Analyseurs

INFO010 – Théorie des langages – Partie pratique

S. Collette G. Geeraerts

Automate à pile

Un automate à pile est un tuple :

$$M = \langle Q, \Sigma, \Gamma, \Omega, \delta, q_0, Z_0, O_0, F \rangle$$

- Q est l'ensemble des états ;
- Σ est l'alphabet d'entrée ;
- Γ est l'alphabet du *stack* ;
- Ω est l'alphabet de sortie ;
- δ est la fonction de transition ;
- q_0 est l'état initial ;

Automate à pile – 2

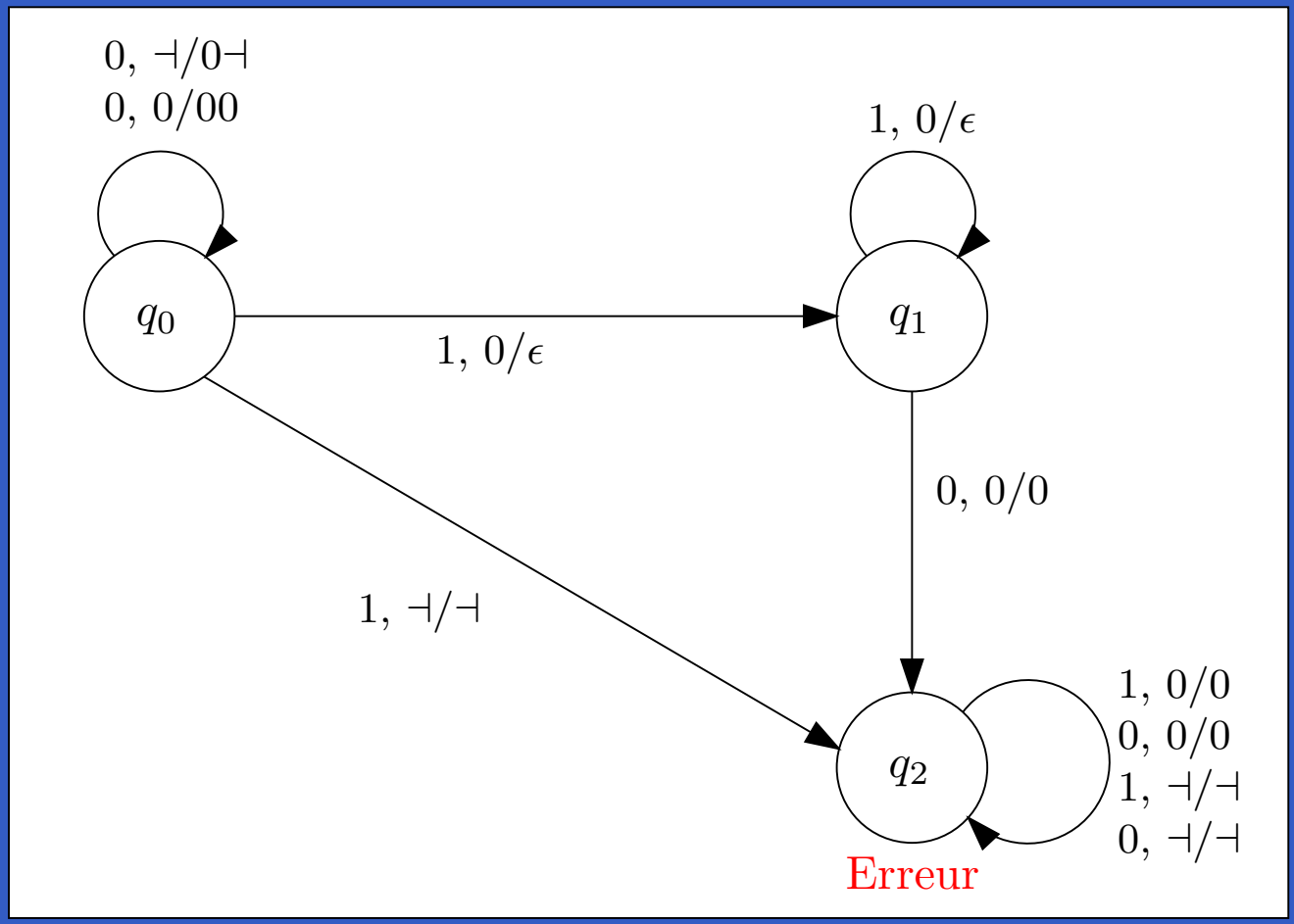
- Z_0 est le contenu du *stack* à l'état initial ;
- O_0 est le contenu de la sortie à l'état initial ;
- F est l'ensemble des états accepteurs. Si $F = \emptyset$, l'automate sera accepteur lorsque la pile sera vide, quel que soit l'état dans lequel il se trouve.

