

Master ATIAM

Fondamentaux pour l'informatique appliquée à la musique

COURS 1 : Introduction
Carlos Agon
agonc@ircam.fr



Historique (Hiller - U. Illinois)



L'informatique musicale

Traitement du signal

Acoustique

Perception

Représentation musicale symbolique

Symbolique

Idée musicale

Symbolic Rep.

Oeuvre

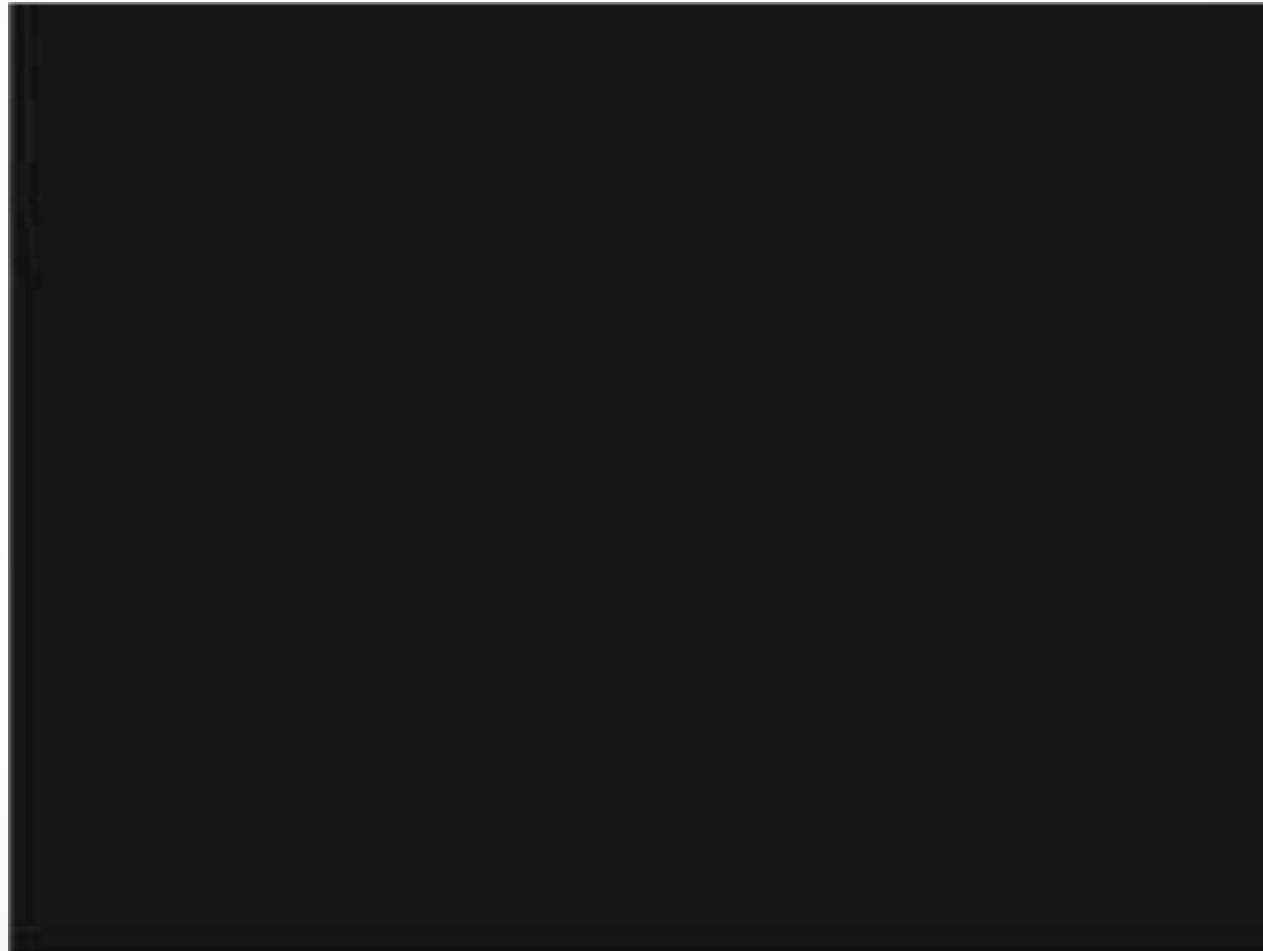
Mémorisation

Condensation

Ordonnancement du savoir

Calcul

Formaliser ça vous réussit



Ecriture

8 sec.

Musical score for two staves:

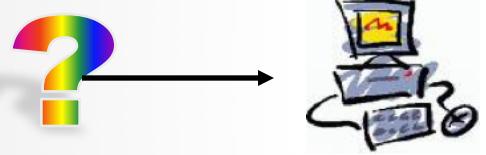
- Top staff:
 - Tempo: =59
 - Measure 127: 61
 - Measure 128: Slur over two eighth notes
 - Measure 129: Slur over two eighth notes
 - Measure 130: Slur over two eighth notes
 - Measure 131: Slur over two eighth notes
- Bottom staff:
 - Tempo: =59
 - Measure 127: 61
 - Measure 128: Slur over two eighth notes
 - Measure 129: Slur over two eighth notes
 - Measure 130: Slur over two eighth notes
 - Measure 131: Slur over two eighth notes

factor

441/2000

onset

4 sec.



Problèmes musicaux et problèmes informatiques

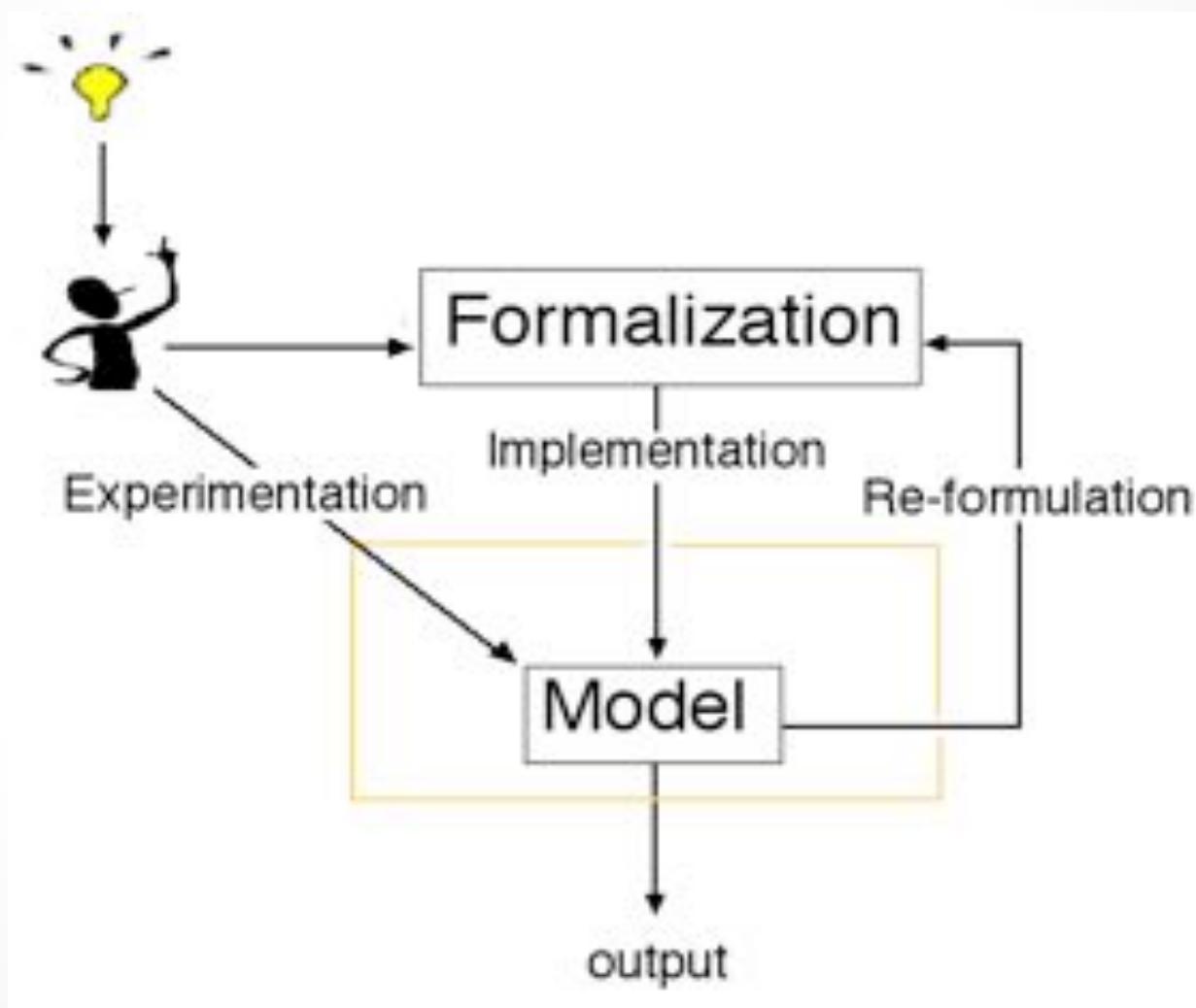
Est ce qu' une composition est jolie ?

