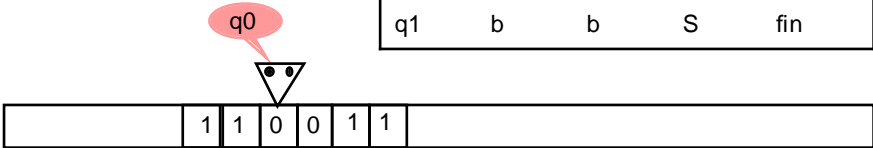
état	sym	écrit	mvt	→état
q0	0	1	D	q0
q0	1	1	D	q0
q0	b	b	G	q1
q1	0	0	G	q1
q1	1	1	G	q1
q1	b	b	S	fin



Exemple

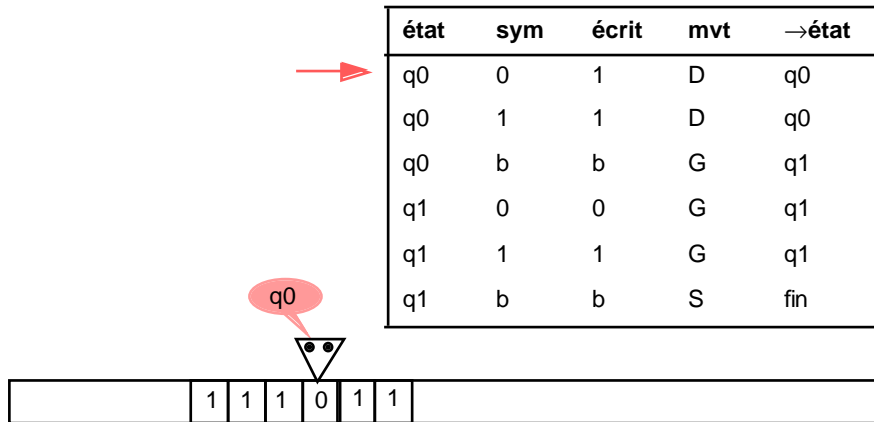
Remplacer les 0 par des 1 et revenir à gauche des symboles

état	sym	écrit	mvt	→état
q0	0	1	D	q0
q0	1	1	D	q0
q0	b	b	G	q1
q1	0	0	G	q1
q1	1	1	G	q1
q1	b	b	S	fin



