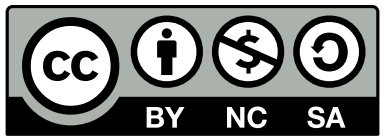
Chapitre 11 - Analyse syntaxique LR

INF7641 Compilation

Jean Privat

Université du Québec à Montréal

INF7641 Compilation
v251



Plan

Langages (rappels)

Langages réguliers

- Expressions régulières
- Automates finis (déterministes ou non)

Langages non contextuels

- Grammaires non contextuelles
- *Context-free grammar* (CFG?)

Analyses syntaxiques

Algos

- Données : une grammaire et une séquence finie de jetons
- Résultat : un arbre syntaxique (ou une erreur de syntaxe)

Problème

- Pas d'analyse syntaxique générale efficace

Solution

- Plusieurs sortes d'analyses et plusieurs types de grammaires
- Exemples
  - Analyses LR: LR(0), LALR(1), LR(1), LR(k), GLR...
  - Analyses LL: LL(1), LL(k), LL(*)...

Analyse syntaxique LR(0)

Analyse syntaxique LR(0)

Analyse ascendante

- Déduit les productions a partir des jetons

Analyse l'entrée de la gauche (**L***eft*)

- Le premier jeton de la séquence est traité

Construit une dérivation droite (**R***ight*)

- Les fils de droite sont construits avant les fils de gauches

Utilise aucun jeton de prévision (0)

- On ne regarde jamais les jetons en avant

Comment faire l'analyse LR(0)

Étapes

- Transformer la grammaire en un automate
- Évaluer l'automate

Quel genre d'automate utiliser

- Automates finis insuffisants
  - Rappel: lemme de la pompe
- Il faut donc un automate plus fort
  - À pile !

Automate à pile (*pushdown automaton*, PDA)

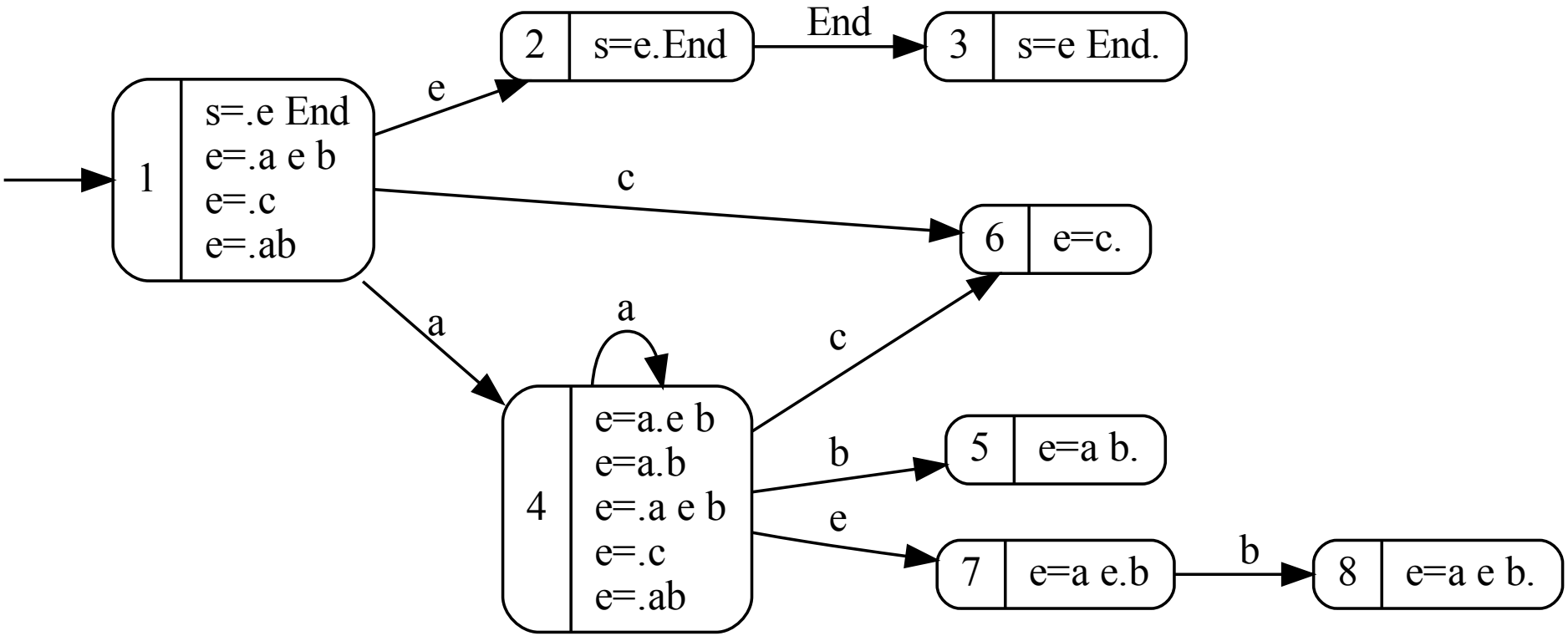
États de l'automate = Ensemble d'*items*

- Un **item** est une **alternative** + une **position**
- La **position** (représentée par un point) peut être au début, à la fin ou entre deux éléments de l'alternative
- Attention: élément = production ou jeton ; élément $\neq$ item

Transitions (déterministes) étiquetées par

- Un **jeton**
- Une **production**
- Ou le jeton spécial *fin de séquence* (noté `$` ou `End`)

Exemple d'automate LR(0)



Grammaire $\rightarrow$ LR(0) : Grammaire augmentée

Ajout d'une nouvelle production

- « départ = ancien_départ End ; »

Exemple

```
e = {list:} '(' l ')' | {id:} i ;
l = {many:} l e | {one:} e ;
```

devient

```
s = e End ;
e = {list:} '(' l ')' | {id:} i ;
l = {many:} l e | {one:} e ;
```

Construction de l'automate : État de départ

On prend la production de départ

- « `s = e End` »

On bâtit l'ensemble des position potentielles

- {« `s = . e End` »}

Si le point est placé avant une production

- On inclue ses alternatives
- {« `s = . e End` », « `e = . '(' l ')'` », « `e = . i` »}
- Éventuellement récursivement

Construction de l'automate : État suivants

Identifier les transitions possibles

- Il y a une transition par élément précédé d'un point
- Traiter chaque transition indépendamment dans la suite

Déterminer l'ensemble des items

- Considérer les items qui progressent sur une transition
- Avancer le point pour chacun d'eux
- Le point devant production $\rightarrow$ inclure ses alternatives

Déterminer l'état destination

- Si l'état existe déjà, le réutiliser
- Si l'état n'existe pas, le créer et penser à déterminer ses états suivants

Exercice

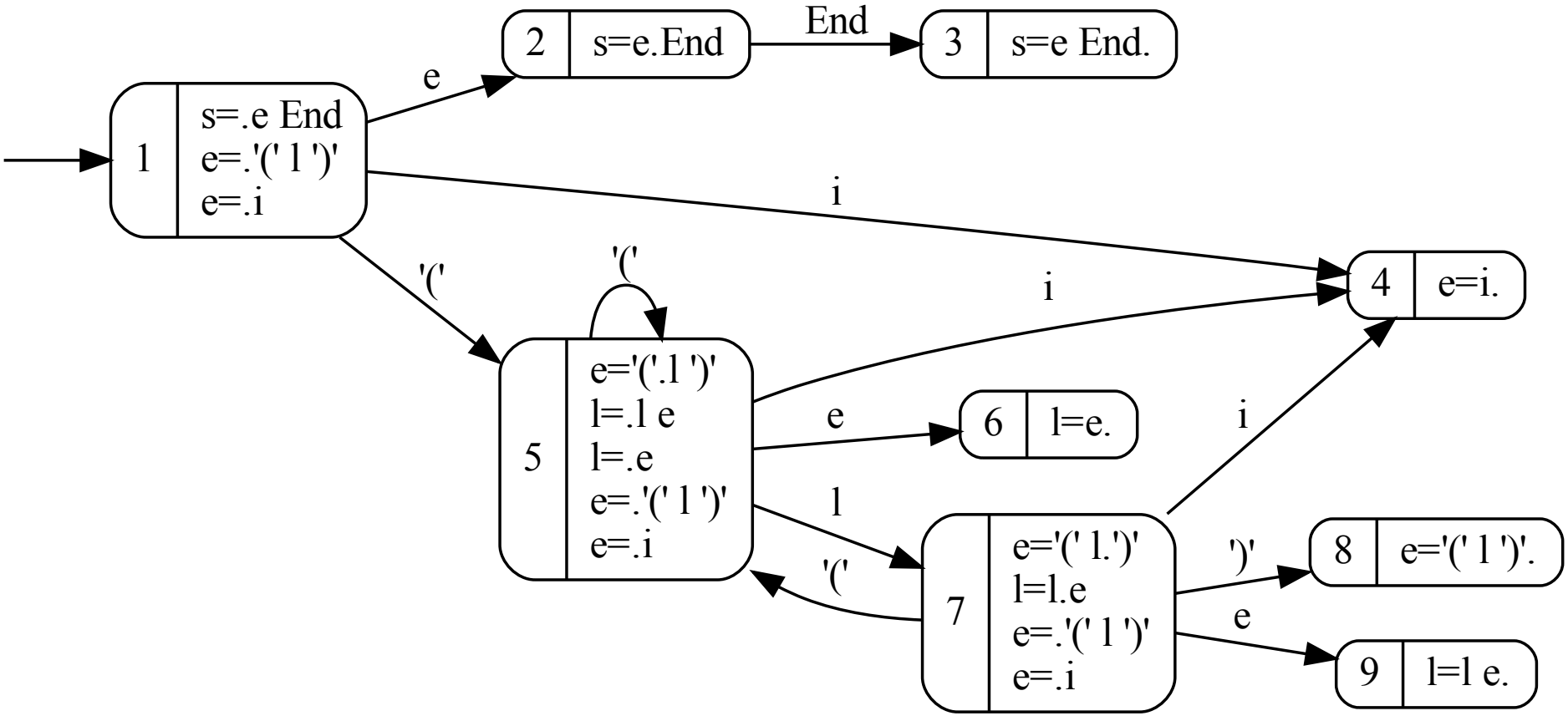
Finir l'automate de la grammaire

```
e = {list:} '(' l ')' | {id:} i ;
l = {many:} l e | {one:} e ;
```

Exercice

Finir l'automate de la grammaire

```
e = {list:} '(' l ')' | {id:} i ;
l = {many:} l e | {one:} e ;
```



