Compilatoare

Mihnea Muraru
mihnea.muraru@upb.ro

2023–2024, semestrul 1

Partea I

Introducere

Cuprins

Cuprins

Notare

- Teste de curs: 1
- Laborator: 1
- Teme: 3 standard (1 + 1.5 + 1.5) + 1 temă bonus (0.5)
- Examen: 4

Desfășurarea cursului

- Recapitularea cursului anterior
- Predare
- Test din cursul anterior
- Feedback despre cursul curent (în formular)

Cuprins

Ce vom studia?

- Concepte fundamentale: analiză lexicală/sintactică/semantică, generare de cod, optimizări, gestiunea automată a memoriei
- Instrumente specifice de dezvoltare: ANTLR, StringTemplate
- Rezultatul: un compilator pentru limbajul COOL, cu generare de cod pentru arhitectura MIPS

Bibliografie

- Cooper, K., Torczon, L. (2023). *Engineering a Compiler (3rd Edition)*. Elsevier.
- Aho, A. V., Lam, M. S., Sethi, R., Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley.
- Parr, T. (2013). *The definitive ANTLR 4 reference*. Pragmatic Bookshelf.
- Parr, T. (2009). *Language implementation patterns: create your own domain-specific and general programming languages*. Pragmatic Bookshelf.
- *CS143, Compilers*. Curs Stanford. `http://web.stanford.edu/class/cs143/`.

Cuprins

Analiza lexicală

- Determinarea structurii cuvintelor și separatorilor (engl. *tokens*), precum și a categoriei acestora

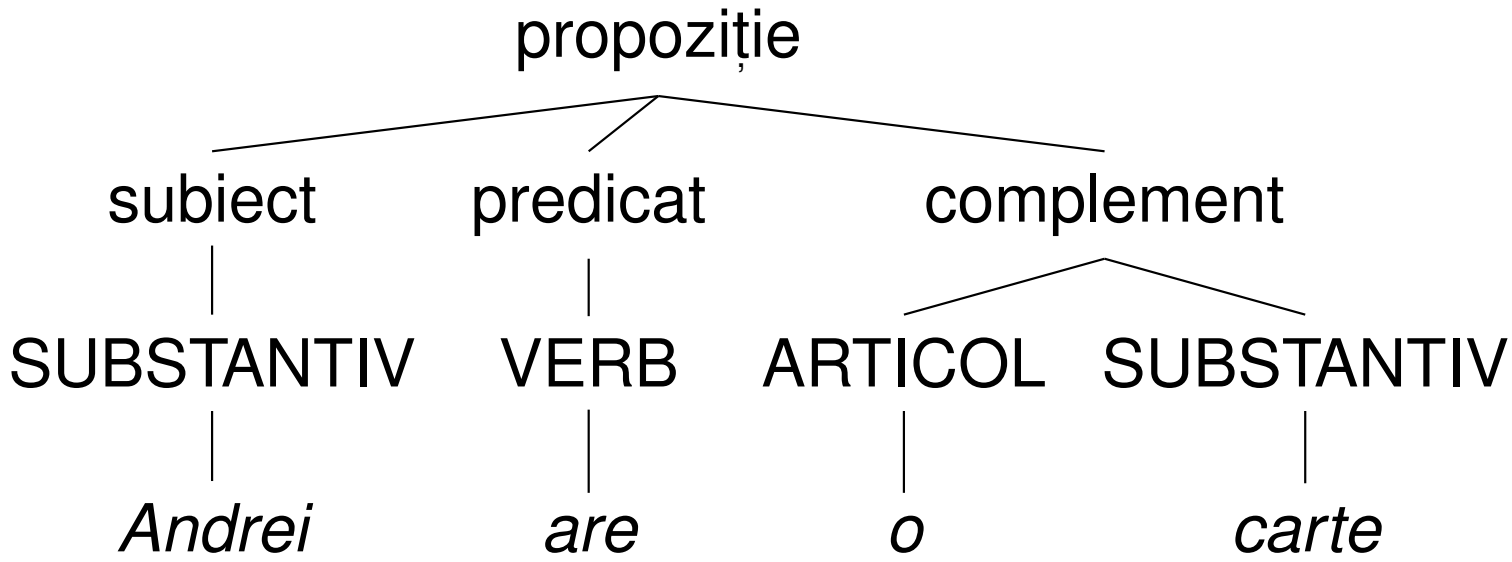
- Exemple:

Limba română		Programare	
Token	Categorie	Token	Categorie
Andrei	substantiv propriu	`if`	cuvânt cheie
carte	substantiv comun	`x`	variabilă
o	articol	`60`	număr
are	verb	`=`	operator
_ , .	separatori	_ `\t ;`	separatori

Analiza sintactică I

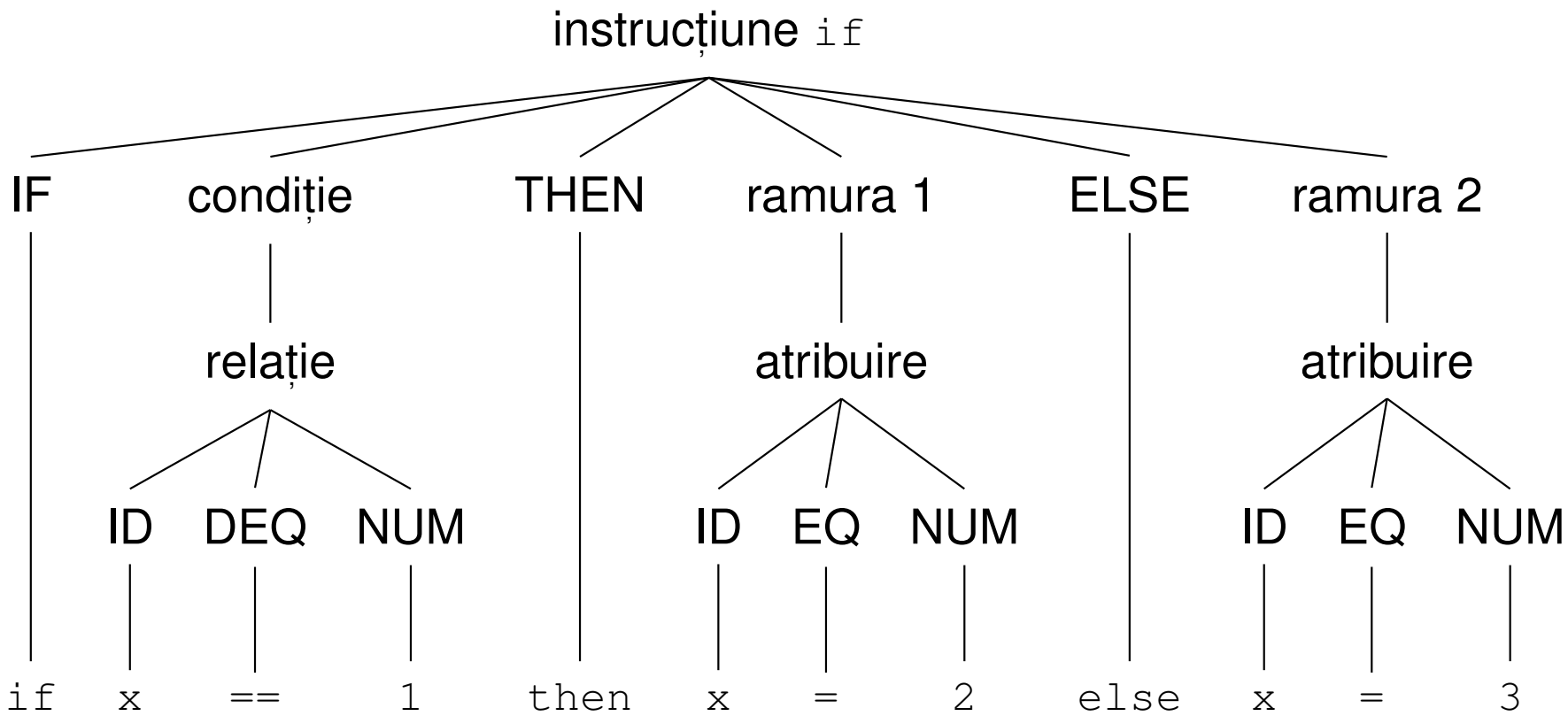
- Determinarea structurii ierarhice a propozițiilor, în baza cuvintelor constitutive și a relațiilor dintre acestea

- Exemplu în limba română:



Analiza sintactică II

- Exemplu în programare:



Analiza semantică I

- Determinarea sensului propozițiilor
- Rezolvarea referințelor: ce desemnează un anumit simbol?

Exemplu în limba română:

Andrei are o carte. Cartea lui este albastră.

La cine se referă *lui*?

Exemplu în programare:

```
1  void f(int x) {
2        int x;
3        return x;
4  }
```

La cine se referă `x`?

Analiza semantică II

- Verificarea compatibilității dintre entități

Exemplu în limba română:

Cartea și-a luat zborul.

Este acest lucru posibil?

Exemplu în programare:

```
1 Carte c;
2 zboara(c);
```

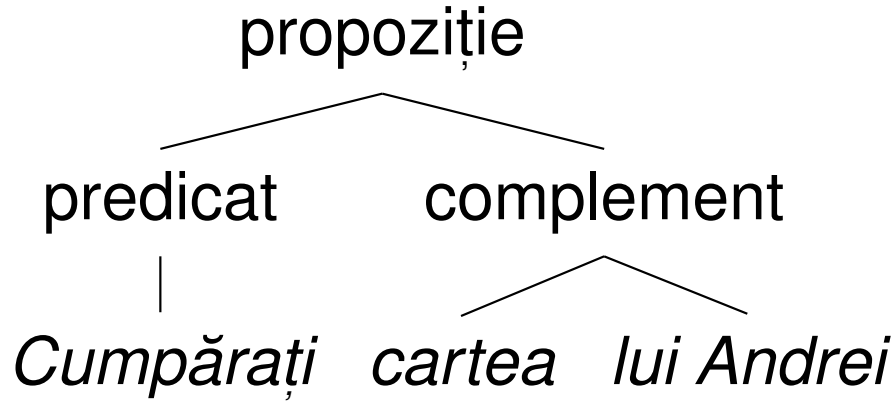
Parametrul lui `zboară` poate fi de tipul `Carte`?

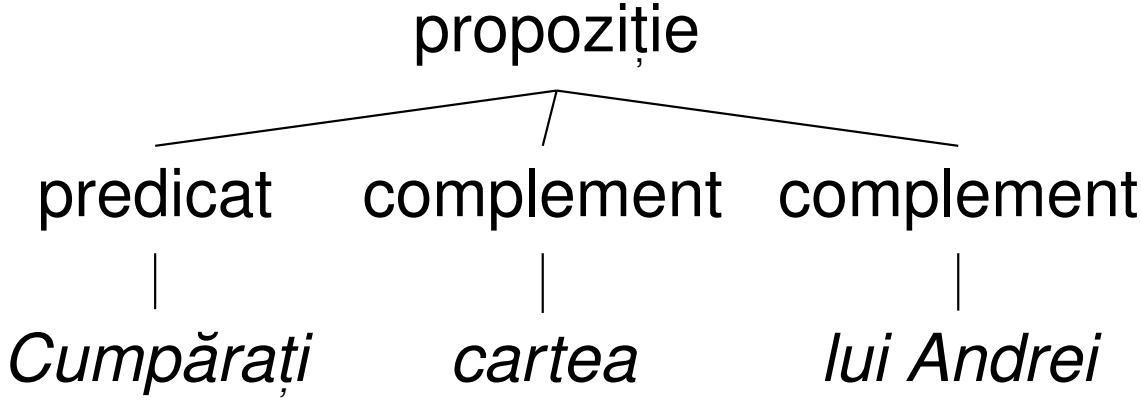
Exemple I

- *Cumăpraţi cartea lui Anrdei.* — eronată lexical
- *Cumpărați cartea Andrei lui.* — corectă lexical, eronată sintactic (structură inexistentă)
- *Cumpărați cartea lui Andrei.* — corectă lexical, eronată sintactic (ambiguă, două structuri posibile, v. slide-ul următor)
- *Cumpărați Victor lui Andrei.* — corectă lexical și sintactic, eronată semantic (sens inexistent)

Exemple II

- *Cumpărați cartea lui Andrei.* (continuare)
  — două structuri sintactice ⇒ ambiguitate





Generare de cod

- Traducerea programului scris în limbajul sursă (engl. *source language*) într-un alt limbaj (engl. *target language*)
- De obicei, într-un limbaj de nivel mai scăzut (e.g. de asamblare)
- Alteori, tot într-un limbaj de nivel înalt (e.g. Haskell $\rightarrow$ JavaScript)
- În interiorul compilatorului, într-un limbaj intermediar (engl. *intermediate language*), pe baza căruia se realizează diverse prelucrări (e.g. optimizări)

Optimizare

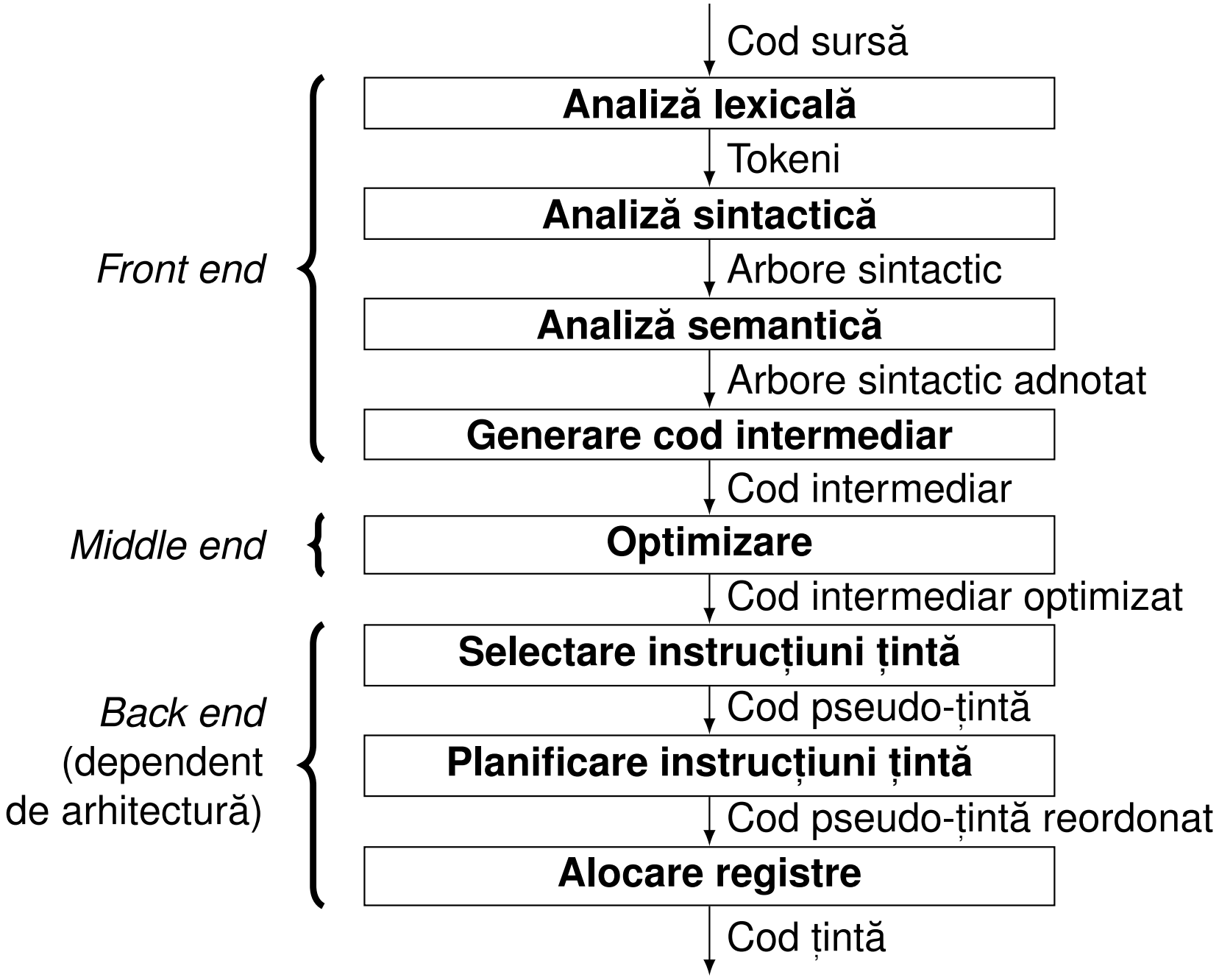
- Prelucrarea automată a programului, în scopul reducerii resurselor utilizate: timp de execuție, memorie, dimensiunea codului generat
- Exemplu: `x + 0` $\rightarrow$ `x` (simplificare algebrică)
- Termen ușor impropriu; alternativ, *îmbunătățire*
- În prezent, cea mai elaborată etapă a compilării!

Gestiunea automată a memoriei

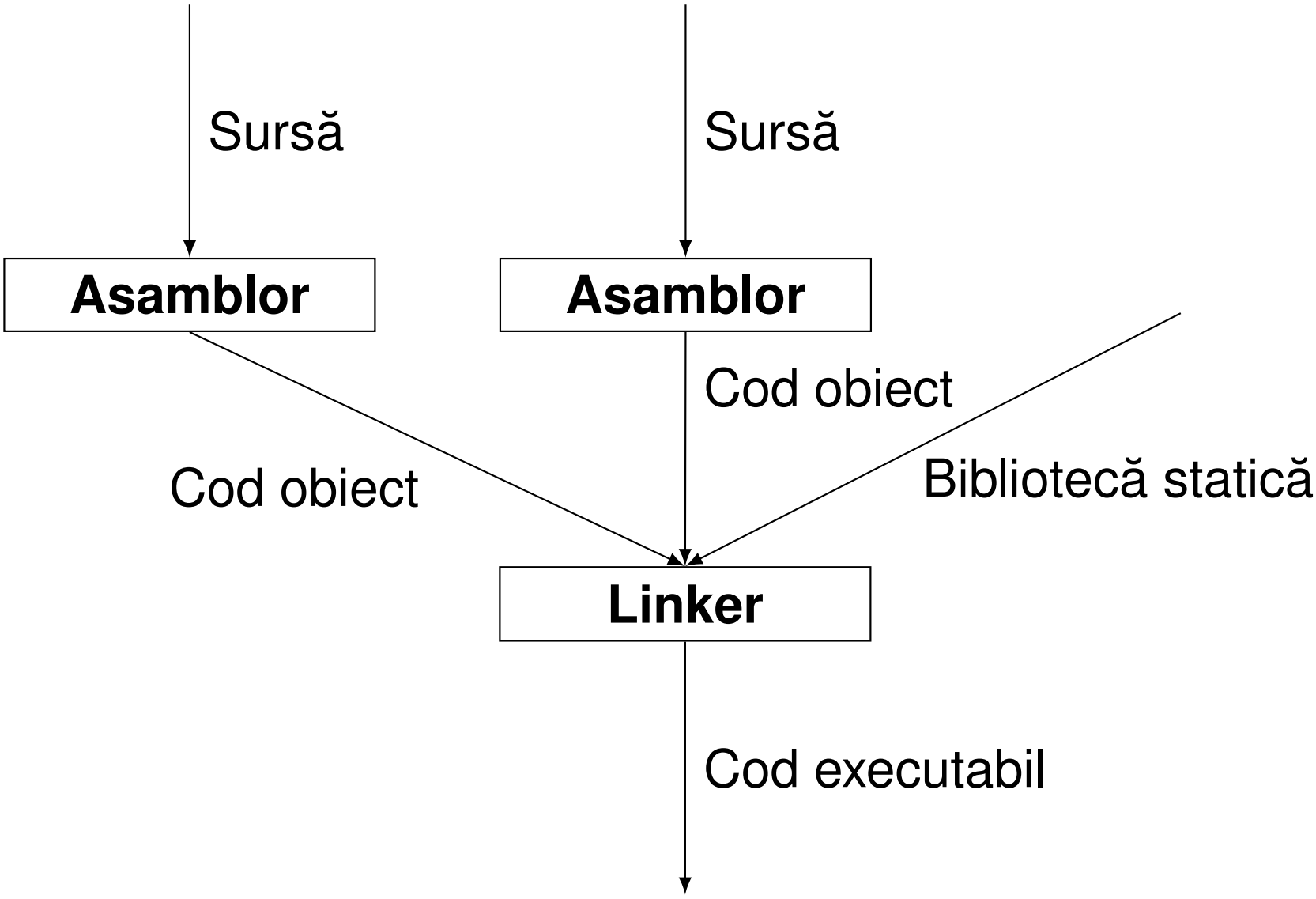
- Adnotarea și mutarea zonelor de memorie, precum și eliberarea automată a celor care nu mai sunt utilizate de program (engl. *garbage collection*)