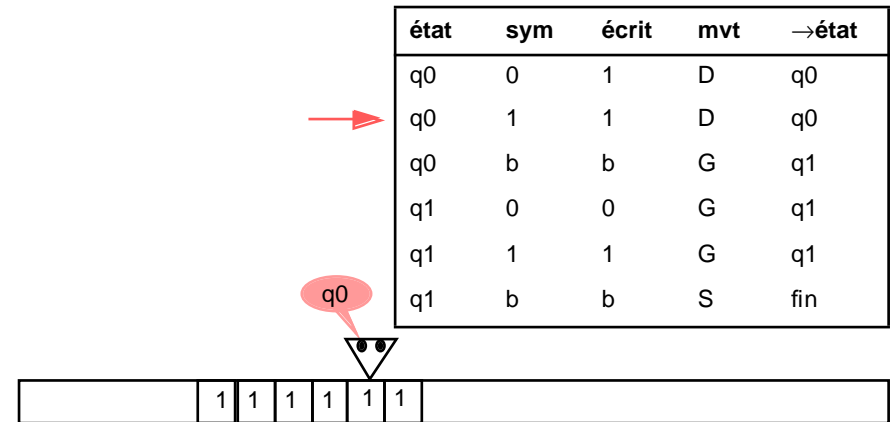
Exemple

Remplacer les 0 par des 1 et revenir à gauche des symboles



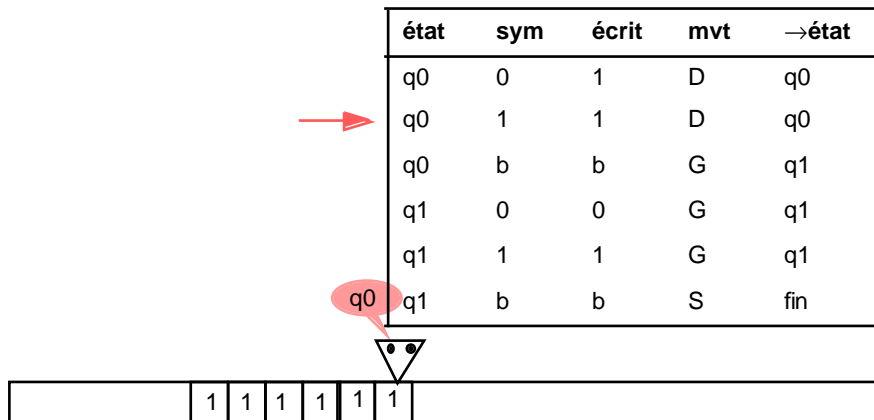
Exemple

Remplacer les 0 par des 1 et revenir à gauche des symboles



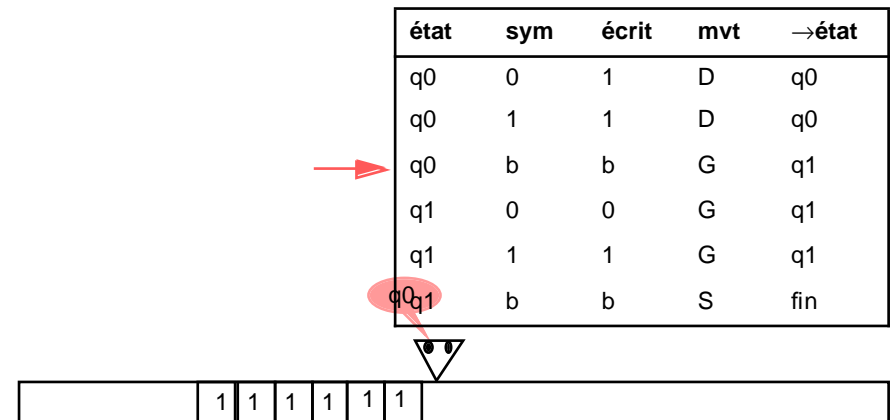
Exemple

Remplacer les 0 par des 1 et revenir à gauche des symboles



