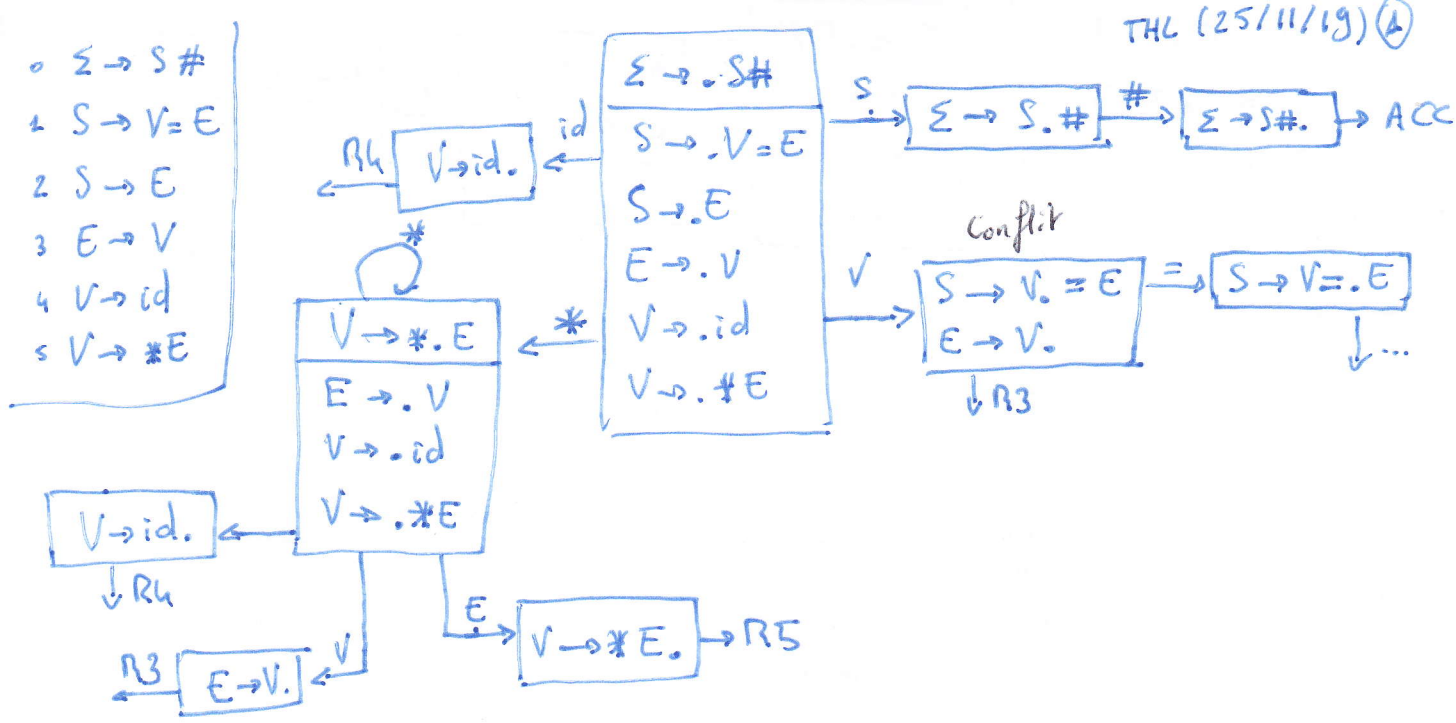
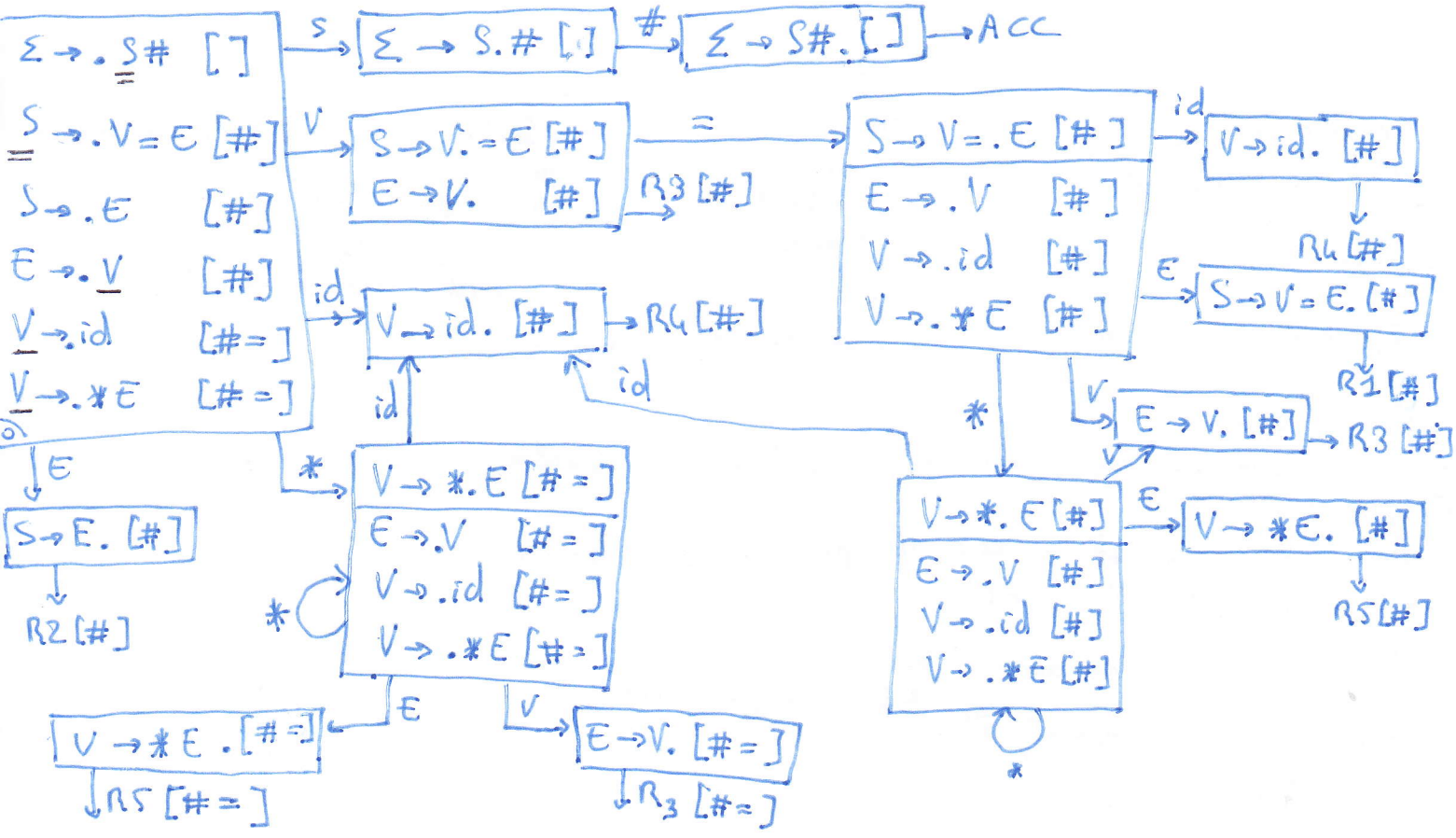