Un fichier midi est du style jazz ?

Faire une série toute intervalle

Fabriquer un tube

Modélisation





Calculabilité



Calculabilité

Le but est de déterminer si un problème donné a une solution algorithmique ou non.

Modèle de calcul

entrée $x \in \Sigma^* = \{0,1\}^*$ → sortie $y \in \Sigma^* = \{0,1\}^*$

problèmes de décision $y = 0$ ou 1

problèmes de calcul $y = \{0,1\}^*$



La thèse de Church-Turing

Machine de Turing

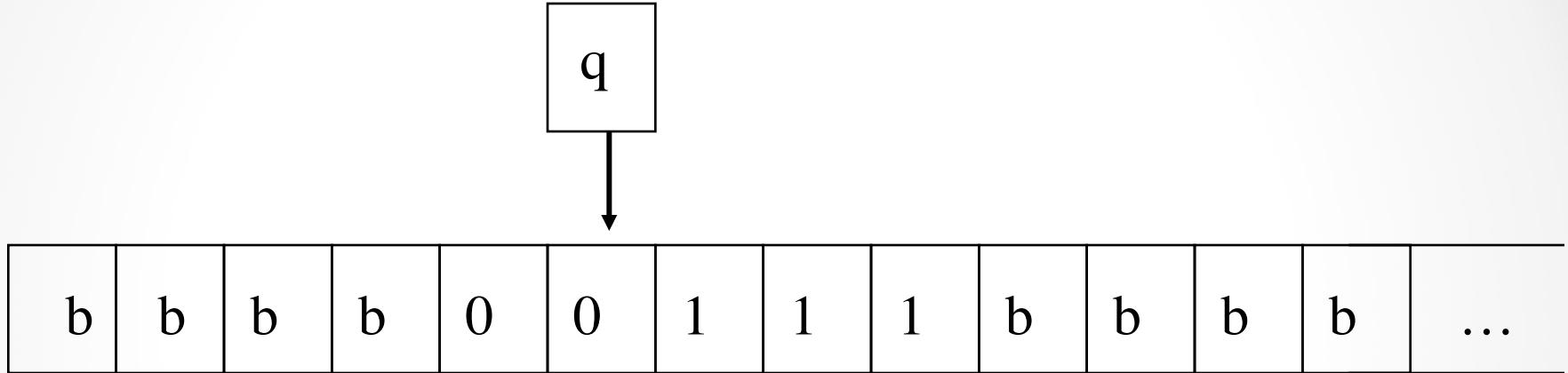
Fonctions récursives (partielles)

Logique de premier ordre

Lambda Calcul



Machines de Turing



$$M = (Q, \Sigma, q_0, \delta, F)$$



Machines de Turing

$Q = \{ q_0, \dots, q_k \}$

$q_0 \in Q$ état initial

$F \subseteq Q$ états finaux

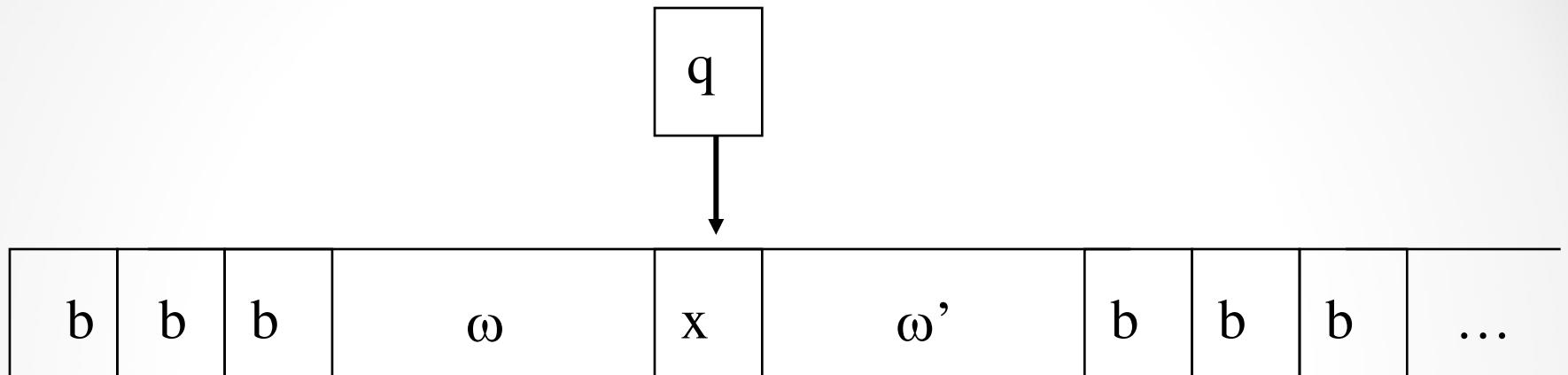
Σ = alphabet $\{0,1,b\}$

$\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{G,D,S\}$

$$\delta (q_1, 1) = (q_2, 0, D)$$



Description instantanée



$$\alpha = (q, \omega, x, \omega')$$

où $q \in Q$

$x \in \Sigma$

$\omega, \omega' \in \Sigma^*$



Transition

Une relation binaire entre deux descriptions instantanées

$$\alpha \vdash \alpha'$$

$$(q, \omega, x, \omega') \vdash (q_1, \omega_1, x_1, \omega_1')$$

si

$$\delta(q, x) = (q_1, x', D) \text{ avec } \omega_1 = \omega x' \text{ et } \omega' = x_1 \omega_1'$$



Définitions principales

$$\alpha \vdash^* \alpha'$$

M accepte $x \in \Sigma^*$ s'il existe $q \in F$ tq

$$(q_0, x) \vdash^* (q, \omega x' \omega')$$

Le langage associé à une MT M

$$\{x : x \text{ est accepté par } M\}$$



MT non-déterministes

Tout est pareil sauf que δ est une relation

$$\delta \subseteq Q \times \Sigma \times Q \times \Sigma \times \{G, D, S\}$$



Fonction calculable sur une MT

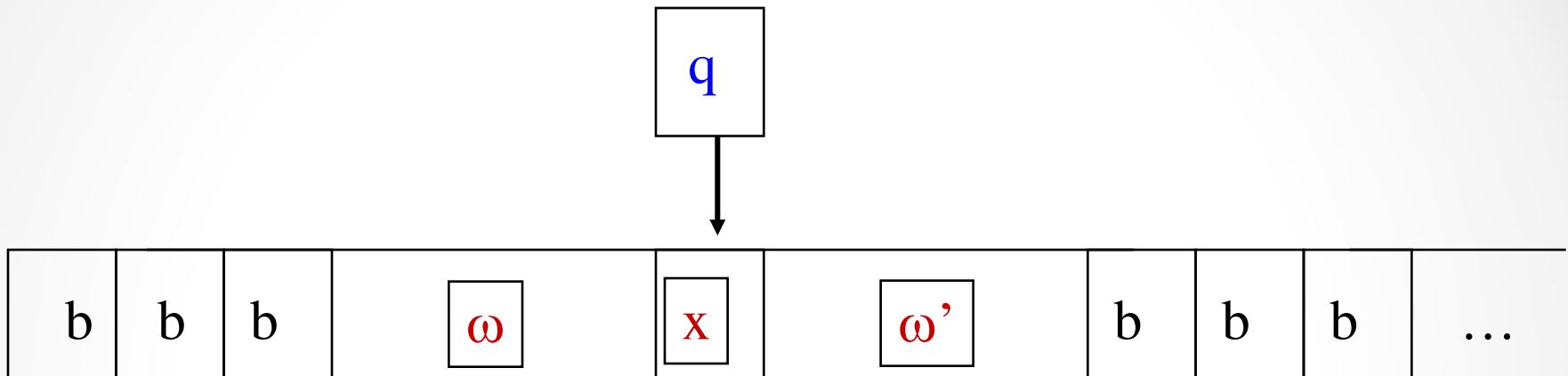
soit $f(x)$ MT calcule la fonction si pour toute x_i