Construction de l'automate : Action des états

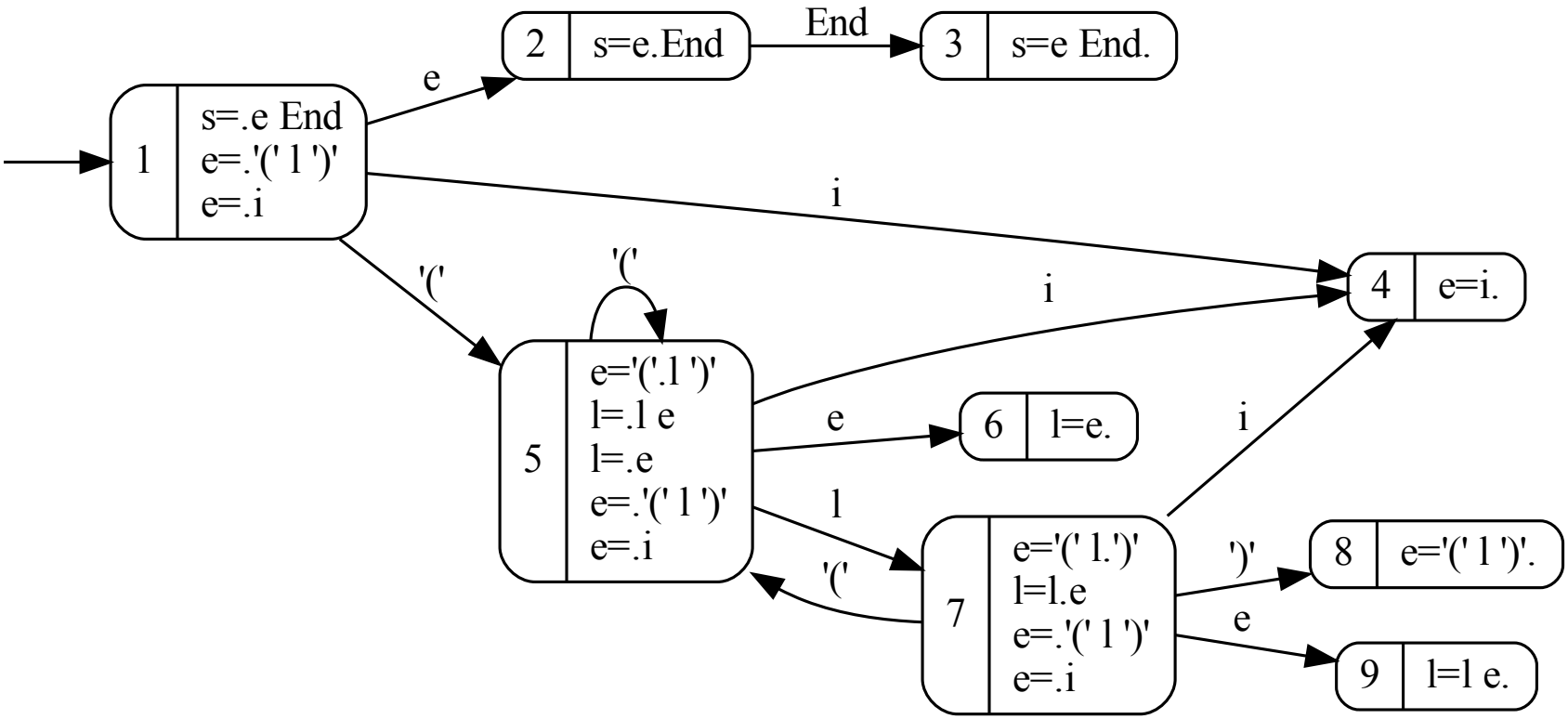
Chaque état à une action

- Décalage : il existe un jeton précédé d'un point
- Réduction $P : A$: il existe un point à la fin de l'alternative $P : A$

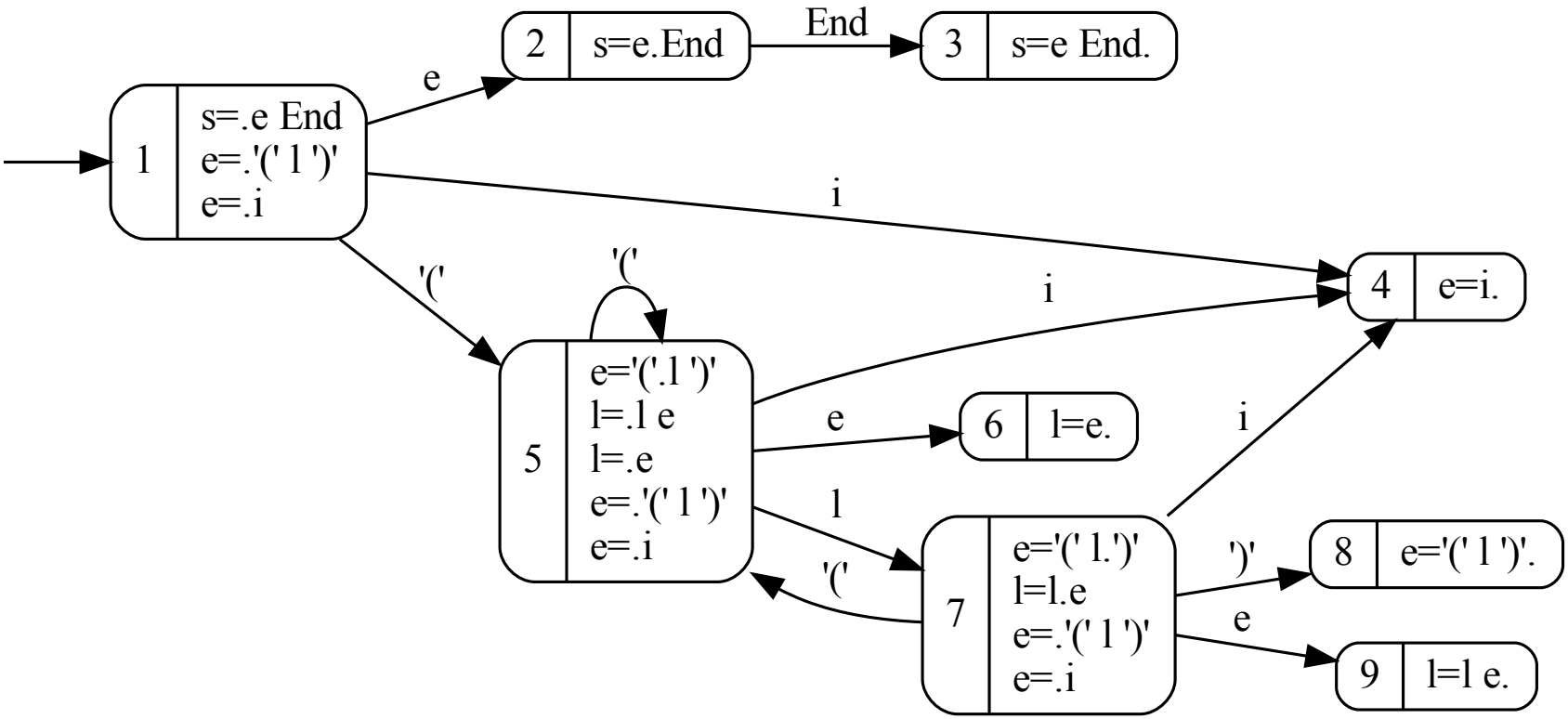
Exemples

- $\{$« `s = . e End` », « `e = . '(' l ')'` », « `e = . i` »$\}$
  $\rightarrow$ décalage
- $\{$« `s = e End .` »$\}$
  $\rightarrow$ réduction

Exercice: déterminer les actions



Exercice: déterminer les actions



- décalages: 1, 2, 5, 7 : décalage
- réductions: 3(s), 4(e:id), 6(l:one), 8(e:list), 9(l:many)

Exercice

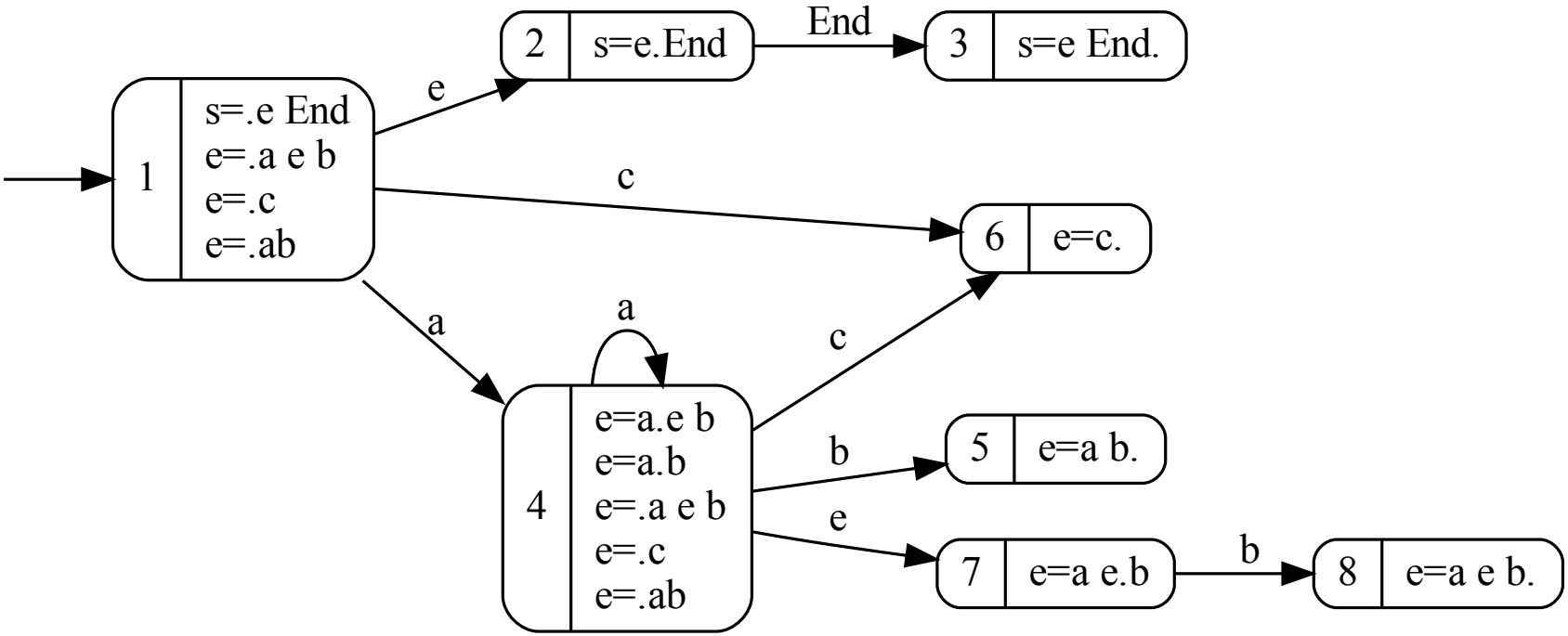
Construire l'automate de la grammaire

```
e = a e b | c | a b ;
```