Automate à pile – Exemple

- Soit le langage formé de toutes les chaînes $0^n 1^n$.
- Construisons un automate à pile le reconnaissant :
  - Pas nécessairement déterministe
  - La pile va servir à *compter*
  - Acceptation par pile vide

Automate à pile – Exemple

- Automate à pile correspondant

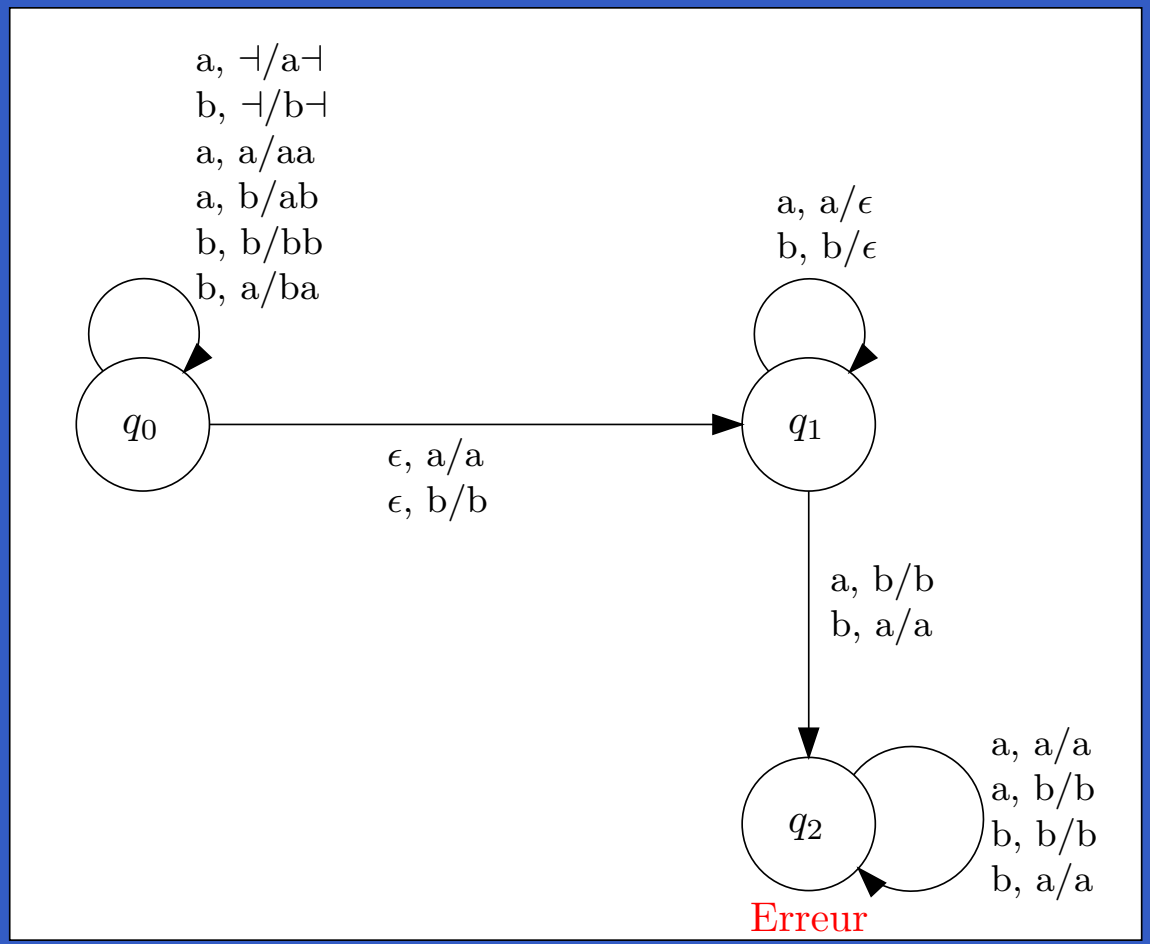


Automate à pile – Exercice

- Construisez l'automate à pile qui accepte le langage composé de tous les mots de la forme ww^R, où w est un mot quelconque sur l'alphabet $\{a, b\}$ et w^R est son image miroir.

Automate à pile – Solution

- Solution



Analyseur Gauche

- L'*analyseur gauche* construit un arbre de dérivation en *top-down*.
- Pour la grammaire $G = \langle V, T, P, S \rangle$, il sera assimilé à *l' automate à pile* suivant :

$$M = \langle \{q\}, T \cup \{\$\}, V \cup T \cup \{\$\}, \{1, \ldots, p\}, \delta, q, S, \emptyset \rangle$$

  - Il n'y a qu'un seul état q (il est initial) ;
  - l'alphabet d'entrée comprend le $ de fin d'input ;
  - l'alphabet de sortie est constitué des numéros des règles de production ;