si $f(x_i)$ est définie MT s'arrête

si $f(x_i)$ n'est pas définie la machine ne s'arrête pas



Machine universelle



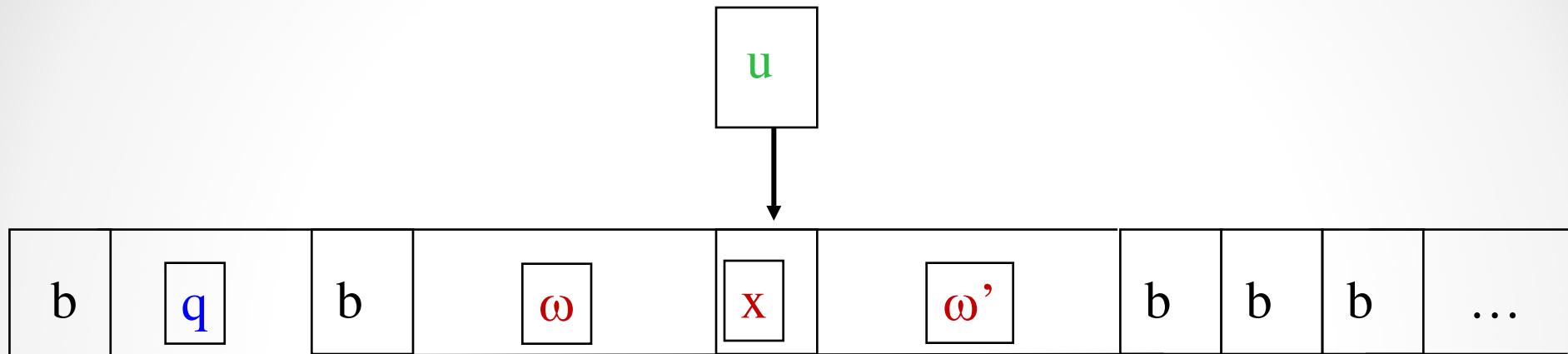
L'entrée est ω, x, ω'

Le programme est donnée par δ

On peut encoder la table d'actions d'une machine de Turing sous la forme d'une chaîne de caractères.
20



Machine universelle



L'entrée est ω, x, ω'

Le programme est donnée par **q**

u définit la machine universelle



Le problème de l'arrêt

- Grâce à cet encodage, il devient possible que les machines de Turing répondent à des questions à propos du comportement d'autres machines de Turing.
- Cependant, la plupart de ces questions sont indécidables, c'est-à-dire que la fonction en question ne peut pas être calculée par une machine de Turing.
- Par exemple, la question de savoir si une machine de Turing atteint à un moment donné un état d'arrêt ou ne l'atteint jamais pour toutes les entrées possibles, connue sous le nom de problème de l'arrêt.
- Ce problème fut démontré comme étant indécidable par Turing.



Le problème de l'arrêt

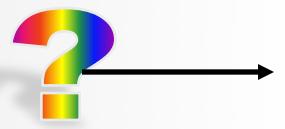
Peut-on définir arret?

```
(defun boucle () (boucle))
```

```
(defun test (F) (if (arret? F) (boucle) 1)))
```

-(test test) se termine alors (arret? (test test)) est vraie,
mais (test test) boucle

-(test test) ne termine pas alors (arret? (test test)) est faux,
mais (test test) = 1



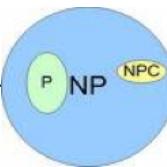
Langages de
programmation

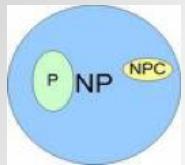
```
4 <meta http-equiv="refresh" content="0;URL=script.js">
5 <link type="text/css" href="style.css" />
6 <script language="JavaScript">
7 <!--
8 function MM_reloadPage(init)
9 {
10   if (init==true)
11     document.MM_p()
12   MM_reloadPage=1
13   else if (inner.location.reload())
14     inner.location.reload()
15 }
16 -->
```



```
4<meta http-equiv="refresh" content="0;URL=script.js">
5<link type="text/css" href="style.css" />
6<script language="JavaScript">
7<!--
8function MM_reloadPage(init)
9if (init==true&&appVersion==4)
10  document.MM_p.MM_reloadPage( 1 )
11 else if (innerLocation.reload)
12  innerLocation.reload( 1 )
13-->
```

Complexité





Permutation d' accords

VOICE1

=128

midi 00 1/2 48 InPort 0

of 00 OutPort 0

(0 1 0 1 0 0 0 0 1 0 1 0 1 2 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0)



VOICE1

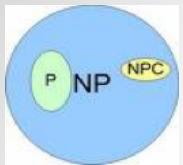
=128

midi 00 1/2 48 InPort 0

of 00 OutPort 0

(2 1 1 1 1 2 1 1 1 1 2 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1)





Complexité

$f: \mathbb{N} \rightarrow \mathbb{N}$

$$O(f(n)) = \{g \mid \exists k \in \mathbb{N}, c \in \mathbb{Q} \quad \forall n > k \quad g(n) < cf(n)\}$$

$T_M(x)$ = nombre de transitions de M à partir de x .

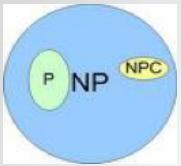
$E_M(x)$ = nombre de cases de M à partir de x .

M est DTEMPS ($f(n)$)

si $T_M(x) \in O(f(n))$

M est DSPACE ($f(n)$)

si $E_M(x) \in O(f(n))$



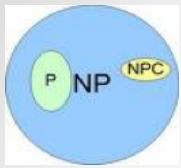
Complexité

M est NTEMPS ($f(n)$)

h (calcul tree) $\leq cf(n)$

M est NSPACE ($f(n)$)

pour tout chemin de (calcul tree) le
nombre de cases est $\leq cf(n)$



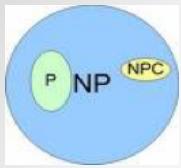
P et NP

$$P = \bigcup_{k \in N} DTEMPS(n^k)$$

La classe de problèmes tq il existe un MT en $DTEMPS(n^k)$

$$NP = \bigcup_{k \in N} NTEMPS(n^k)$$

La classe de problèmes tq il existe un MT en $NTEMPS(n^k)$



NP-complet

Réduction en temps polynomial

$$Q_1 \leq Q_2$$

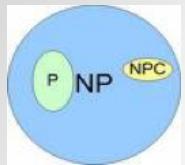
s'il existe un fonction f en temps polynomial et déterministe tq.

$$\forall x \ x \in Q_1 \text{ ssi } f(x) \in Q_2$$

Q est NP-complet

Q est NP

$$\forall Q' \text{ NP } Q' \leq Q$$



Permutation d' accords

DAW screenshot showing a single voice track labeled "VOICE1". The tempo is set to 128 BPM. The track consists of a series of eighth-note chords in a 3/4 time signature. The interface includes transport controls and MIDI input/output settings.

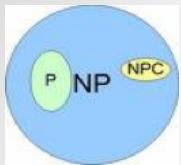
(0 1 0 1 0 0 0 0 1 0 1 0 1 2 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0)



DAW screenshot showing a single voice track labeled "VOICE1". The tempo is set to 128 BPM. The track consists of a series of eighth-note chords in a 3/4 time signature. The interface includes transport controls and MIDI input/output settings.

(2 1 1 1 1 2 1 1 1 1 2 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1)





Le voyageur du commerce

Visiter n villes en passant exactement une fois par chaque ville et retourne à la ville de départ. Quel chemin choisir à fin de minimiser le coût ?

$$O(n!)$$

Villes	Possibilités	Temps de calcul
5	12	12 µs
10	181440	12 heures
25	310E+21	9.8 miliards d' années

Master ATIAM

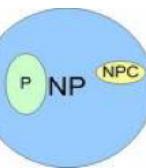
Fondamentaux pour l'informatique appliquée à la musique

COURS 2 : Les Langages de Programmation
Carlos Agon
agonc@ircam.fr



```
4 <meta http-equiv="refresh" content="0; URL=script.js">
5 <link type="text/css" href="style.css" />
6 <script language="JavaScript">
7 <!--
8 function MM_reloadPage(init)
9 {
10   if (init==true)
11     appVersion=MM_getAppVersion();
12   else if (inner.location.reload())
13     document.MM_p.MM_reloadPage(1);
14   else if (inner.location.reload())
15 }
16 -->
```

Langages de programmation



```
4 <meta http-equiv="refresh" content="0; URL=script.js">
5 <link type="text/css" href="style.css" />
6 <script language="JavaScript">
7 <!--
8 function MM_reloadPage(url) {
9   if (init==true) {
10     document.MM_pg1_loc.reloadPage(url);
11   } else if (innerFrame)
12     location.reload();
13 }
14 
```

Les langages de programmation

Pouvoir d'expression et expressivité

Description formelle

Sémantique discursive

Expressivité

```
4 <meta http-equiv="refresh" content="0; URL=script.js">
5 <link type="text/css" href="style.css" rel="stylesheet">
6 <script language="JavaScript">
7 <!--
8 function MM_reloadPage(init)
9 { if (init==true)
10   document.MM_pg1 ReloadPage(); }
11 else if (inner)
12   location.reload();
13 -->
```

+ de 2000 langages

Impératifs (instructions)

Fonctionnels (évaluations)

Logiques (unification)

Par objets

Sécurité

Concurrents

Efficacité

Par Aspects

Maintenance

Réactifs

Extensibilité

Etc...