Follow(E): { =, # }

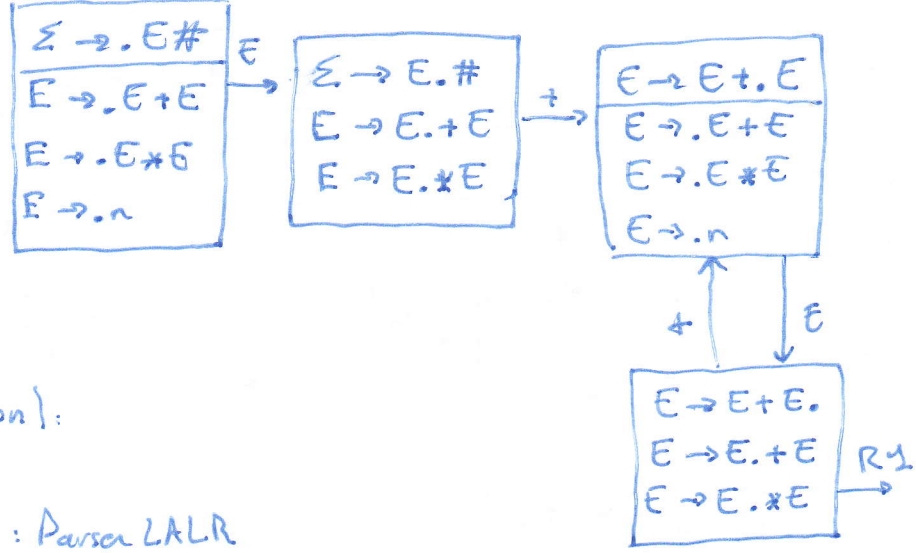


⇒ LALR(1) [Look Ahead LR(1)] (après simplification)

⇒ CLR(1) [Complex LR(1)] (schema ci-dessus)

$E \to E+E$
$E \to E*E$
$E \to n$
⇒ Grammar ambigüe

$E \to E+T \mid T$
$T \to T*F \mid F$
$F \to n$
⇒ Gram Ok



Générateur de parser (Flex et Bison):

Lex : Analyseur lexical
⇓ ↳ Language Type 3
Flex

Yacc : Parser LALR
⇓ ↳ Langage Type ≤ 2
Bison : Retrocompatible
→ Ajout d'option.

fic.l → LEX → fic.c → Compilateur → fic.o

fic.y → YACC → fic.c → Compilateur → fic.o

Bison:
- Format de fichier
  - Prologue : def + options
  %%% ← Separateur
  - Règles
  %%%
  - Epilogue : code utilisateur (C)
- Options
  --xml
  --report=all :
  --graph
- Appel
  bison [options] fic.y -o fic.c
- Sortie: Bison génère un programme du parser: int yyparse()
- Dependances:
  - int yylex();
  - void yyerror(const char *s);

- Gestion d'erreurs:
  - Message d'erreur plus explicit : dans le prologue ⇒ '%define parse.error verbose'
- Conflits shift-reduce:
  - Specifier combien de conflits on attend : prologue ⇒ '%expect n'
  - Dans la pratique : '%expect 0'
- Règles

```
exp
  : exp "+" exp { $$ = $1 + $3; }
  | exp "*" exp
  ;
```

Valeur après la reviews ↙
Action à accomplir pendant ↙

- Axiome : 1ere regles ou %start