- Subtilități datorate faptului că algoritmii de eliberare a memoriei nu au foarte multă... memorie disponibilă

Structura unui compilator



Etapele ulterioare



Compilatoare vs. interpretoare I

Criteriu	Compilatoare	Interpretoare
Rol	Traducerea programului în alt limbaj	Rularea programului
Prelucrări	Procesarea masivă a codului	Utilizarea codului (aproape) ca atare
Limbaje	Asociate mai degrabă celor de nivel scăzut	Asociate mai degrabă celor de nivel înalt

Compilatoare vs. interpretoare II

- Primele etape, până la analiza semantică, inclusiv, comune compilatoarelor și interpretoarelor
- Prezența amândurora în cazul anumitor limbaje, ca Java (javac, JVM, compilator JIT)

Cuprins

Limbajul COOL

```
1  class Main inherits IO {
2      main() : IO {
3          out_string("Compile me, please!\n")
4      };
5  };
```

Limbajul COOL

- *Classroom Object-Oriented Language*
- Clase, moștenire, tip static vs. tip dinamic, *dynamic dispatch*
- Verificare statică și dinamică a tipurilor
- Construcții de limbaj: `if`, `while`, `case`, `let`

Partea II

Analiza lexicală

Cuprins

Cuprins

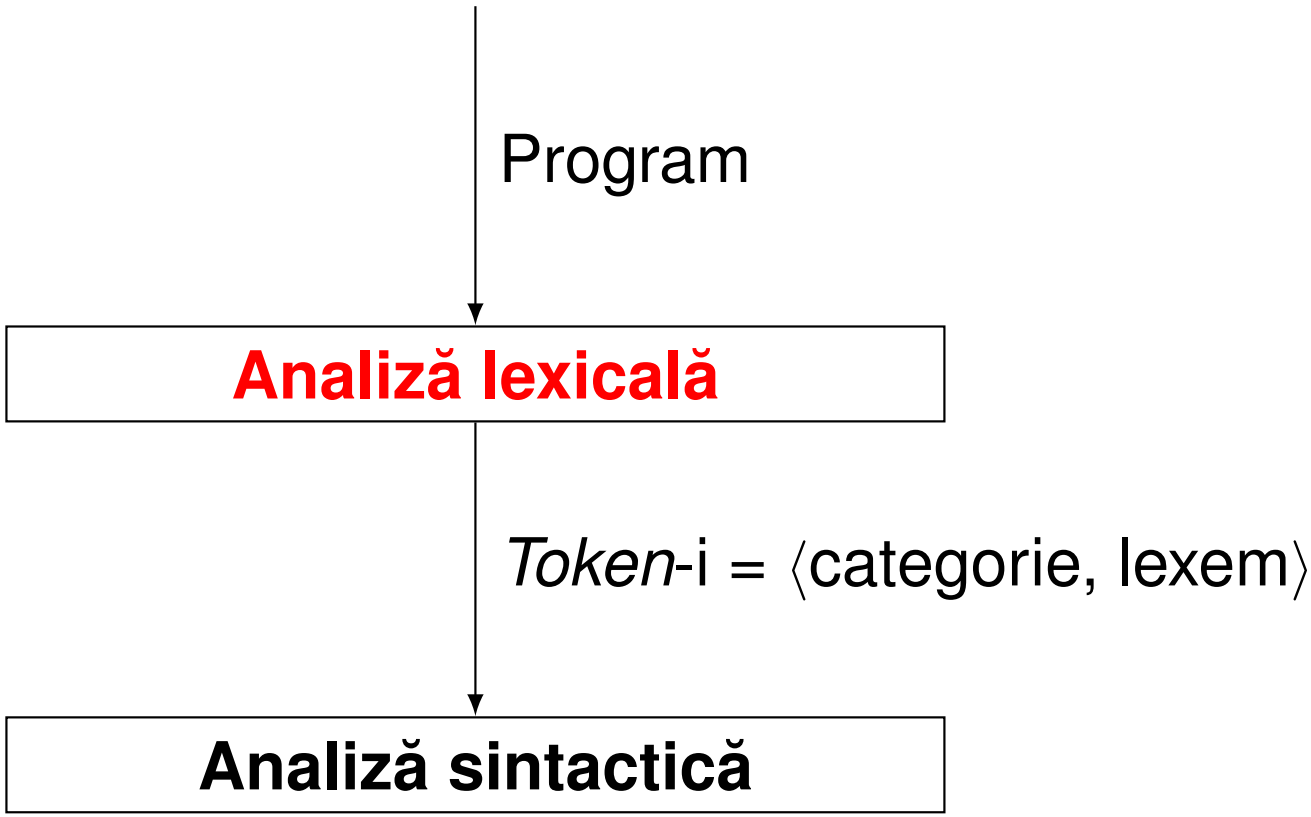
Analiza lexicală

- Determinarea structurii cuvintelor și separatorilor (engl. *tokens*), precum și a categoriei acestora

- Exemple:

Limba română		Programare	
Token	Categorie	*Token*	Categorie
Andrei	substantiv propriu	`if`	cuvânt cheie
carte	substantiv comun	`x`	variabilă
o	articol	`60`	număr
are	verb	`=`	operator
_ , .	separatori	_ `\t ;`	separatori

Context


Program
Analiză lexicală
Token-i = ⟨categorie, lexem⟩
Analiză sintactică

Exemplu

```
"let x:Int<-0\n\tin x+1"
```

Exemplu

```
"let x:Int<-0\n\tin x+1"
```

- ⟨*Let*, `"let"`⟩

Exemplu

```
"let x:Int<-0\n\tin x+1"
```

- ⟨*Let*, `"let"`⟩
- ⟨*Whitespace*, `" "`⟩

Exemplu

```
"let x:Int<-0\n\tin x+1"
```

- ⟨*Let*, `"let"`⟩
- ⟨*Whitespace*, `" "`⟩
- ⟨*ObjectId*, `"x"`⟩

Exemplu

```
"let x:Int<-0\n\tin x+1"
```

- ⟨*Let*, `"let"`⟩
- ⟨*Whitespace*, `" "`⟩
- ⟨*ObjectId*, `"x"`⟩
- ⟨*Colon*, `":"`⟩

Exemplu

```
"let x:Int<-0\n\tin x+1"
```

- ⟨*Let*, `"let"`⟩
- ⟨*Whitespace*, `" "`⟩
- ⟨*ObjectId*, `"x"`⟩
- ⟨*Colon*, `":"`⟩
- ⟨*TypeId*, `"Int"`⟩

Exemplu

```
"let x:Int<-0\n\tin x+1"
```

- ⟨*Let*, `"let"`⟩
- ⟨*Whitespace*, `" "`⟩
- ⟨*ObjectId*, `"x"`⟩
- ⟨*Colon*, `":"`⟩
- ⟨*TypeId*, `"Int"`⟩
- ⟨*Assign*, `"<-"`⟩

Exemplu

```
"let x:Int<-0\n\tin x+1"
```

- ⟨*Let*, `"let"`⟩
- ⟨*Whitespace*, `" "`⟩
- ⟨*ObjectId*, `"x"`⟩
- ⟨*Colon*, `":"`⟩
- ⟨*TypeId*, `"Int"`⟩
- ⟨*Assign*, `"<-"`⟩
- ⟨*Int*, `"0"`⟩

Exemplu

```
"let x:Int<-0\n\tin x+1"
```

- ⟨*Let*, `"let"`⟩
- ⟨*Whitespace*, `" "`⟩
- ⟨*ObjectId*, `"x"`⟩
- ⟨*Colon*, `":"`⟩
- ⟨*TypeId*, `"Int"`⟩
- ⟨*Assign*, `"<-"`⟩
- ⟨*Int*, `"0"`⟩
- ⟨*Whitespace*, `"\n\t"`⟩

Exemplu

```
"let x:Int<-0\n\tin x+1"
```

- ⟨*Let*, `"let"`⟩
- ⟨*Whitespace*, `" "`⟩
- ⟨*ObjectId*, `"x"`⟩
- ⟨*Colon*, `":"`⟩
- ⟨*TypeId*, `"Int"`⟩
- ⟨*Assign*, `"<-"`⟩
- ⟨*Int*, `"0"`⟩
- ⟨*Whitespace*, `"\n\t"`⟩
- ⟨*In*, `"in"`⟩

Exemplu

```
"let x:Int<-0\n\tin x+1"
```

- ⟨*Let*, `"let"`⟩
- ⟨*Whitespace*, `" "`⟩
- ⟨*ObjectId*, `"x"`⟩
- ⟨*Colon*, `":"`⟩
- ⟨*TypeId*, `"Int"`⟩
- ⟨*Assign*, `"<-"`⟩
- ⟨*Int*, `"0"`⟩
- ⟨*Whitespace*, `"\n\t"`⟩
- ⟨*In*, `"in"`⟩
- ⟨*Whitespace*, `" "`⟩

Exemplu

```
"let x:Int<-0\n\tin x+1"
```

- ⟨*Let*, `"let"`⟩
- ⟨*Whitespace*, `" "`⟩
- ⟨*ObjectId*, `"x"`⟩
- ⟨*Colon*, `":"`⟩
- ⟨*TypeId*, `"Int"`⟩
- ⟨*Assign*, `"<-"`⟩
- ⟨*Int*, `"0"`⟩
- ⟨*Whitespace*, `"\n\t"`⟩
- ⟨*In*, `"in"`⟩
- ⟨*Whitespace*, `" "`⟩
- ⟨*ObjectId*, `"x"`⟩

Exemplu

```
"let x:Int<-0\n\tin x+1"
```

- ⟨*Let*, `"let"`⟩
- ⟨*Whitespace*, `" "`⟩
- ⟨*ObjectId*, `"x"`⟩
- ⟨*Colon*, `":"`⟩
- ⟨*TypeId*, `"Int"`⟩
- ⟨*Assign*, `"<-"`⟩
- ⟨*Int*, `"0"`⟩
- ⟨*Whitespace*, `"\n\t"`⟩
- ⟨*In*, `"in"`⟩
- ⟨*Whitespace*, `" "`⟩
- ⟨*ObjectId*, `"x"`⟩
- ⟨*Plus*, `"+"`⟩

Exemplu

```
"let x:Int<-0\n\tin x+1"
```

- ⟨*Let*, `"let"`⟩
- ⟨*Whitespace*, `" "`⟩
- ⟨*ObjectId*, `"x"`⟩
- ⟨*Colon*, `":"`⟩
- ⟨*TypeId*, `"Int"`⟩
- ⟨*Assign*, `"<-"`⟩
- ⟨*Int*, `"0"`⟩
- ⟨*Whitespace*, `"\n\t"`⟩
- ⟨*In*, `"in"`⟩
- ⟨*Whitespace*, `" "`⟩
- ⟨*ObjectId*, `"x"`⟩
- ⟨*Plus*, `"+"`⟩
- ⟨*Int*, `"1"`⟩

Lookahead

- Cum realizăm distincția dintre operatorii `<` și `<-`, sau dintre variabila `i` și cuvântul cheie `in`?

- *Lookahead* = numărul de caractere următoare care trebuie consultate pentru a decide finalul sau continuarea unui *token*

- Ideal, *lookahead* mărginit de o constantă

Ambiguitate

- Cum realizăm în Java, la nivel lexical, distincția dintre operatorul `>>` (deplasare la dreapta) și două caractere `>` care închid o declarație generică, `Map<String, List<Integer>>`?
- Imposibil! Necesitatea informației contextuale, accesibilă de-abia la nivelul analizei sintactice!
- Soluția: extragerea a doi *token*-i `>` în locul celui compus, `>>` (Parr, 2013)

Cuprins

Motivație

- Structura lexicală a limbajului = mulțimea categoriilor *token*-ilor

- Descrierea lexemelor aferente categoriilor, realizată cu expresii regulate

Expresii regulate (ER)

Definiție în raport cu un alfabet Σ; A și B sunt ER:

- ε: șirul **vid**
- `'a'`: șirul cu un **singur** caracter, `"a"`
- $A + B$ (**reuniune**): șiruri descrise fie de A, fie de B
- AB (**concatenare**): șiruri rezultate din concatenarea unui șir descris de A cu un șir descris de B
- A^* (**închiderea Kleene**): șiruri rezultate din concatenarea oricâtor (inclusiv 0) șiruri descrise de A

Abrevieri și notații alternative

- $\texttt{'ab'} = \texttt{'a''b'}$
- $\sim\texttt{'a'}$: șiruri formate dintr-un unic caracter, diferit de $\texttt{'a'}$
- $[\texttt{ab}] = \texttt{'a'} + \texttt{'b'}$
- $[\texttt{a} - \texttt{z}] = \texttt{'a'} + \ldots + \texttt{'z'}$
- $\sim[\texttt{a} - \texttt{z}]$: șiruri formate dintr-un unic caracter, diferit de cele dintre $\texttt{'a'}$ și $\texttt{'z'}$
- $A \mid B = A + B$
- $A? = A + \varepsilon$
- $A^+ = AA^*$

Semantica formală a expresiilor regulate

$L(A)$ = limbajul (mulțimea șirurilor) descris de A

- $L(\varepsilon) = \{\texttt{""}\}$
- $L(\texttt{'a'}) = \{\texttt{"a"}\}$
- $L(A + B) = L(A) \cup L(B)$
- $L(AB) = \{ab \mid a \in L(A),\ b \in L(B)\}$
- $L(A^*) = \bigcup_{i \geq 0} L(A^i)$, cu $A^0 = \varepsilon$

Exemple

$\Sigma = \{0, 1\}$

- $L(0^*) =$

Exemple

$\Sigma = \{0, 1\}$

- $L(0^*) = \{\text{""}, 0, 00, \ldots\}$
- $L(0(0+1)) =$

Exemple

$\Sigma = \{0, 1\}$

- $L(0^*) = \{\text{""}, 0, 00, \ldots\}$
- $L(0(0+1)) = \{00, 01\}$
- $L((0+1)^*) =$

Exemple

$\Sigma = \{0, 1\}$

- $L(0^*) = \{\text{""}, 0, 00, \ldots\}$
- $L(0(0+1)) = \{00, 01\}$
- $L((0+1)^*) = \Sigma^*$ (toate șirurile)
- $L((0+1)^*(0^*+1^*)1) =$

Exemple

$\Sigma = \{0, 1\}$

- $L(0^*) = \{\text{""}, 0, 00, \ldots\}$
- $L(0(0+1)) = \{00, 01\}$
- $L((0+1)^*) = \Sigma^*$ (toate șirurile)
- $L((0+1)^*(0^*+1^*)1) =$
  $L((0+1)^*(0^*+1^*+00+01+10+11)1)$
  (corespondență mai mulți la unu între ER și limbaje)

Cuprins

Categorii uzuale I

- **Cuvinte cheie**: lexeme predefinite

Categorii uzuale I

- **Cuvinte cheie**: lexeme predefinite

  If = `'if'`

- **Număr întreg** fără semn: șir nevid de cifre

Categorii uzuale I

- **Cuvinte cheie**: lexeme predefinite

  $If = \texttt{'if'}$

- **Număr întreg** fără semn: șir nevid de cifre

  $Digit = [\texttt{0} - \texttt{9}]$
  $Int \quad = Digit^{+}$

- **Identificator**: cel puțin o literă/cifră/*underscore*, primul caracter nefiind cifră

Categorii uzuale I

- **Cuvinte cheie**: lexeme predefinite

  $If = \texttt{'if'}$

- **Număr întreg** fără semn: șir nevid de cifre

  $Digit = [\texttt{0} - \texttt{9}]$
  $Int \quad = Digit^{+}$

- **Identificator**: cel puțin o literă/cifră/*underscore*, primul caracter nefiind cifră

  $Letter = [\texttt{a} - \texttt{z}\texttt{A} - \texttt{Z}]$
  $Id \quad = (Letter + \texttt{'\_'})(Letter + Digit + \texttt{'\_'})^{*}$

Categorii uzuale II

- **Număr real** fără semn: parte întreagă, parte fracționară, exponent

Categorii uzuale II

- **Număr real** fără semn: parte întreagă, parte fracționară, exponent

  $$\begin{aligned} \textit{Digits} &= \textit{Digit}^{+} \\ \textit{Fraction} &= (\texttt{'.'}\ \textit{Digits}?)? \\ \textit{Exponent} &= (\texttt{'e'}\ (\texttt{'+'} + \texttt{'-'})?\ \textit{Digits})? \\ \textit{Real} &= \textit{Digits Fraction Exponent} \end{aligned}$$

- **Șir de caractere**: șir flancat de `'"'`, putând conține acest caracter doar dacă este precedat de *backslash* (*escaped*)

Categorii uzuale II

- **Număr real** fără semn: parte întreagă, parte fracționară, exponent

  Digits = *Digit*$^+$
  Fraction = (`'.'` *Digits*?)?
  Exponent = (`'e'` (`'+'` + `'-'`)? *Digits*)?
  Real = *Digits Fraction Exponent*

- **Șir de caractere**: șir flancat de `'"'`, putând conține acest caracter doar dacă este precedat de *backslash* (*escaped*)

  String = `'"'` (`'\\"'` + ∼`'"'`)* `'"'`
  (*backslash* însuși este *escaped*)

Spații albe

- **Spații albe**: unul sau mai multe caractere *blank*, linie nouă, *tab* etc.

Spații albe

- **Spații albe**: unul sau mai multe caractere *blank*, linie nouă, *tab* etc.

  Whitespace = [\r\n\t]$^+$

- De obicei, eliminate la nivelul analizei lexicale

- Alteori, importante în analiza sintactică, pentru descrierea corectă a structurii programului (v. Python)

Partiționarea unui șir folosind ER I

- Rolul unei ER: descrierea apartenenței unui șir la o anumită categorie lexicală
- Cum partiționăm un șir în *token*-i pornind de la întreaga specificație lexicală (toate ER)?

Partiționarea unui șir folosind ER II

Algoritm:

1. Construiește o unică ER, pornind de la ER aferente categoriilor indiviuale.

   $E = If + Int + Id + \ldots$

2. Pentru șirul de intrare $x_1 \ldots x_n$, verifică, pentru fiecare $1 \leq i \leq n$, dacă $x_1 \ldots x_i \in L(E)$.

3. În caz de succes, elimină prefixul și repetă de la (1).

Probleme în partiționare

- Succes pentru mai multe prefixe
  - Exemplu: `"5"` $\in L(Int)$ și `"5.2"` $\in L(Real)$
  - Soluție: alegerea celui mai lung (*maximal munch*)
- Succes pentru același prefix și mai multe ER
  - Exemplu: `"5"` $\in L(Int)$ și `"5"` $\in L(Real)$
  - Soluție: prioritizarea primei ER definite
- Eșec pentru toate prefixele (eroare lexicală)
  - Soluție: definirea unei categorii speciale, cu prioritate minimă, pentru șiruri incorecte

Cuprins

Motivație

- ER: modalitate de specificare a limbajelor

- Automate finite (AF): modalitate de implementare a răspunsului la întrebarea: pentru un șir s și o ER E, este adevărat că $s \in L(E)$?