Exercice

Construire l'automate de la grammaire

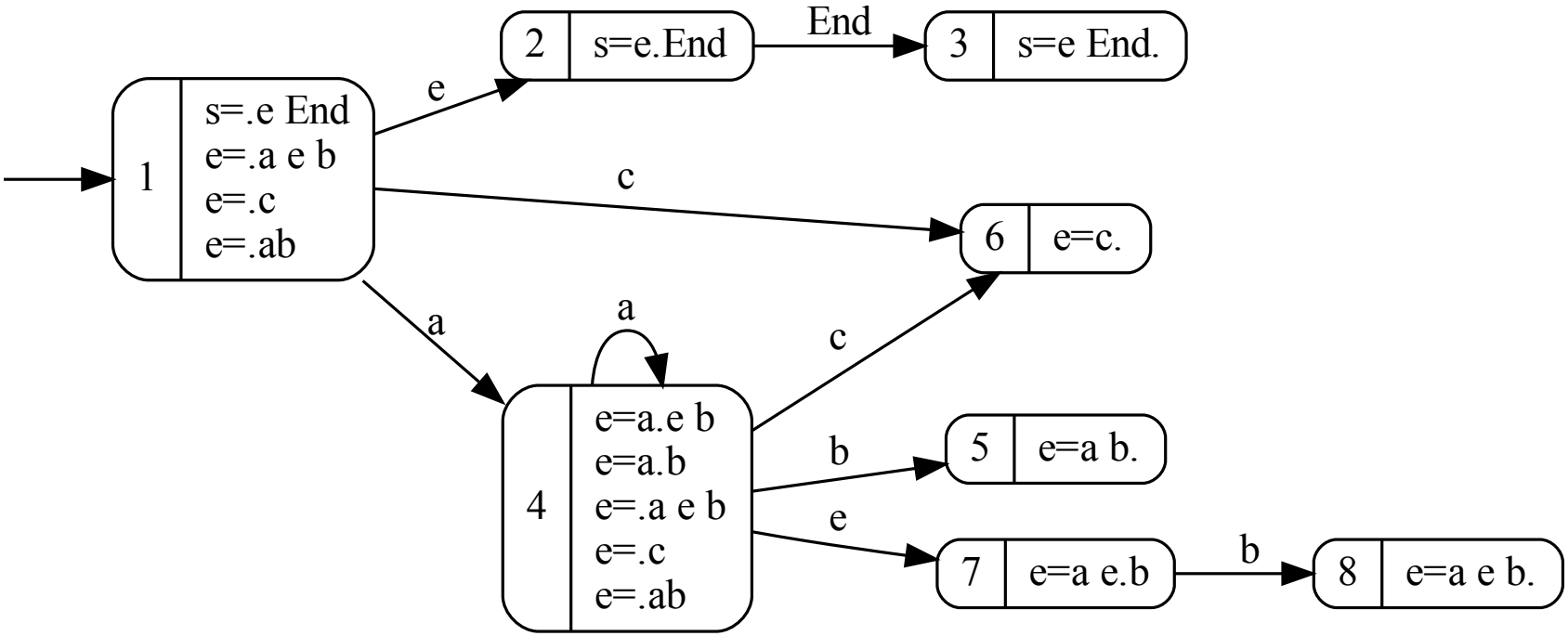
```
e = a e b | c | a b ;
```



Exercice

Construire l'automate de la grammaire

```
e = a e b | c | a b ;
```

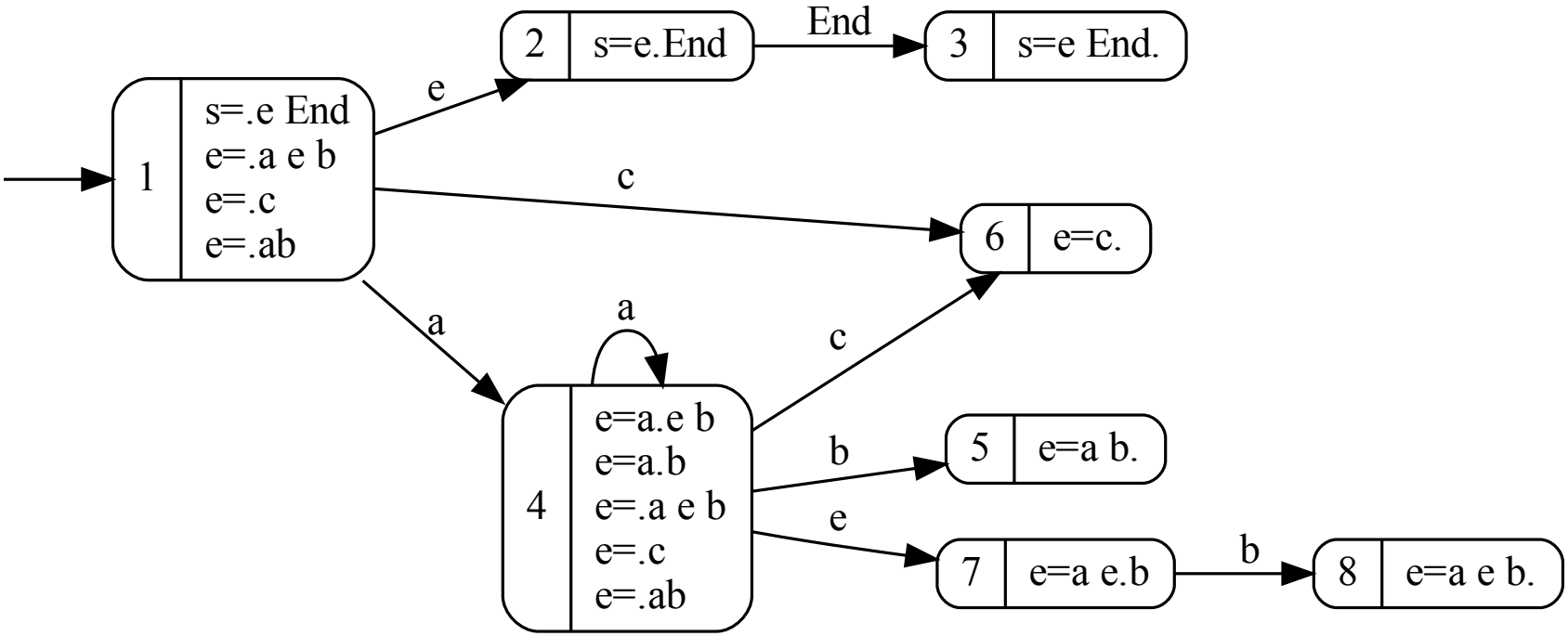


Déterminer les actions des états de l'automate

Exercice

Construire l'automate de la grammaire

```
e = a e b | c | a b ;
```



Déterminer les actions des états de l'automate

- décalages : 1, 2, 4, 7
- réductions : 3(s), 5(e:3), 6(e:2), 8(e:1)

Exercice

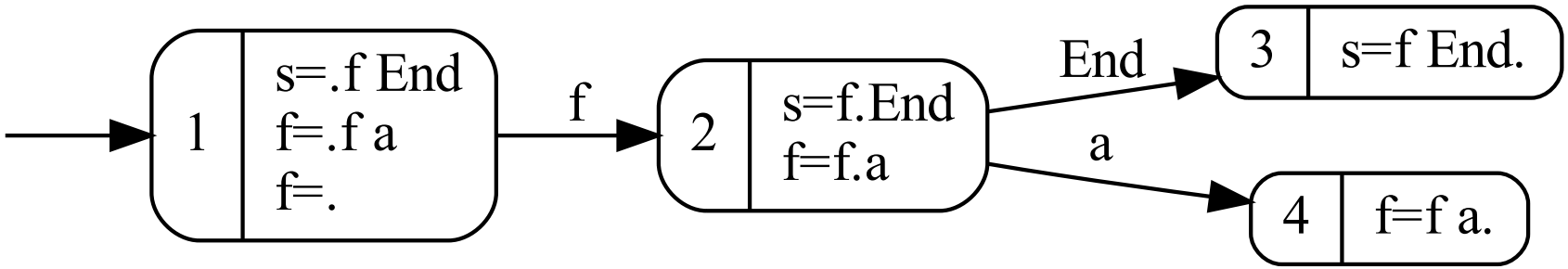
Construire l'automate de la grammaire

```
f = {devant:} f a | {rien:} ;
```

Exercice

Construire l'automate de la grammaire

```
f = {devant:} f a | {rien:} ;
```

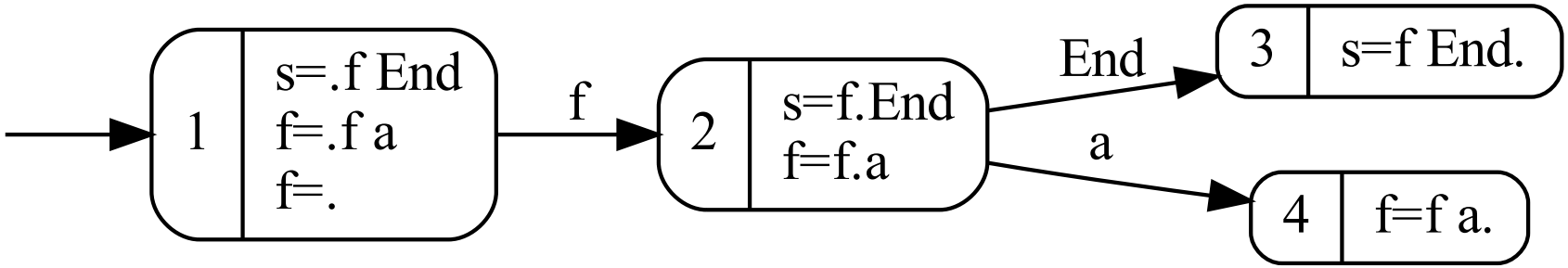


Déterminer les actions des états de l'automate

Exercice

Construire l'automate de la grammaire

```
f = {devant:} f a | {rien:} ;
```



Déterminer les actions des états de l'automate

- 1 : réduction(f:rien)
- 2 : décalage
- 3 : réduction(s)
- 4 : réduction(f:devant)

