

Theorie des Langages

Qu'est-ce que la théorie des langages ?

Poly sur jo.fabrizio.free.fr

* Qui l'utilise ?

- Les compilateurs : pour traduire $\xrightarrow{\text{processus de lecture}}$ concentration sur la lecture en THL
- La linguistique (si le mot existe) $\xrightarrow{\text{processus de traduction/compilation}}$ entre autres pour les traductions
- La génétique

Ensemble : pas d'ordre
Suite : ordre

Qu'est-ce qu'un langage ?

C'est un ensemble de mots pouvant être infini, vide, pas (ex : l'ensemble des nombres en base 2 est infini, attention il est dénombrable) obligatoirement ordonné.

Qu'est-ce qu'un mot ?

Un mot est une suite de symboles $\forall x$ trait d'un alphabet pouvant être vide.

Qu'est-ce qu'un alphabet ? mot $\in \Sigma^*$ est classe

Un alphabet est un ensemble de symboles ne pouvant pas être vide ou infini.

$\Delta \quad \{\epsilon\} \neq \emptyset$
Langage contient langage vide
 ϵ mot vide

Σ^* c'est l'ensemble alphabet créé avec tout le mot qui on peut créer dans Σ

Les opérations sur les mots

o) La concaténation : non commutatif, associatif, unitaire (le mot vide n'a rien), non symétrisable \Leftrightarrow c'est un monoïde
ex : $\forall a, b \in \Sigma^*, \forall (a, b) \in \Sigma^{*2}, |a.b| = |a| + |b|$

Qu'est-ce qu'un préfixe ? ex : pref(abcd) = { $\epsilon, a, ab, abc, abcd$ }

Qu'est-ce qu'un suffixe ? ex : suff(abcd) = {abcd, bcd, cd, d, ϵ }

$(L_1, L_2) \subset \Sigma^{*2}, L_1 \cdot L_2 = \{ \forall x \in \Sigma^*, \exists (a, b) \in L_1 \times L_2, x = ab \}$

ex : $\begin{array}{c} \bullet ab \\ \bullet cd \end{array} \sim L_1 \quad \begin{array}{c} \bullet aa \\ \bullet bb \end{array} \sim L_2 = \{ abaa, abbb, cd aa, cd bb \}$
Concaténation L_1 à L_2 (\Rightarrow chercher pref(a).suff(b))

$\forall n \in \mathbb{N}, \forall a \in \Sigma^*, a^n = \begin{cases} a.a.a \dots a & \text{si } n \neq 0 \\ \epsilon & \text{si } n=0 \end{cases}$

$\forall n \in \mathbb{N}, \forall L \subset \Sigma^*, L^n = \begin{cases} L \cdot L \cdot L \dots \cdot L & \text{si } n \neq 0 \\ \{\epsilon\} & \text{si } n=0 \end{cases}$

⑧ $\forall L \subset \Sigma^*$; $L^* = \bigcup_{i=0}^{\infty} L^i = L^\circ \cdot UL^* \cdot UL^2 \cdot UL^3 \dots$

symbol
3 char, 4 ...

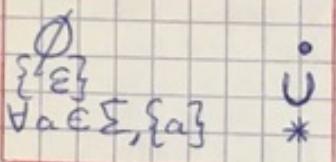
ex: Language de la base 2: $(\{0\} \cup \{1\})^* = \{\epsilon\} \cup \{0\} \cup \{1\} \cup \{00\} \cup \{01\} \cup \{10\} \cup \{11\} \dots$

Sans ϵ : $(\{0\} \cup \{1\})^* = (\{0\} \cup \{1\})^*$

Soit $L^+ = L \cdot L^* = \bigcup_{i=1}^{\infty} L^i$

+1 $L^+ = L \cdot L^* = \bigcup_{i=1}^{\infty} L^i$

U union



Représentation des langages rationnels

Language récursivement énumérable: Language qui peut être listé avec un algorithme

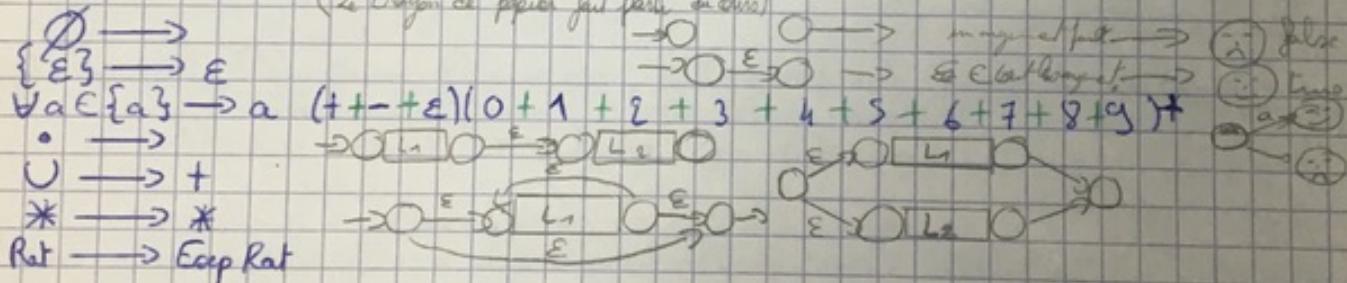
Language récursif: si un algorithme peut tester dans un temps fini si un mot appartient à un language.

definition d'une distance:

$$\begin{aligned} d: E \times E &\rightarrow \mathbb{R}^+ \\ \forall (a, b) \in E^2, d(a, b) &= 0 \Leftrightarrow a = b \\ \forall (a, b, c) \in E^3, d(a, b) &\leq d(a, c) + d(c, b) \\ \forall (a, b) \in E^2, d(a, b) &= d(b, a) \end{aligned}$$

ex: Language de la base de 10: $(\{+\} \cup \{-\} \cup \{\epsilon\}) (\{0\} \cup \{1\} \cup \{2\} \cup \{3\} \cup \{4\} \cup \{5\} \cup \{6\} \cup \{7\} \cup \{8\} \cup \{9\})^+$

(Le crayon de papier fait parti du langage)

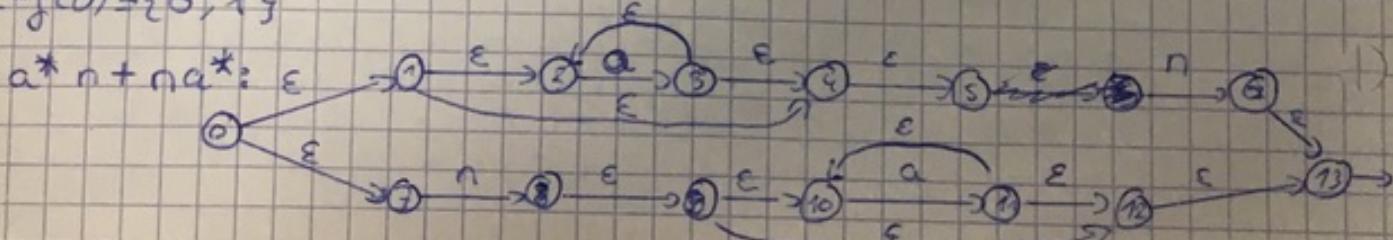


On peut aussi l'écrire $[-+]^* [0-9]^*$

On crée à chaque un automate non déterministe à état fini à transition spontanée (ϵ , Q, I, F) ensemble de sorties ensemble d'entrée S → ensemble de transitions qui permet de franchir les barrières