  - À l'état initial, le *stack* contient le *start symbol*.

Analyseur Gauche – 2

- On représente une configuration de l'automate par un tuple : (q, i, S, o) :
  - q est l'état courant ;
  - i est l'état courant de l'input ;
  - S est l'état courant du *stack*. Le sommet est à gauche du string S ;
  - o est l'état de la sortie.
- Il y a deux types de transitions (dans δ) :
  - Le *matching* : $(q, ax, a\gamma, y) \rightarrow (q, x, \gamma, y)$
  - L'expansion : $(q, x, A\gamma, y) \rightarrow (q, x, \alpha\gamma, yi)$ tel que la règle numéro i est de la forme $A \rightarrow \alpha$

Analyseur Droit

- L'*analyseur droit* construit un arbre de dérivation en *bottom-up*.
- Pour la grammaire $G = \langle V, T, P, S \rangle$, il sera assimilé à l'*automate à pile* suivant :

$$M = \langle \{q\}, T \cup \{\$\}, V \cup T \cup \{\$, \vdash\}, \{1, \ldots, p\}, \delta, q, \vdash, \emptyset \rangle$$

- Donc :
  - L'alphabet du *stack* contient le symbole «*stack vide*» : $\vdash$;
  - Le contenu initial du stack est $\vdash$