Évaluation de l'automate : Principes

Une pile pour plus de puissance

- On maintient une pile d'analyse
- On avance dans l'automate et empile des bouts d'arbre syntaxique
- On construit des bouts d'arbre syntaxique en utilisant les têtes de piles

Dans un état

- Ce qui précède le point est un passé **avéré**
  $\rightarrow$ C'est dans la pile
- Ce qui suit le point est un futur **possible**

Évaluation de l'automate

Données

- Un automate LR(0) A
- Une séquence de jetons J

Résultat

- Un arbre syntaxique

Départ

- Une pile initialisé à départ(A)

Évaluation de l'automate

État courant

- C'est celui du sommet de pile

Quatre actions

- Décalage : on consomme un jeton
- Réduction : on construit une production (sous-arbre)
- Acceptation : on retourne l'arbre syntaxique
- Erreur : on retourne une erreur de syntaxe

Action de décalage

État de décalage

- Le jeton j de la séquence J est consommé
- On empile j dans la pile
- La transition j dans l'automate est suivie vers l'état E
- On empile E dans la pile

Erreur de syntaxe

- Si la transition j n'existe pas, on retourne une erreur de syntaxe sur le jeton j

Action de réduction

État de réduction $P : A$

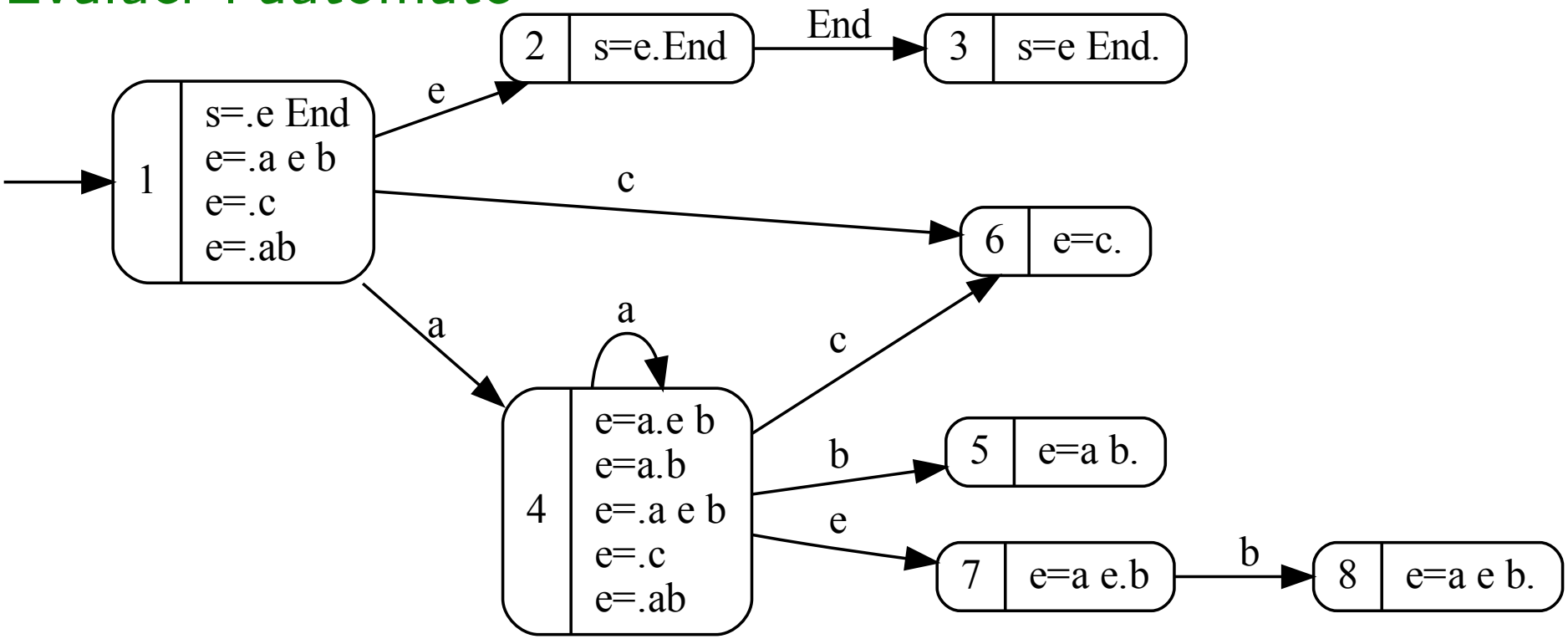
- On construit le nœud pour l'alternative $P : A$ avec les derniers éléments de la pile
- On dépile tous ces éléments de la pile
- L'état courant est le nouveau sommet de pile
- On empile P dans la pile
- La transition P dans l'automate est suivie vers l'état E
- On empile E dans la pile

Acceptation

- Si on réduit la production de départ, alors on retourne le nœud construit

Exercice

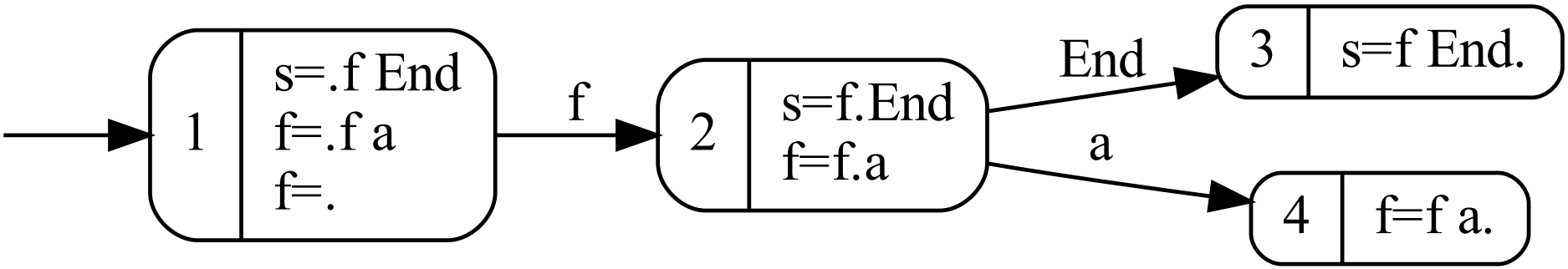
Évaluer l'automate



- Séquence de jetons = « a a c b b End »

Exercice

Évaluer l'automate



- Séquence de jetons = « a a a End »

Conflits dans l'automate

Problème

- L'algo est valide si une seule action est possible par état
- L'automate construit ne l'est pas forcément

Conflits

- Réduction-réduction (*reduce-reduce*)
- Décalage-réduction (*shift-reduce*)

Exemple

```
g = {nonvide:} a g | {vide:} ;
```

Analyse syntaxique LR(1)

Conflit LR(0) (rappel)

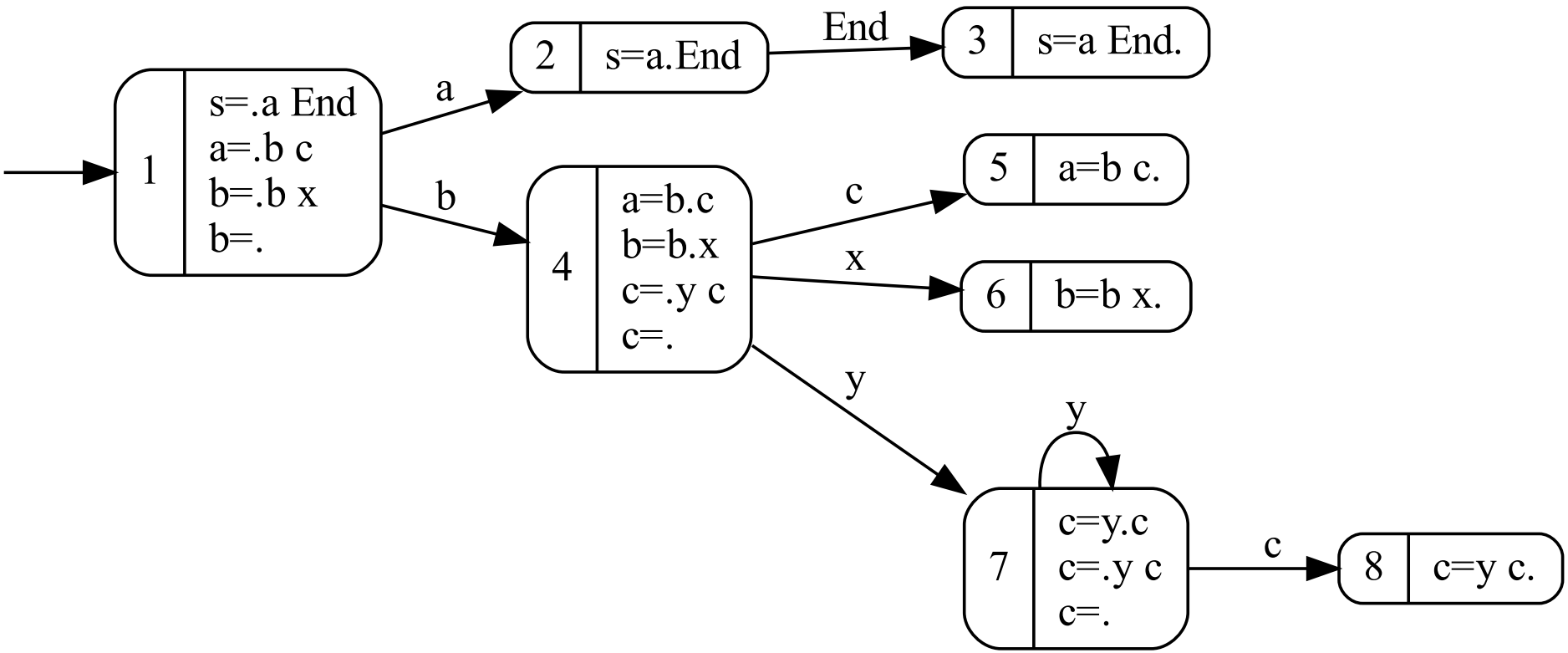
Certaines grammaires ne sont pas LR(0)

- Ssi on obtient un automate LR(0) conflictuel

Exercice : construire l'automate LR(0)