La construction de l'automate est l'algorithme de Thompson

ϵ -fermeture
 $\epsilon-f(O) = \{0, 1\}$

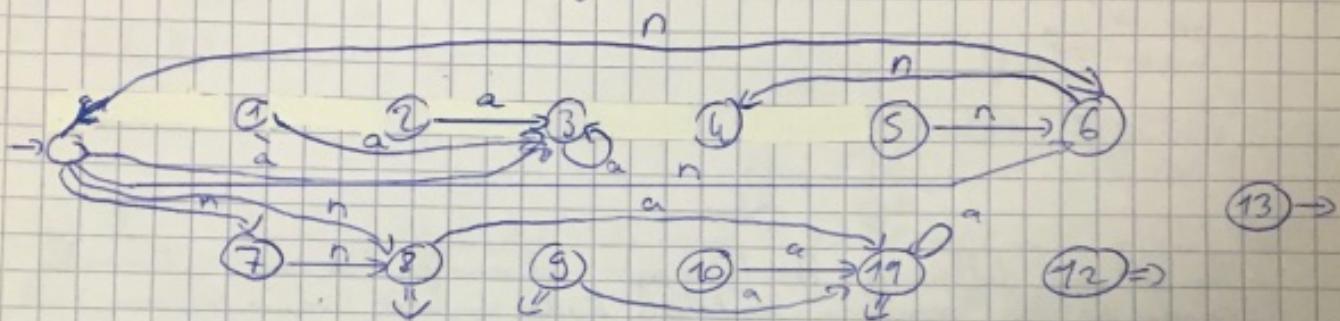


$$\Sigma - \delta(0) = \{0, 1, 2, 4, 5, 7\}$$

Théorie des langages (3)

$$\Sigma - \delta(0) = \{0, 1, 7, \cup \Sigma - \delta(1) \cup \Sigma - \delta(7)\}$$

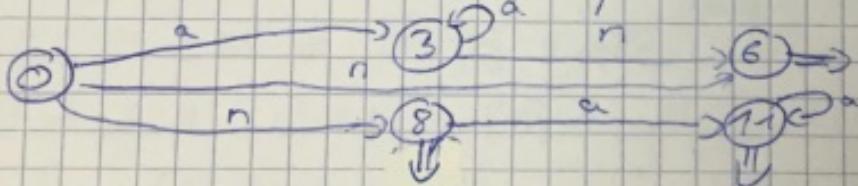
0	1	2	3
0	0, 1, 7	0, 1, 2, 4, 7	0, 1, 2, 4, 5, 7
1	1, 2, 4	1, 2, 4, 5	1, 2, 4, 5
2	2	2	2
3	3, 2, 4	3, 2, 4, 5	3, 2, 4, 5
4	4, 5	4, 5	4, 5
5	5	5	5
6	6, 13	6, 13	6, 13
7	7	7	7
8	8, 9	8, 9, 10, 12	8, 9, 10, 12, 13
9	9, 10, 12	9, 10, 12, 13	9, 10, 12, 13
10	10	10	10
11	11, 10, 12	11, 10, 12, 13	11, 10, 12, 13
12	12, 13	12, 13	12, 13
13	13	13	13



Un état est dit accessible si il existe un chemin depuis l'entrée jusqu'à cet état.

Un état est dit co-accessible si il existe un chemin depuis l'état jusqu'à une sortie.

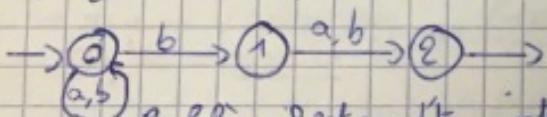
Un état est dit utile si il est à la fois accessible et co-accessible.



Conclusion : Thompson créer des automates compliqués.

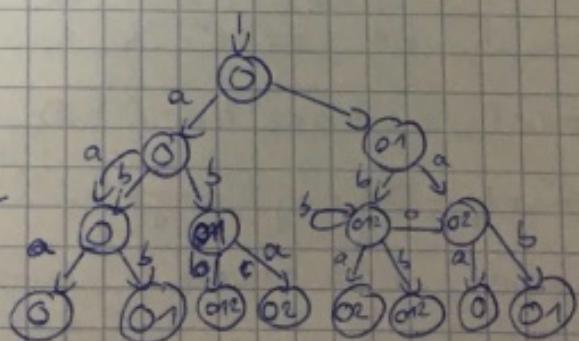
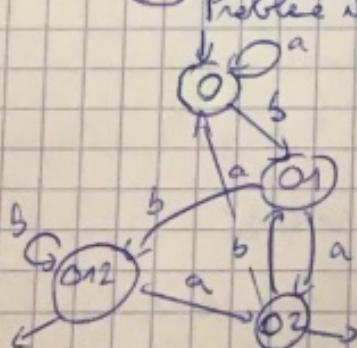
Chen et Liawne & Ratin elles → Expresses Ratin elles → ϵ -NFA → NFA (Σ, Q, I, F, δ)

$$\Sigma = \{a, b\} \quad (a+b)^* b (a+b)$$



Problème il est non déterministe

En retournant les états inutiles (



6) Théorie des langages

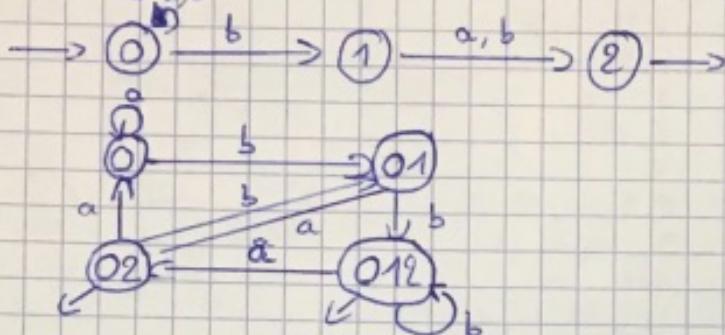
Cours 3:

→ Rat. \rightarrow Expr. Rat. \rightarrow E-NFA \rightarrow NFA \rightarrow DFA (Σ, Q, i, F, δ)

$$\{a, b\}^*$$

$$(Q \times \Sigma) \rightarrow Q$$

$$\Sigma = \{a, b\}$$

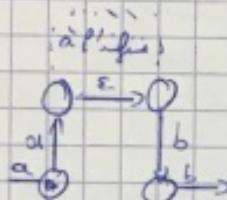


	a	b
0	0	01
01	02	012
02	0	01
012	02	012

$$e \leftarrow i$$

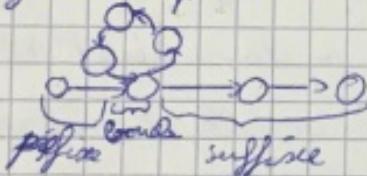
$$[s \leftarrow \overline{T}[e, s]$$

$$a^n b^n \in \Sigma^*, n \in \mathbb{N}$$



On dit donc différencier les langages rationnels des non-rationnels afin de savoir si notre expression peut être faite de façon linéaire ou non.