Analyseur Droit – 2

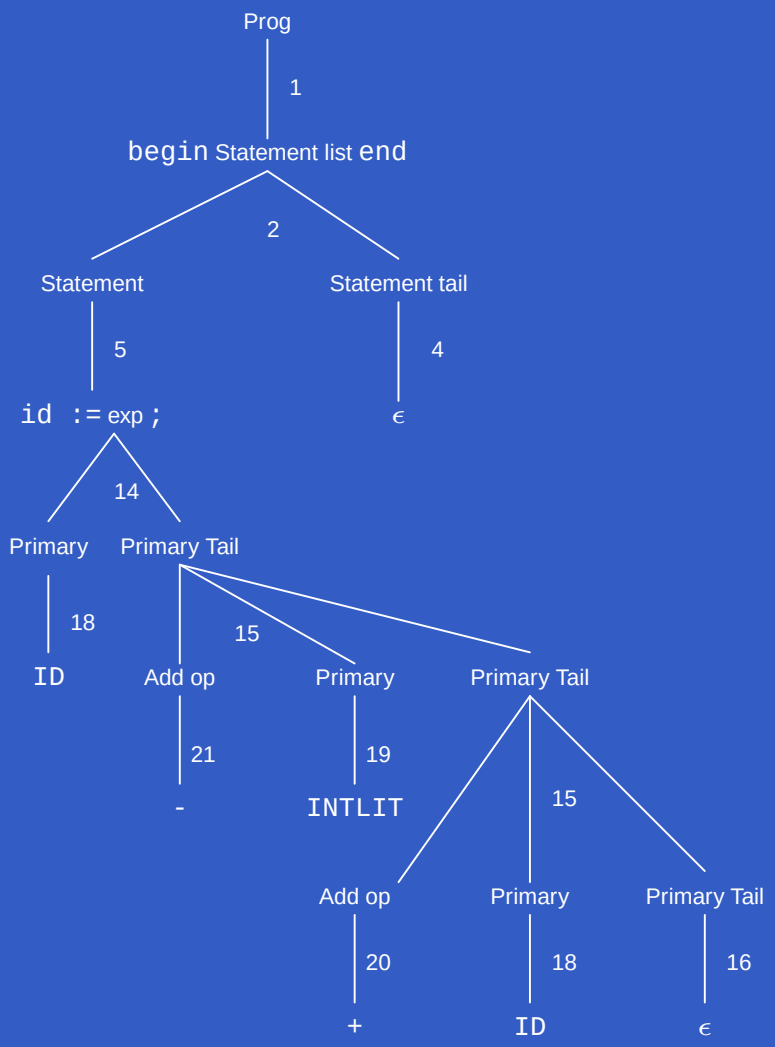
- On représente une configuration de l'analyseur droit comme pour l'analyseur gauche (le sommet du *stack* est à droite).
- Il y a trois types de transitions (dans δ) :
  - La réduction : $(q, x, \gamma\alpha, y) \rightarrow (q, x, \gamma A, yi)$ si la règle P_i est la forme $A \rightarrow \alpha$;
  - Le *shift* : $(q, \alpha x, \gamma, y) \rightarrow (q, x, \gamma\alpha, y)$;
  - L'*accept* : $(q, \varepsilon, \vdash S, y) \rightarrow (q, \varepsilon, \varepsilon, y)$

Exercices

1	<program>	→	**begin** <statement list> **end**
2	<statement list>	→	<statement> <statement tail>
3	<statement tail>	→	<statement> <statement tail>
4	<statement tail>	→	λ
5	<statement>	→	ID := <expression> ;
6	<statement>	→	**read** (<id list>) ;
7	<statement>	→	**write** (<expr list>) ;
8	<id list>	→	ID <id tail>
9	<id tail>	→	, ID <id tail>
10	<id tail>	→	λ
11	<expr list>	→	<expression> <expr tail>
12	<expr tail>	→	, <expression> <expr tail>
13	<expr tail>	→	λ
14	<expression>	→	<primary> <primary tail>
15	<primary tail>	→	<add op> <primary> <primary tail>
16	<primary tail>	→	λ
17	<primary>	→	(<expression>)
18	<primary>	→	ID
19	<primary>	→	INTLIT
20	<add op>	→	+
21	<add op>	→	−
22	<system goal>	→	<program> $

1. Construisez l'arbre de dérivation de `begin ID := ID - INTLIT + ID; end`
2. Simulez l'analyseur gauche sur la *string* `begin A := BB - 314 + A; end`
3. Simulez l'analyseur droit sur cette *string*

Correction de l'exercice 1



Correction de l'exercice 2

- Étape : 1
- *Input* restante :

  `begin A :=BB-314+A; end $`
- Contenu du *Stack* :

  <system goal>
- Action : Expansion 22
- Règle :

  <system goal> → <program> $

Correction de l'exercice 2

- Étape : 2
- *Input* restante :

  `begin A :=BB-314+A; end $`
- Contenu du *Stack* :