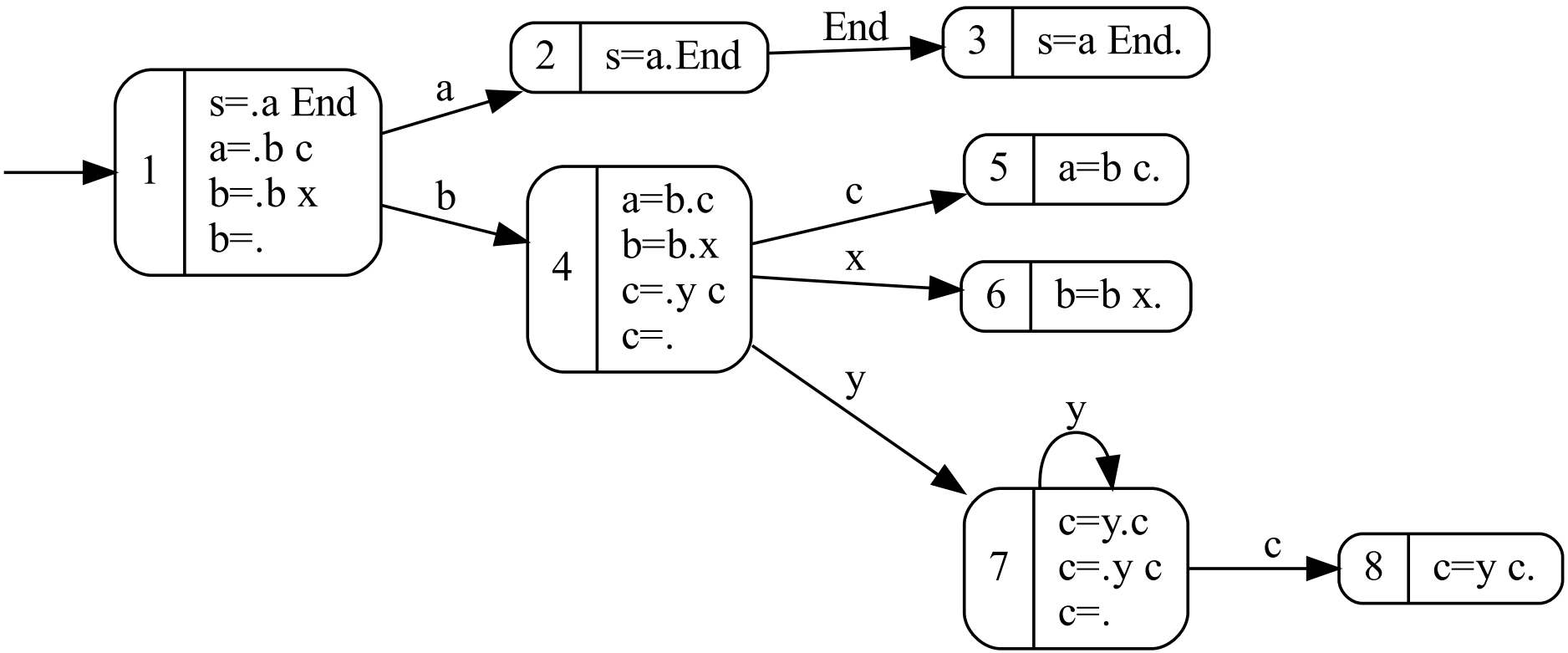
```
s = a End ;
a = b c ;
b = b x | ;
c = y c | ;
```

Conflit LR(0) (rappel)



Exercice : déterminer les actions

Conflit LR(0) (rappel)



Exercice : déterminer les actions

- Conflit à l'état 4 : Décalage ; Réduction(c:2)
- Conflit à l'état 7 : Décalage ; Réduction(c:2)

Conflit LR(0)

Conflits gênants

- En pratique, la plupart des grammaires ne sont pas LR(0)

Solution

- Il faut une analyse plus forte : LR(1)

Analyse syntaxique LR(1)

Analyse ascendante

- Déduit les productions a partir des jetons

Analyse l'entrée de la gauche (**L***eft*)

- Le premier jeton de la séquence est traité

Construit une dérivation droite (**R***ight*)

- Les fils de droite sont construits avant les fils de gauches

Utilise un jeton de prévision (1)

- On se permet de regarder un jeton en avant

Fonctions d'aide

Ensemble Nullable

- Indique si une production peut englober aucun jeton
- Exemple : b $\in$ Nullable

Fonction Premier

- Indique l'ensemble des jetons qui commencent (au sens large) une production
- Exemple : x $\in$ Premier(a)

Ensemble Nullable

Une production p est nullable ssi

- Elle possède une alternative vide, ou
- Elle possède une alternative composée uniquement de productions nullables

Calcul par point fixe

- Approximer successivement l'ensemble Nullable
- Jusqu'à l'atteinte de la stabilité

Exercice

Déterminer l'ensemble Nullable

```
s = a End ;
a = b c ;
b = b x | ;
c = y c | ;
```

Exercice

Déterminer l'ensemble Nullable

```
s = a End ;
a = b c ;
b = b x | ;
c = y c | ;
```

- 0 : Nullable $= \{\ \}$
- 1 : Nullable $= \{$b, c$\}$
- 2 : Nullable $= \{$a, b, c$\}$
- 3 : Nullable $= \{$a, b, c$\} \rightarrow$ Stabilité atteinte

Fonction Premier

Un jeton j appartient à Premier(p) ssi pour une alternative a de la production p

- $p = \alpha\ j\ \beta$, ou
- $p = \alpha\ q\ \beta$ et $j \in$ Premier(q)

avec

- α une séquence (possiblement vide) de productions nullables
- β une séquence quelconque (possiblement vide) de productions ou de jetons
  $\rightarrow$ Calcul par point fixe aussi

Exercice

Déterminer la fonction Premier

```
s = a End ;
a = b c ;
b = b x | ;
c = y c | ;
```

Exercice

Déterminer la fonction Premier

```
s = a End ;
a = b c ;
b = b x | ;
c = y c | ;
```

- P(s)={ } ; P(a)={ } ; P(b)={ } ; P(c)={ }
- P(s)={End} ; P(a)={ } ; P(b)={x} ; P(c)={y}
- P(s)={End} ; P(a)={x, y} ; P(b)={x} ; P(c)={y}
- P(s)={End, x, y} ; P(a)={x, y} ; P(b)={x} ; P(c)={y}
- P(s)={End, x, y} ; P(a)={x, y} ; P(b)={x} ; P(c)={y} $\rightarrow$ Stabilité atteinte

Automate LR(1)

Être plus fort que LR(0)

- Idée : regarder un jeton en avant pour décider entre les actions possibles
- Besoin : maintenir les jetons suivants les productions dans les *items*

Item LR(1)

- Une *alternative*
- Une *position* (indiquée par le point)
- Un *jeton suivant la production* (indiqué après la virgule)
- Exemple : « a = b . c , End »

Construction de l'automate LR(1)

Item de départ

- « s $=$. a End , End »

Dérouler les production avec le bon jeton suivant

- « a $=$ γ . b α c β , j »
  $\rightarrow$ dérouler b, suivant $\in$ P(c)
- « a $=$ γ . b α k β , j »
  $\rightarrow$ dérouler b, suivant $=$ k
- « a $=$ γ . b α , j »
  $\rightarrow$ dérouler b, suivant $=$ j

avec

- α une séquence (possiblement vide) de productions nullables
- β et γ deux séquences quelconques (possiblement vide) de productions ou de jetons

Construction de l'automate

Exemple : État de départ

- Item de départ

```
s = . a End , End
```

- On déroule a avec jeton suivant End

```
a = . b c , End
```

- On déroule b avec jetons suivants y (car P(c) = {y}) et End (car c est nullable)

```
b = . b x , y
b = . , y
b = . b x , End
b = . , End
```

Construction de l'automate

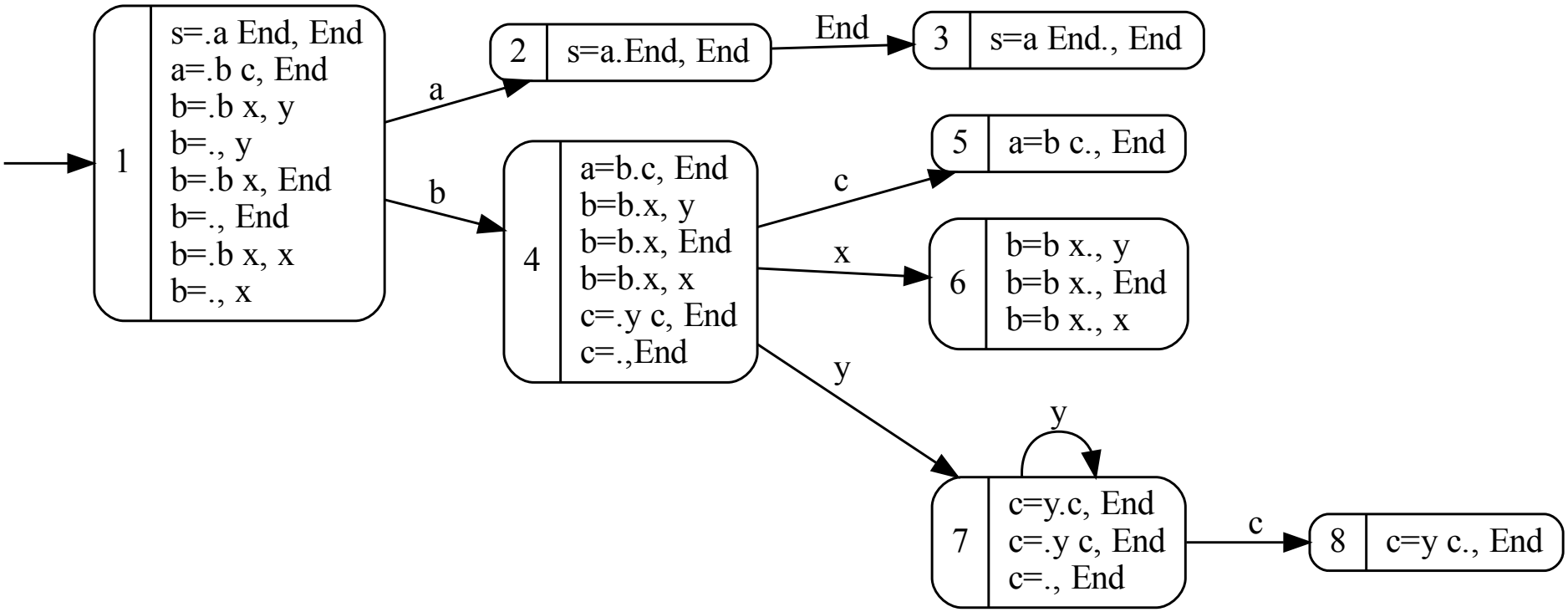
Exemple : État de départ (suite)