Si le langage n'est pas rationnel, on peut avoir une boucle exemple :



$$x \in L$$

$$\exists k \in \mathbb{N} \quad |x| > k$$

$$x = uvw$$

$$|uv| < k \quad |v| > 1$$

$$\forall i \in \mathbb{N}$$

$$uvw^i \in L$$

$$a^n b^n \quad n \in \mathbb{N} \quad a \vdash a b \vdash b$$

$$u = a^i \quad v = b^j \quad w = a^{n-i} b^{n-j}$$

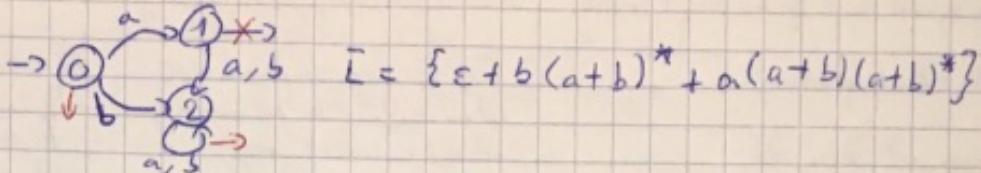
$$|uvw|_a = i + n - i + n - i + 0$$

$$|uvw|_b = 0 + n - j + n - j + j$$

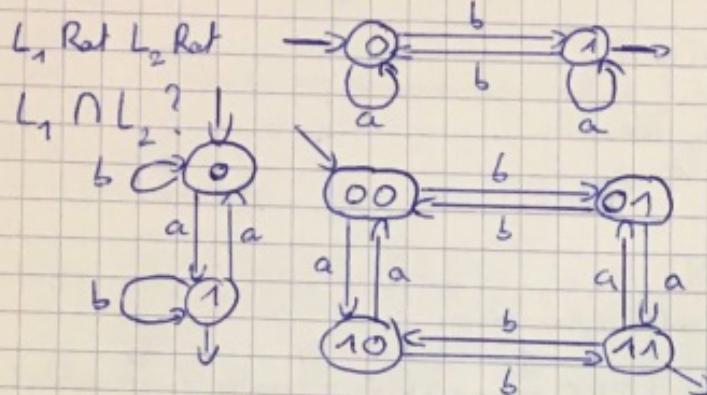
$$|uvw|_a \neq |uvw|_b \quad uvw \notin L$$

$$\text{Si } i=j : |uvw|_a = |uvw|_b \text{ , par contre } uvw \notin L$$

⚠ Si la pomme de pompage ne fonctionne pas, alors le langage n'est pas rationnel, l'inverse n'est pas vrai.



$L \text{ Rat} \Rightarrow \bar{L} \text{ Rat}$



$L_1 \text{ Rat} L_2 \text{ Rat} \Rightarrow L_1 \cap L_2 \text{ Rat}$

Autre preuve : $L_1 \cap L_2 = \bar{L}_1 \cap \bar{L}_2 = \bar{L}_1 \cup \bar{L}_2$ $\bar{L}_1 \text{ Rat} \bar{L}_2 \text{ Rat} \Rightarrow \bar{L}_1 \cup \bar{L}_2 \text{ Rat} \xrightarrow{\text{Rat}} \bar{L} \text{ Rat}$
Donc $\bar{L}_1 \cup \bar{L}_2 \text{ Rat} \Rightarrow \bar{L}_1 \cap \bar{L}_2 \text{ Rat}$

$L_1 \text{ Rat } \bar{L}_2$? $L_1 \subset L_2 \not\Rightarrow L_1 \text{ Rat}$: contre exemple : $a b \text{ Rat } a b c \in "b" \text{ or } a^nb^n \not\text{Rat}$

$L_1 \text{ Rat } L_2$? $L_2 \subset L_1 \not\Rightarrow L_2 \text{ Rat}$: contre exemple : $a^n b^n \in a^* b^* \text{ or } a^* b^* \text{ Rat}$ et $a^n b^n \not\text{Rat}$

Connaitre l'algo Brzozowski McEliece qui est dans le poly

Cours 6 :

$\text{pref}(L) \Rightarrow ?$

$\forall L \subset \Sigma^* L \text{ Rat} \Rightarrow \emptyset \quad \{\epsilon\} \quad \forall a \in \Sigma \{a\} \cup \cdot \ast$

$\emptyset \Rightarrow \text{pref}(\emptyset) = \emptyset \text{ Rat} \quad \{\epsilon\} \Rightarrow \text{pref}(\{\epsilon\}) = \{\epsilon\} \text{ Rat}$

$\{a\} \Rightarrow \text{pref}(\{a\}) = \{\epsilon + a\} \text{ Rat}$

Hypothèse : $\forall L_1 \subset \Sigma^* L_2 \subset \Sigma^*$ $L_1, L_2 \text{ Rat}$, $\text{pref}(L_1)$ et $\text{pref}(L_2)$ Rat.

$\text{pref}(L_1 \cup L_2) = \underbrace{\text{pref}(L_1)}_{\text{Rat par hyp}} \cup \underbrace{\text{pref}(L_2)}_{\text{Rat par hyp}}$

$\text{pref}(L_1 \cdot L_2) = \underbrace{\text{pref}(L_1)}_{\text{Rat}} \cup \underbrace{\text{pref}(L_2) L_1}_{\text{Rat/Rat}}$

$\text{pref}(L_1^*) = \underbrace{L_1^*}_{\text{Rat}} \cdot \underbrace{\text{pref}(L_1)}_{\text{Rat}}$

Par récurrence, on prouve que $\text{pref}(L)$ est rationnel.

⑥ Exemple de langage :

Une phrase est de la forme sujet verbe $\Sigma = \{\text{il, elle, parle, écoute}\}$

Un sujet est un pronom

Ce langage comporte le moté :
 (les mots sont des phrases d'épis)
 (la définition du langage)

Un pronom est il ou elle

il écoute
 elle écoute
 il parle
 elle parle

Un verbe est écouter ou parler

$N = \{\text{phrase, verbe, pronom, sujet}\}$

P \rightarrow SV

$(N\Sigma)^* \rightarrow (N\Sigma)^*$

S \rightarrow PN

$\Sigma = V = N\Sigma$

PN \rightarrow il | elle

List-arg \rightarrow arg List

V \rightarrow parle | écoute

List \rightarrow , | List arg

A \rightarrow aABC

A \rightarrow abC

CB \rightarrow BC

BB \rightarrow bb

bC \rightarrow bc

CC \rightarrow cc

Quel langage est engendré par cette grammaire ?

Axiome : A $\xrightarrow{①} aABC \xrightarrow{②} aa bCB C \xrightarrow{③} aab BCC \xrightarrow{④} aabb CCC \xrightarrow{⑤} aabbccC$

A $\xrightarrow{②} ab C \xrightarrow{③} a bc$

Le langage engendré est $a^n b^n c^n$

On peut donc engendré $a^n b^n c^n$ avec une grammaire ce qui était impossible avec les outils d'avant.