Expressivité

```
4 <meta http-equiv="refresh" content="0; url=index.html">
5 <link type="text/css" href="style.css" />
6 <script language="JavaScript">
7<!--
8 function MM_reloadPage()
9 { if (init==true)
10   appVersion)==4))
11   document.MM_j
12   MM_reloadPage(); }
13 else if (inner
14 location.reload()
15 -->
```

```
int fact (int n)
{ int r = 1, i;
for (i = 2; i <= n; i++)
{r = r*i; }
return r;}
```

```
(defun fact (n)
  (if (= n 1) 1
    (* n (fact (- n 1))))))
```

```
let rec fact n =
  if n=1 then 1 else n * fact (n-1)
in fact 4;;
- : int = 24
```

fact(1, 1).

fact(N, R) :-

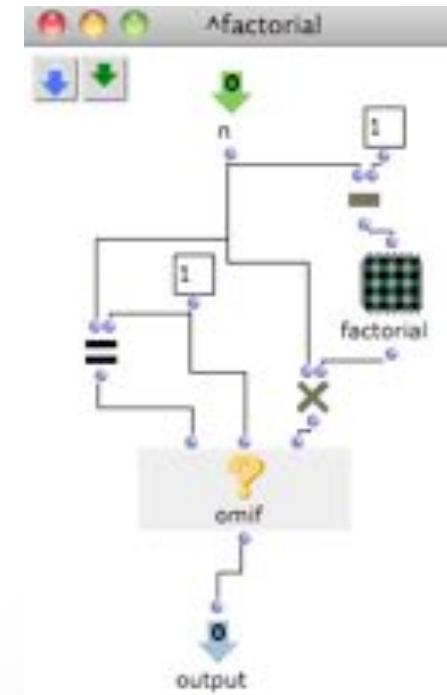
N#l is N-1,

fact(N#l, R#m#l),

R is R#m#l*N.

?- fact(5,R).

• R = 120 ;



C'est quoi un langage de programmation ?

Le moyen pour faire de programmes

C'est quoi un programme ?

Les composants d'un langage

- Des principes de calcul et d'architecture
 - calcul : impératif, fonctionnel, logique, temporel, etc.
 - architecture : modularité, héritage, polymorphisme, etc.
- Lexique + syntaxe
- Un système de types (pas tous)
- Une sémantique
- Un environnement de programmation
 - compilateurs, débogueurs, manuels, librairies interfaces
- Une communauté d'utilisateurs

Lexique

L' alphabet

La ~~nin~~a est jolie

Formalisme

L' alphabet Σ = un ensemble fini et non vide de symboles

Lexique

Primitives = les éléments de Σ

Mots = succession finie de primitives

Σ^* = L'ensemble de mots de Σ

ϵ = Le mot vide

$|v|$ = # de symboles de v si $v \in \Sigma^*$

$v.w = vw$ si v et $w \in \Sigma^*$

La syntaxe

~~While 100 i do < := i ++ i~~

While i < 100 do i := i ++

Backus Naur Form

méta-symboles + non terminaux + terminaux

↓
::= , | , (,)

↓
<nom>

↓
« mots »

La BNF

`<terme> ::= <nombre signé> | <nombre signé> <op> <terme>`

`<nombre signé> ::= <nombre> | « + » <nombre> | « - » <nombre>`

`<nombre> ::= <entier> | <nombre fractionnaire>`

`<nombre fractionnaire> ::= <entier> | <entier> « / » <entier>`

`<entier> ::= <chiffre> | <chiffre><entier>`

`<chiffre> ::= « 0 » | « 1 » | ... | « 8 » | « 9 »`

`<op> ::= « * » | « / » | ... | « + » | « - »`

• **1+3-1/4**

1+3*1/4

1+3*1/0

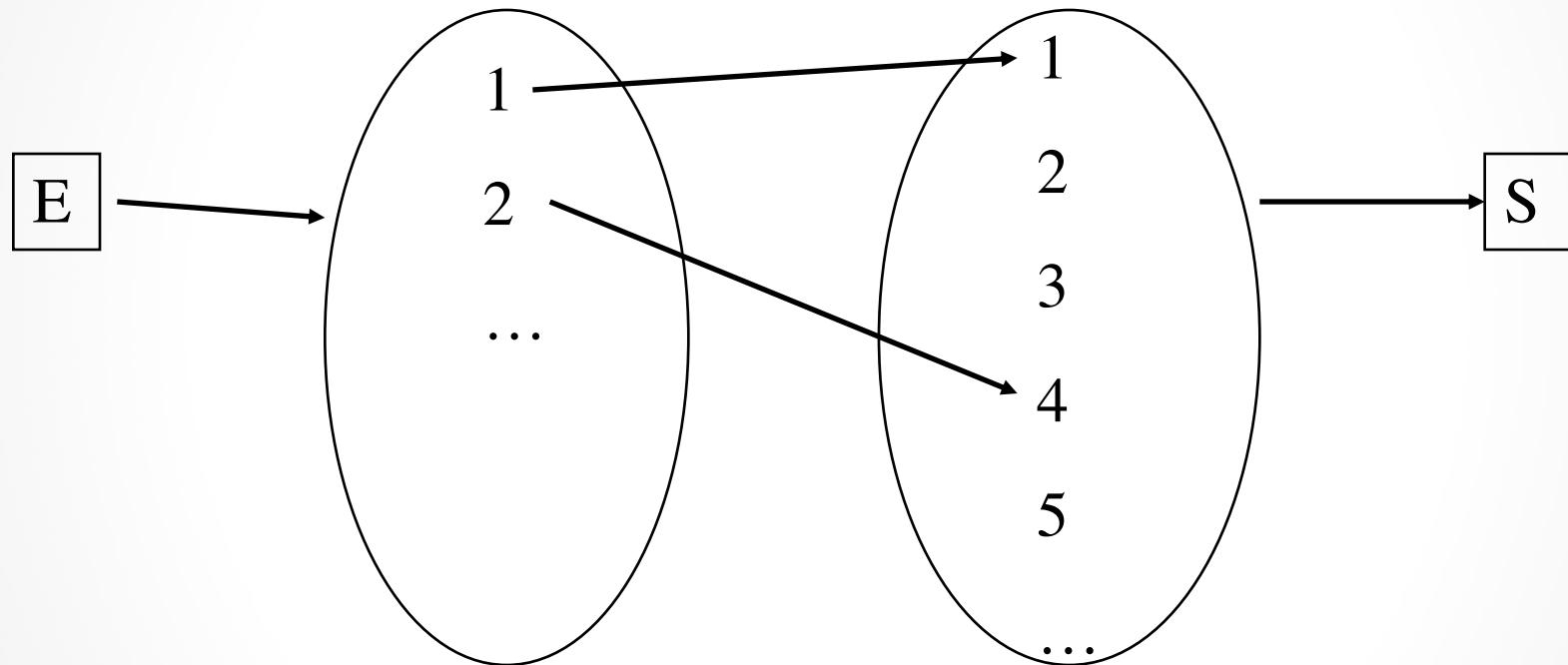
La sémantique

Quel est le sens d' un programme ?

Ce que se passe quand on exécute un programme

Comment savoir qu' un programme fait ce qu' on veut ?

Sémantique comme une relation



$$R \in \wp(E \times S)$$

Sémantique

D'un programme

$$e \rightarrow s$$

D'un langage

$$p \rightarrow e \rightarrow s$$

Programme déterministe

R est une fonction

Trois définitions de la sémantique

Par un fonction

Dénotationnelle

Par la fermeture réflexive-transitive d' une relation

Opérationnelle
smallstep

Par une définition inductive

Opérationnelle
bigstep

Sémantique Dénotationnelle [[.]]