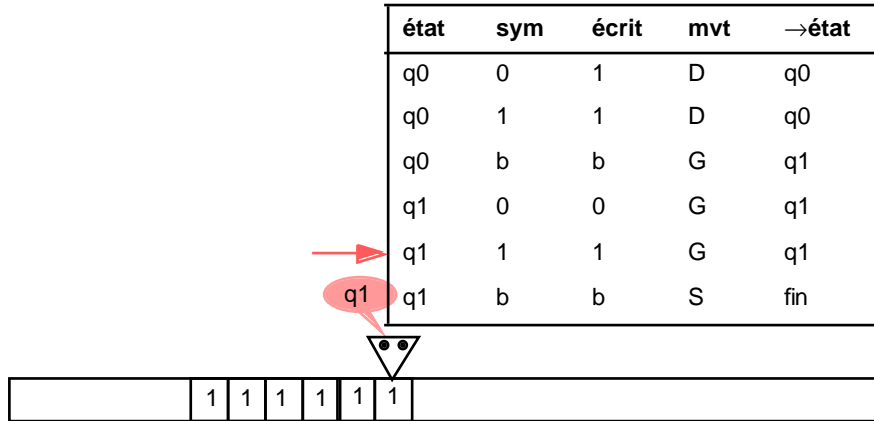
Exemple

Remplacer les 0 par des 1 et revenir à gauche des symboles



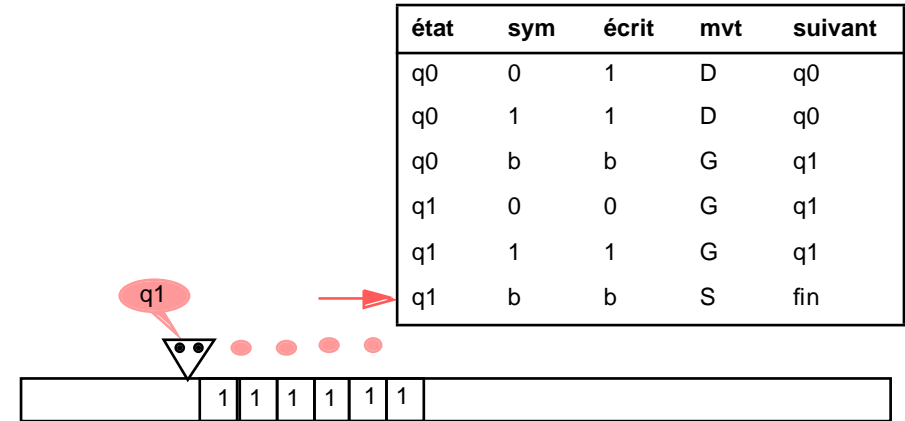
Exemple

Remplacer les 0 par des 1 et revenir à gauche des symboles

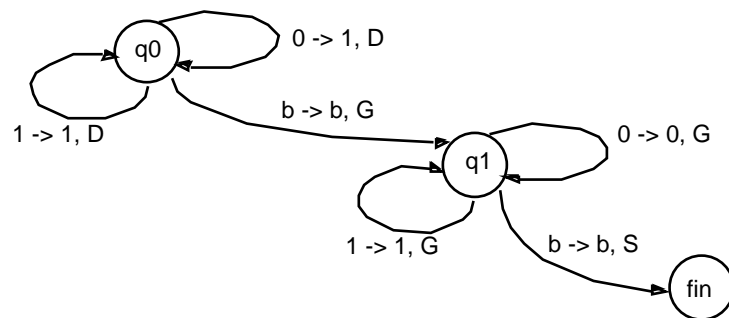


Exemple

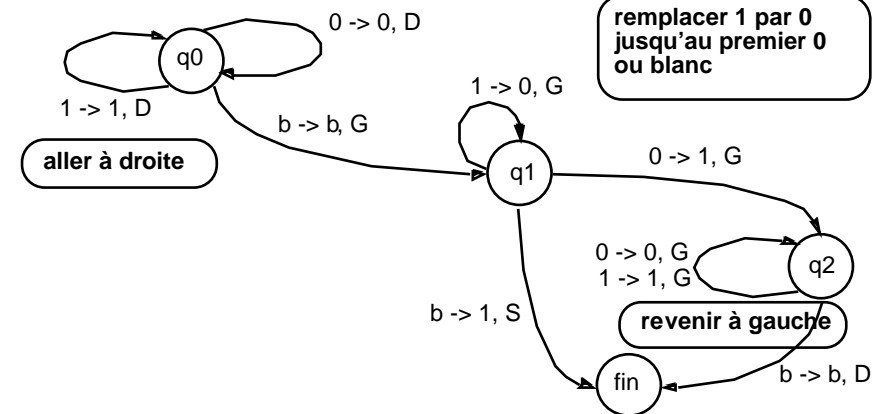
Remplacer les 0 par des 1 et revenir à gauche des symboles