Pour faire $a^n b^n$: Z $\rightarrow aZb \mid \epsilon$

S \rightarrow Name | List End

Name \rightarrow 'ceriel' | 'dick' | 'soam'

List \rightarrow Name | Name' | List

' | ' Name End \rightarrow ' and' Name. Puis que le langage soit monotone (' and' Name \rightarrow and' Name)

Axiome : S \rightarrow List End \rightarrow Name | ' List End \rightarrow Name | ' Name End \rightarrow Name and' Name

Les contraintes sur les grammaires:

Monotonie : $\alpha \rightarrow \beta \quad |\alpha| \leq |\beta|$

Contexte Sensitive : $\alpha AB \xrightarrow{\downarrow \downarrow \downarrow} \alpha S \beta$

Contexte Sensitive \Rightarrow Monotone

✓

$a^n b^n c^n$

$\left\{ \begin{array}{l} S \rightarrow abc \\ S \rightarrow aSQ \\ bQC \rightarrow bLCC \\ CQ \rightarrow QC \\ C \rightarrow c \end{array} \right.$	Monotone 1 ≤ 3 1 ≤ 3 3 ≤ 4 2 ≤ 2 Pas Contexte Sensitive
---	--

Contexte Sensitive

Contexte Sensitive

Contexte Sensitive

Contexte Sensitive

Pas Contexte Sensitive

$\left\{ \begin{array}{l} S \rightarrow abc \\ S \rightarrow aSQ \\ bQC \rightarrow bLCC \\ CQ \rightarrow QC \\ C \rightarrow c \end{array} \right.$	$CQ \rightarrow XQ$ $\Sigma QL \rightarrow X_1 QL$ $XC \rightarrow QC$
---	--

Cours 5:

Une grammaire se définit par: (N, Σ, P, S)

Vocabulaire $V^+ \rightarrow V^*$

Pour que une grammaire soit monotone: contraint de monotone $\gamma \rightarrow \delta \quad |\gamma| \leq |\delta|$

Pour que une grammaire soit contexte sensible: $\gamma A \delta \in V^* \rightarrow \gamma \alpha \delta \in V^*$

Pour que une grammaire soit contexte libre: $A \rightarrow \infty$
Hors contexte \downarrow $\in N$ \downarrow V^+

Exemple:

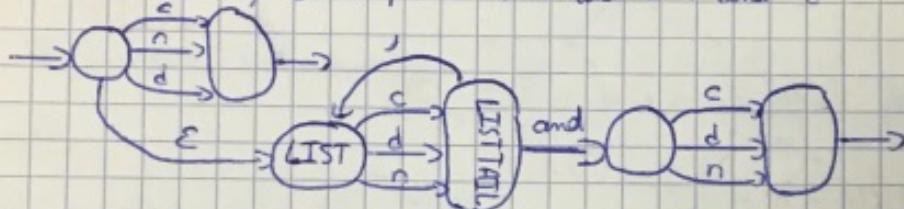
$S \rightarrow \text{NAME} \mid \text{LIST} \mid \text{and} \mid \text{NAME}$
 $\text{LIST} \rightarrow \text{NAME} \mid \text{LIST} \mid \text{NAME}$
 $\text{NAME} \rightarrow 'c' \mid 'd' \mid 'n'$

On réécrit la grammaire pour avoir une représentation plus simple:

$S \rightarrow 'c' \mid 'd' \mid 'n' \mid \text{LIST}$

$\text{LIST} \rightarrow 'c' \mid \text{LISTTAIL} \mid 'd' \mid \text{LISTTAIL} \mid 'n' \mid \text{LISTTAIL}$

$\text{LISTTAIL} \rightarrow 'c' \mid \text{LIST} \mid \text{and} \mid 'n' \mid \text{and} \mid 'c' \mid \text{and} \mid 'd'$



Une grammaire linéaire à un non-terminal au plus et l'autre du même côté.

exemple: $A \rightarrow \alpha B$
 $B \rightarrow \beta \epsilon \rightarrow 1 \in N$
 $C \rightarrow \beta$
 $C \epsilon^*$

Les types de grammaires:

	Type 0	
Type 1	monotone contexte sensible	$\rightarrow a^n b^n c^n$
Type 2	Hors contexte	$\rightarrow a^n b^n$
Type 3	Rationnel	$\rightarrow a^n b^m$
Type 4	Choise Finie	$\rightarrow a^n b^m$ Grammaire Linéaire = Grammaire Rationnel

Hiérarchie Chomsky (ou by)

Pour reconnaître un langage de type 3, il faut un automate à état fini.

Pour reconnaître un langage de type 1, il faut un automate Camé à pile et

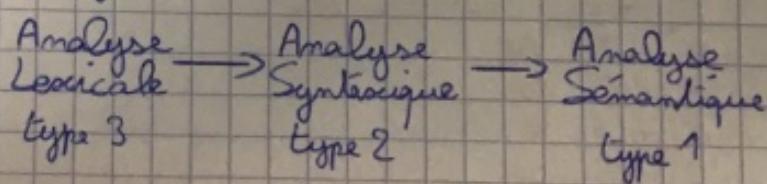
Pour reconnaître un langage de type 0, il faut une machine de Turing (machine à mémoire illimitée)

Pour reconnaître un langage de type 2, il faut un automate à pile.

Déterminer le type de Chomsky de la grammaire puis du langage car:

type Langage \geq Type grammaire

Le débit d'un compilateur:



③

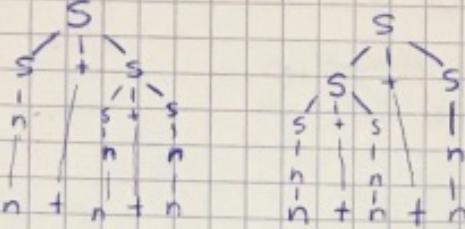
$\langle A \rangle ::= \langle B \rangle C$ BNF Back plus Non Faisable Possible dans RegLien

Si on ajoute $A \rightarrow E$ on reste sur un type 2 (strict)

Exemple: ① $S \rightarrow S' + S$
 ② $S \rightarrow 'n'$

Langage engendré $n(t+n)^*$ (grammaire type 2, Brugge type 3)

$n + n + n$:



$$\begin{aligned} \text{fert } a &= 1000 \dots 000 \\ \text{fert } b &: 0,000 \dots 0001 \\ \text{fert } c &: A+B-A=0 \\ \text{fert } d &: A-A+B=0,000 \dots 0001 \\ \text{On privilieze donc le deuxième arbre.} \end{aligned}$$

Deux arbres de dérivation différents portent vers une grammaire ambiguë.

Cette grammaire laisse donc beaucoup d'arbres possibles, on réfléchit donc à comment extraire le problème:

$$\begin{aligned} S &\rightarrow S_+ | S_n \\ S_+ &\rightarrow S_+ + S_n \\ S_n &\rightarrow n \end{aligned}$$

$$\text{Test: } S_+ + S_+ (((n + n) + (n + n))) \text{ OK}$$

$$S_+ + S_n (((n + n) + n)) \text{ OK}$$

$$S_n + S_+$$

$$S_n + S_n \quad n \neq n \text{ OK}$$

(1): résultat
 (2): voulu

$$((((n + (n)) + n))) \text{ OK}$$

Pour lever les ambiguïtés, on a donc:

$$\begin{aligned} S &\rightarrow S_+ | S_n \\ S_+ &\rightarrow S_+ + S_n | S_n + S_n \quad (=) \\ S_n &\rightarrow n \end{aligned} \quad \begin{aligned} S &\rightarrow S_+ | S_n \\ S_+ &\rightarrow S_+ + S_n \quad (=) \\ S_n &\rightarrow n \end{aligned} \quad \begin{aligned} S &\rightarrow S + S_n | S_n \\ S_n &\rightarrow n \end{aligned}$$