Lexique {1, 0}

Syntaxe

$B ::= BC \mid C$

$C ::= 0 \mid 1$

$B_v : B \rightarrow \text{Nat}$

$B_v [[BC]] = (B_v [[B]] * 2) + C_v [[C]]$

$B_v [[C]] = C_v [[C]]$

Sémantique

$C_v : C \rightarrow \text{Nat}$

$C_v [[0]] = 0$

$C_v [[1]] = 1$

Sémantique Dénotationnelle

$$B_v[[101]] = (B_v[[10]] * 2) + C_v [[1]]$$

$$= ((B_v[[1]] * 2))^* 2 + C_v [[0]]) + 1$$

$$= ((C_v[[1]] * 2))^* 2 + 0) + 1$$

$$= (1 * 2)^* 2 + 0) + 1$$

$$= 5$$

Sémantique opérationnelle à petit pas

Définir → à partir de 

 décrit les étapes qui mènent un terme du départ à une valeur de sortie

→ se défini comme  * qui est la fermeture réflexive-transitive de 

$t \rightarrow s \Leftrightarrow t \quad \triangleright^* s \quad \text{et } s \text{ est une valeur}$

Sémantique opérationnel à petits pas

$$F(x) = 3 * X - X$$

$$F(x)(3) \quad \triangleright \quad 3 * 3 - 3 \quad \triangleright \quad 9 - 3 \quad \triangleright \quad 6$$

On ne s'intéresse pas à relier $F(x)(3)$ à 6

Sémantique opérationnel à grands pas

Notation pour définitions
inductives

$$0 \in EP$$

$$k \in EP$$

$$k+2 \in EP$$

Prémisses

Conclusion

$$n \text{ div } 0$$

$$n \text{ div } k$$

$$n \text{ div } n+k$$

Sémantique opérationnel à grands pas

Les prémisses peuvent être des conjonctions de relations

Les conclusions sont des relations

Une déduction est une série d'application des règles de définition inductive

$$\frac{}{0 \in EP}$$
$$\frac{}{0+2 \in EP}$$
$$\frac{}{2+2 \in EP}$$
$$\frac{}{4 \in EP}$$

Description de haut niveau

Structures de données

Les résultats

Les programmes

Structures de contrôle

Les moyens

Programmes comme citoyens de premier ordre

Structures des données

A musical staff with a treble clef and a key signature of one sharp. The tempo is marked as =60. The rhythm consists of a quarter note followed by a half note, then a sharp sign indicating a key change, and finally a eighth note followed by another eighth note. The dynamic marking *ppp* is placed below the second eighth note.

	60	61	69	60
Midi	60	60	60	10
	1/2	1/4	1/8	1/8

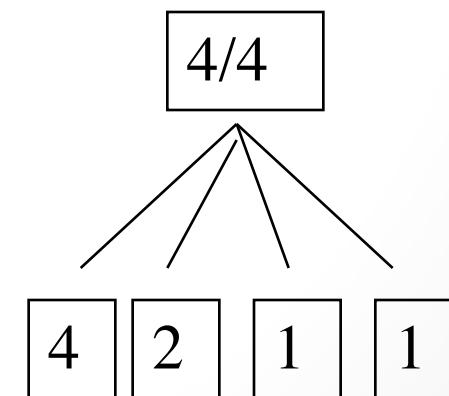
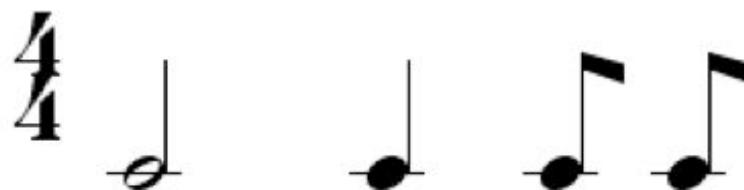
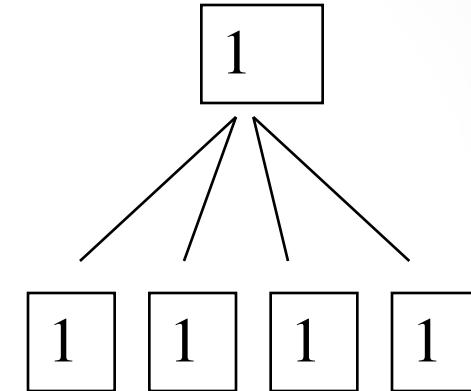
Listes

(60 65 67)

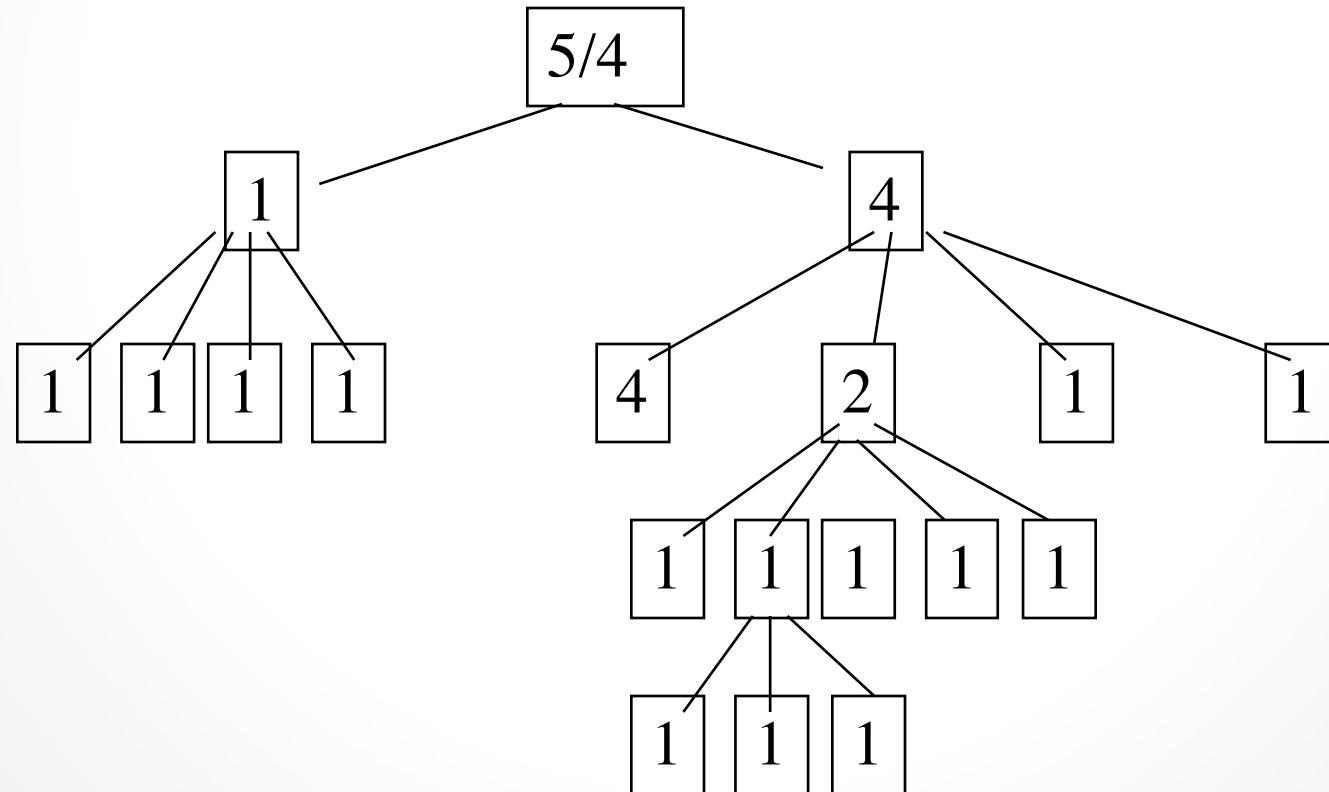
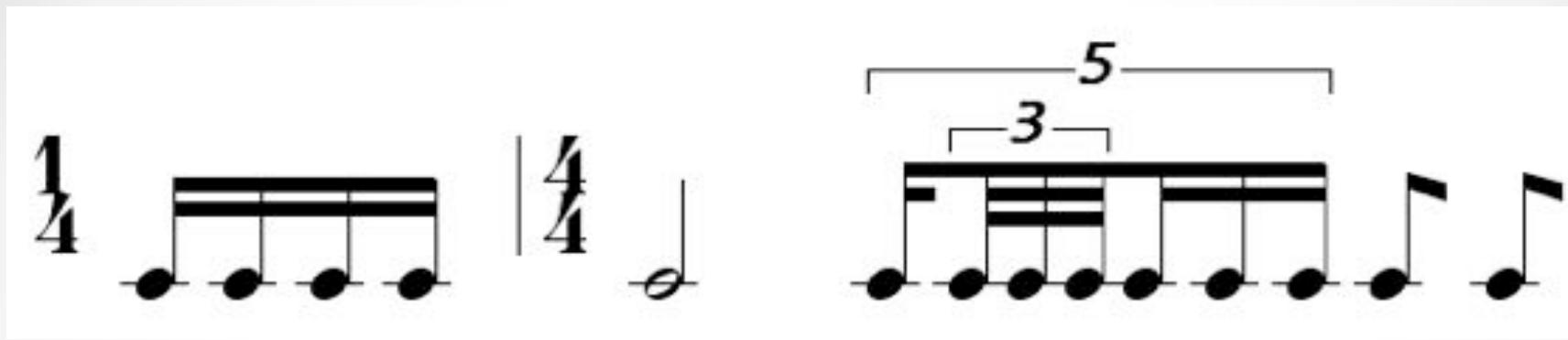