Représentation par un diagramme

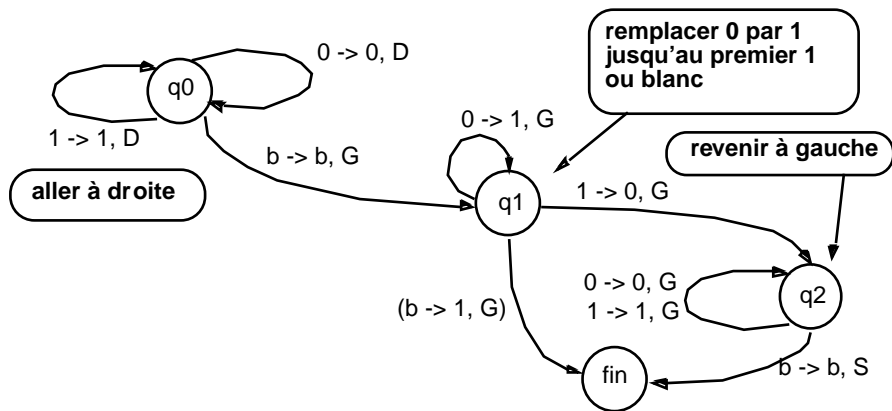


Calcul de X + 1 en binaire



10	1	110110101111
+ 1	+ 1	+ 1
--	--	-----
11	10	1101101 10000

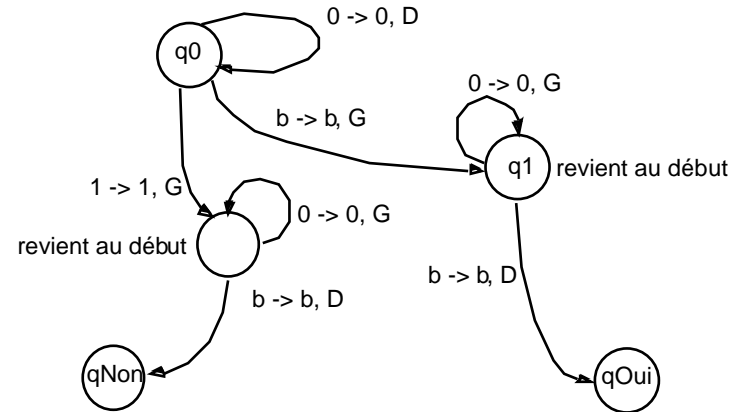
Calcul de $X - 1$ en binaire



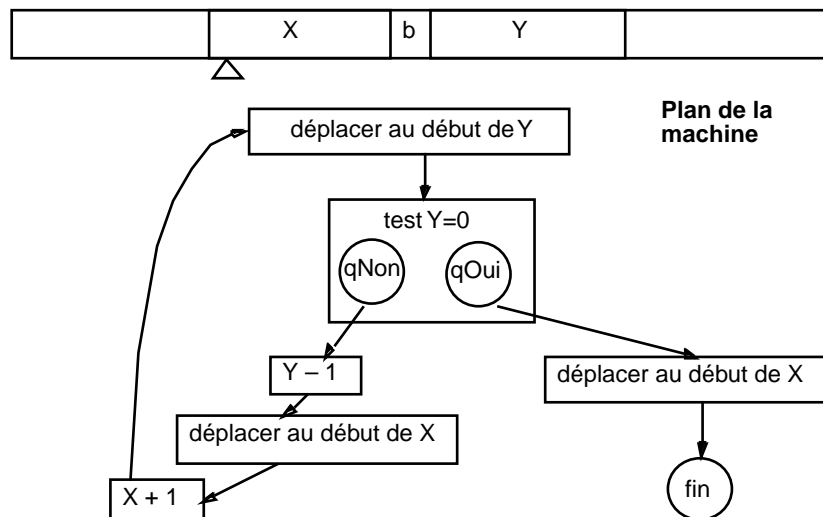
11	10	110110100000
- 1	- 1	- 1
--	--	-----
10	01	110110 011111

Teste si $X = 0$

Fragment de machine qui arrive dans l'état qOui ou qNon si $X=0$ ou $X \neq 0$



Calcul de $X+Y$



Thèse de Church-Turing (tentative de définition formelle)

Tout processus mathématique que l'on peut raisonnablement qualifier de «purement mécanique», peut s'obtenir à l'aide d'un programme pour une machine de Turing.

Il s'agit d'une thèse \neq énoncé mathématique

À l'appui de la thèse:

On n'a jamais trouvé de machine plus puissante que la machine de Turing.

La machine de Turing est universelle: elle peut se simuler elle-même

Il y a cependant des formalismes plus «lisibles» pour exprimer les algorithmes

Expression des algorithmes

Formalismes pour exprimer des algorithmes

Le langage **S** : variables entières, +1, -1, SI ... ALLER

Le langage **Sx** : extension de S avec affectation, saut, expressions arithmétiques

Organigrammes : un formalisme graphique

Un **langage algorithmique structuré** : si ... sinon ; tant que ... ; affectation

Ces formalismes sont tous équivalents à celui de la machine de Turing, on peut s'en servir pour exprimer des algorithmes.