  <program>$
- Action : Expansion 1
- Règle :

  <program> → `begin` <statement list> `end`

Correction de l'exercice 2

- Étape : 3
- *Input* restante :

  `begin A :=BB-314+A; end $`
- Contenu du *Stack* :

  `begin` <statement list> `end $`
- Action : Matching
- Règle :

Correction de l'exercice 2

- Étape : 4
- *Input* restante :

  `A :=BB-314+A; end $`
- Contenu du *Stack* :

  <statement list> end $
- Action : Expansion 2
- Règle :

  <statement list> → <statement> <statement tail>

Correction de l'exercice 2

- Étape : 5
- *Input* restante :

  `A :=BB-314+A; end $`
- Contenu du *Stack* :

  <statement> <statement tail> `end $`
- Action : Expansion 5
- Règle :

  <statement> → `ID :=` <expression>

Correction de l'exercice 2

- Étape : 6
- *Input* restante :

  `A :=BB-314+A; end $`
- Contenu du *Stack* :

  `ID :=` <expression> `;` <statement tail> `end $`
- Action : Matching
- Règle :

Correction de l'exercice 2

- Étape : 7
- *Input* restante :

  `:=BB-314+A; end $`
- Contenu du *Stack* :

  `:=` <expression> `;` <statement tail> `end $`
- Action : Matching
- Règle :

Correction de l'exercice 2

- Étape : 8
- *Input* restante :

  `BB-314+A ; end $`
- Contenu du *Stack* :

  <expression> ; <statement tail> end $
- Action : Expansion 14
- Règle :

  <expression> → <primary> <primary tail>

Correction de l'exercice 2

- Étape : 9
- *Input* restante :

  `BB-314+A; end $`
- Contenu du *Stack* :

  <primary> <primary tail> `;` <statement tail> `end $`
- Action : Expansion 18
- Règle :

  <primary> → `ID`

Correction de l'exercice 2

- Étape : 10
- *Input* restante :

  `BB-314+A; end $`
- Contenu du *Stack* :

  `ID` <primary tail> `;` <statement tail> `end $`
- Action : Matching
- Règle :

Correction de l'exercice 2

- Étape : 11
- *Input* restante :

  `-314+A ; end $`
- Contenu du *Stack* :

  <primary tail> `;` <statement tail> `end $`
- Action : Expansion 15
- Règle :

  <primary tail> → <add op> <primary> <primary tail>

Correction de l'exercice 2

- Étape : 12
- *Input* restante :

  `-314+A; end $`
- Contenu du *Stack* :

  <add op> <primary> <primary tail> `;` <statement tail> `end $`

- Action : Expansion 21
- Règle :

  <add op> → `-`

Correction de l'exercice 2

- Étape : 13
- *Input* restante :

  `-314+A ; end $`
- Contenu du *Stack* :
  - <primary> <primary tail> ; <statement tail> end $
- Action : Matching
- Règle :

Correction de l'exercice 2

- Étape : 14
- *Input* restante :

  `314+A ; end $`
- Contenu du *Stack* :

  <primary> <primary tail> `;` <statement tail> `end $`
- Action : Expansion 19
- Règle :

  <primary> → `INTLIT`

Correction de l'exercice 2

- Étape : 15
- *Input* restante :

  `314+A ; end $`
- Contenu du *Stack* :

  `INTLIT` <primary tail> `;` <statement tail> `end $`
- Action : Matching
- Règle :

Correction de l'exercice 2

- Étape : 16
- *Input* restante :

  +A ; end $
- Contenu du *Stack* :

  <primary tail> ; <statement tail> end $
- Action : Expansion 15
- Règle :

  <primary tail> → <add op> <primary> <primary tail>

Correction de l'exercice 2

- Étape : 17
- *Input* restante :

  `+A ; end $`
- Contenu du *Stack* :

  <add op> <primary> <primary tail> `;` <statement tail> `end $`

- Action : Expansion 20
- Règle :