Structura unui AF

- **Alfabet** de intrare, Σ
- Mulțime de **stări**, S
- Stare **inițială**, $s_0 \in S$
- Mulțime de stări **finale**, $F \subseteq S$
- Mulțime de **tranziții**, $s \rightarrow^a s'$

Acceptare și respingere

- **Acceptare**, dacă, după parcurgerea întregului șir, AF se află într-o stare finală

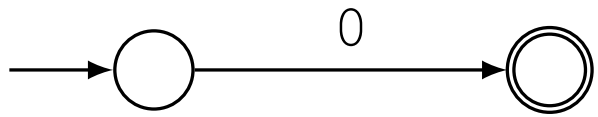
- **Respingere**, dacă
  - După parcurgerea șirului, AF nu este într-o stare finală SAU
  - Nicio tranziție nu mai poate fi efectuată înainte de sfârșitul șirului

Exemple I

- $E = 0$

Exemple I

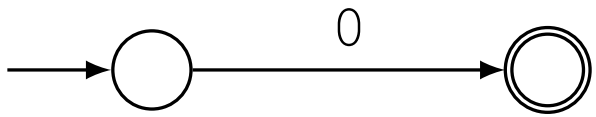
- $E = 0$



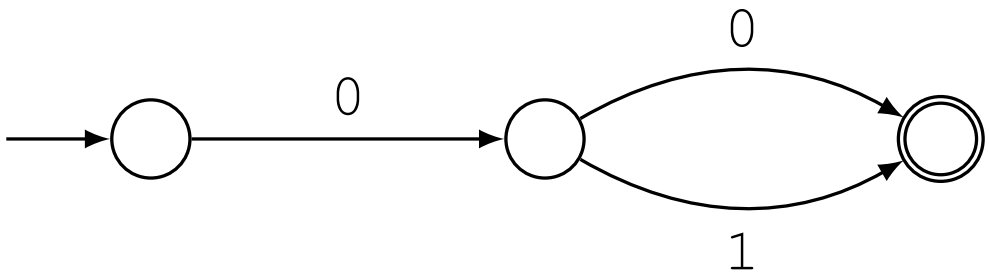
- $E = 0(0 + 1)$

Exemple I

- $E = 0$



- $E = 0(0+1)$

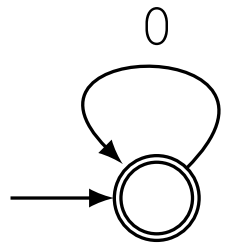


Exemple II

- $E = 0^*$

Exemple II

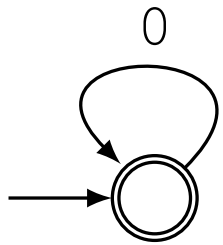
- $E = 0^*$



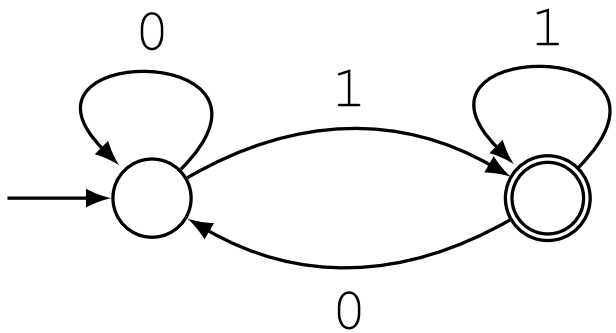
- $E = (0 + 1)^* 1$

Exemple II

- $E = 0^*$



- $E = (0 + 1)^* 1$

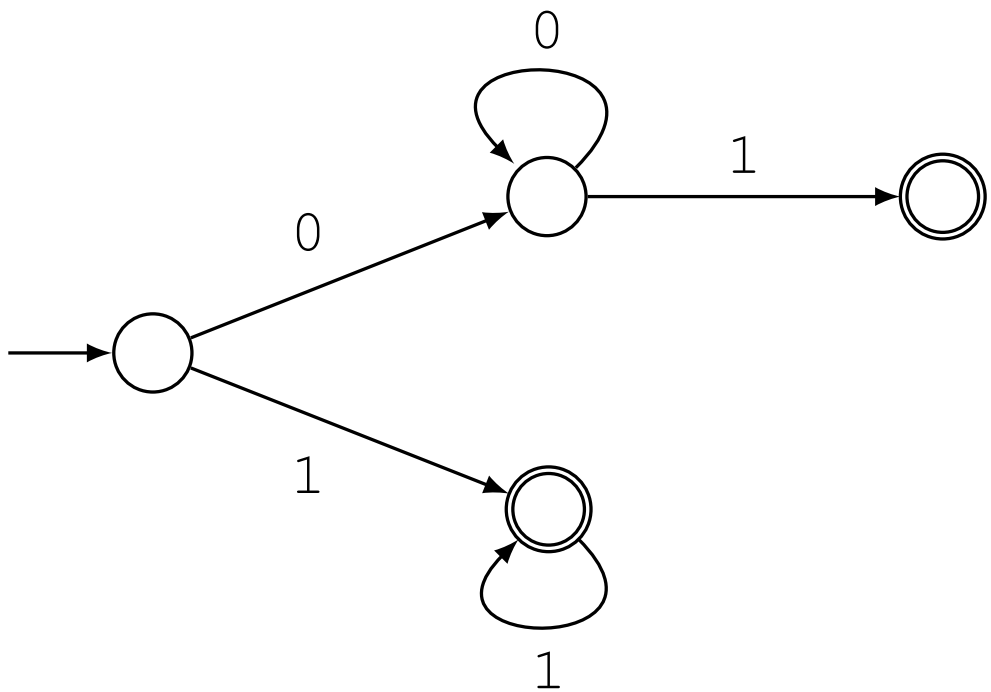


Exemple III

- $E = (0^* + 1^*)1$

Exemple III

- $E = (0^* + 1^*)1$

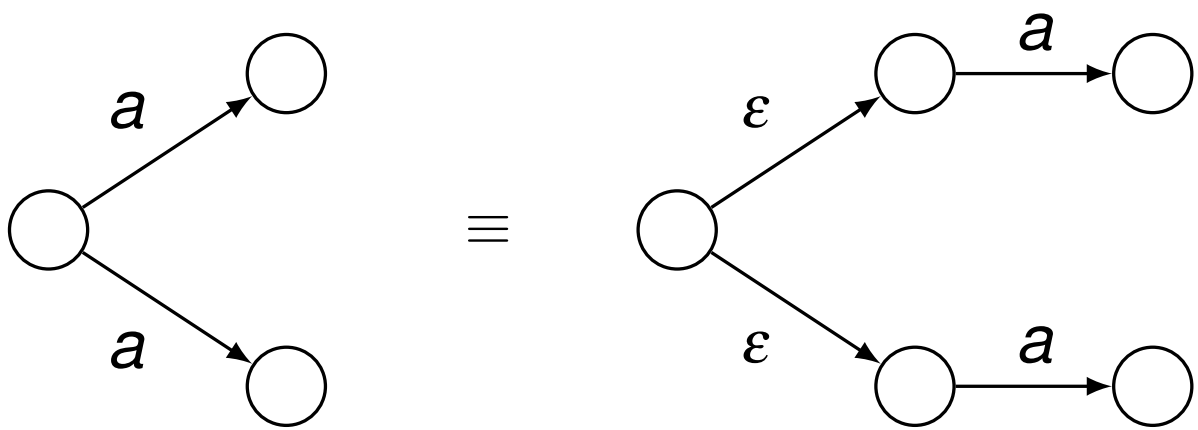


ε-tranziții

- $s \rightarrow^{\varepsilon} s'$: tranziția din starea s în starea s', fără citirea vreunui simbol
- AF **determinist** (AFD): cel mult o tranziție pentru o stare și un simbol, și absența ε-tranzițiilor
- AF **nedeterminist** (AFN): posibilitatea tranzițiilor multiple pentru o stare și un simbol, precum și a ε-tranzițiilor

Automate finite nedeterministe

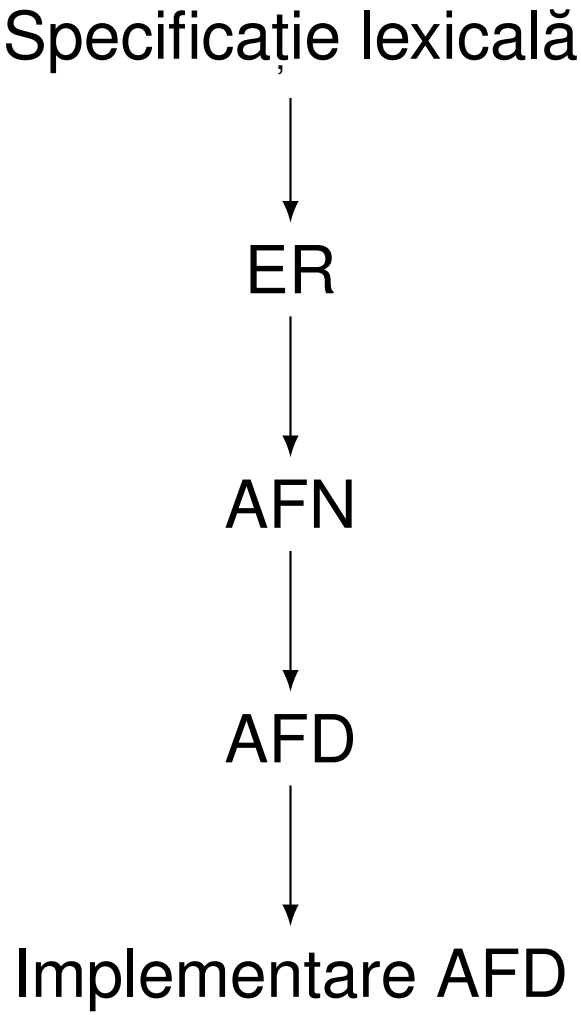
- Suficiența ε-tranzițiilor:



- Posibilitatea situării în mai multe stări simultan
- Acceptare dacă, după parcurgerea șirului, cel puțin o stare este finală
- De obicei, de dimensiune mai redusă decât un AFD echivalent, dar mai costisitor de executat

Cuprins

Etape



Transformarea ER în AFN I

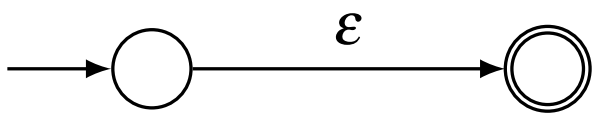
- AFN aferent ER E

- $E = \varepsilon$

Transformarea ER în AFN I

- AFN aferent ER E

- $E = \varepsilon$

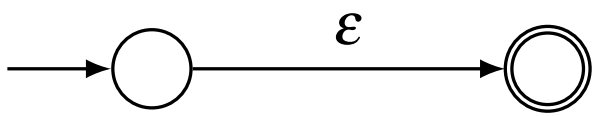


- $E = a$ (un simbol)

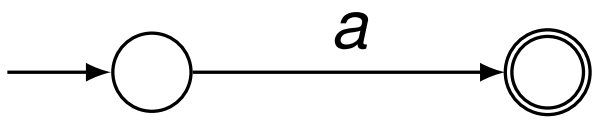
Transformarea ER în AFN I

- AFN aferent ER E

- $E = \varepsilon$



- $E = a$ (un simbol)