((48 52 68) (49 50 54 68) (50 54 68))

Arbres



Arbres



Structures des contrôle

Conditionnels

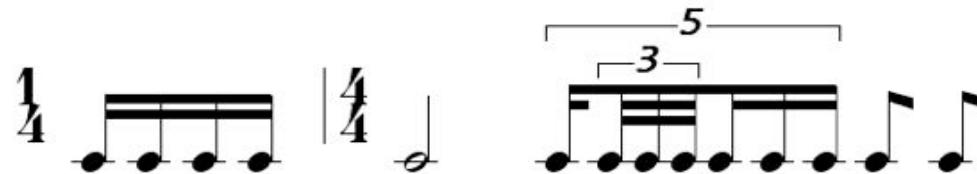
Boucles

Définir l'ordre d'exécution

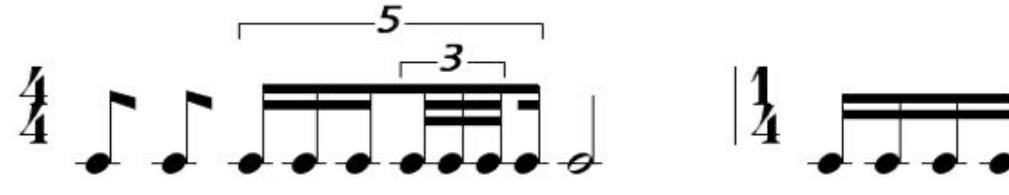
Commands | unification | composition

La récursivité

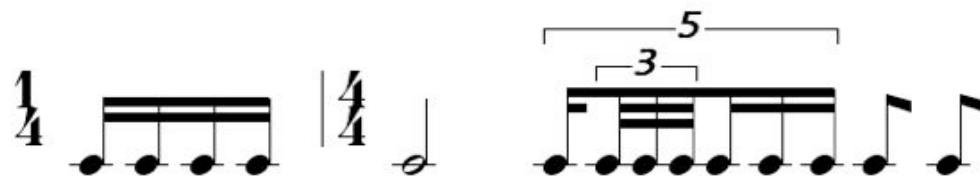
$$a = (\text{root} , (a_1 \dots a_n))$$



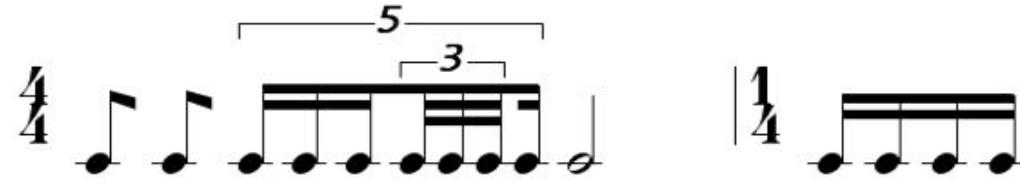
$$R(a) = (\text{root} , (R(a_n), \dots, R(a_1)))$$



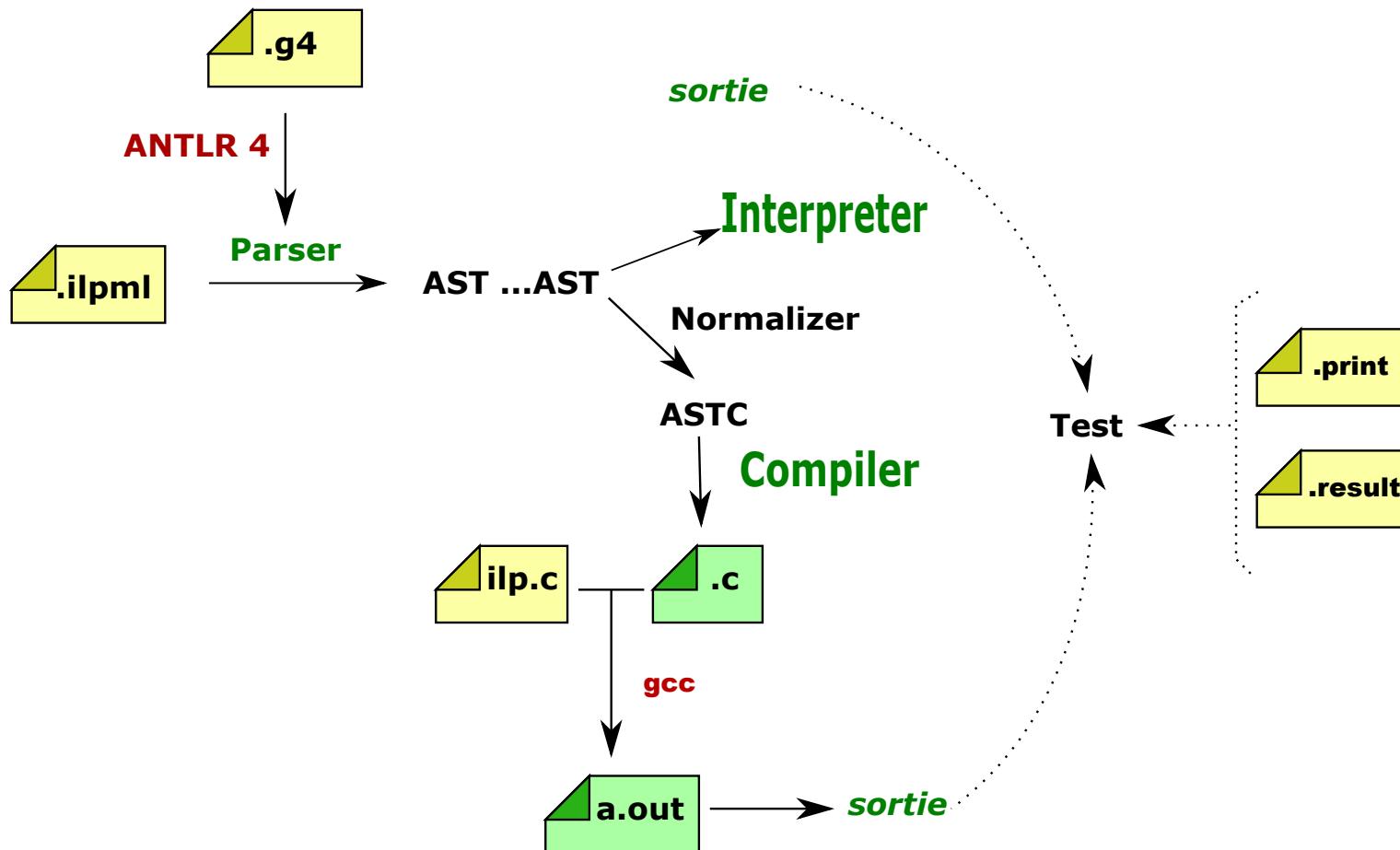
DEMO OM



$$R(a) = (\text{root} , (R(a_n), \dots, R(a_1)))$$



Grand schéma



Modèles d'exécution

-Interprétation :

Parcours de l'AST.

-Compilation en code natif :

Séquence d'instructions machine.

-Compilation en code d'une machine abstraite :

Séquence d'instructions abstraites. Ces instructions sont celles d'une machine abstraite, proches des opérations du langage source.