- On déroule b avec jeton suivant x

```
b = . b x , x
b = . , x
```

- Il ne reste plus rien de nouveau à dérouler
  (on vient de dérouler b avec jeton suivant x)

Automate construit



Actions gardées : un jeton de prévision

On agit en fonction du futur

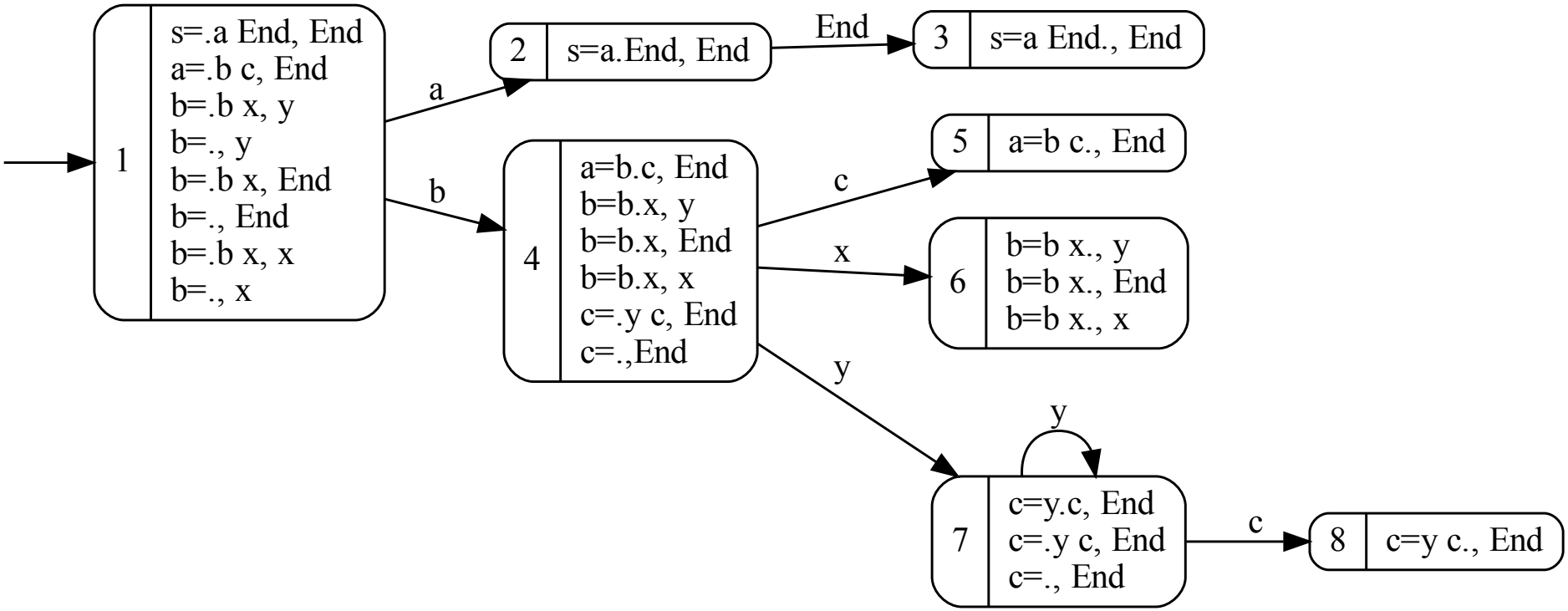
- Plusieurs actions par état
- Un jeton de prévision conditionne (garde) une action

Règle des actions gardées

- Décalage : il existe un jeton j précédé d'un point
  $\rightarrow$ gardée par le jeton j
- Réduction $P : A$: il existe un point à la fin de l'alternative
  $\rightarrow$ gardée par le jeton suivant la production

Exercice

Déterminer les actions pour les états



Exercice

Déterminer les actions pour les états

- 1 : $y \rightarrow R(b:2)$; $End \rightarrow R(b:2)$; $x \rightarrow R(b:2)$
- 2 : $End \rightarrow D$
- 3 : $End \rightarrow R$
- 4 : $x \rightarrow D$; $y \rightarrow D$; $End \rightarrow R(c:2)$
- 5 : $End \rightarrow R(a)$
- 6 : $y \rightarrow R(b:1)$; $End \rightarrow R(b:1)$; $x \rightarrow R(b:1)$
- 7 : $y \rightarrow D$; $End \rightarrow R(c:2)$
- 8 : $End \rightarrow R(c:1)$

Évaluation de l'automate LR(1)

Principe

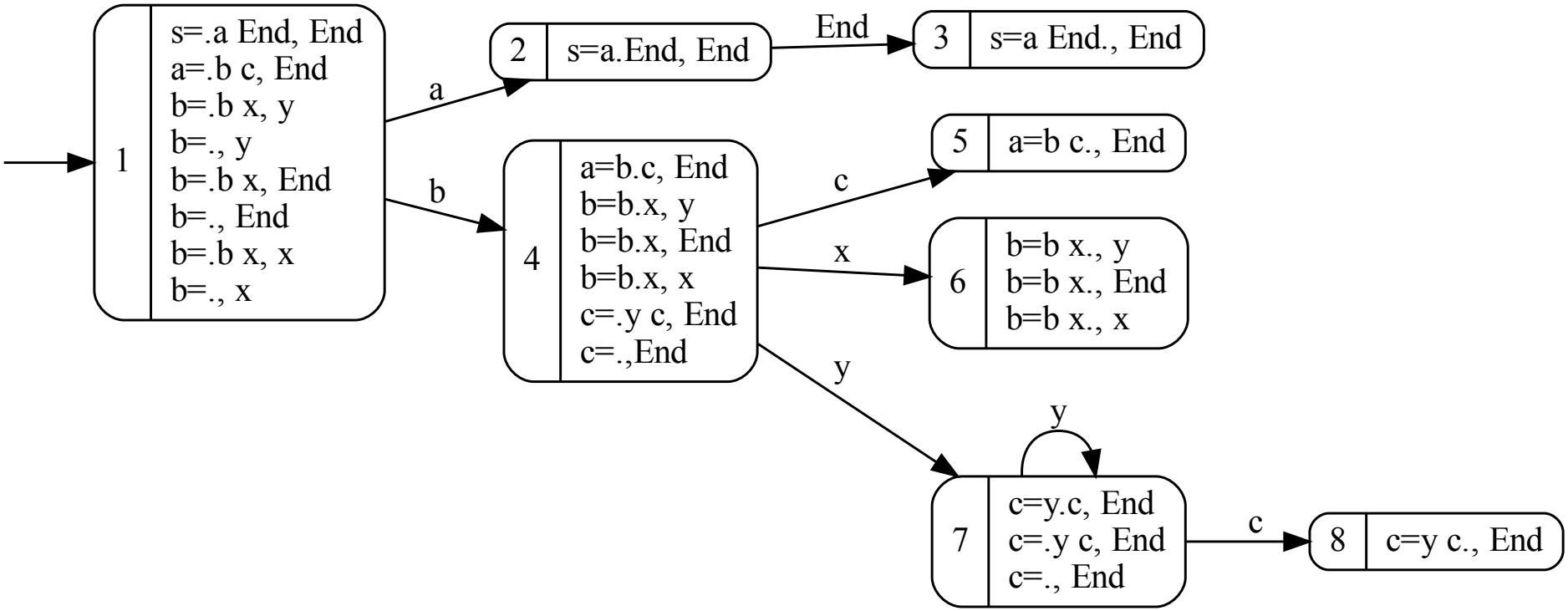
- On conserve le principe de LR(0)
- Mais l'action à effectuer dépend du prochain jeton

Concrètement

- Regarder le prochain jeton : ne pas le consommer!
- En déduire l'action pour l'état courant
- Effectuer l'action : même comportement que pour LR(0)
- Recommencer à la première étape

Exercice

Évaluer l'automate



- Séquence de jetons = « x x y y End »