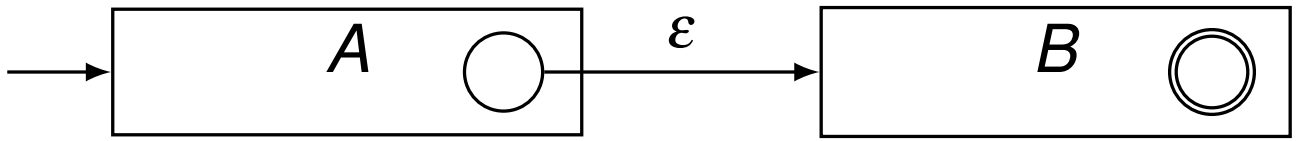


Transformarea ER în AFN II

- $E = AB$

Transformarea ER în AFN II

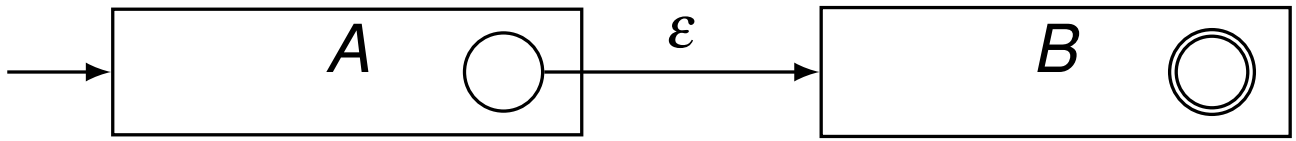
- $E = AB$



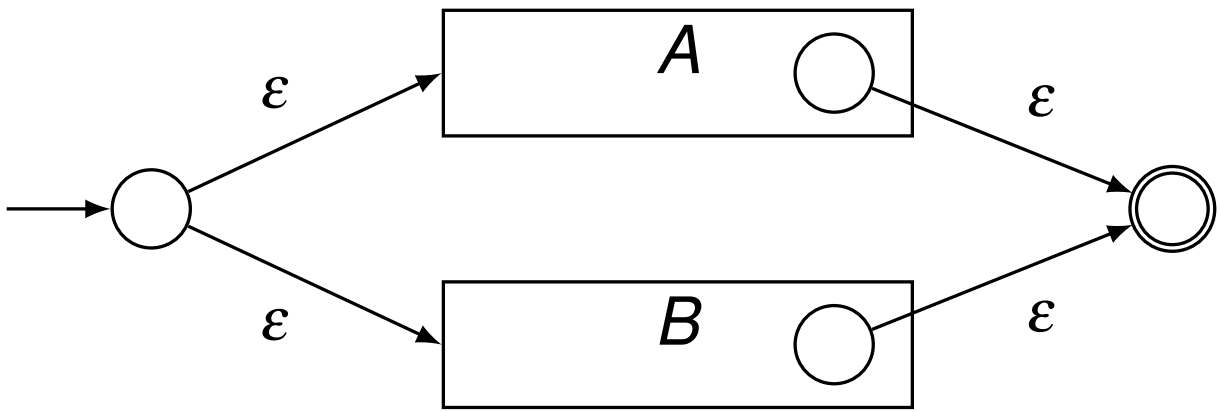
- $E = A + B$

Transformarea ER în AFN II

- $E = AB$



- $E = A + B$

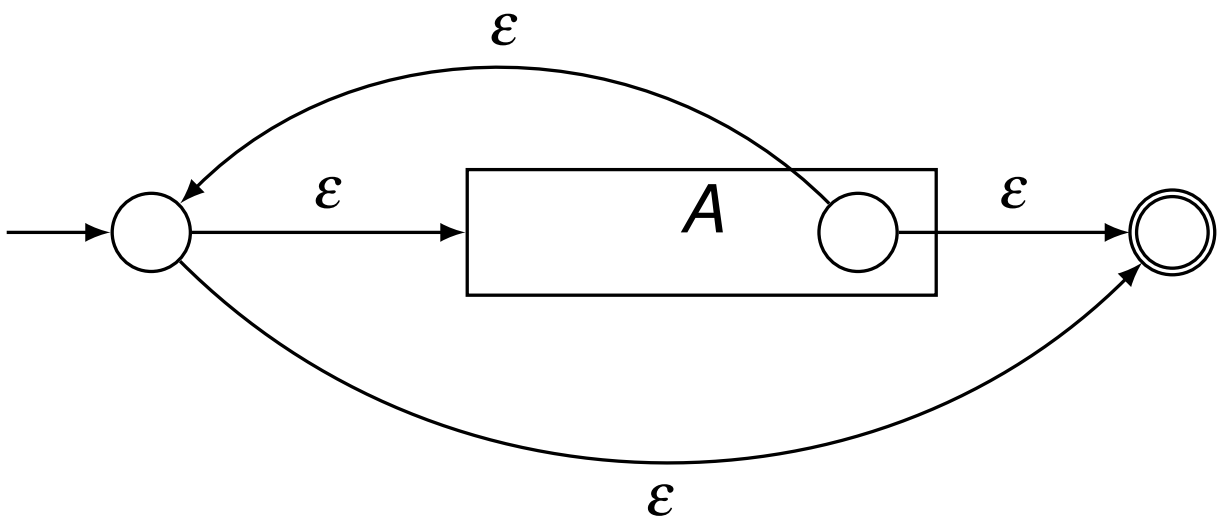


Transformarea ER în AFN III

- $E = A^*$

Transformarea ER în AFN III

- $E = A^*$

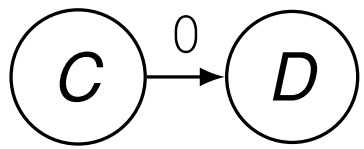


Exemplu

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

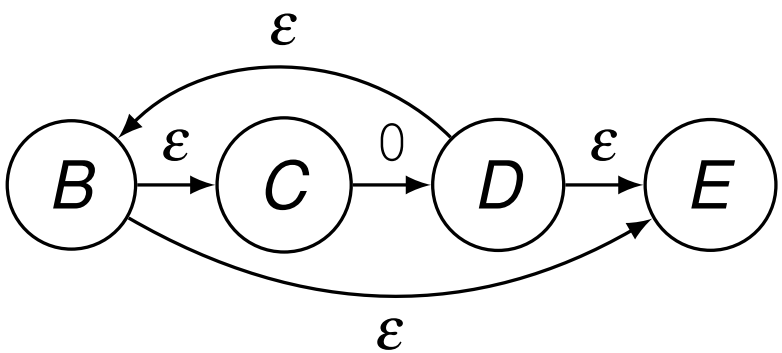
Exemplu

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



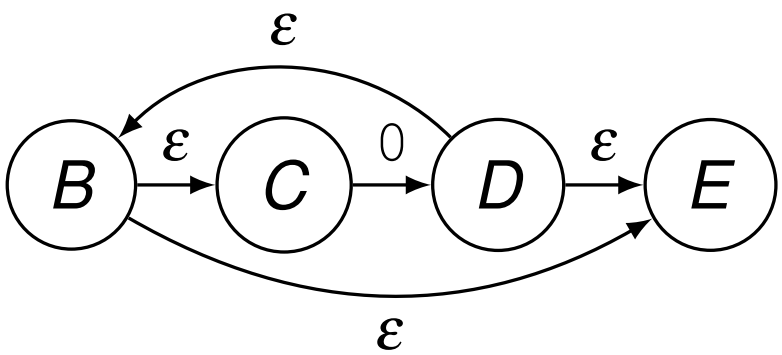
Exemplu

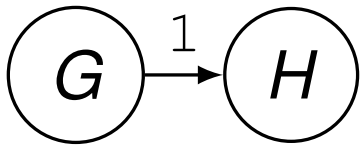
$$E = (0^* + 1^*)1$$



Exemplu

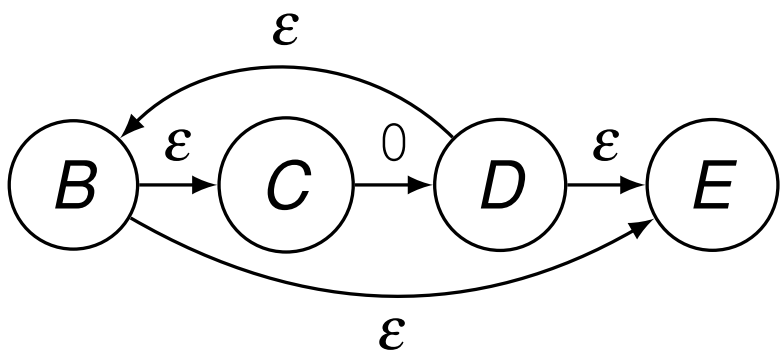
$$E = (0^* + 1^*)1$$

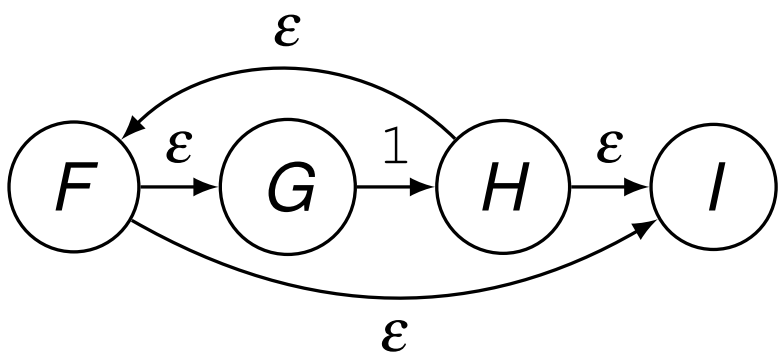




Exemplu

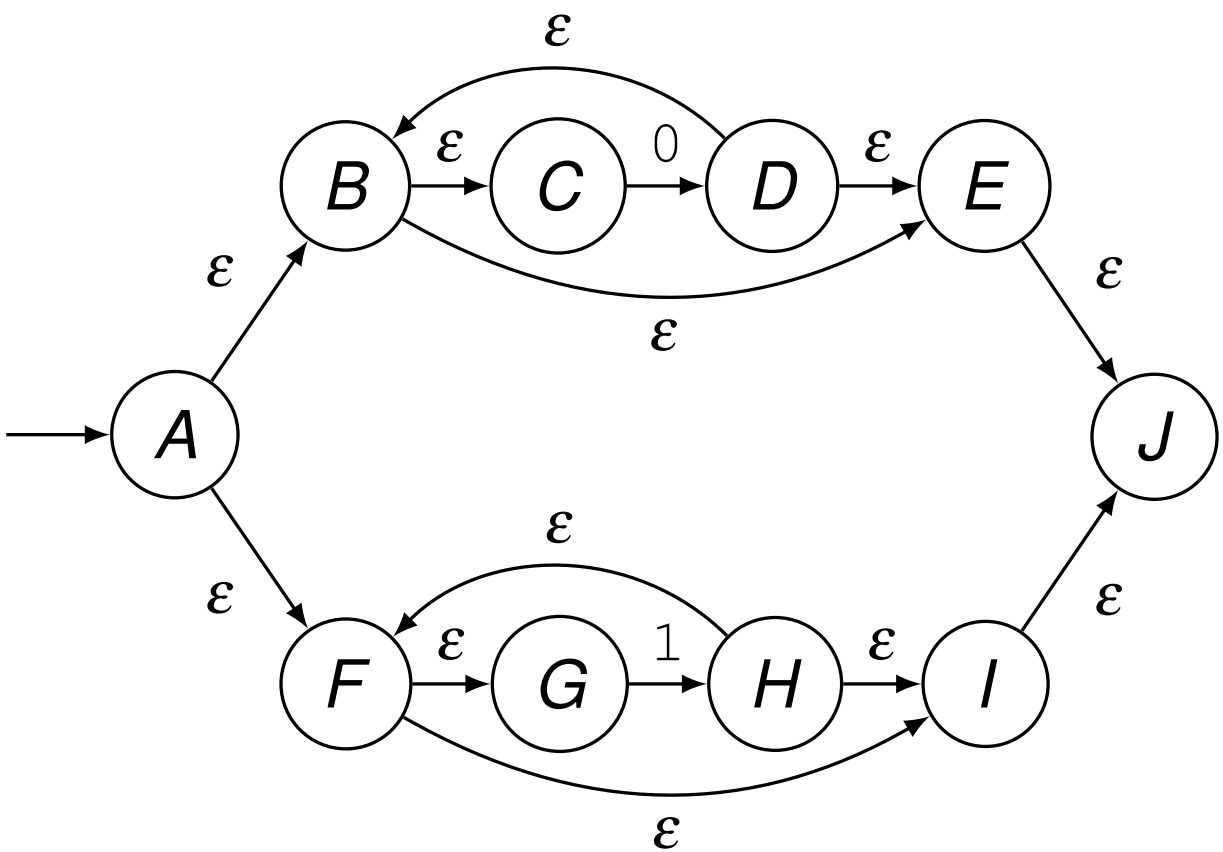
$$E = (0^* + 1^*)1$$





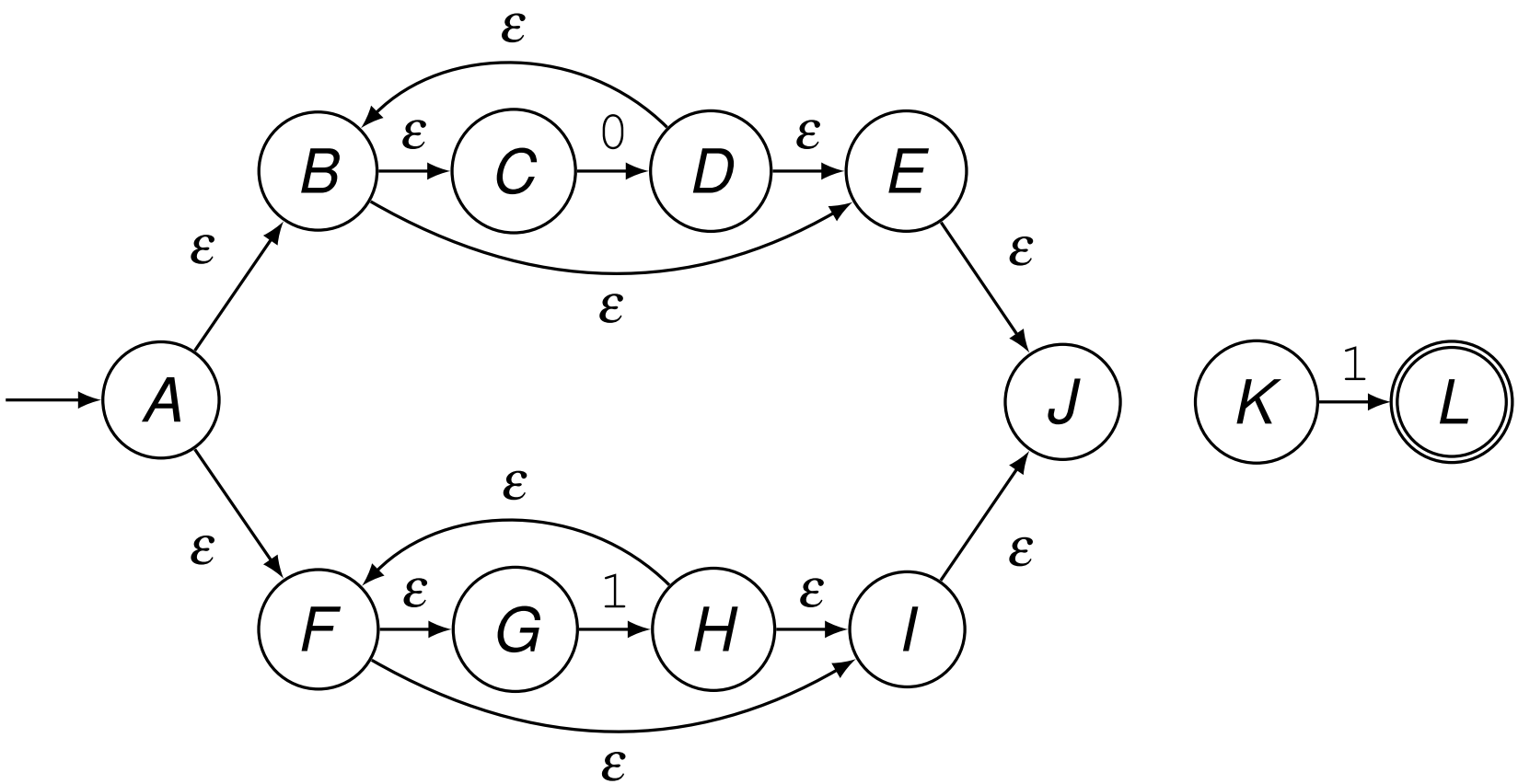
Exemplu

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



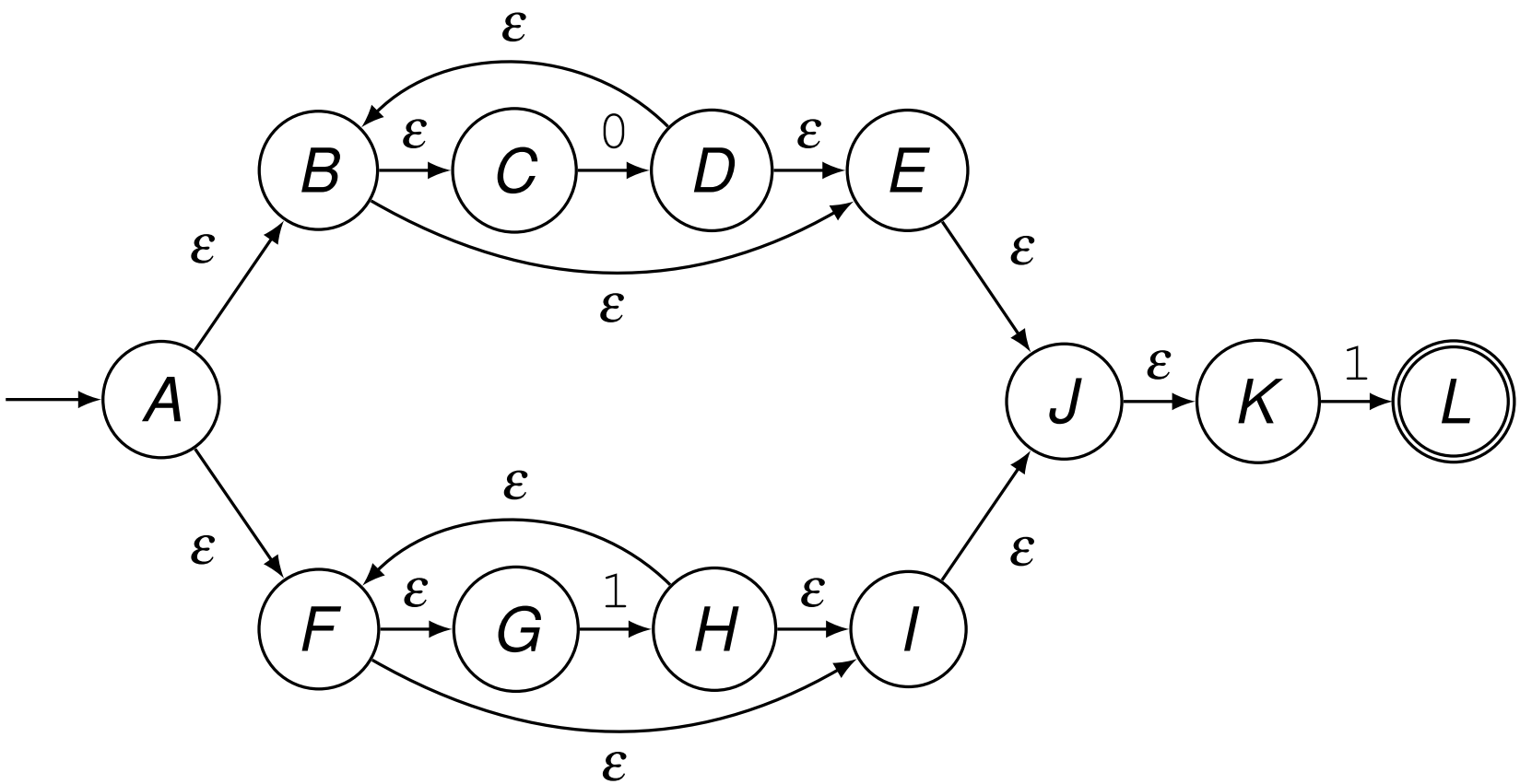
Exemplu

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



Exemplu

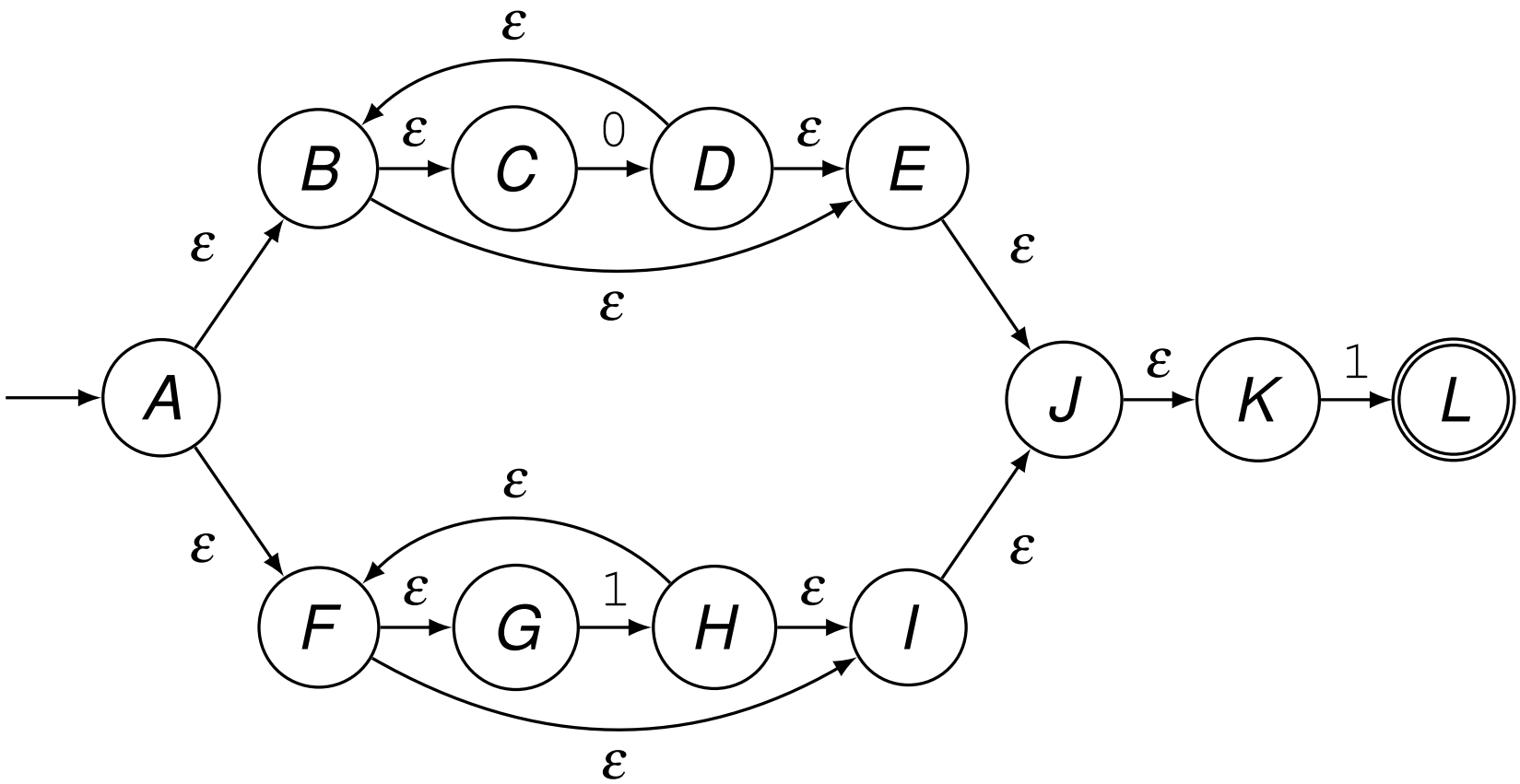
$$E = (0^* + 1^*)1$$



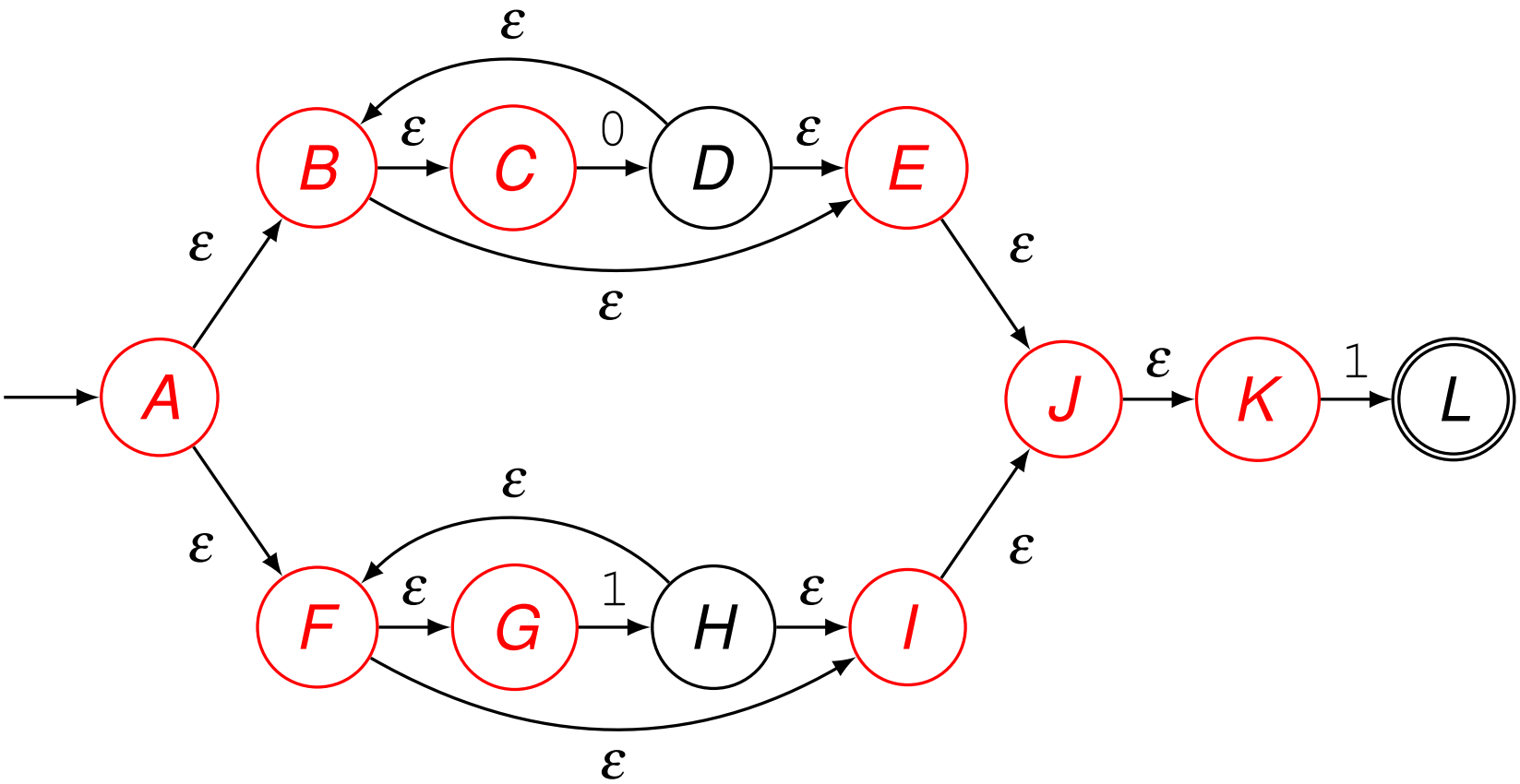
Transformarea unui AFN într-un AFD echivalent

- **Stările** AFD: mulțimi de stări ale AFN
- Starea **inițială** a AFD: mulțimea care conține starea inițială a AFN și orice alte stări accesibile prin ε-tranziții
- Stările **finale** ale AFD: mulțimi care conțin cel puțin o stare finală a AFN
- **Tranziții** $S \rightarrow^a S'$ în AFD: dacă S' este mulțimea tututor stărilor AFN în care se poate ajunge pornind de la stările din S, pentru simbolul a, ținând cont și de stările accesibile prin ε-tranziții