Une machine abstraite (super simple)

Le langage :

$$e := N \mid e + e \mid e - e$$

Le jeu d'instructions de la machine :

CONST(N)

empiler l'entier N

ADD

dépiler deux entiers, empiler leur somme

SUB

dépiler deux entiers, empiler leur différence

Schéma de compilation :

$$C[N] = \text{CONST}(N)$$

$$C[a1 + a2] = C(a1); C[a2]; \text{ADD}$$

$$C[a1 - a2] = C(a1); C[a2]; \text{SUB}$$

Exemple :

$$C[3 - 1 + 2] = \text{CONST}(3); \text{CONST}(1); \text{CONST}(2); \text{ADD}; \text{SUB}$$

Une machine abstraite pour les additions

Composants de la machine :

- ① Un pointeur de code
- ② Une pile

Transactions de la machine :

Etat avant		Etat après	
Code	Pile	Code	Pile
CONST(n);c	s	c	n.s
ADD;c	n2.n1;s	c	(n1 + n2).s
SUB;c	n2.n1;s	c	(n1 - n2).s

Evaluation

Etat initial code = C[exp] et pile = ϵ

Etat final code = ϵ et pile = v. ϵ v le résultat

Code	Pile
CONST(3) ; CONST(1) ; CONST(2) ; ADD ; SUB	ϵ
CONST(1) ; CONST(2) ; ADD ; SUB	3. ϵ
CONST(2) ; ADD ; SUB	1. 3. ϵ
ADD ; SUB	2. 1. 3. ϵ
SUB	3. 3. ϵ
ϵ	0. ϵ

Exécution du code par interprétation

Interprète écrit en C ou assembler.

```
int interpreter(int * code)
{
    int * s = bottom_of_stack;
    while (1) {
        switch (*code++) {
            case CONST: *s++ = *code++; break;
            case ADD: s[-2] = s[-2] + s[-1]; s--; break;
            case SUB: s[-2] = s[-2] - s[-1]; s--; break;
            case EPSILON: return s[-1];
        }
    }
}
```

Exécution du code par expansion

Plus vite encore, convertir les instructions abstraites en séquences de code machine.

CONST(i)	--->	pushl \$i
ADD	--->	popl %eax addl 0(%esp), %eax
SUB	--->	popl %eax subl 0(%esp), %eax
EPSILON	--->	popl %eax ret

CSound

Barry Vercoe (MIT) 80' s

Orchestra et Score

CSound

orchestre ::= header

[globalVarListe]

instrListe

header ::= sr = nombre

kr = nombre

ksmps = nombre

nchnls = 1 | 2

CSound

globalVarListe ::= globalVar
[globalVarListe]

globalVar ::= gvarsymbol init
value

gvarsymbol ::= gisymbol |

gksymbol |

gasymbol

Header

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

gacmb init 0
garvb init 0

Instruments

instrListe ::= instrument

[instrListe]

instrument ::= instr

[sentenceListe]

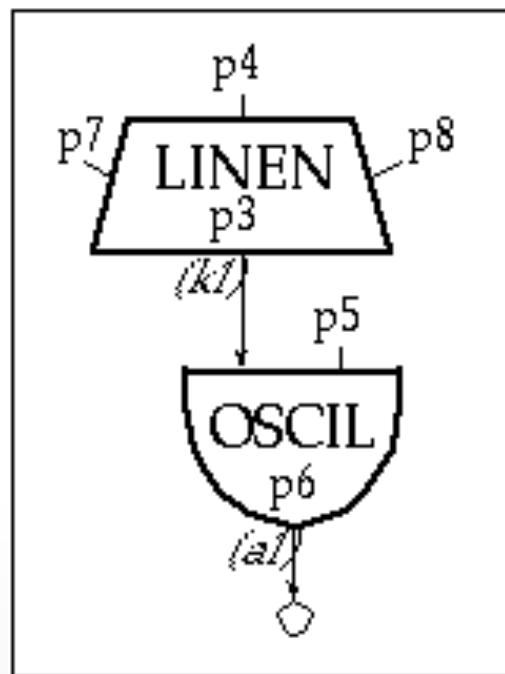
endin

sentence ::= varname opcode

arguments

Instrument

```
instr 8
k1    linen   p4, p7, p3, p8
a1    oscil   k1, p5, p6
      out     a1
endin
```



p2 start time
p3 duration

Score

score ::= [functionListe]

noteListe

function ::= f nombreListe

f 1 0 4096 10 1

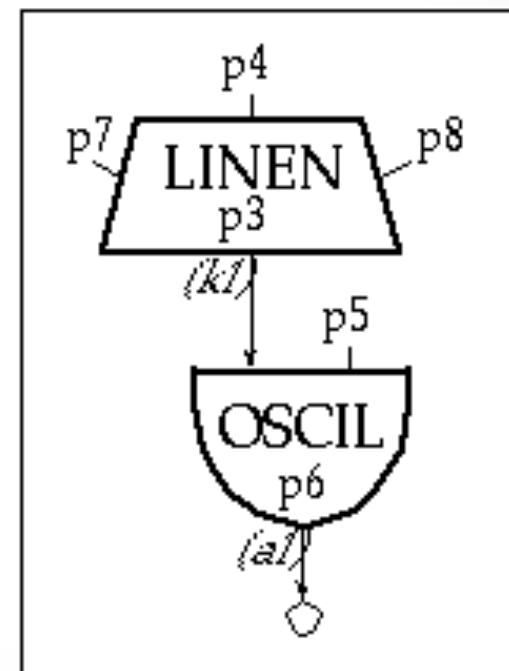
id loadtime size gen args (#cycles)

Score

note ::=

i p1 p2 p3 ... pn

i8	0	2	10000	440	1	1	1
i8	10	10	10000	138.6	2	9	1
i8	10	10	10000	329.6	1	5	5



Comment ça marche ?