- Aucun n'est plus puissant que la MdT (on peut toujours les traduire en MdT)
- Ils sont aussi puissants que la MdT (on peut traduire MdT en ces formalismes)

Langage S

X1, X2, ... variables d'entrée, entiers illimités

Y variable de sortie, entier illimité

Z1, Z2, ... variables locales (travail), entiers illimités

Le programme est formé d'instructions de la forme

$V \leftarrow V + 1$

$V \leftarrow V - 1$

SI $V \neq 0$ ALLER A [k]

Exemple

[a] $X \leftarrow X - 1$

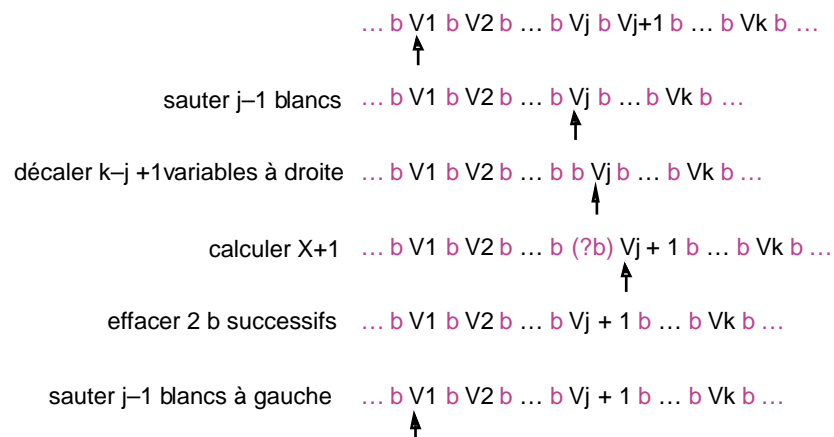
$Y \leftarrow Y + 1$

SI $X \neq 0$ ALLER A [a]

Calcule $f(x) = 1$ si $x = 0$; $= x$ sinon

On peut traduire S en MdT

p.ex. Opération $V_j \leftarrow V_j - 1$



Macro instructions en S

Mise à zéro : " $V \leftarrow 0$ "

[a] $V \leftarrow V - 1$
SI $V \neq 0$ ALLER A [a]

Saut sans condition: "ALLER A [b]"

$Z_i \leftarrow Z_i + 1$
SI $Z_i \neq 0$ ALLER A [b]

Affectation entre variables: " $V \leftarrow V'$ "

$V \leftarrow 0$
[a] SI $V' \neq 0$ ALLER A [b]
 $ALLER A [c]$
[b] $V' \leftarrow V' - 1$
 $V \leftarrow V + 1$
 $Z \leftarrow Z + 1$
 $ALLER A [a]$
[c] SI $Z \neq 0$ ALLER A [d]
 $ALLER A [e]$
[d] $Z \leftarrow Z - 1$
 $V' \leftarrow V' + 1$
 $ALLER A [c]$

Arithmétique en S

Addition "X1 + X2"

$Y \leftarrow X1$

$Z \leftarrow X2$

[b] SI $Z \neq 0$ ALLER A [a]

ALLER A [e]

[a] $Z \leftarrow Z-1$

$Y \leftarrow Y+1$

ALLER A [b]

Multiplication "X1 x X2"

$Z2 \leftarrow X2$

[b] SI $Z2 \neq 0$ ALLER A [a]

ALLER A [e]

[a] $Z2 \leftarrow Z2-1$

$Z1 \leftarrow X1+Y$

$Y \leftarrow Z1$

ALLER A [b]

Sx: Extension de S

On peut étendre S pour obtenir un langage plus "confortable" Sx.

On définit les macro opérations

1. ALLER A [L]

2. $V \leftarrow \langle \text{expression arithmétique} \rangle$

$X \leftarrow 2*Y / (Z1+Z3)$

3. SI $\langle \text{expression logique} \rangle$ ALLER A [L]

SI $(X < 3)$ OU $(Y/2 > Z)$ ALLER A [K]

Ces macro opérations sont traduisibles en S.

Exemple: l'algorithme d'Euclide en Sx

Cet algorithme, datant du III^{ème} siècle av. J.C, permet de calculer le plus grand commun diviseur (pgcd) de deux nombres entiers $x1$ et $x2$, en supposant $x1 > x2$.