  <add op> → +

Correction de l'exercice 2

- Étape : 18
- *Input* restante :

  +A ; end $
- Contenu du *Stack* :

  + <primary> <primary tail> ; <statement tail> end $
- Action : Matching
- Règle :

Correction de l'exercice 2

- Étape : 19
- *Input* restante :

  A ; end $
- Contenu du *Stack* :

  <primary> <primary tail> ; <statement tail> end $
- Action : Expansion 18
- Règle :

  <primary> → ID

Correction de l'exercice 2

- Étape : 20
- *Input* restante :

  `A ; end $`
- Contenu du *Stack* :

  `ID` <primary tail> `;` <statement tail> `end $`
- Action : Matching
- Règle :

Correction de l'exercice 2

- Étape : 21
- *Input* restante :

  ; end $
- Contenu du *Stack* :

  <primary tail> ; <statement tail> end $
- Action : Expansion 16
- Règle :

  $\langle \text{primary tail} \rangle \longrightarrow \lambda$

Correction de l'exercice 2

- Étape : 22
- *Input* restante :

  ; end $
- Contenu du *Stack* :

  ; <statement tail> end $
- Action : Matching
- Règle :

Correction de l'exercice 2

- Étape : 23
- *Input* restante :

  end $
- Contenu du *Stack* :

  <statement tail> end $
- Action : Expansion 4
- Règle :

  $<\text{statement tail}> \longrightarrow \lambda$

Correction de l'exercice 2

- Étape : 24
- *Input* restante :

  end $
- Contenu du *Stack* :

  end $
- Action : Matching
- Règle :

Correction de l'exercice 2

- Étape : 25
- *Input* restante :

  $

- Contenu du *Stack* :

  $

- Action : Matching
- Règle :

Correction de l'exercice 2

- Étape :
- *Input* restante :

- Contenu du *Stack* :

- Action :
- Règle :

Bingo !

Correction de l'exercice 2

- Étape : 1
- *Input* restante :

```
begin A :=BB-314+A; end $
```

- Contenu du *Stack* :

- Action : Shift
- Règle :

Correction de l'exercice 2

- Étape : 2
- *Input* restante :

  `-314+A; end $`
- Contenu du *Stack* :

  $\vdash$ `begin ID := ID`
- Action : Reduce 18
- Règle :

  $\langle$primary$\rangle \longrightarrow$ `ID`

Correction de l'exercice 2

- Étape : 3
- *Input* restante :

  `-314+A; end $`
- Contenu du *Stack* :

  ⊢ `begin ID :=` <primary>
- Action : Shift
- Règle :

Correction de l'exercice 2

- Étape : 4
- *Input* restante :

  `314+A ; end $`
- Contenu du *Stack* :

  ⊢ `begin ID :=` <primary> `-`
- Action : Reduce 21
- Règle :

  <add op> ⟶ `-`

Correction de l'exercice 2

- Étape : 5
- *Input* restante :

  `314+A ; end $`
- Contenu du *Stack* :

  ⊢ `begin ID :=` <primary> <add op>
- Action : Shift
- Règle :

Correction de l'exercice 2

- Étape : 6
- *Input* restante :

  `+A ; end $`
- Contenu du *Stack* :

  ⊢ `begin ID :=` <primary> <add op> `INTLIT`
- Action : Reduce 19
- Règle :

  <primary> ⟶ `INTLIT`

Correction de l'exercice 2

- Étape : 7
- *Input* restante :

  `+A ; end $`
- Contenu du *Stack* :

  ⊢ `begin ID :=` <primary> <add op> <primary>
- Action : Shift
- Règle :

Correction de l'exercice 2

- Étape : 8
- *Input* restante :

  `A ; end $`
- Contenu du *Stack* :

  ⊢ `begin ID :=` <primary> <add op> <primary> `+`
- Action : Reduce 20
- Règle :

  <add op> → `+`

Correction de l'exercice 2

- Étape : 9
- *Input* restante :