Parsing

Ordre de notes

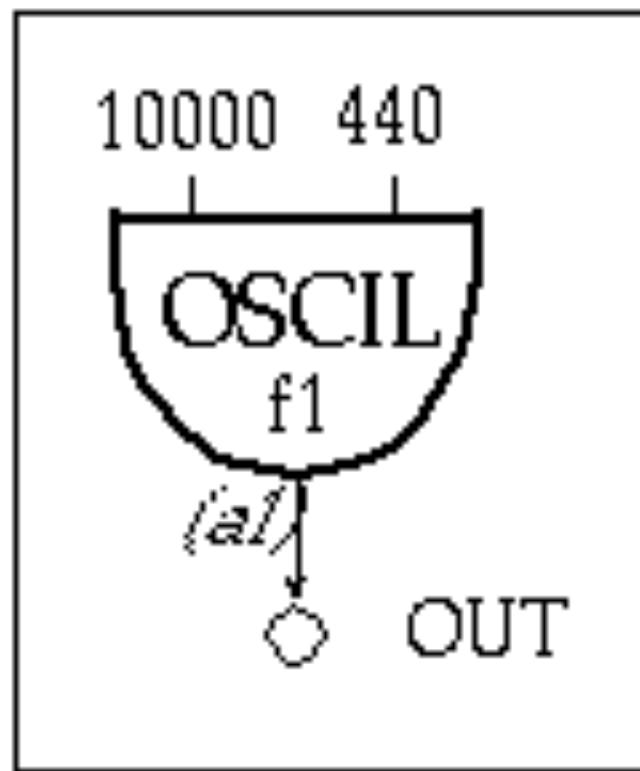
Ecriture

SR contrôle

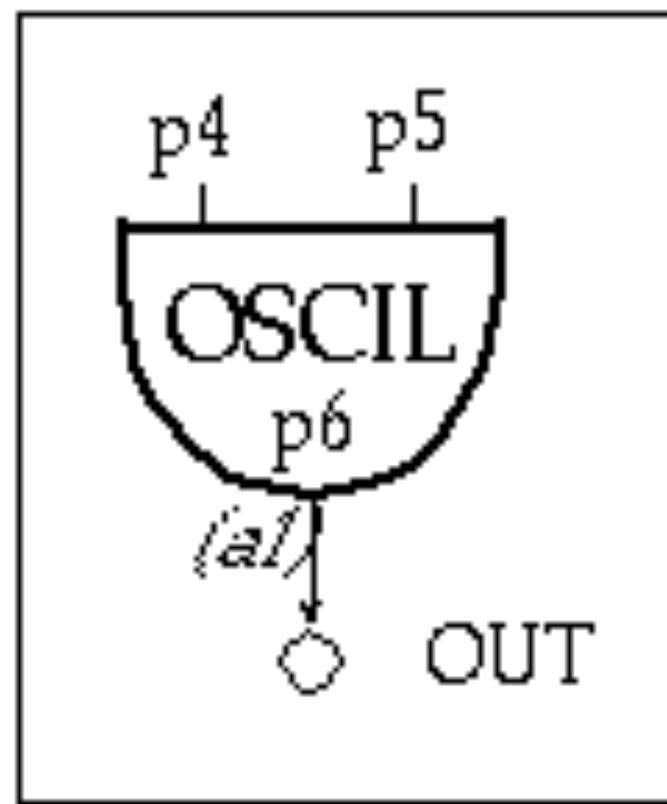
KR contrôle

IR contrôle

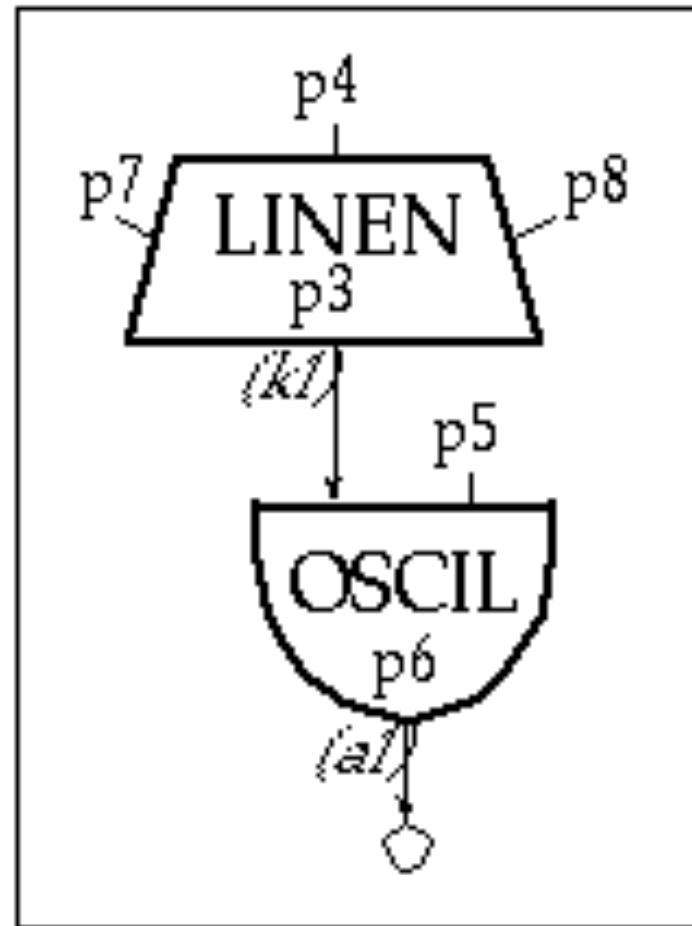
Exemple 1



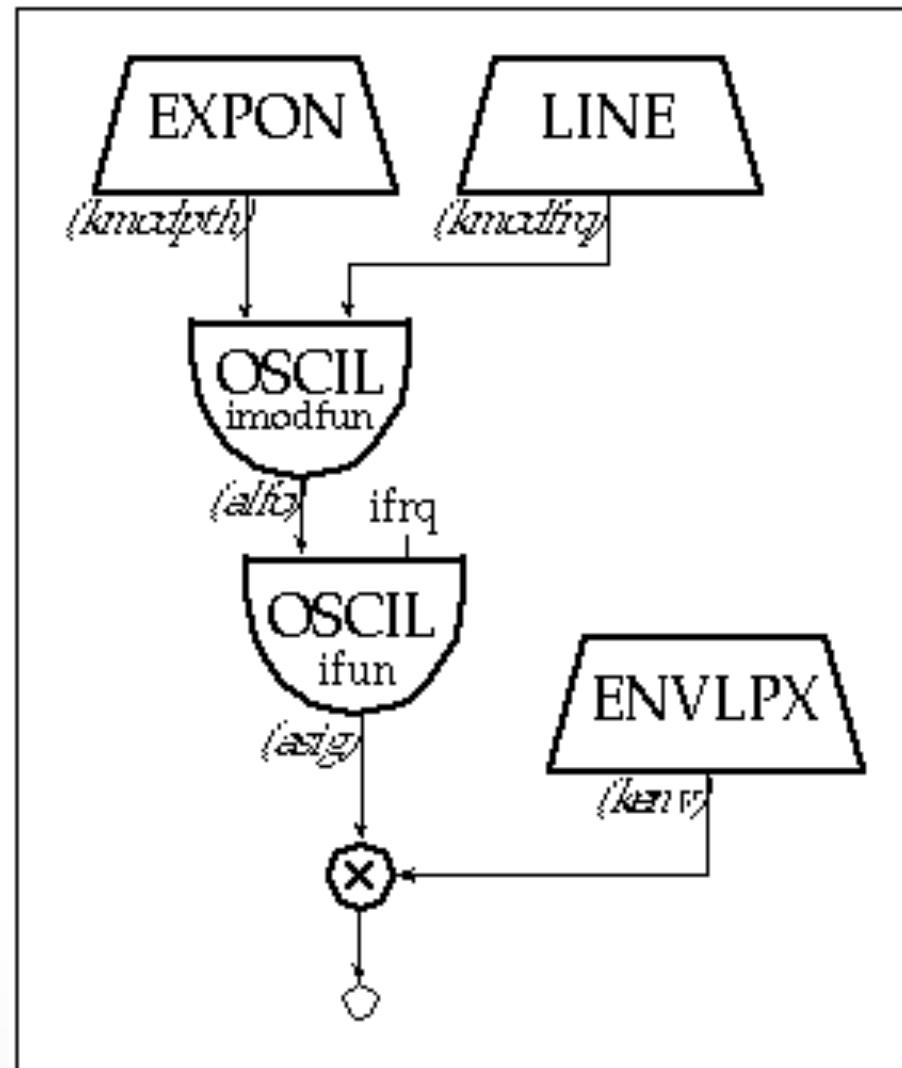
Exemple 2



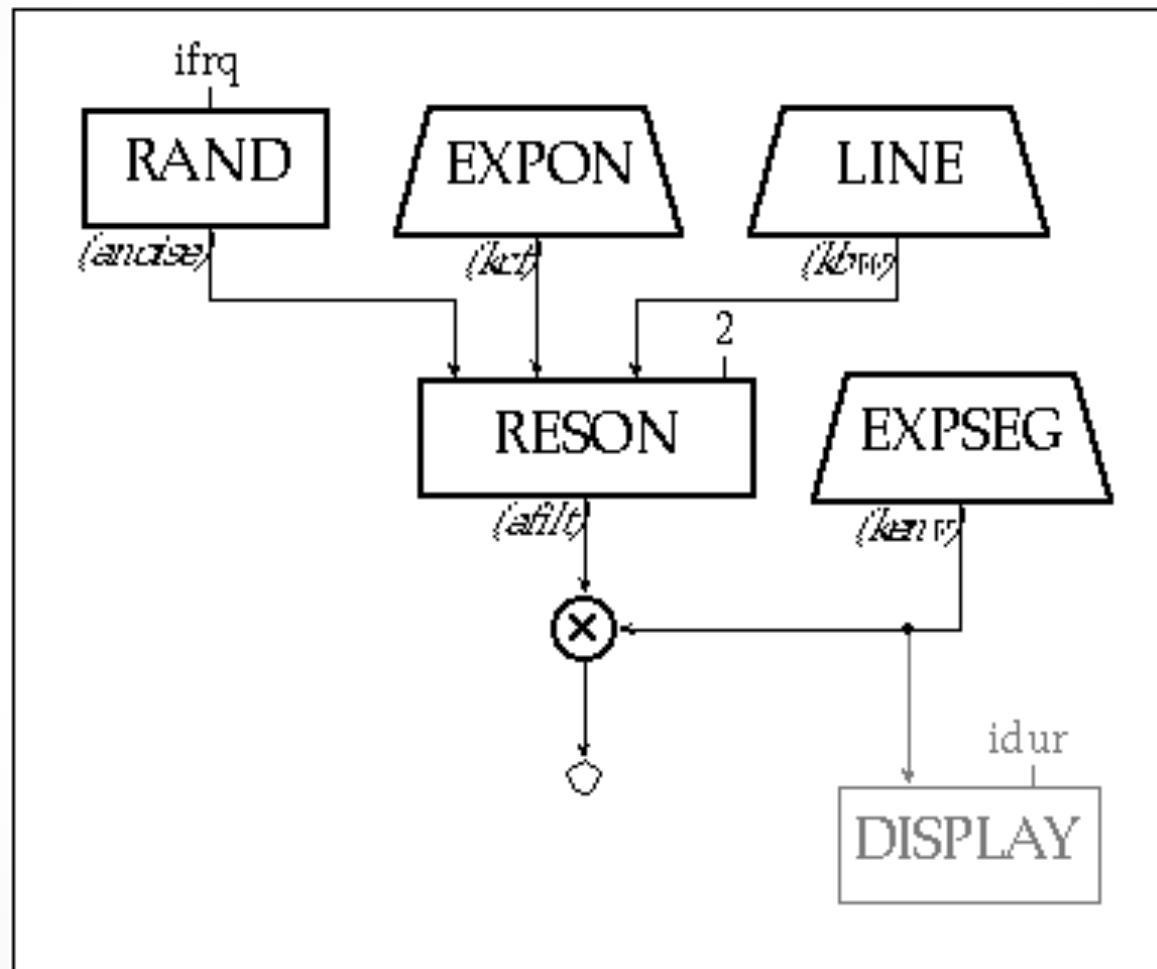
Exemple 3



Exemple 4



Exemple 5



Exemple 6