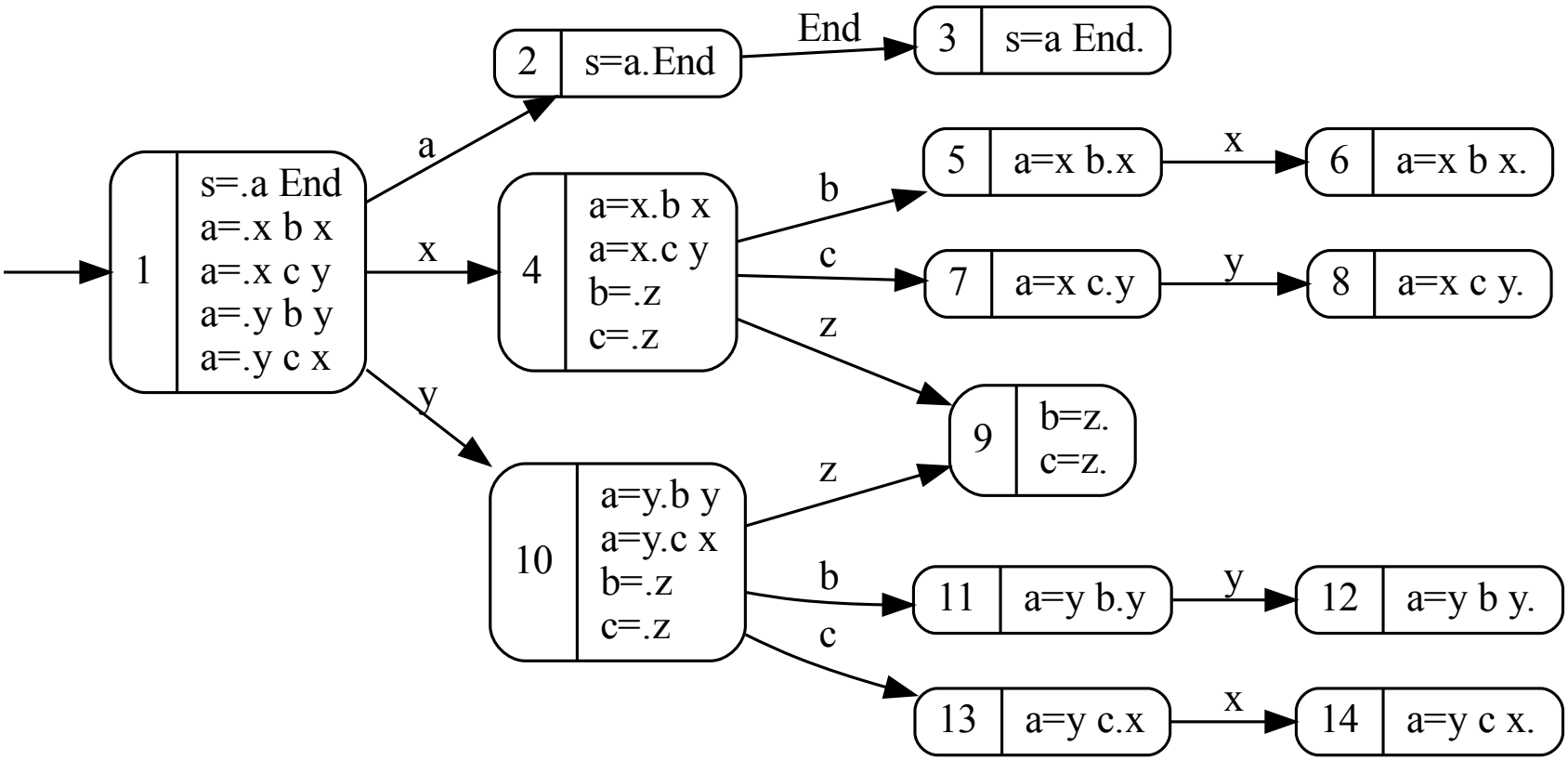
Exercice

Construire l'automate LR(0)

```
a = x b x | x c y | y b y | y c x ;
b = z ;
c = z ;
```

Exercice

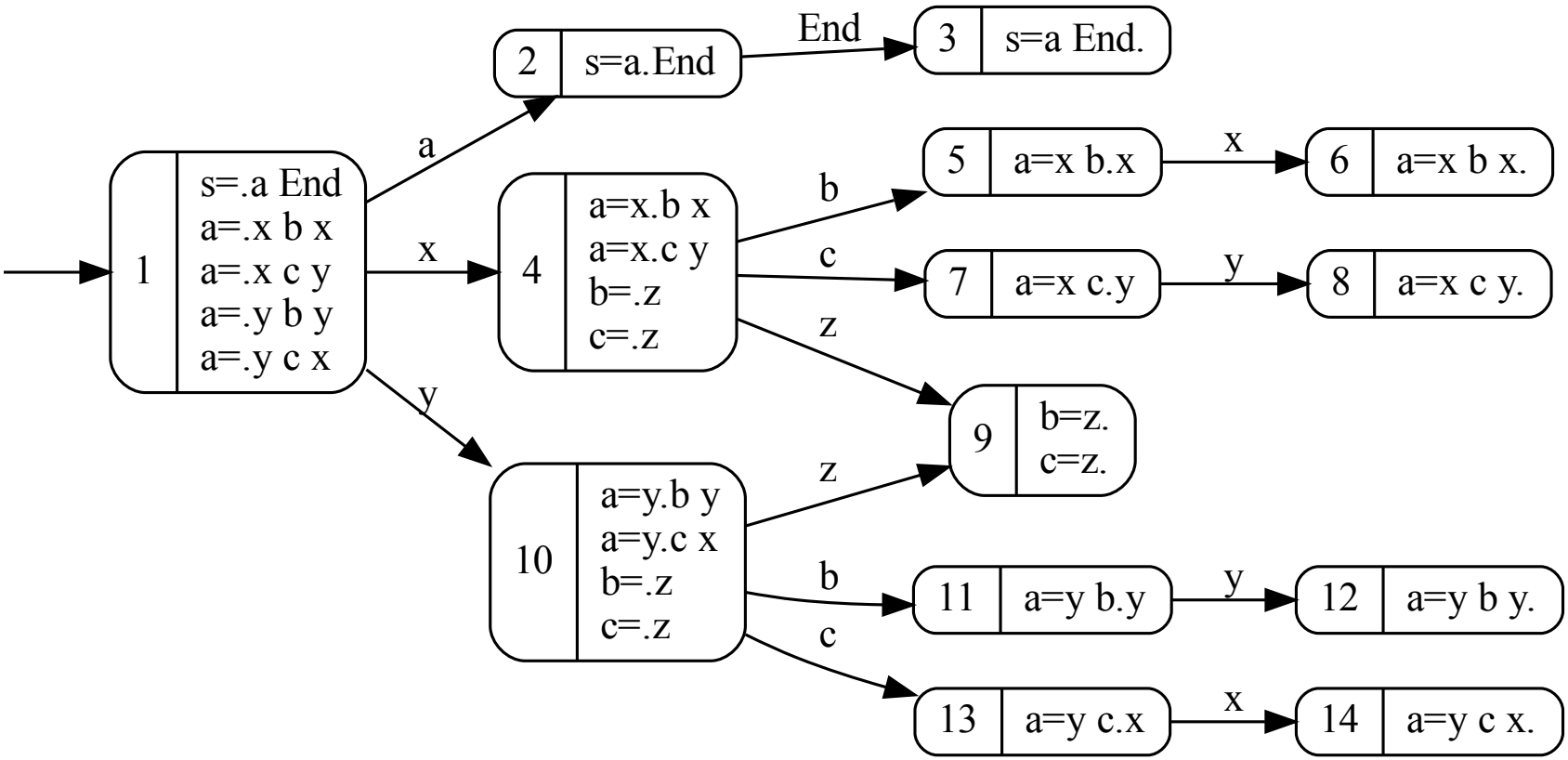
Construire l'automate LR(0)



Déterminer les actions

Exercice

Construire l'automate LR(0)



Déterminer les actions

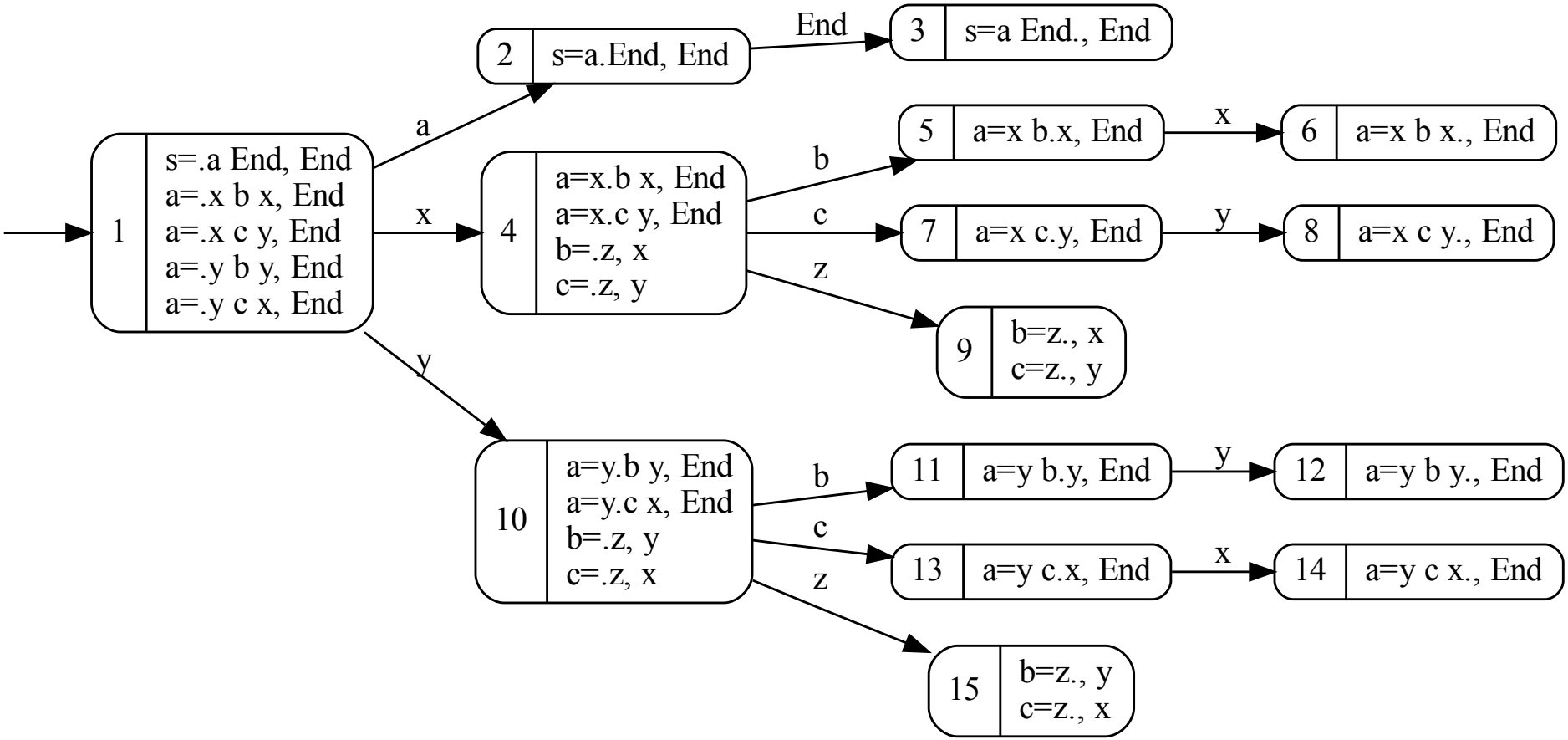
- Conflit à l'état 9 : R(b) ; R(c)

Exercice

Construire l'automate LR(1)