Exemplu

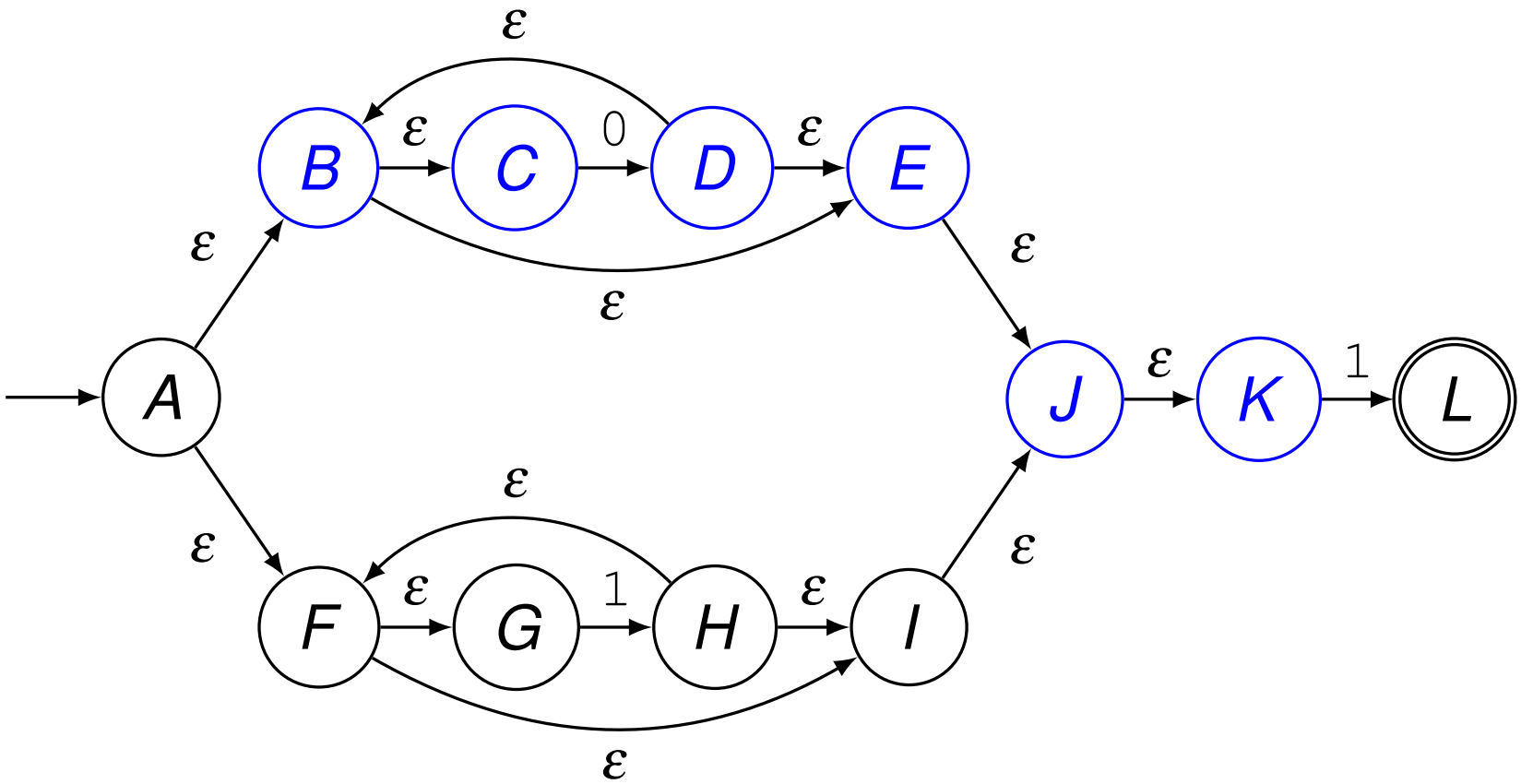


Exemplu



Starea inițială: $S_0 = \{A, B, C, E, J, K, F, G, I\}$

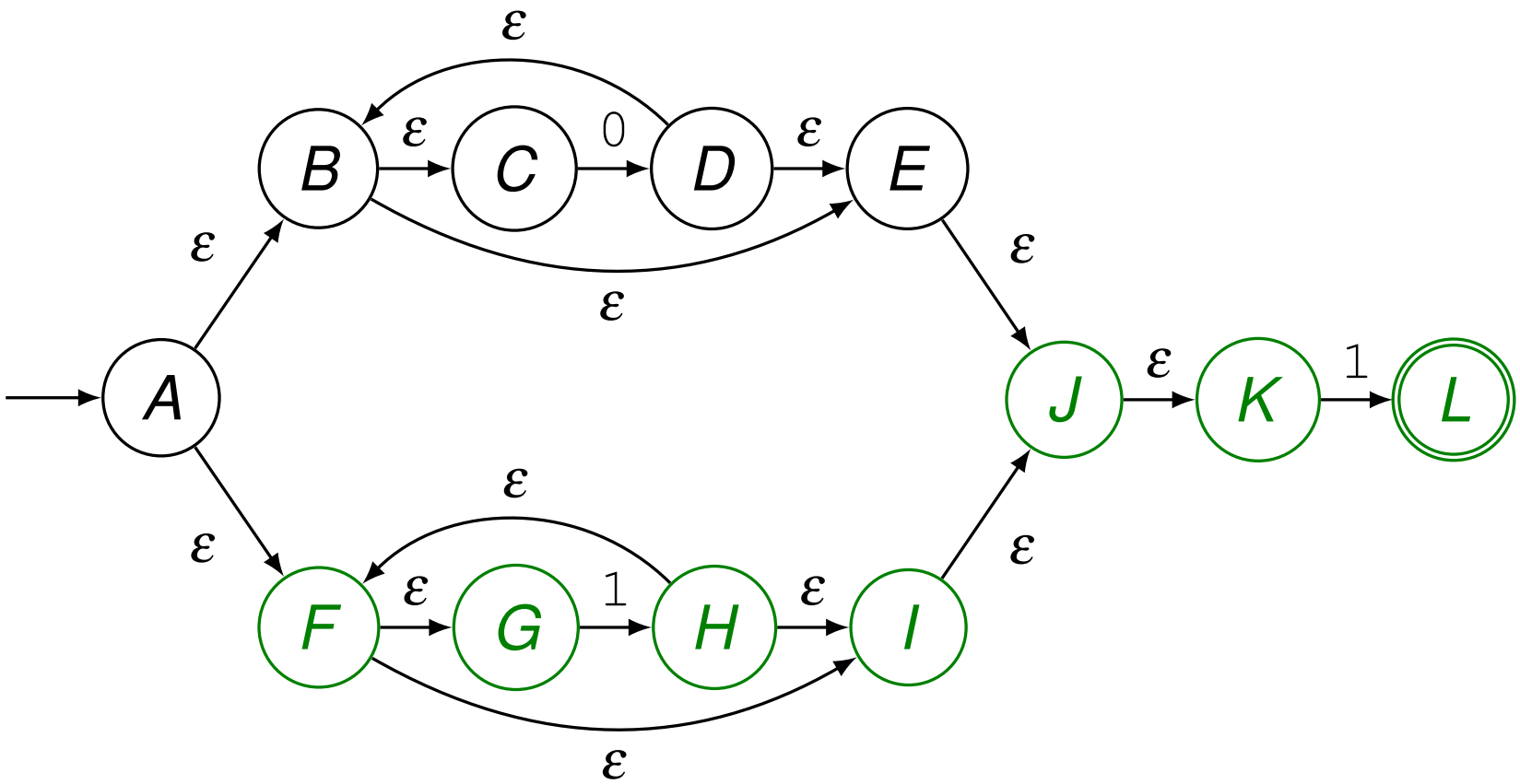
Exemplu



$S_0 \rightarrow^0 S_1 = \{D, E, J, K, B, C\}$

$S_0 \rightarrow^1 S_2 = \{H, I, J, K, F, G, L\}$ (stare finală)

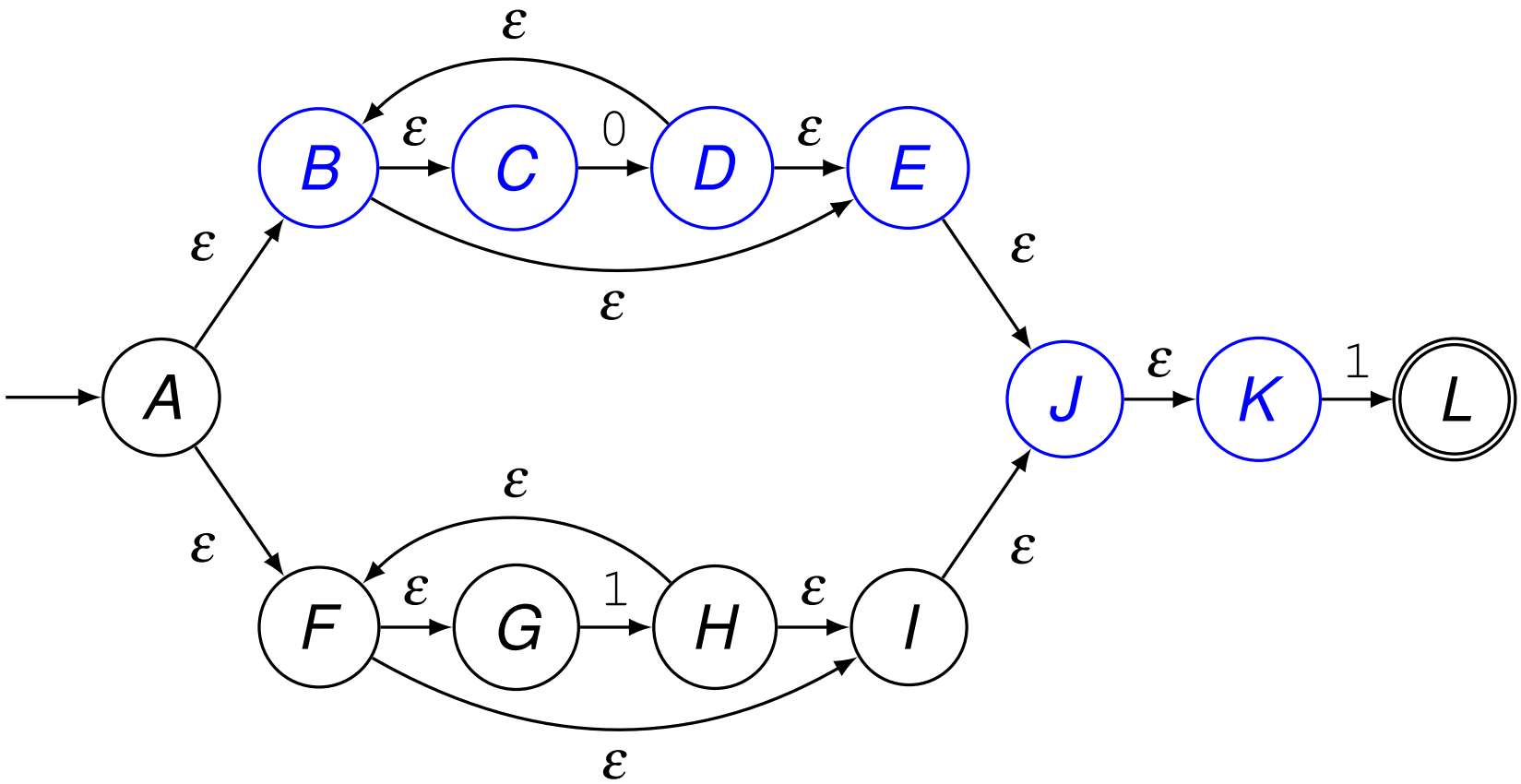
Exemplu



$S_0 \rightarrow^0 S_1 = \{D, E, J, K, B, C\}$

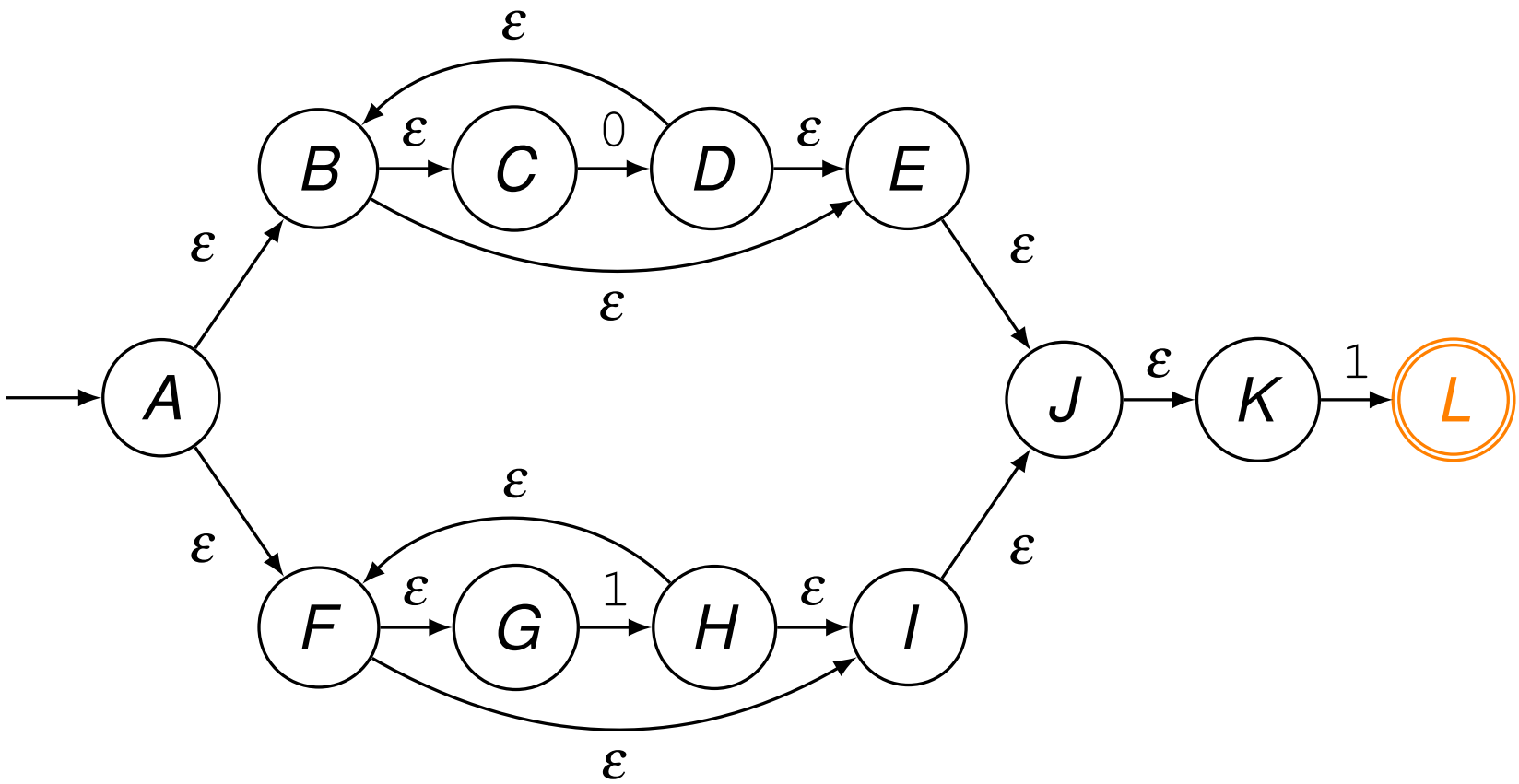
$S_0 \rightarrow^1 S_2 = \{H, I, J, K, F, G, L\}$ (stare finală)

Exemplu



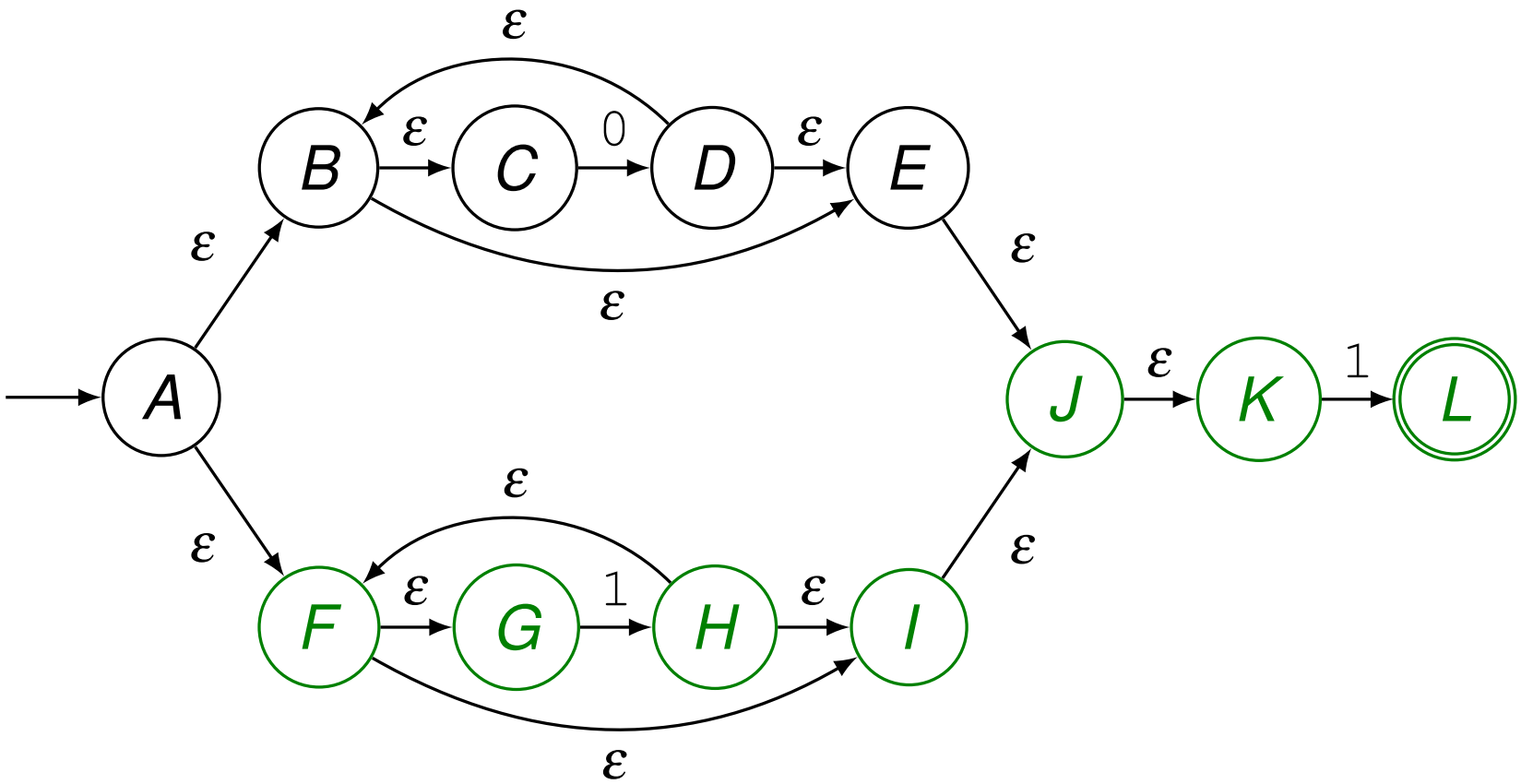
$S_1 \rightarrow^0 S_1 = \{D, E, J, K, B, C\}$
$S_1 \rightarrow^1 S_3 = \{L\}$ (stare finală)

Exemplu



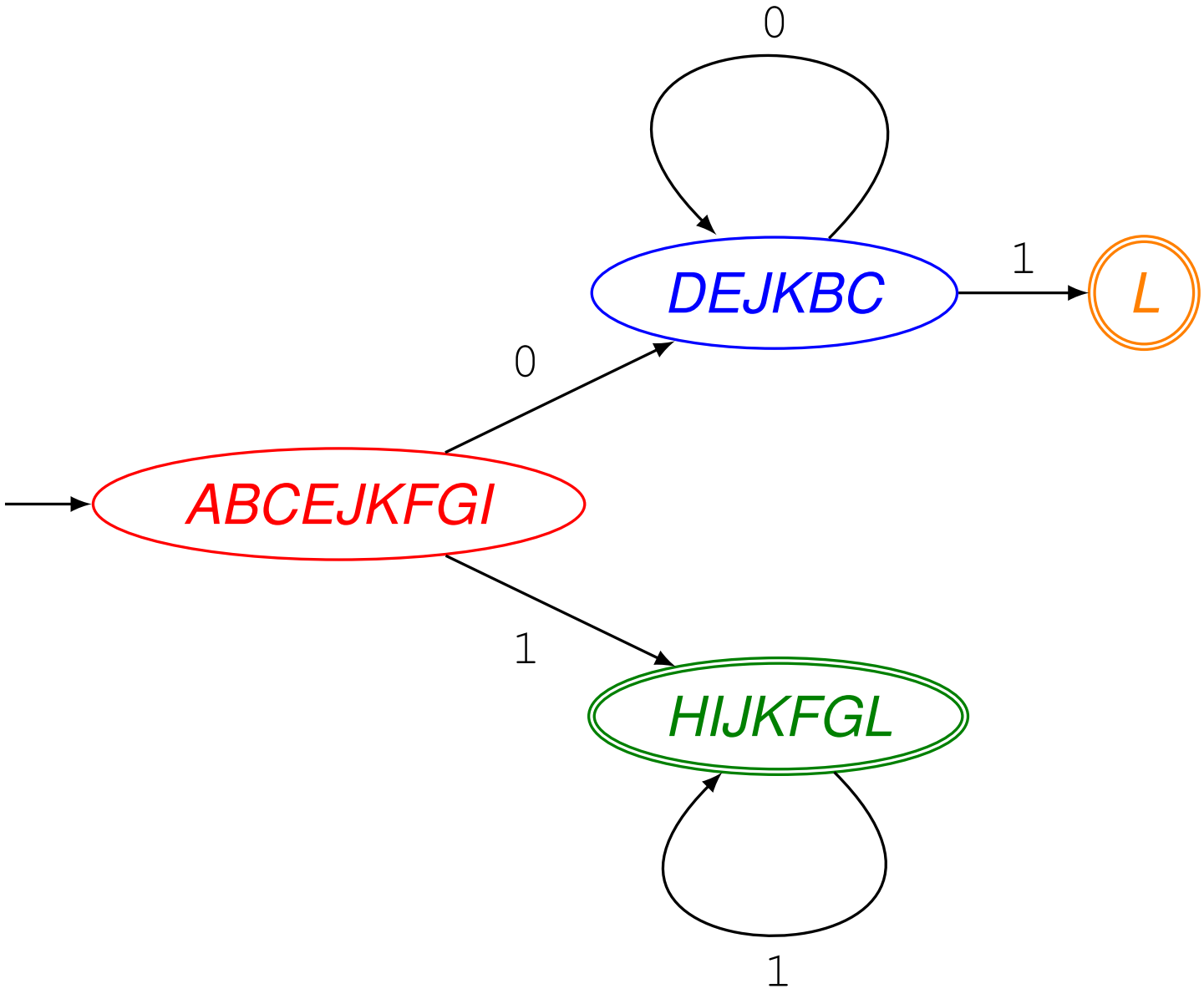
$S_1 \rightarrow^0 S_1 = \{D, E, J, K, B, C\}$
$S_1 \rightarrow^1 S_3 = \{L\}$ (stare finală)