Exemple: $S \rightarrow S' + 'S$
 $S \rightarrow S' * 'S$
 $S \rightarrow S_n$

On a des ambiguïtés sur les additions à chaîne, les multiplications à chaîne et le mélange addition/multiplication, on créer donc:

$$\begin{aligned} S &\rightarrow S_+ | S_n | S_n \\ S_+ &\rightarrow S_+ + S \\ S_+ &\rightarrow S_+ * S \\ S_n &\rightarrow n \end{aligned}$$

$$\begin{aligned} \text{Test: } S_+ + S_+ ((+1 + 1)) &\text{ OK} \\ S_+ + S_* ((+1 + 1) * (+1)) &\text{ OK, ou: } (1+2) + (1*3) \\ S_+ + S_n &\text{ OK} \\ S_* + S_+ (((*1 + 1) * (+1))) &\text{ OK} \\ S_* + S_* &\text{ OK} \\ S_* + S_n &\text{ OK} \\ S_n + S_* &\text{ OK} \\ S_n + S_* &\text{ OK} \\ S_n + S_n &\text{ OK} \end{aligned}$$

$$\begin{aligned} S_* &\ast S_+ \text{ OK} \\ S_* &\ast S_* \text{ OK} \\ S_* &\ast S_n \text{ OK} \\ S_* &\ast S_+ \text{ OK} \\ S_* &\ast S_n \text{ OK} \\ S_* &\ast S_n \text{ OK} \\ S_n &\ast S_+ \text{ OK} \\ S_n &\ast S_* \text{ OK} \\ S_n &\ast S_n \text{ OK} \end{aligned}$$

$$\begin{aligned} S &\rightarrow S_+ \mid S_* \mid S_n \\ S_+ &\rightarrow S_+ + S_* \mid S_n + S_n \mid S_* + S_n \mid S_n + S_* \mid S_n \\ S_* &\rightarrow S_* * S_n \mid S_n * S_n \end{aligned}$$

Théorie des langages

⑨

$$\begin{aligned} S &\rightarrow S_+ \mid S_* \mid S_n \\ S_+ &\rightarrow S_+ + S_* \mid S_n + S_n \\ S_* &\rightarrow S_* * S_n \mid S_n * S_n \\ T &\rightarrow S_* \mid S_n \\ S &\rightarrow S_+ \mid T \\ S_* &\rightarrow T \mid S_n \\ S_+ &\rightarrow S + T \\ S_n &\rightarrow n \\ S &\rightarrow S + T \mid T \\ T &\rightarrow T \mid S_n \\ S_n &\rightarrow n \mid ('S') \mid '-' \mid S_n \mid \text{sgnt}('S') \end{aligned}$$

$\text{INST-IF} \rightarrow \text{'if' EXP 'then' INST}$
else INST

Les parseurs:

Il y en a trois autres deux types LR → left to right



→ left to right

Exemple: LL(0) → derived at predictively Left mode derivation

$\text{INST} \rightarrow \text{'if' EXP 'then' INST}$
 $\text{INST} \rightarrow \text{'print' 'for'}$
 $\text{EXP} \rightarrow \text{'true' } \mid \text{'false'}$

Grammaire type 2

Language engendré: $(\text{if}(\text{true} \mid \text{false}) \text{then})^* \text{ print for }$ type 3

- ① $\text{INST} \rightarrow \text{'if' EXP 'then' INST 'fi'}$
- ② $\text{INST} \rightarrow \text{'print' 'for'}$
- ③ $\text{EXP} \rightarrow \text{'true' } \mid \text{'false'}$

	if	print	true	false
INST	①	②		
EXP		③	④	

$\text{FIRST}(\text{INST}) = \{\text{'if'}, \text{'print'}\}$

- ① $\text{INST} \rightarrow \text{'if' EXP 'then' INST 'fi'}$
- ② $\text{INST} \rightarrow \text{'print' 'for'}$
- ③ $\text{EXP} \rightarrow \text{COND}$
- ④ $\text{COND} \rightarrow \text{'true' } \mid \text{'false'}$

	if	print	true	false
INST	①	②		
EXP		③	④	
COND		⑤	⑥	⑦

$\text{First}(\text{EXP}) = \text{First}(\text{COND}) = \{\text{'true' } \mid \text{'false' }\}$

10) Cours n°6 :

- ① $\text{stmt} \rightarrow \text{'if' expr 'then' stmt 'fi'}$
- ② $\text{stmt} \rightarrow \text{'print' 'for'}$
- ③ $\text{expr} \rightarrow \text{neg cond} \rightarrow \text{First(expr)} \cup \text{First(neg cond)} = \text{First(neg)} \cup \text{First(cond)} = \{\text{not, true, false}\}$
- ④ $\text{cond} \rightarrow \text{'true' / 'false'}$
- ⑤ $\text{neg} \rightarrow \text{'not' neg} \mid \epsilon \rightarrow \text{Follow(neg)} = \text{First(cond)}$

LL(R)

i

		First			
		if	print	true	false
LL(1)	stmt	1	2		
	expr			3	3
	cond			4	5
	neg			7	7
					6

$$\textcircled{1} Z \rightarrow XYZ \rightarrow \text{FIRST}(Z) \cap \text{FIRST}(XYZ) = \text{FIRST}(X) \cup \text{FIRST}(Y) \cup \text{FIRST}(Z) *$$

$$\textcircled{2} Z \rightarrow d \rightarrow \text{FIRST}(Z) \cap \{d\} \rightarrow \emptyset \quad \alpha \in \text{NULL}$$

$$\textcircled{3} Y \rightarrow c \rightarrow \text{FIRST}(Y) \cap \{c\} \rightarrow \emptyset$$

$$\textcircled{4} Y \rightarrow \epsilon \rightarrow \emptyset \quad (\text{car } \epsilon \text{ est dans le First}) \rightarrow \emptyset$$

$$\textcircled{5} X \rightarrow Y \rightarrow \text{FIRST}(x) \cap \text{FIRST}(Y) \rightarrow \text{Follow}(Y) \cap \text{Follow}(X) *$$

$$\textcircled{6} X \rightarrow a \rightarrow \text{FIRST}(x) \cap \{a\} \rightarrow \emptyset \quad (\text{car } a \text{ pertient pas dans } \text{Follow}(Y))$$

$$\text{NULL} = \{Y, X\}$$

$$\text{FIRST}(X) = \{a, c\}$$

$$\text{Follow}(X) = \{a, c, d\}$$

$$\text{FIRST}(Y) = \{c\}$$

$$\text{Follow}(Y) = \{a, c, d\}$$

$$\text{FIRST}(Z) = \{d, a, c\}$$

$$\text{Follow}(Z) = \emptyset$$

$$* \rightarrow \text{Follow}(X) \cap \text{First}(YZ) = \text{First}(Y) \cup \text{First}(Z)$$

$$\rightarrow \text{Follow}(Y) \cap \text{First}(Z)$$

$$\rightarrow \text{Follow}(Z) \cap \text{Follow}(Z)$$

	First(x)	Follow(A)
A	$A \rightarrow x$	$A \sim \epsilon E$

	a	c	d
x	$\frac{a}{c} \frac{a}{c}$	$\frac{c}{c} \frac{c}{c}$	$\frac{c}{c}$
y	$\frac{a}{c}$	$\frac{c}{c}$	$\frac{c}{c}$
z	$\frac{a}{c}$	$\frac{c}{c}$	$\frac{c}{c}$

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow n$$

Problème pour faire le parseur car on ne peut pas calculer le first.

$$E \rightarrow E + T \rightarrow E + T + T \rightarrow E + T + T + T \rightarrow T + T + T + T$$

$$(T + T)^* \rightarrow T + T + \dots + T \rightarrow \underbrace{T + T + \dots + T}_{T + T}^*$$

$$E \rightarrow T + E \mid T$$

$$\text{Donc } T \rightarrow F * T \mid T$$

$$F \rightarrow n$$

On a donc : (Ce n'est pas une grammaire car on a des expressions anglaises.)