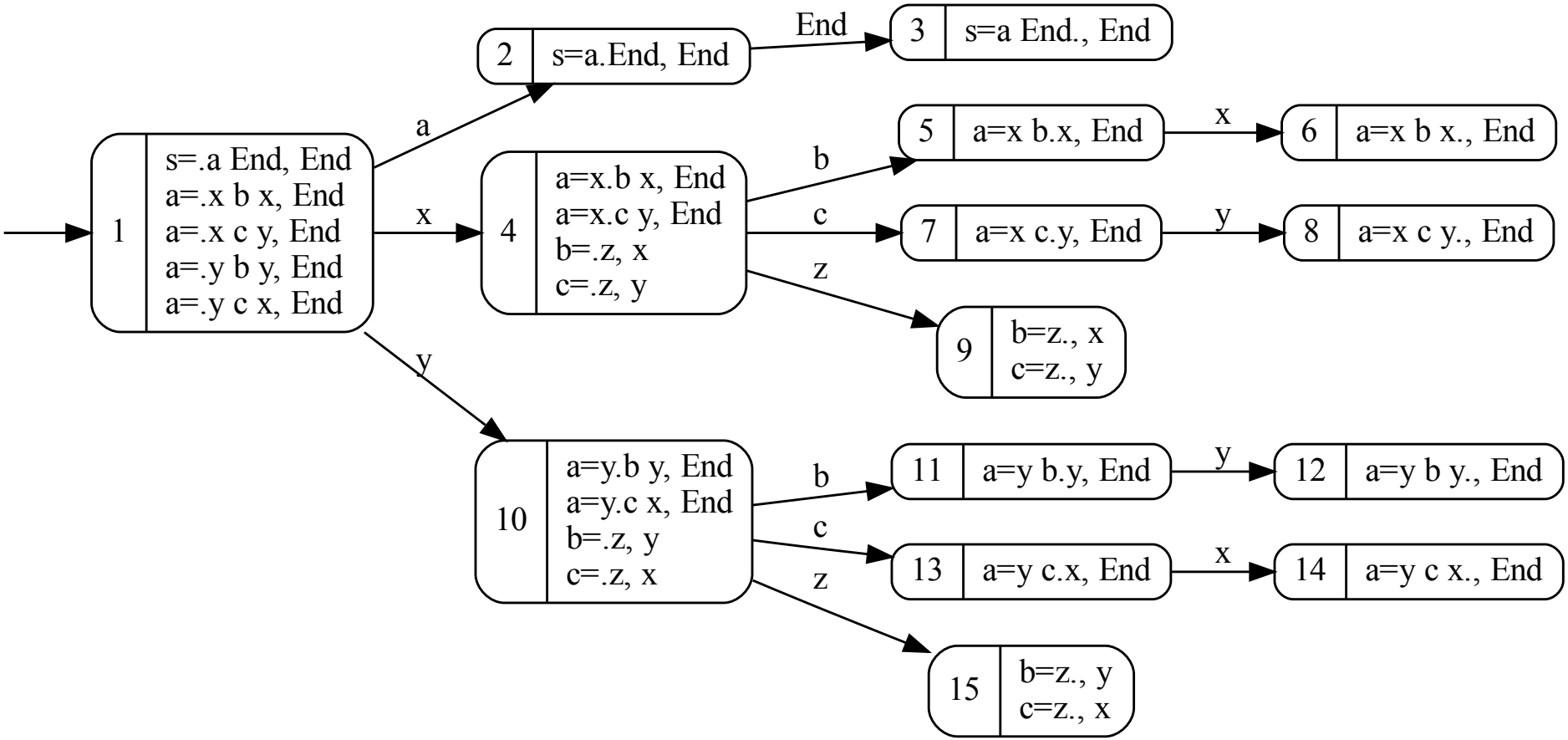
```
a = x b x | x c y | y b y | y c x ;
b = z ;
c = z ;
```

Exercice
Construire l'automate LR(1)



Déterminer les actions

Exercice
Construire l'automate LR(1)

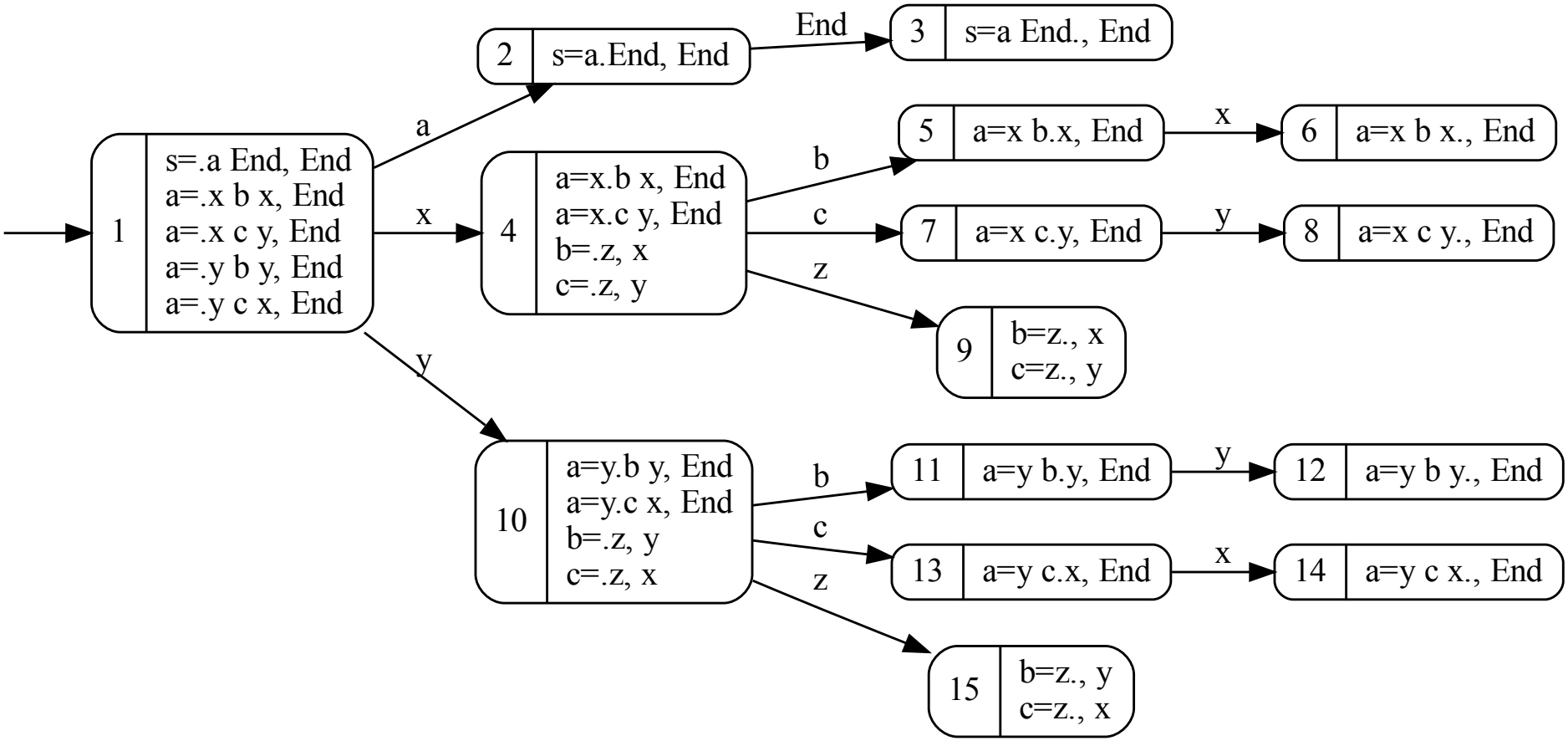


Déterminer les actions

- État 9 : x→R(b) ; y→R(c)
- État 15 : y→R(b) ; x→R(c)

Exercice

Évaluer l'automate



- Séquence de jetons = « x z y End »

Conflits LR(1)

Problème

- L'algo est valide si une seule action est possible par jeton de prévision par état
- L'automate construit ne l'est pas forcément

Solutions

- Utiliser une analyse plus forte
- Réécrire la grammaire

Conflits LR(1)

Grammaire ambiguë

```
e = e '+' e | i ;
```

Grammaire LR(k) pour $k > 1$

```
e = f x x | g x y ;
f = z ;
g = z ;
```

Grammaire pas LR(k) pour aucun k

```
e = x e x | ;
```

Autres analyses syntaxiques

Analyse syntaxique LALR(k)

- *Look-Ahead Left-to-right, Rightmost derivation parser*
- Idée : compresser les automates LR(k) qui sont **gros**
- Fusionner les nœuds LR(k) qui ont les même items (en excluant les prévisions)

Gain

- Même taille finale que LR(0)

Pertes

- Possiblement des conflits réductions/réductions
- Pas si gênant en pratique

Analyse syntaxique GLR

- *Generalized Left-to-right Rightmost derivation parser*
- Idée : en cas de conflit, on tente toutes les actions possibles
- On forke la pile et on tente
- Optimisation : partager les morceaux de pile
- Peut retourner tous les arbres (ou le premier qu'on trouve)

Gain

- Gères les grammaires conflictuelles
- Dont les grammaires ambiguës

Perte

- Complexité $O(n^3)$
- Mais reste $O(n)$ si entrée non-conflictuelle
- Le premier arbre qu'on trouve n'est peut-être pas le « bon »
  $\rightarrow$ Se méfier des outils (et des gens) qui utilisent GLR

Analyse syntaxique LL(k)

- *Left-to-right, Leftmost derivation parser*

Principe d'un parseur descendant récursif

- On regarde le(s) jeton(s) qui s'en vien(nen)t
- Pour savoir quelle dérivation appliquer
- Sans *backtrack* !
  - On veut la **bonne** dérivation !
  - On veut du $O(n)$!

Contrainte

- Il ne peut y avoir qu'un seule **bonne** dérivation
- Des conflits sont possibles s'il y a des **récursivité gauches**

Mise en œuvre LL(1)

Une pile

- Indiquant le **futur** à reconnaître
- La liste des jetons et productions qu'on cherche
- (c'est aussi un automate à pile quelque part dessous)

Une table indiquant

- Pour la production P courante
- Et le jeton j qui s'en vient
- Quelle dérivation $P \rightarrow \alpha$ appliquer
- $\rightarrow$ C'est en gros celle où $j \in \text{Premier}(\alpha)$
- $\rightarrow$ Mais c'est plus compliqué si α est nullable

Récursivité gauche

Directe

- Une alternative de la forme « $A \rightarrow A\alpha$ »

Indirecte

Des alternatives de la forme

- $A_0 \rightarrow \beta_0 A_1 \alpha_0$
- $A_1 \rightarrow \beta_1 A_2 \alpha_1$
- ...
- $A_n \rightarrow \beta_n A_0 \alpha_n$

Où tous les β sont nullables

- Exercice: montrer que $A_0 \rightarrow^* A_0\alpha$

Solution pour LL-iser ?

- Éliminer les récursions gauche
- $\rightarrow$ Ça complique la grammaire
- $\rightarrow$ C'est pas toujours faisable

Conclusion générale

- Beaucoup de sujets et sous-domaines en compilation
- Modernité et tradition
- On a simplement effleuré des concept de base
- Les graphes c'est quand même utile (des fois)

- Merci pour votre écoute