Exemplu



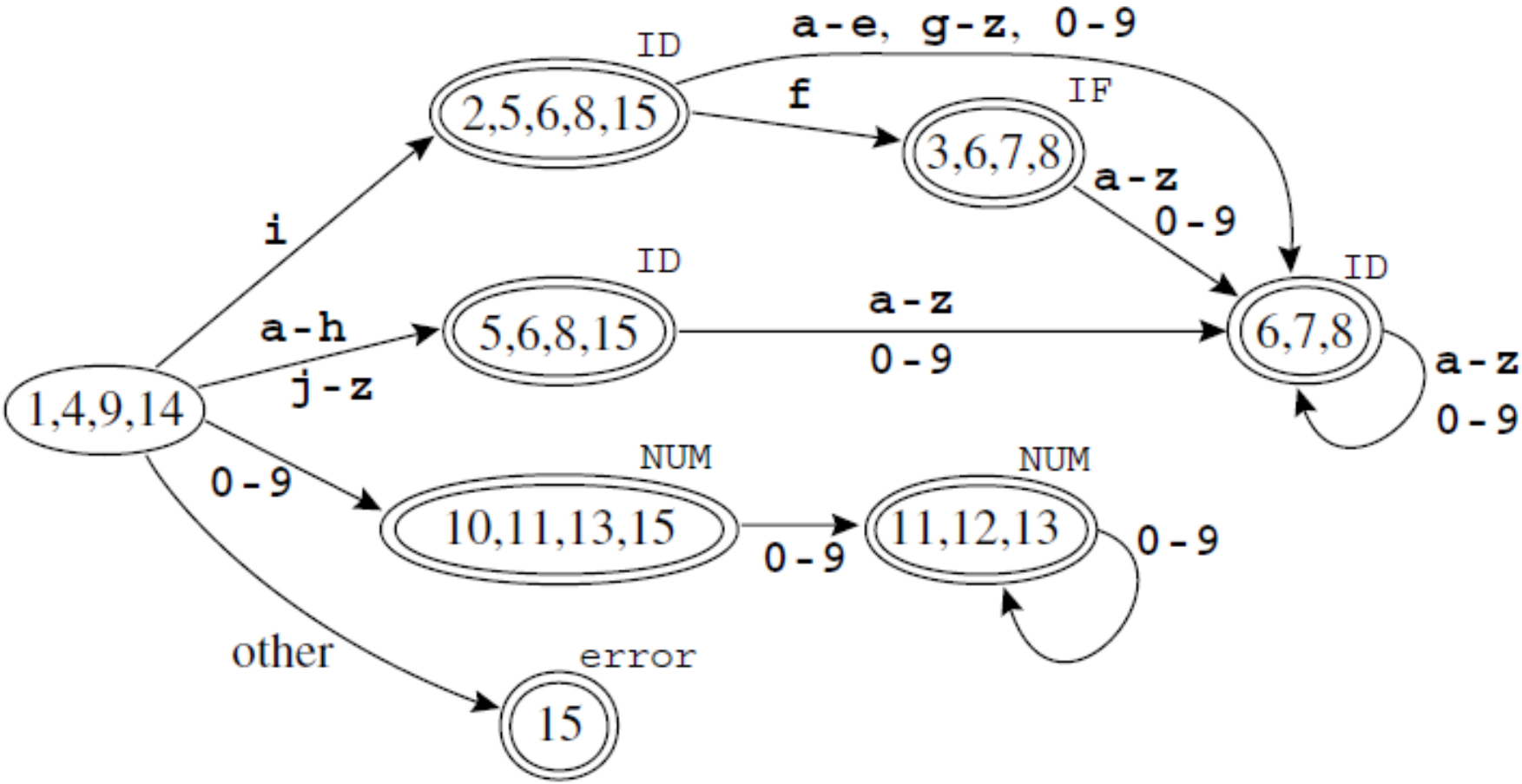
$S_2 \rightarrow^1 S_2 = \{H, I, J, K, F, G, L\}$ (stare finală)

Exemplu



v. slide-ul 55

Exemplu de automat corespunzător unei specificații lexicale



Sursă: Appel, A., Palsberg, J. (2002). *Modern Compiler Implementation in Java (2nd ed.)*. Cambridge University Press.

Partea III

Analiza sintactică

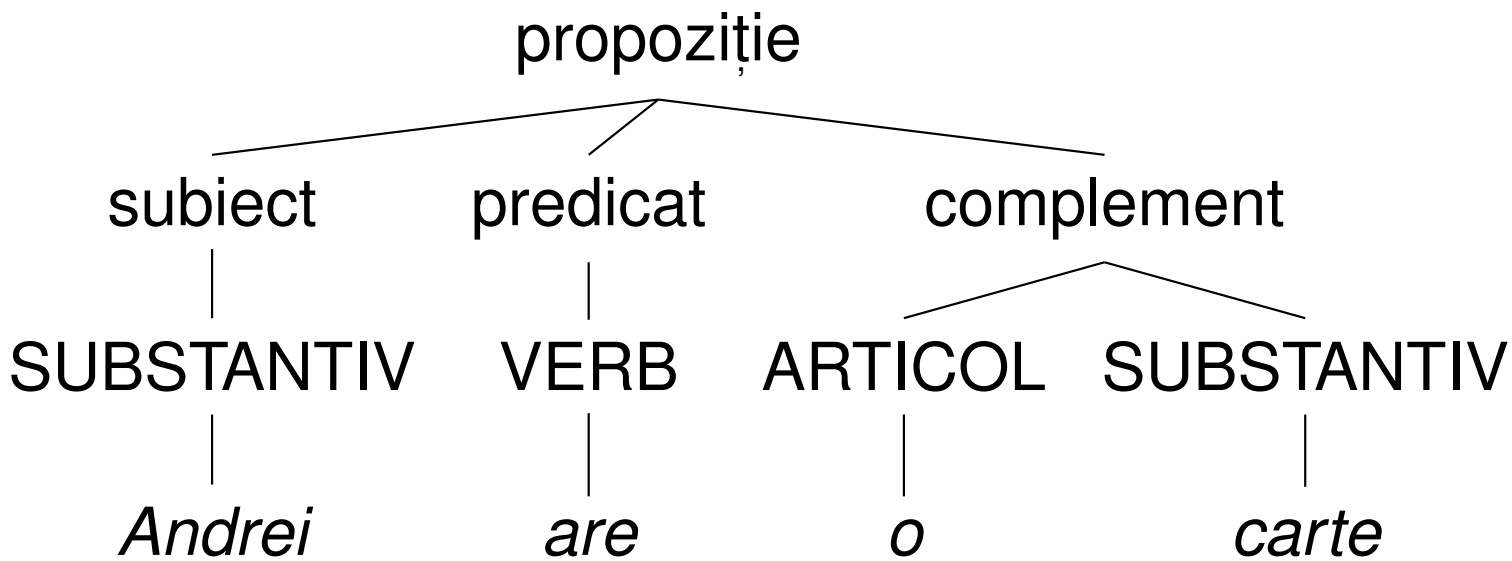
Cuprins

Cuprins

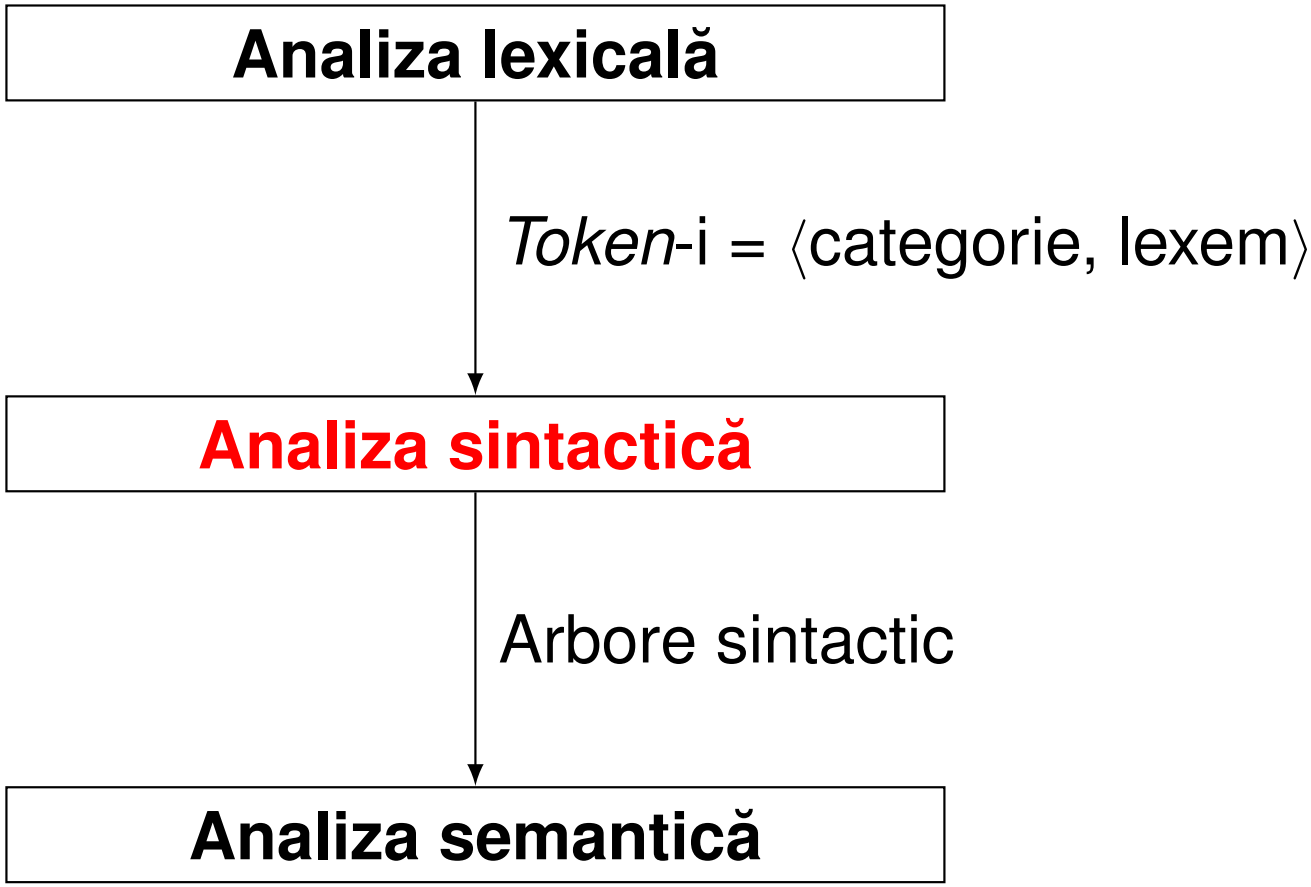
Analiza sintactică

- Determinarea structurii ierarhice a propozițiilor, în baza cuvintelor constitutive și a relațiilor dintre acestea

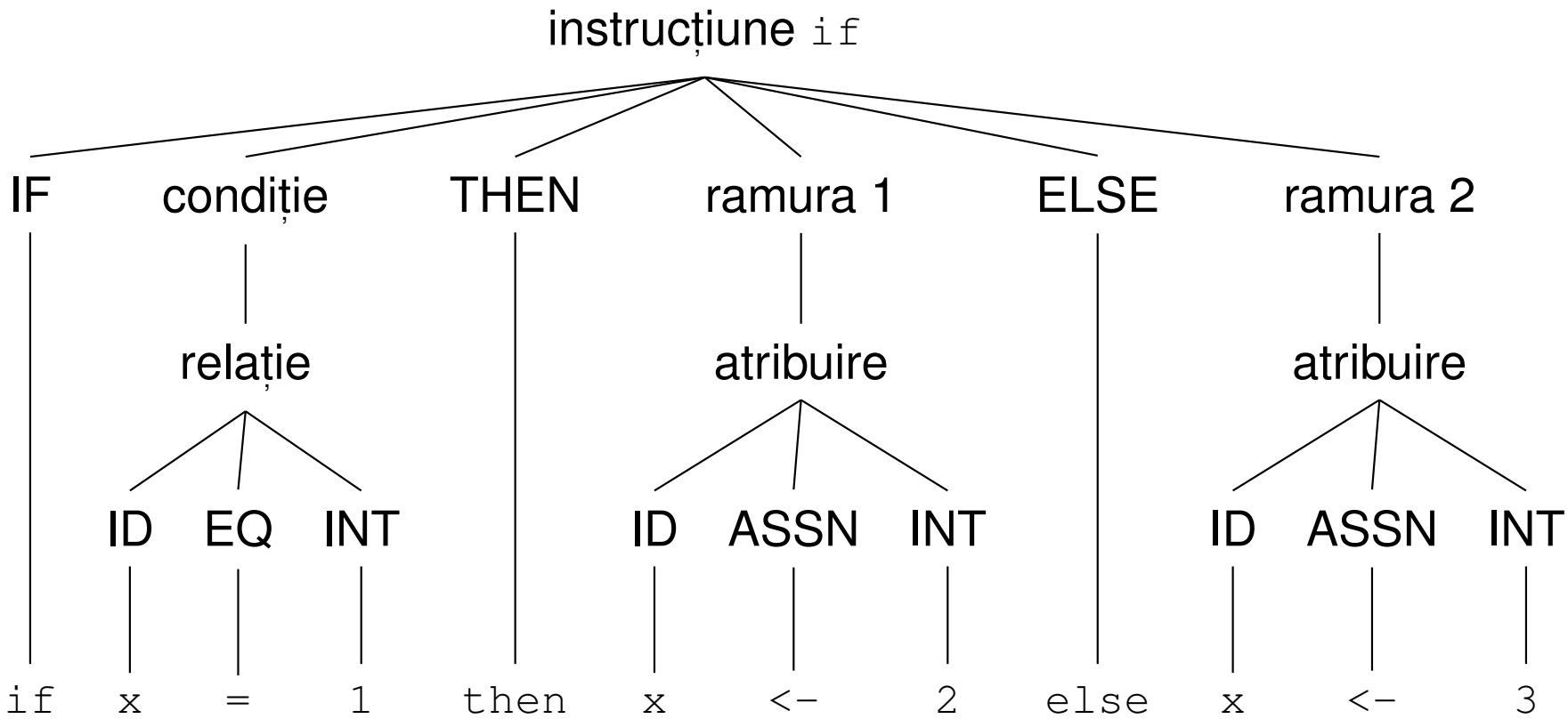
- Exemplu în limba română:



Context


Analiza lexicală
Token-i = ⟨categorie, lexem⟩
Analiza sintactică
Arbore sintactic
Analiza semantică

Exemplu


instrucțiune if
IF
condiție
THEN
ramura 1
ELSE
ramura 2
relație
atribuire
atribuire
ID
EQ
INT
ID
ASSN
INT
ID
ASSN
INT
if
x
=
1
then
x
<-
2
else
x
<-
3

Cuprins

Motivație

- Cum exprimăm natura inerent recursivă
  a construcțiilor de limbaj?
- Exemplu de expresie:
  if E then E else E fi este un *E*
- Expresiile regulate, insuficient de puternice
  pentru a surprinde această structură recursivă
- Apel la **gramaticile independente de context** (GIC)!

Structura unei GIC

- Mulțimea de **terminali**, T (categoriile *token*-ilor din limbaj)
- Mulțimea de **neterminali**, N
- Simbolul de **start**, $S \in N$
- Mulțimea **producțiilor**, având forma $X \to Y_1 \dots Y_n$, cu $X \in N$ și $Y_i \in T \cup N \cup \{\varepsilon\}$

Exemplu

Expresii în COOL:

$$
\begin{aligned}
E \rightarrow\ & \mathit{if}\ E\ \mathit{then}\ E\ \mathit{else}\ E\ \mathit{fi} \\
|\ & \mathit{id} \leftarrow E \\
& \cdots \\
|\ & E + E \\
|\ & E * E \\
|\ & (E) \\
|\ & \mathit{id}
\end{aligned}
$$

Limbajul descris de o GIC

- Aplicarea unei producții $X_i \to Y_1 \dots Y_m$:

$$X_1 X_2 \dots X_{i-1} X_i X_{i+1} \dots X_n \to X_1 X_2 \dots X_{i-1} Y_1 \dots Y_m X_{i+1} \dots X_n$$

- Secvența de aplicare a zero sau mai multe producții:

$$X_1 \dots X_n \to^* Y_1 \dots Y_p$$

- Limbajul descris de gramatica G, cu simbolul de start S și mulțimea de terminali T:

$$L(G) = \{t_1 \dots t_n \mid S \to^* t_1 \dots t_n, \text{ și } t_i \in T\}$$

Cuprins

Derivare

- Secvență de aplicări ale producțiilor, pornind cu simbolul de start

- Reprezentabilă sub forma unui arbore:
  - Simbolul de **start**, în rădăcină
  - Pentru fiecare **producție** $X \to Y_1 \dots Y_n$, nodurile $Y_1, \dots, Y_n$, adăugate drept copii ai nodului X

Exemplu

- Gramatica:

$$\begin{aligned} E \rightarrow\ & E + E \\ |\ & E * E \\ |\ & (E) \\ |\ & id \end{aligned}$$

- Șirul:

$$id + id * id$$

Derivare **la stânga** (*leftmost derivation*)

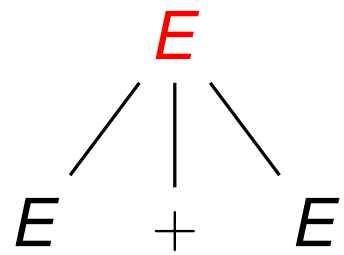
E

E

Expandarea celui mai din stânga neterminal

Derivare **la stânga** (*leftmost derivation*)

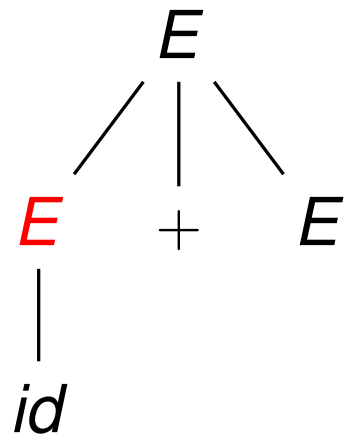
$$E$$
$$\rightarrow E + E$$



Expandarea celui mai din stânga neterminal

Derivare **la stânga** (*leftmost derivation*)