  `A ; end $`
- Contenu du *Stack* :

  ⊢ `begin ID :=` <primary> <add op> <primary> <add op>
- Action : Shift
- Règle :

Correction de l'exercice 2

- Étape : 10
- *Input* restante :

  `; end $`
- Contenu du *Stack* :

  ⊢ `begin ID :=` <primary> <add op> <primary> <add op> `ID`
- Action : Reduce 18
- Règle :

  <primary> → `ID`

Correction de l'exercice 2

- Étape : 11
- *Input* restante :

  ; end $
- Contenu du *Stack* :

  ⊢ begin ID := <primary> <add op> <primary> <add op> <primary>
- Action : Reduce 16
- Règle :

  $\langle \text{primary tail} \rangle \longrightarrow \lambda$

Correction de l'exercice 2

- Étape : 12
- *Input* restante :

  `; end $`
- Contenu du *Stack* :

  ⊢ `begin ID :=` <primary> <add op> <primary> <add op> <primary> <primary tail>
- Action : Reduce 15
- Règle :

  <primary tail> → <add op> <primary> <primary tail>

Correction de l'exercice 2

- Étape : 13
- *Input* restante :

  `; end $`
- Contenu du *Stack* :

  ⊢ `begin ID :=` <primary> <add op> <primary> <primary tail>
- Action : Reduce 15
- Règle :

  <primary tail> → <add op> <primary> <primary tail>

Correction de l'exercice 2

- Étape : 14
- *Input* restante :

  ; end $
- Contenu du *Stack* :

  ⊢ `begin ID :=` <primary> <primary tail>
- Action : Reduce 14
- Règle :

  <expression> ⟶ <primary> <primary tail>

Correction de l'exercice 2

- Étape : 15
- *Input* restante :

  `; end $`
- Contenu du *Stack* :

  ⊢ `begin ID :=` <expression>
- Action : Shift
- Règle :

Correction de l'exercice 2

- Étape : 16
- *Input* restante :

  `end $`
- Contenu du *Stack* :

  ⊢ `begin ID :=` <expression> `;`
- Action : Reduce 5
- Règle :

  <statement> → `ID :=` <expression> `;`

Correction de l'exercice 2

- Étape : 17
- *Input* restante :

  `end $`
- Contenu du *Stack* :

  $\vdash$ `begin` <statement>
- Action : Reduce 4
- Règle :

  $\langle \text{statement tail} \rangle \longrightarrow \lambda$

Correction de l'exercice 2

- Étape : 18
- *Input* restante :

  `end $`
- Contenu du *Stack* :

  ⊢ `begin` <statement> <statement tail>
- Action : Reduce 2
- Règle :

  <statement list> → <statement> <statement tail>

Correction de l'exercice 2

- Étape : 19
- *Input* restante :

  `end $`
- Contenu du *Stack* :

  ⊢ `begin` <statement list>
- Action : Shift
- Règle :

Correction de l'exercice 2

- Étape : 20
- *Input* restante :

  $
- Contenu du *Stack* :

  $\vdash$ `begin` <statement list> `end`
- Action : Reduce 1
- Règle :

  <program> $\longrightarrow$ `begin` <statement list> `end`

Correction de l'exercice 2

- Étape : 21
- *Input* restante :

  $
- Contenu du *Stack* :

  ⊢ <program>
- Action : Shift
- Règle :

Correction de l'exercice 2

- Étape : 22
- *Input* restante :

- Contenu du *Stack* :

$$\vdash \langle \text{program} \rangle \$$$

- Action : Reduce 22
- Règle :

$$\langle \text{system goal} \rangle \longrightarrow \langle \text{program} \rangle \$$$

Correction de l'exercice 2

- Étape : 23
- *Input* restante :

- Contenu du *Stack* :

  $\vdash$ <system goal>

- Action : Accept
- Règle :

Correction de l'exercice 2

- Étape :
- *Input* restante :

- Contenu du *Stack* :

- Action :
- Règle :

Gagné !