$$E \rightarrow T + T^*$$

$$T \rightarrow F (*F)^*$$

$$F \rightarrow n$$

```

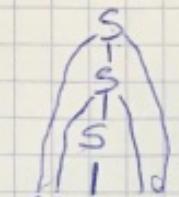
void parse_E() {
    NODE t = parse_T();
    while (get_next_token() == TOK_PLUS) {
        eat('+');
        NODE tp;
        tp = parse_T();
        NODE c = new_node(t, '+', tp);
    }
    return t;
}

NODE void parse_T() {
    NODE f = parse_F();
    while (get_next_token() == TOK_STAR) {
        eat('*');
        NODE fp;
        fp = parse_F();
        NODE p = newnode(f, '*', fp);
    }
    return p;
}

NODE void parse_F() {
    NODE n = newleaf_number();
    return n;
}

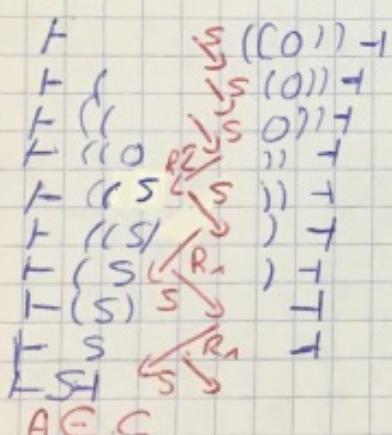
```

shunt NODE



((O))
↑↑↑↑↑↑
→ → →

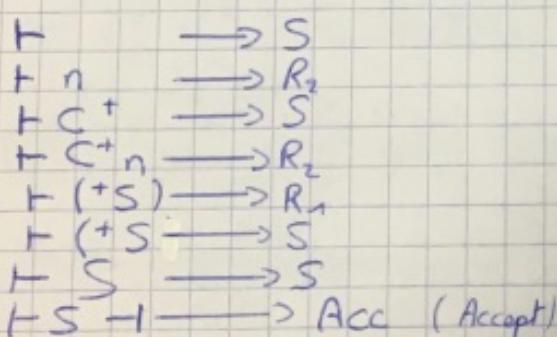
On a construit l'arbre en bas, on a commencé par notre résultat puis on remonte jusqu'à l'arbre.



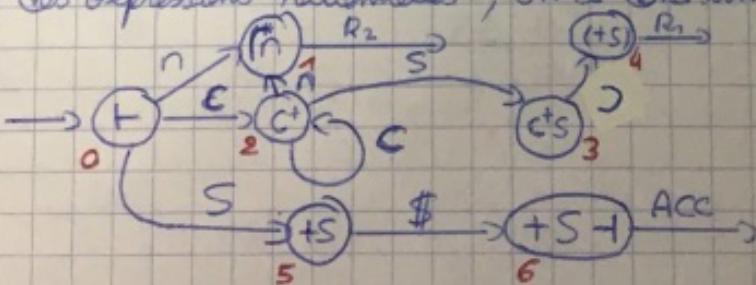
Actions:

$S \rightarrow \text{shift}$
 $R \rightarrow \text{Reduce}$

\vdash : début de la pile \dashv : fin de la pile

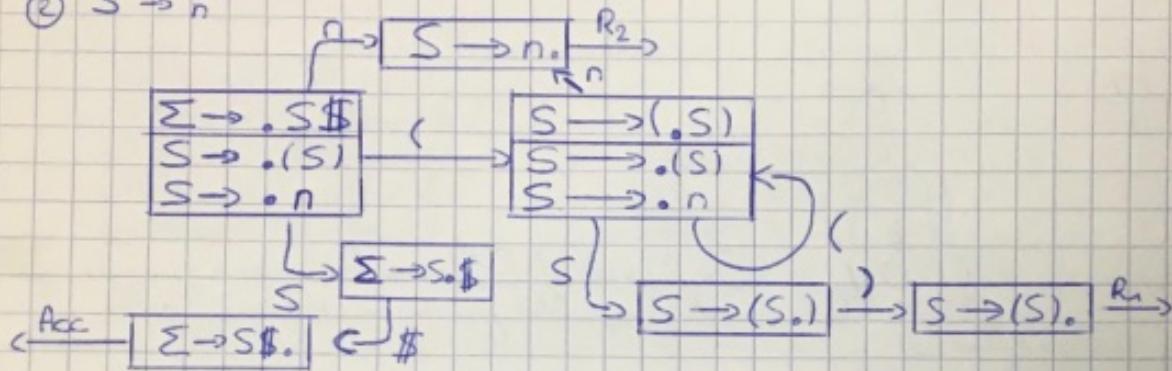


On obtient des expressions rationnelles, on a donc un automate :

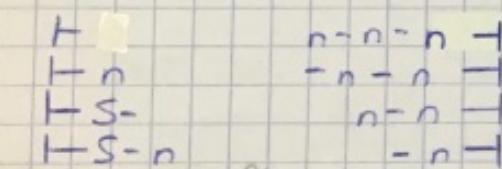


	Action	(n)	\$	S
0	S	2	1			5
1	R2					
2	S	2	1			3
3	S					
4	R1					
5	S					
6	Acc					

- ① $S \rightarrow (S)$
 - ② $S \rightarrow n$



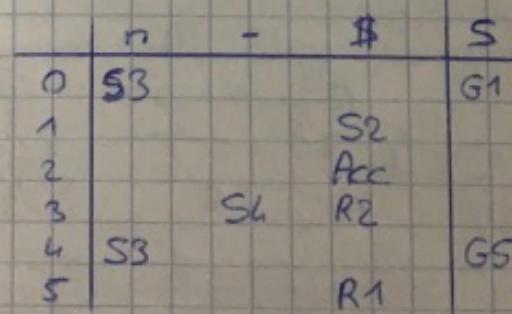
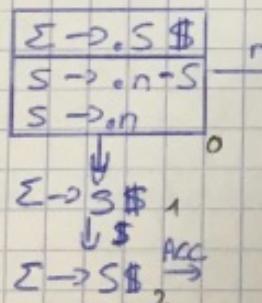
- ① $\Sigma \rightarrow S\$$
 - ② $S \rightarrow S - n$
 - ③ $S \rightarrow n$