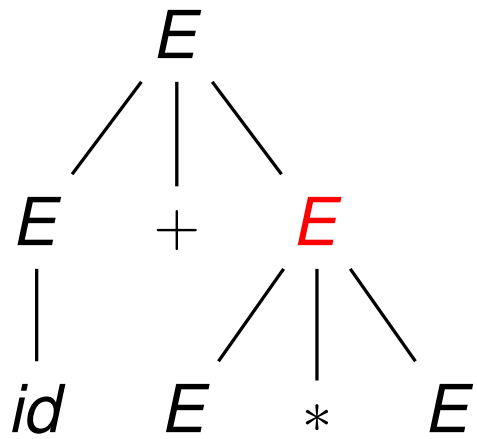
$$
\begin{aligned}
& E \\
\rightarrow\ & E + E \\
\rightarrow\ & id + E
\end{aligned}
$$



Expandarea celui mai din stânga neterminal

Derivare **la stânga** (*leftmost derivation*)

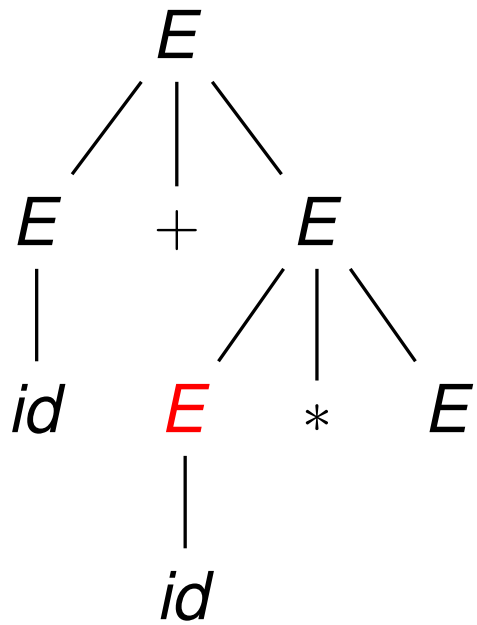
$$
\begin{aligned}
& E \\
\rightarrow\ & E + E \\
\rightarrow\ & id + E \\
\rightarrow\ & id + E * E
\end{aligned}
$$



Expandarea celui mai din stânga neterminal

Derivare **la stânga** (*leftmost derivation*)

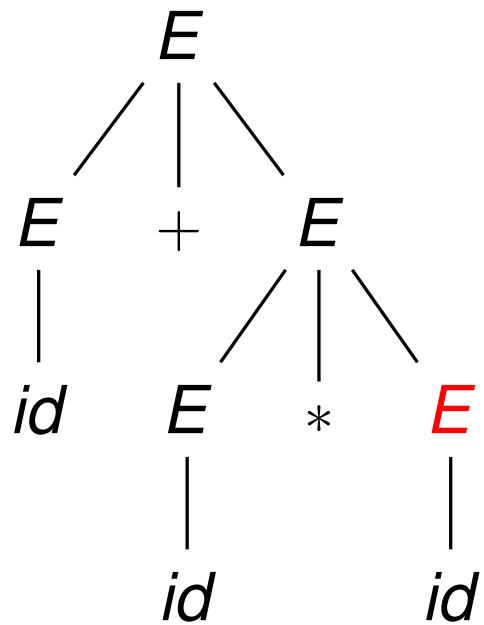
$$
\begin{aligned}
& E \\
\rightarrow\ & E + E \\
\rightarrow\ & id + E \\
\rightarrow\ & id + E * E \\
\rightarrow\ & id + id * E
\end{aligned}
$$



Expandarea celui mai din stânga neterminal

Derivare **la stânga** (*leftmost derivation*)

$$
\begin{aligned}
& E \\
\rightarrow\ & E + E \\
\rightarrow\ & id + E \\
\rightarrow\ & id + E * E \\
\rightarrow\ & id + id * E \\
\rightarrow\ & id + id * id
\end{aligned}
$$



Expandarea celui mai din stânga neterminal

Derivare **la dreapta** (*rightmost derivation*)

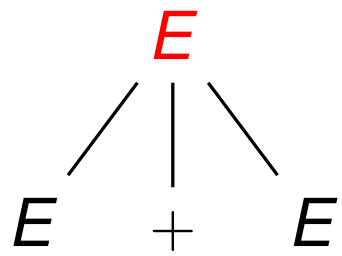
E

E

Expandarea celui mai din dreapta neterminal

Derivare **la dreapta** (*rightmost derivation*)

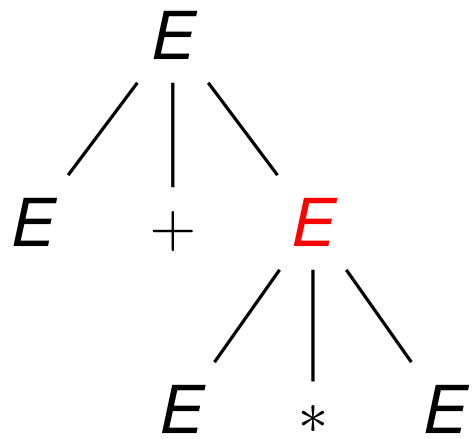
$$E$$
$$\rightarrow E + E$$



Expandarea celui mai din dreapta neterminal

Derivare **la dreapta** (*rightmost derivation*)

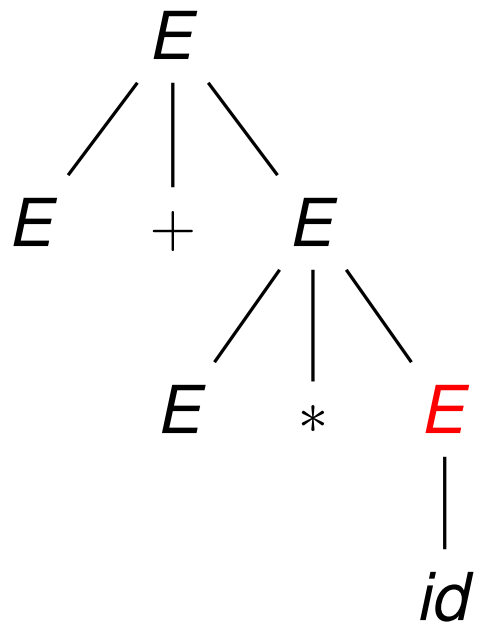
$$
\begin{aligned}
& E \\
\rightarrow\ & E + E \\
\rightarrow\ & E + E * E
\end{aligned}
$$



Expandarea celui mai din dreapta neterminal

Derivare **la dreapta** (*rightmost derivation*)

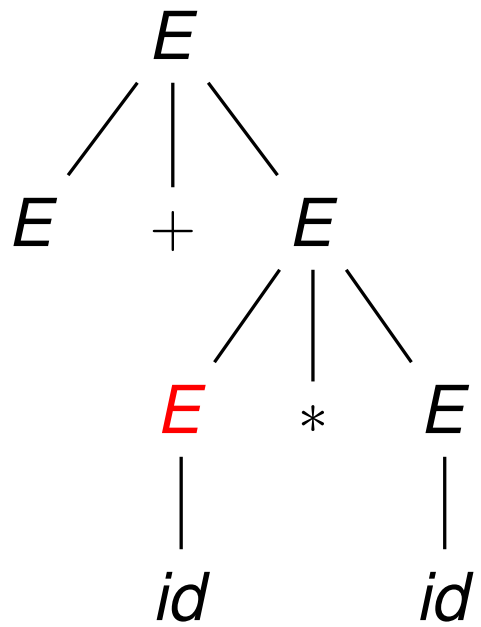
$$
\begin{aligned}
& E \\
\rightarrow\ & E + E \\
\rightarrow\ & E + E * E \\
\rightarrow\ & E + E * id
\end{aligned}
$$



Expandarea celui mai din dreapta neterminal

Derivare **la dreapta** (*rightmost derivation*)

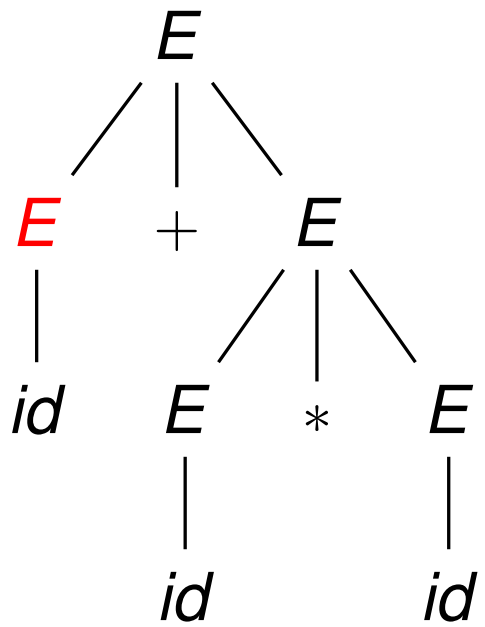
$$
\begin{aligned}
& E \\
\rightarrow\ & E + E \\
\rightarrow\ & E + E * E \\
\rightarrow\ & E + E * id \\
\rightarrow\ & E + id * id
\end{aligned}
$$



Expandarea celui mai din dreapta neterminal

Derivare **la dreapta** (*rightmost derivation*)

$$
\begin{aligned}
& E \\
\rightarrow\ & E + E \\
\rightarrow\ & E + E * E \\
\rightarrow\ & E + E * id \\
\rightarrow\ & E + id * id \\
\rightarrow\ & id + id * id
\end{aligned}
$$



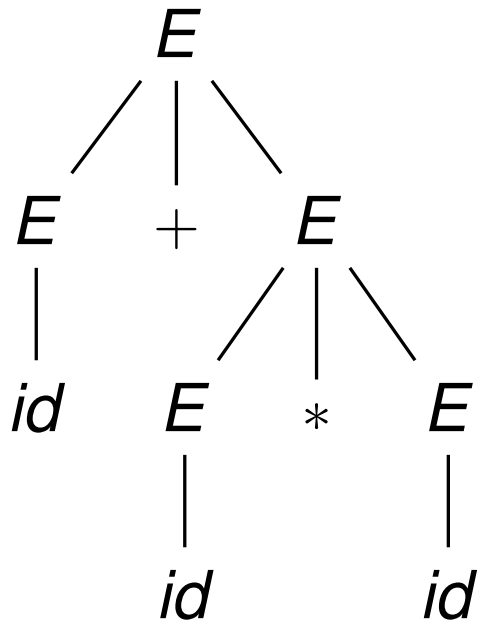
Expandarea celui mai din dreapta neterminal

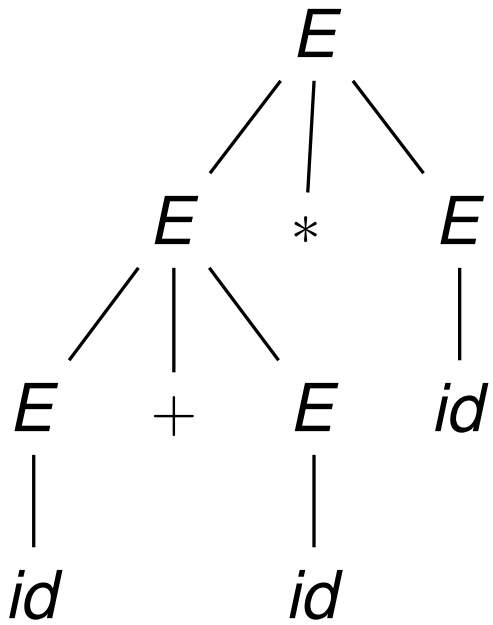
Arbori de derivare

- **Arbore de derivare** (*parse tree*): **neterminali**, ca noduri interne, și **terminali**, ca frunze
- Corespondență mai mulți la unu între derivări și arbori de derivare
- Șirul original, obținut printr-o parcurgere în inordine
- Prezența informației de precedență și asociere a operațiilor, absentă din șirul liniar

Cuprins

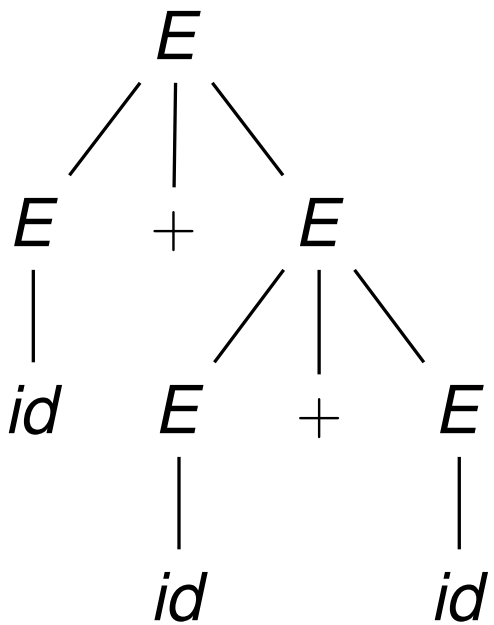
Expresii aritmetice I

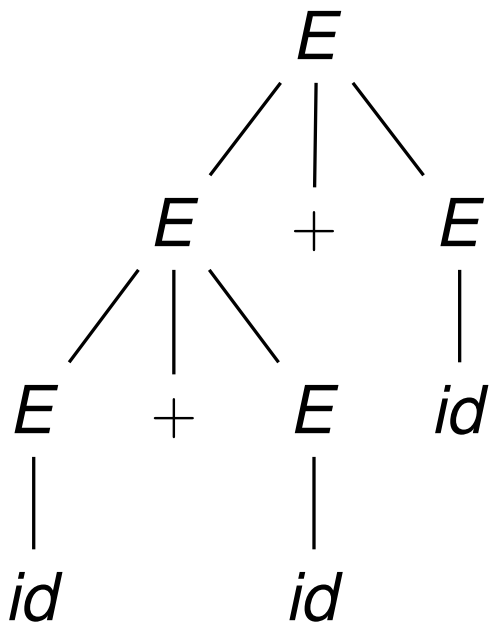




Existența a doi arbori de derivare diferiți
pentru șirul $id + id * id$ (problemă de **precedență**)

Expresii aritmetice II





Existența a doi arbori de derivare diferiți
pentru șirul $id + id + id$ (problemă de **asociere**)

Ambiguitate

- Existența unui șir pentru care se pot construi cel puțin doi arbori de derivare diferiți
- Indiciu al subspecificării gramaticii
- Gestiune prin rescrierea gramaticii sau prin adăugarea de precizări suplimentare, de precedență și/sau de asociere

Rescrierea gramaticii

$E \to T + E \mid T$	Expresie
$T \to F * T \mid F$	Termen
$F \to (E) \mid id$	Factor

Impunerea precedenței lui $*$ față de $+$
prin modul de scriere a gramaticii

Specificarea precendenței și a asocierii

ANTLR:

- **Precedența**, specificată prin ordinea alternativelor
- **Asocierea**, explicitată cu `assoc`; implicit, la stânga

$$
\begin{aligned}
E \rightarrow\ & E \ \wedge\texttt{<assoc=right>}\ E \\
\mid\ & E * E \\
\mid\ & E + E \\
\mid\ & (E) \\
\mid\ & id
\end{aligned}
$$

Împerecherea *if-then-else* (*dangling else*) I

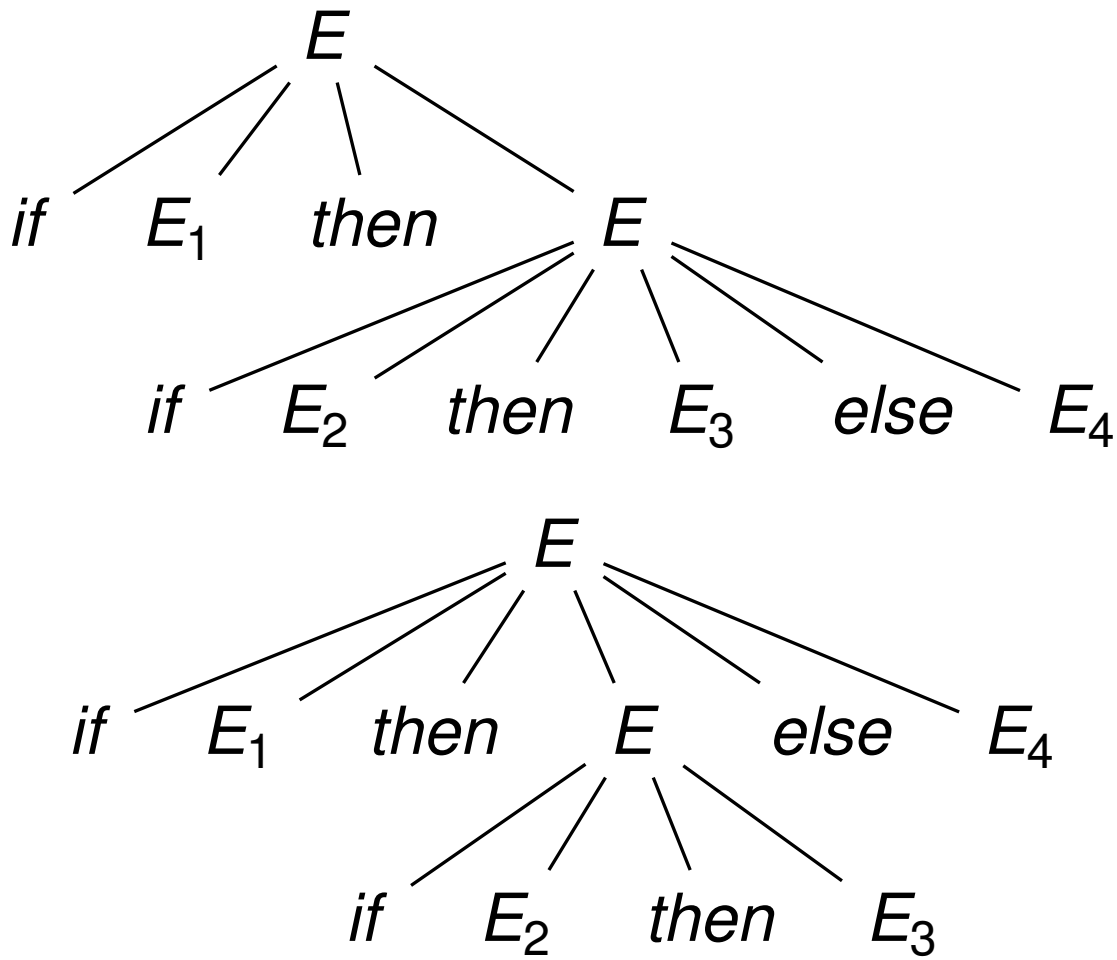
- Gramatica:

$$\begin{aligned} E \rightarrow\ & \mathit{if}\ E\ \mathit{then}\ E \\ \mid\ & \mathit{if}\ E\ \mathit{then}\ E\ \mathit{else}\ E \\ \mid\ & E' \end{aligned}$$

- Șirul:

$$\mathit{if}\ E_1\ \mathit{then}\ \mathit{if}\ E_2\ \mathit{then}\ E_3\ \mathit{else}\ E_4$$

Împerecherea *if-then-else* (*dangling else*) II



Existența a doi arbori de derivare diferiți — de preferat, *else*, lângă cel mai apropiat *then* (primul arbore)

Împerecherea *if-then-else* (*dangling else*) III

Rescriere: între *then* și *else*, permise doar *if*-uri complete

$$E \rightarrow Compl \mid Incompl$$

$$Compl \rightarrow \mathit{if}\ E\ \mathit{then}\ Compl\ \mathit{else}\ Compl \mid E'$$

$$Incompl \rightarrow \mathit{if}\ E\ \mathit{then}\ E \mid \mathit{if}\ E\ \mathit{then}\ Compl\ \mathit{else}\ Incompl$$

Ambiguitate în C++

- În C++, secvența `T(5)`, interpretabilă fie drept **aplicație** de funcție, fie drept ***cast***
- Ambiguitate imposibil de rezolvat la nivel sintactic!
- Necesitatea informației semantice, despre natura lui `T` (funcție sau tip)

Cuprins

Obiective

- **Raportarea** unor mesaje sugestive de eroare, de obicei incluzând informații despre linia și coloana din fișer unde a apărut eroarea

- **Recuperarea** din eroare, în vederea continuării analizei

Strategii de gestiune a erorilor

- Adăugarea sau eliminarea unui *token*
- Modul de „panică”
- Producții dedicate erorilor