$z1 \leftarrow x1$

$z2 \leftarrow x2$

A: SI $(z1 = z2)$ ALLER E

SI $(z1 < z2)$ ALLER B

$z1 \leftarrow z1 - z2$

ALLER A

B: $z2 \leftarrow z2 - z1$

ALLER A

E: $y \leftarrow z1$

Algorithme basé sur les propriétés arithmétique :

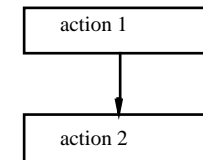
1. si $a > b$ alors $\text{pgcd}(a, b) = \text{pgcd}(a, a-b)$

2. si $a=b$ alors $\text{pgcd}(a, b) = a = b$

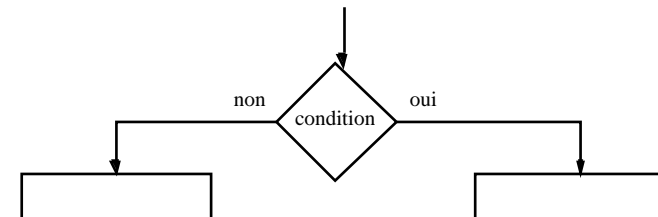
L'algorithme finit toujours par s'arrêter car $z1$ ou $z2$ diminue tout en restant positif

Les Organigrammes («flowcharts»)

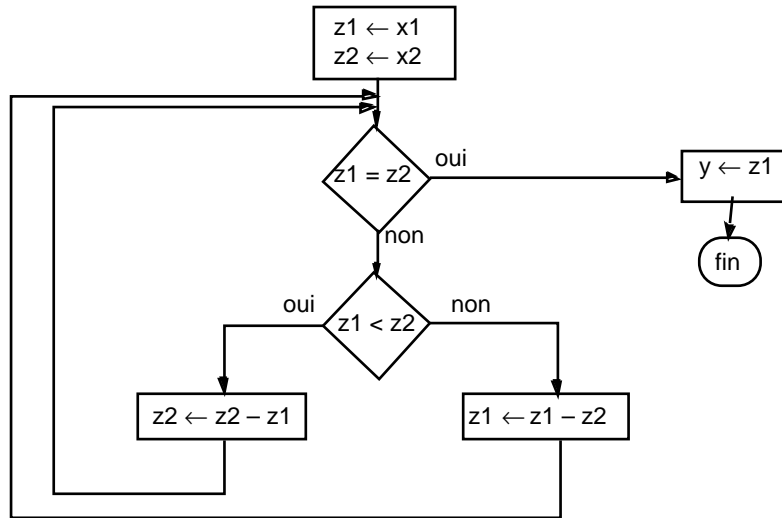
Il s'agit d'un formalisme graphique pour visualiser l'enchaînement en séquence des actions



et l'alternative



Euclide organigramme



L'expression Structurée des Algorithmes

L'idée est de représenter un algorithme à l'aide de blocs qui possèdent **une** entrée et **une** sortie et sont composables entre eux.

- Affectation:** $v \leftarrow \text{expression}$
- Séquence:** $\{\text{instr1}; \text{instr2}; \text{instr3}; \dots; \text{instrn}\}$
- Alternative:** **si** (condition) **instr**
- Alternative complète:** **si** (condition) **instr1** **sinon** **instr2**
- Itération conditionnelle:** **tant que** (condition) **instr**

Instr = affectation ou séquence ou alternative ou itération conditionnelle.

Chaque construction peut s'exprimer dans le langage Sx non structuré.

Euclide structuré

```

z1 ← x1;
z2 ← x2;
tant que (z1 ≠ z2)
{
  si (z1 < z2)
    z2 ← z2 - z1
  sinon
    z1 ← z1 - z2
}
y ← z1
  
```

(Böhm et Jacopini 1966):

Tout algorithme écrit sous forme non structurée peut s'écrire sous forme structurée (il est en général nécessaire d'ajouter des variables).

Le non calculable

On ne peut pas tout exprimer sous forme d'algorithme.

\Leftrightarrow Il y a des fonctions que les machines de Turing ne peuvent calculer

\Leftrightarrow Il y a des fonctions pour lesquelles il n'existe pas de MdT

Preuve

On peut dénombrer les MdT et attribuer à chaque machine **M** un numéro **num(M)**

Soit m_p la fonction que calcule la machine numéro **p**.

Soit $g(x)$ la fonction $x \rightarrow m_x(x) + 1$.

(« ce que calcule la machine no x quand on lui donne x comme input ») + 1

g est une fonction calculable \Rightarrow il y a une machine **MG** qui calcule **g**.

Soit **G = num(MG)**, le numéro de la machine **MG**

On a alors $m_G(G) = g(G) = m_G(G) + 1$

Donc **0 = 1**

Donc **G** n'existe pas