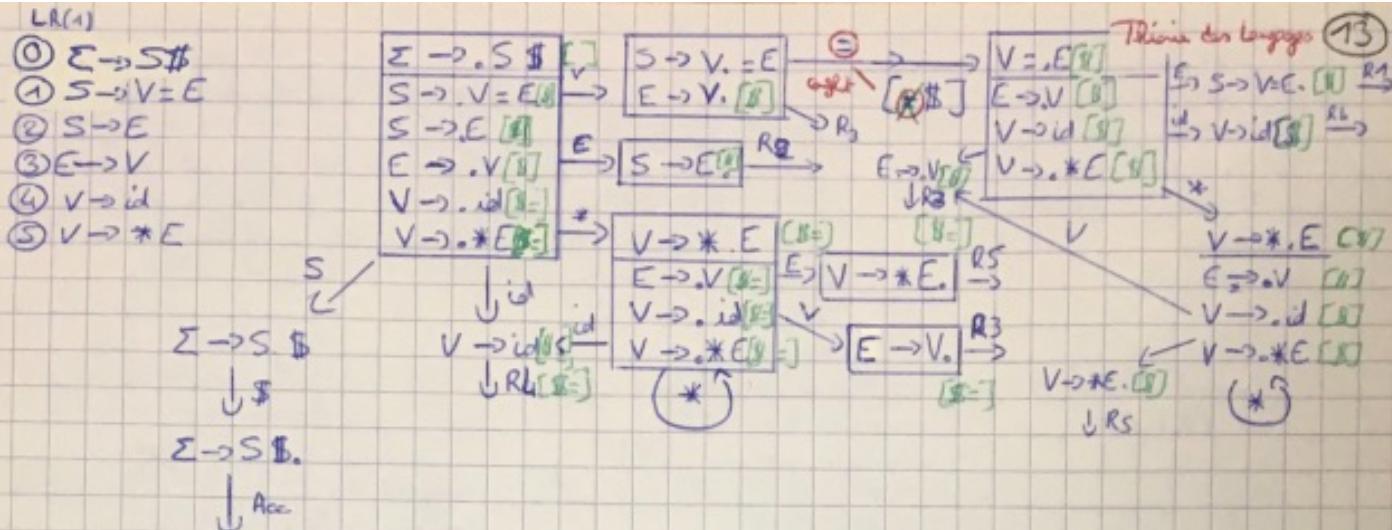
Exemplos à fin

5.8(1)

- $$\begin{array}{ll} \textcircled{O} & \sum \rightarrow S\$ \\ \textcircled{1} & S \rightarrow n - S \\ \textcircled{2} & S \rightarrow n \end{array}$$



$\text{SLR}(1)$
↓
simple



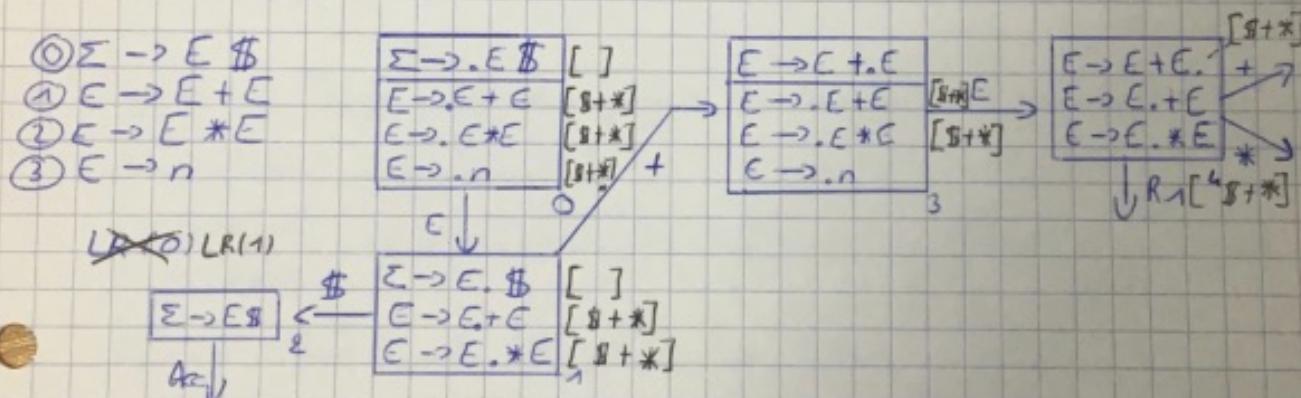
Follow(V) ⊇ {=}
Follow(S) ⊆ Follow(Σ)

$$\begin{cases} \text{Follow}(E) \subseteq \text{Follow}(V) \\ \text{Follow}(V) \subseteq \text{Follow}(E) \end{cases} \Rightarrow \text{Follow}(E) = \text{Follow}(V)$$

$$\begin{aligned} \text{Follower}(S) &= \{\$\} \\ \text{Follower}(E) &= \{=\$ \} \\ \text{Follower}(V) &= \{=\$ \} \end{aligned}$$

Résumé : On a vu le LR(0), & SLR(1) (GLR) (LR(1)) et le LR(0)

Cela crée un lexer
qui crée un parser explication (slide) sur la page du web



reduce RR PS diff
 $I - C + E$ + m-1 C diff reduce diff
bus chain B reduce: $90 \text{ Cff } +$

On peut récrire de deux façons : $E \rightarrow E + T \mid T$
 $T \rightarrow T * F \mid E$
 $F \rightarrow n$

on $E \rightarrow E + E$ /left "+"
 $E \rightarrow E * E$ /left "*"
 $E \rightarrow n$

Makah Bison:

BTSOU-*lamin*

~~BISONFLAGS=--report=all --graph --xml~~

XSLTProc = xsltproc

`XSLT PROFLAGS = $B($B(SON)) --print-data-dir /xml2xhtml.xml`

%-c % - output % - xml % - font % - it

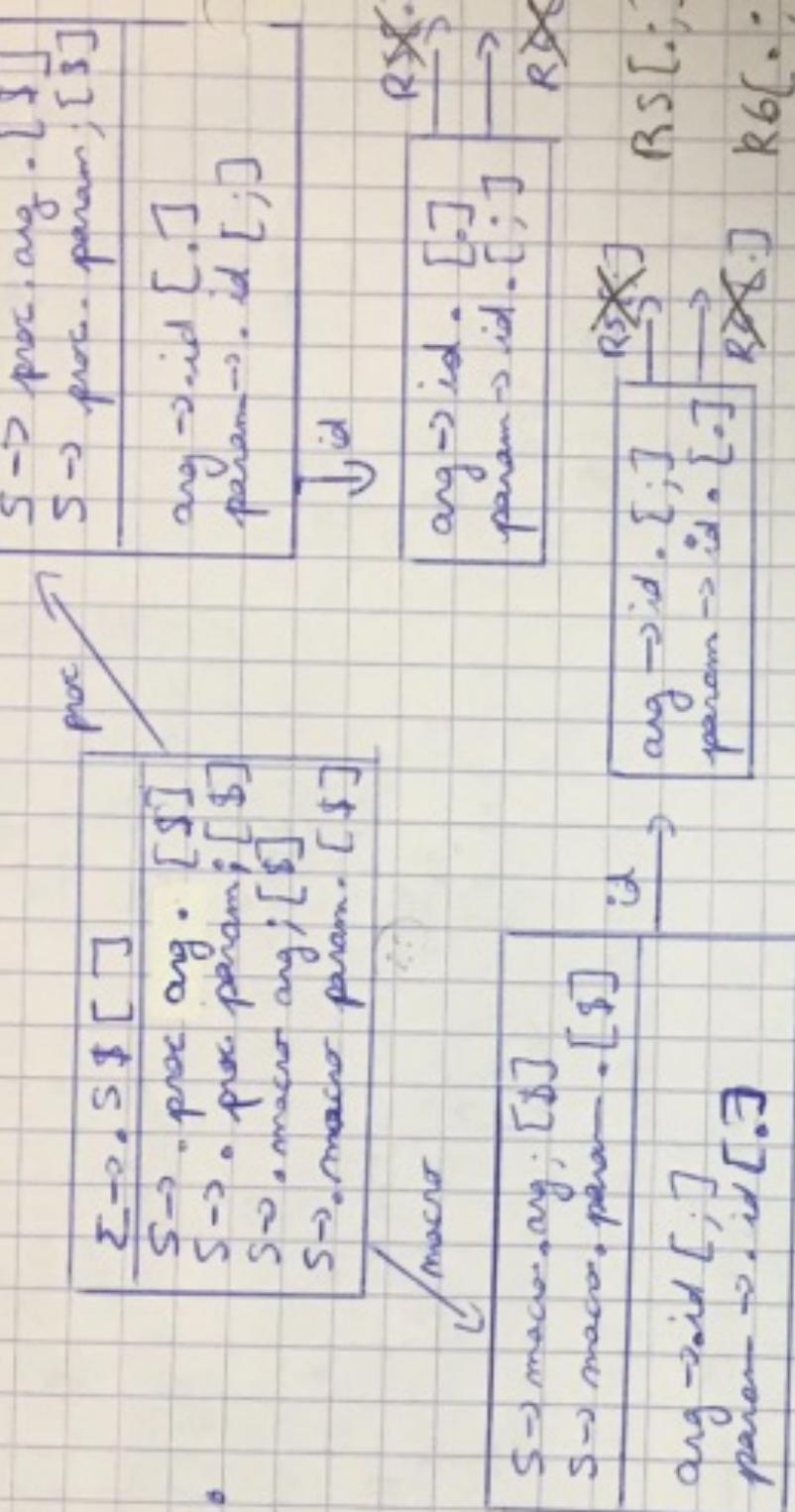
\$@ISON) \$(BISONFLAGS) \$<

~~10130~~

$$f(x) = \frac{1}{2}x^2 - \frac{1}{2}x + \frac{1}{2}$$

exit 98 sta

④ $S \rightarrow S \#$
 $S \rightarrow \text{proc array};$
 $S \rightarrow \text{proc param};$
 $S \rightarrow \text{macro arg};$
 $S \rightarrow \text{macro param} ;$
 $\text{arg} \rightarrow \text{id}$
 $\text{param} \rightarrow \text{id}$



LR(1)

LR(1)

Définition: Morphisme de Groupes

Soit $(G; \square)$ et $(H; *)$ deux groupes.

$\varphi: G \rightarrow H$ est un morphisme de groupes lorsque:

- $\forall g_1, g_2 \in G \quad \varphi(g_1 \square g_2) = \varphi(g_1) * \varphi(g_2)$
- $\varphi(e_G) = e_H$

Propriété: Si $(G; *)$ groupe et φ : morphisme de groupe du $\varphi(G)$ est un groupe.
(opération unique implicite)

Définition: φ est un isomorphisme de groupes lorsque φ est un morphisme de groupe bijectif.

Propriété: φ isomorphisme de groupe entraîne $\exists \psi$: isomorphisme de groupes avec $\varphi \circ \psi = \text{id}$
△ en général isomorphisme \neq (morphisme \wedge bijection)

Exemple:

$$\begin{pmatrix} a & b & c \\ a & b & c \\ b & a & c \\ c & c & a \end{pmatrix}$$

$$\begin{array}{c|ccc} \mathbb{Z}/3\mathbb{Z} & 0 & 1 & 2 \\ \hline 0 & 0 & 1 & 2 \\ 1 & 1 & 2 & 0 \\ 2 & 2 & 0 & 1 \end{array}$$

Notation: $(G, *) \simeq (H, \square)$ signifie il existe un isomorphisme

Propriété: \simeq relation d'équivalence

Etudions $(\mathbb{Z}/n\mathbb{Z}^*, \times)$
sans 0

Propriété: $(\mathbb{Z}/n\mathbb{Z}^*, \times)$ est un groupe si et seulement si n est premier

Démonstration: trop invraisemblable?

$$(a, b) \in \mathbb{Z}/n\mathbb{Z}^2 \quad ab \equiv 1 [n] \Leftrightarrow \exists k \in \mathbb{Z}, ab - kn = 1$$

$(a \neq 0, b \neq 0)$

Question: Quel lien entre $(\mathbb{Z}/p\mathbb{Z}^*, \times)$ et $(\mathbb{Z}/(p-1)\mathbb{Z}; +)$?

$(\mathbb{Z}/5\mathbb{Z}^*, \times)$

$(\mathbb{Z}/4\mathbb{Z}; +)$

$$\begin{array}{c|cccc} \times & 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 2 & 3 & 4 \\ 2 & 2 & 4 & 1 & 3 \\ 3 & 3 & 1 & 4 & 2 \\ 4 & 4 & 3 & 2 & 1 \end{array} \mapsto \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 2 & 3 & 4 \\ 2 & 2 & 4 & 1 & 3 \\ 3 & 3 & 1 & 4 & 2 \\ 4 & 4 & 3 & 2 & 1 \end{array}$$

$$\varphi(1) = e = \varphi(0)$$

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 1 & 2 & 3 & 4 & 5 \\ 2 & 2 & 4 & 1 & 3 & 5 \\ 3 & 3 & 1 & 4 & 2 & 5 \\ 4 & 4 & 3 & 2 & 5 & 1 \\ 5 & 5 & 1 & 3 & 4 & 2 \end{array}$$

$\varphi(4) = b = \varphi(2)$

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 1 & 2 & 3 & 4 & 5 \\ 2 & 2 & 4 & 1 & 3 & 5 \\ 3 & 3 & 1 & 4 & 2 & 5 \\ 4 & 4 & 3 & 2 & 5 & 1 \\ 5 & 5 & 1 & 3 & 4 & 2 \end{array}$$

$\varphi(3) = a = \varphi(1)$

$$\varphi(2) = c = \varphi(3)$$

$(\mathbb{Z}/5\mathbb{Z}^*, \times) \simeq (\mathbb{Z}/4\mathbb{Z}, +)$

(16)

Les anneaux:

Un ensemble A muni de deux opérations binaires $+$ et $*$ est dit anneau, noté $(A; +, *)$ lorsque:

- $(A; +)$ est un groupe commutatif. (semi-anneau)

- $*$ distributif sur $+$

$$\forall a \forall b \forall c \left\{ \begin{array}{l} a * (b+c) = a * b + a * c \\ (b+c) * a = b * a + c * a \end{array} \right.$$

- $*$ est associative

- e_+ est absorbant pour $+$ d'où la notation $0_A = e_+$

Un anneau (non minimaliste) peut être muni de contraintes permi:

- commutatif

- $*$ est commutative

- Unitaire

$$\exists 1_A \forall a \quad \overbrace{a * 1_A}^{= 1_A * a} = a \text{ unité à droite unité à gauche}$$

- À division

$$\forall a \forall b \forall c \quad \frac{\text{division à gauche}}{a * b = a * c} \Rightarrow b = c$$

division à droite

- Intégré: $\forall a \forall b \quad a * b = 0_A \Rightarrow a = 0_A \vee b = 0_A$

- Idempotence: $\forall a \quad a * a = a \quad \begin{cases} A \wedge A \leq A & a * a = a \\ A \vee A \geq A & a + a = a \end{cases}$

Définition:

Un anneau $(A; +, *)$ est dit Anneau de Boole lorsque:

- Idempotent ; commutatif et unitaire ($1 = \bar{1}$)

Exemple: $(\{0, 1\}, \vee, \wedge)$ anneau de Boole`CC = gcc``CFLAGS =``LDLFLAGS = -ll``OBJ_FILES = scan-calc.o``PRECIOUS: scan-calc.c` → empêche de le supprimer alors qu'il a été créé par une règle intermédiaire
`all: calc``calc: $(OBJ_FILES)``$(CC) $(CFLAGS) -o $@ $^ $(LDLFLAGS)``% .o: %.c``$(CC) -c $(CFLAGS) -o $@ $^``% .c: % .l``$(FLEX) -o $@ $^`