Adăugarea sau eliminarea unui *token*

- **Eliminarea** unui *token* (*single-token deletion*):

$$\textit{if}\ a\ \textit{then}\ 1\ \textit{else}\ 2\ b\ \textit{fi}$$
$$\rightarrow \textit{if}\ a\ \textit{then}\ 1\ \textit{else}\ 2\ \textit{fi}$$

- **Adăugarea** unui *token* (*single-token insertion*):

$$\textit{if}\ a\ \textit{then}\ 1\ 2\ \textit{fi}$$
$$\rightarrow \textit{if}\ a\ \textit{then}\ 1\ \textit{else}\ 2\ \textit{fi}$$

Modul de „panică”

- Consumarea *token*-ilor până la identificarea unora de resincronizare
- ***Token* de resincronizare** = *token* care îi poate urma referinței unei reguli sintactice, pe lanțul actual de invocare (pot fi mai mulți)
- Exemplu:

$$E \rightarrow \mathit{if}\ E\ \mathit{then}\ E\ \mathit{else}\ E\ \mathit{fi}$$

- Determinare dinamică a *token*-ilor de resincronizare — referințe diferite pot avea *token*-i de resincronizare diferiți

Producții dedicate erorilor

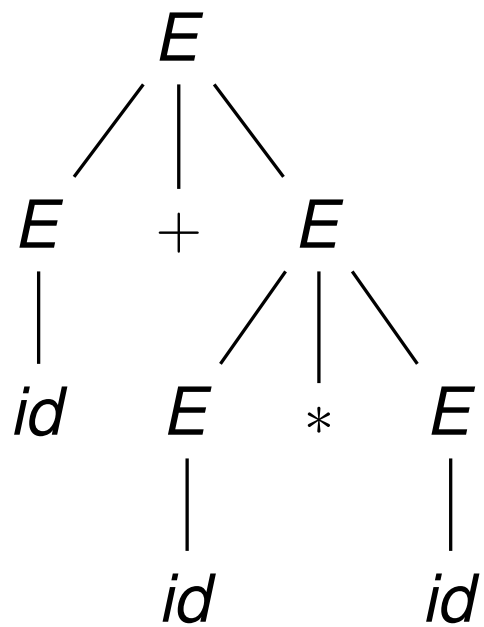
- Adăugarea de producții care anticipează erori
- Exemplu:

$$E \rightarrow E * E$$
$$\mid \quad EE$$
$$\ldots$$

Cuprins

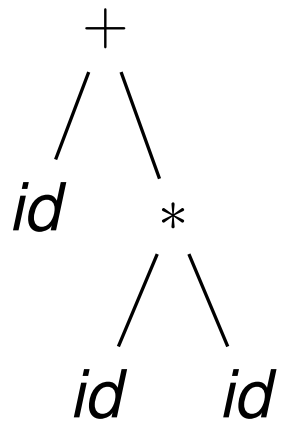
Sintaxă concretă

- Sintaxa propriu-zisă utilizată în scrierea programelor
- Surprinsă de **arborii de derivare**, sensibili atât la notație, cât și la structura gramaticii:



Sintaxă abstractă

- Structura esențială a expresiilor
- Exemplu: Conceptul de înmulțire, independent de notația aleasă pentru operatorul de înmulțire, ($*$ sau $\times$), sau de forma particulară a gramaticii
- Exemplu: Parantezele, nenecesare odată ce forma ierarhică a expresiei a fost obținută
- Surprinsă de **arborii de sintaxă abstractă** (*abstract syntax trees*)



Partea IV

Analiza sintactică *top-down*

Cuprins

Cuprins

Principii

- Arbore de derivare construit de sus în jos și de la stânga la dreapta
- Pornire de la simbolul de start al gramaticii
- La fiecare pas, alegerea unei producții aferente celui mai din stânga neterminal (cum?), cu expandarea nodului aferent

Strategii *top-down*

- ***Recursive descent***: încercarea producțiilor în ordine, cu *backtracking* în caz de eșec

- **Analiză predictivă**: alegerea producției adecvate în funcție de *lookahead*

Cuprins

Pașii *recursive descent* (RD)

- Încercarea producțiilor în ordine, cu *backtracking* în caz de eșec
- La generarea unui **neterminal**, imposibilitatea de a cunoaște încă utilitatea căii curente
- La generarea unui **terminal**, verificarea potrivirii între acesta și următorul *token* din șir
- Eșec în caz de:
  - Nepotrivire cu *token-ul* curent SAU
  - Încheiere a derivării/șirului, unul înaintea celuilalt

Exemplu

E

- Gramatica:

$$E \to T \mid T + E$$
$$T \to id \mid id * T \mid (E)$$

- Șirul:

$$id + id * id$$

Exemplu

- Gramatica:

$$E \to T \mid T + E$$
$$T \to id \mid id * T \mid (E)$$

- Șirul:

$$id + id * id$$

Exemplu

- Gramatica:

$$E \to T \mid T + E$$
$$T \to id \mid id * T \mid (E)$$

- Șirul:

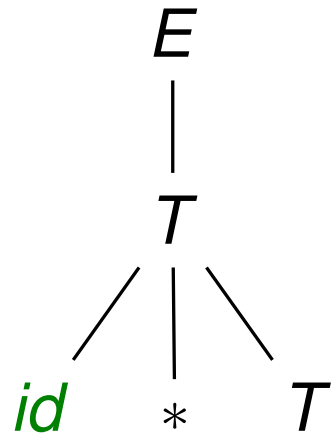
$$id + id * id$$

Eroare: șir neterminat

Exemplu

- Gramatica:

$$E \to T \mid T + E$$
$$T \to id \mid id * T \mid (E)$$



- Șirul:

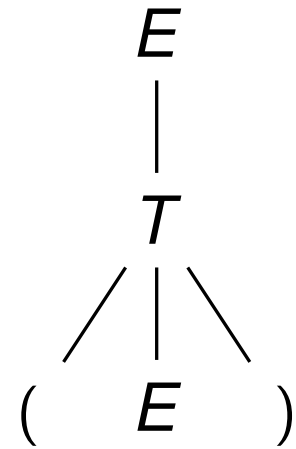
$$id + id * id$$

Eroare: nepotrivire
între $*$ și $+$

Exemplu

- Gramatica:

$$E \to T \mid T + E$$
$$T \to id \mid id * T \mid (E)$$

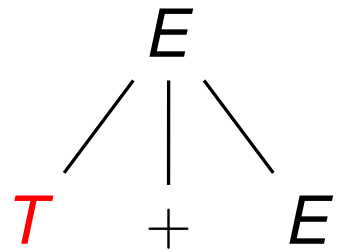


- Șirul:

$$id + id * id$$

Eroare: nepotrivire
între (și *id*

Exemplu



- Gramatica:

$$E \to T \mid T + E$$
$$T \to id \mid id * T \mid (E)$$

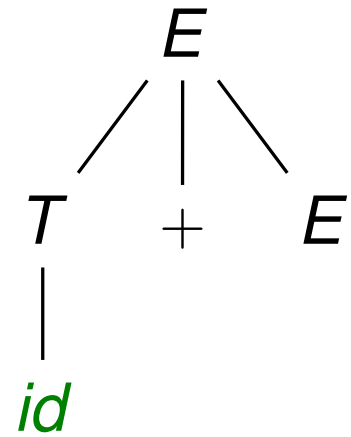
- Șirul:

$$id + id * id$$

Exemplu

- Gramatica:

$$E \to T \mid T + E$$
$$T \to id \mid id * T \mid (E)$$



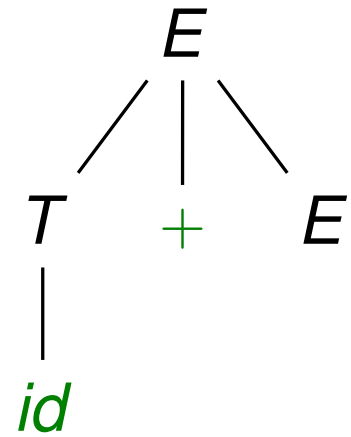
- Șirul:

$$id + id * id$$

Exemplu

- Gramatica:

$$E \to T \mid T + E$$
$$T \to id \mid id * T \mid (E)$$



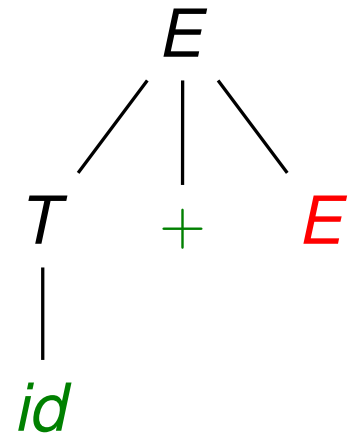
- Șirul:

$$id + id * id$$

Exemplu

- Gramatica:

$$E \to T \mid T + E$$
$$T \to id \mid id * T \mid (E)$$



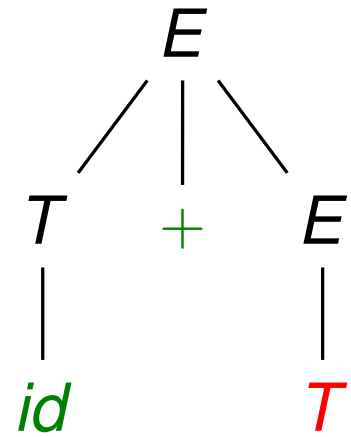
- Șirul:

$$id + id * id$$

Exemplu

- Gramatica:

$$E \to T \mid T + E$$
$$T \to id \mid id * T \mid (E)$$



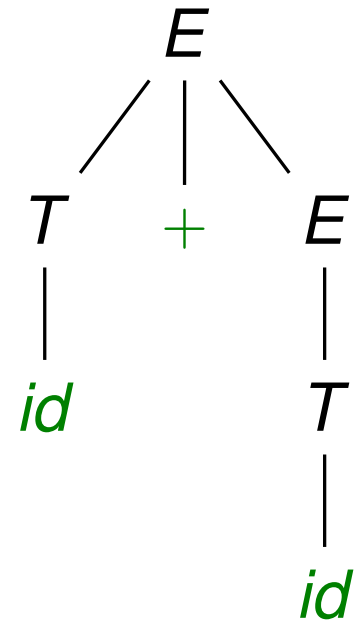
- Șirul:

$$id + id * id$$

Exemplu

- Gramatica:

$$E \to T \mid T + E$$
$$T \to id \mid id * T \mid (E)$$



- Șirul:

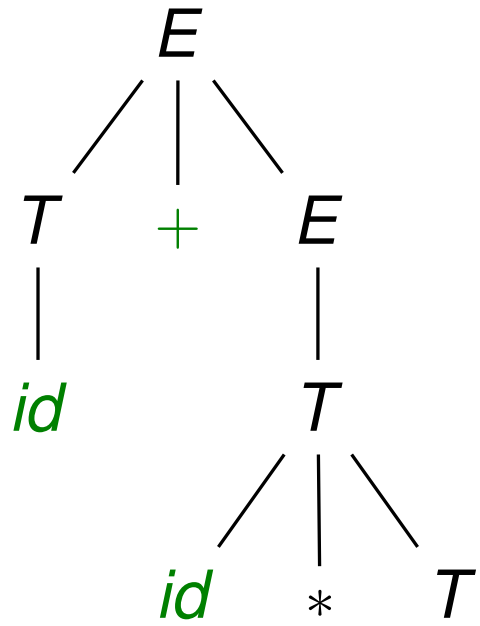
$$id + id * id$$

Eroare: șir neterminat

Exemplu

- Gramatica:

$$E \to T \mid T + E$$
$$T \to id \mid id * T \mid (E)$$



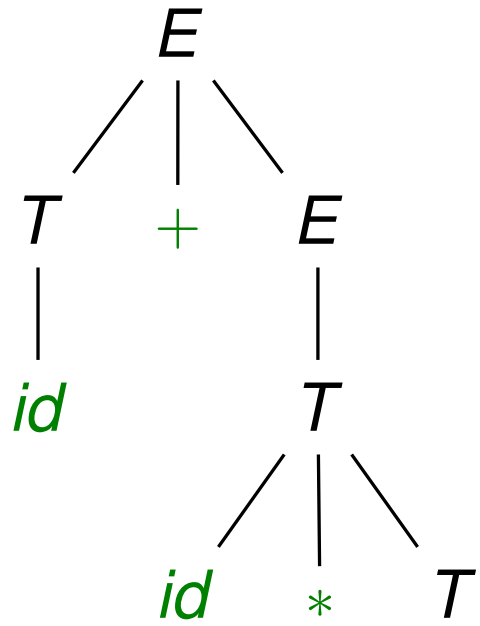
- Șirul:

$$id + id * id$$

Exemplu

- Gramatica:

$$E \to T \mid T + E$$
$$T \to id \mid id * T \mid (E)$$



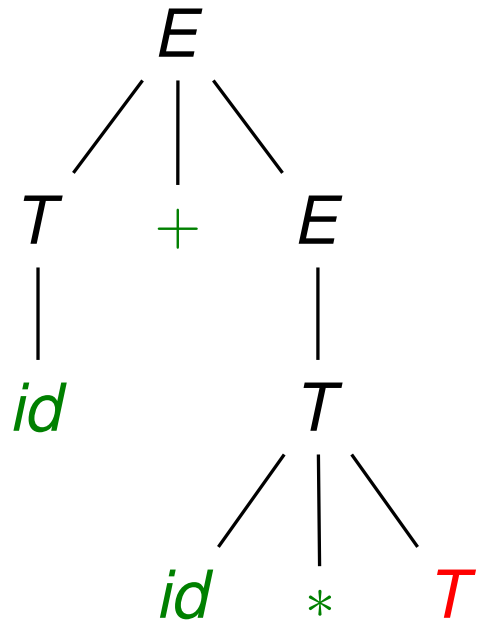
- Șirul:

$$id + id * id$$

Exemplu

- Gramatica:

$$E \to T \mid T + E$$
$$T \to id \mid id * T \mid (E)$$



- Șirul:

$$id + id * id$$

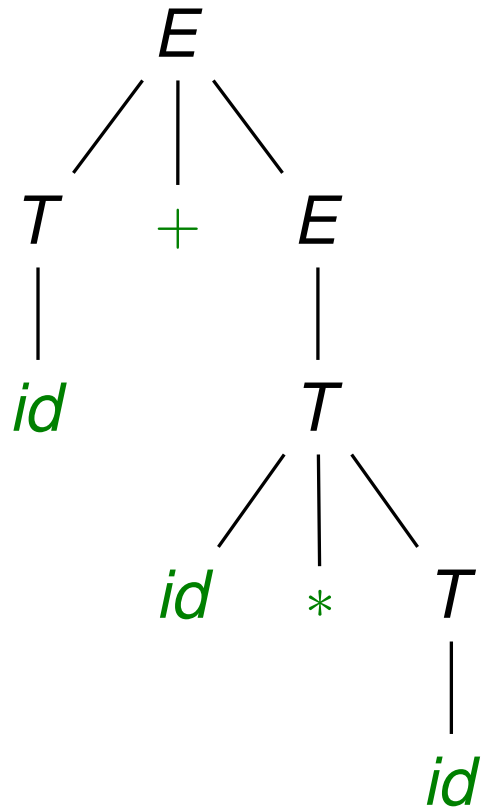
Exemplu

- Gramatica:

$$E \to T \mid T + E$$
$$T \to id \mid id * T \mid (E)$$

- Șirul:

$$id + id * id$$



Succes!

Implementare I

- `consume()`: întoarce *token*-ul curent și avansează poziția în șirul de intrare
- `peek()`: similar cu `consume()`, dar fără a avansa poziția
- `mark()`: reține poziția curentă din șirul de intrare într-o stivă
- `reset()`: revine la poziția din vârful stivei, pe care o elimină

Implementare II

- Potrivirea unui ***token***:

```
bool match(token) { consume() == token }
```

- Încercarea **producției** *i* aferente neterminalului *X*:

```
bool Xi() { ... }
```

- Încercarea **tuturor producțiilor** aferente neterminalului *X*:

```
1 bool X() {
2   (mark(), X1())
3     or (reset(), mark(), X2())
4     or ... or (reset(), Xn())
5 }
```

Implementare III

- **Inițializare**: situarea în dreptul primului *token* din șir
- **Rulare**: apelul funcției aferente simbolului de start

Exemplu

$$E \to T \mid T + E$$
$$T \to id \mid id * T \mid (E)$$

```
1  bool E1() { T() }
2  bool E2() { T() and match('+') and E() }
3
4  bool T1() { match(id) }
5  bool T2() { match(id) and match('*') and T() }
6  bool T3() { match('(') and E() and match(')') }
```

Limitări ale strategiei RD descrise

- Absența backtracking-ului în situațiile în care aplicarea producțiilor se realizează cu succes, fără a parcurge întregul șir (v. exemplul anterior)

- Buclă infinită în cazul **recursivității la stânga**

$$E \to E + T \mid \ldots$$

```
1 bool E1() { E() and match('+') and T() }
2 bool E()  { ... E1() ... }
```

Cuprins

Definiție

- Gramatică **recursivă la stânga** (*left-recursive*), dacă există un neterminal X și o secvență de aplicări ale producțiilor, astfel încât

$$X \rightarrow^* X\alpha$$

- Varianta **directă**:

$$X \rightarrow X\alpha \mid \beta$$

- Varianta **indirectă**:

$$X \rightarrow Y\alpha \mid \beta$$
$$Y \rightarrow X\delta$$

Eliminarea recursivității la stânga directe I

- Gramatica **recursivă la stânga**:

$$X \to X\alpha \mid \beta$$

- Secvența de aplicare a producțiilor, cu generare de la dreapta la stânga:

$$X \to X\alpha \to X\alpha\alpha \to \ldots \to \beta\alpha\ldots\alpha$$

- Rescrierea **nerecursivă la stânga**:

$$X \to \beta X'$$
$$X' \to \alpha X' \mid \varepsilon$$

Eliminarea recursivității la stânga directe II

- Gramatica **recursivă la stânga**:

$$X \to X\alpha_1 \mid \ldots \mid X\alpha_n \mid \beta_1 \mid \ldots \mid \beta_m$$

- Secvența de aplicare a producțiilor, cu generare de la dreapta la stânga:

$$X \to X\alpha_{i_1} \to X\alpha_{i_2}\alpha_{i_1} \to \ldots \to \beta_j\alpha_{i_p}\ldots\alpha_{i_1}$$

- Rescrierea **nerecursivă la stânga**:

$$X \to \beta_1 X' \mid \ldots \mid \beta_m X'$$
$$X' \to \alpha_1 X' \mid \ldots \mid \alpha_n X' \mid \varepsilon$$

Cuprins

Motivație

- Costuri mari ale *backtracking*-ului utilizat în cadrul strategiei *recursive descent*, pentru alegerea producțiilor

- Pe cât posibil, restricționarea gramaticii, în vederea reducerii costurilor

Gramatici $LL(k)$

- ***L***$L(k)$: ***L****eft to right* (ordinea de parcurgere a șirului)
- L***L***(k): ***L****eftmost derivation*
- $LL($***k***$)$: maximum ***k*** *token*-i de *lookahead* pentru alegerea unei producții
- $LL(1)$: existența unei singure producții aplicabile pentru cel mai din stânga neterminal și pentru *token*-ul curent, sau eroare

Exemplu

$$E \to T + E \mid T$$
$$T \to id * T \mid id \mid (E)$$

- Non-$LL(1)$, din cauza prezenței lui T la începutul ambelor producții ale lui E!
- $LL(2)$
- Rescriere $LL(1)$ prin **factorizare la stânga** (*left factoring*)

Factorizare la stânga

- Înainte:

$$E \to T + E \mid T$$
$$T \to id * T \mid id \mid (E)$$

- După:

$$E \to TE'$$
$$E' \to +E \mid \varepsilon$$
$$T \to (E) \mid idT'$$
$$T' \to *T \mid \varepsilon$$

Tabele $LL(1)$

$$
\begin{aligned}
E &\to TE' \\
E' &\to +E \mid \varepsilon \\
T &\to (E) \mid id\,T' \\
T' &\to *T \mid \varepsilon
\end{aligned}
$$

	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε

Producția curentă, aleasă de la intersecția neterminalului din linii cu terminalul din coloane ($ = EOF)

Algoritmul de analiză *LL*(1) I

- Alegerea producției următoare din tabel, pe baza celui mai din stânga neterminal și a *token*-ului curent
- Frontiera arborelui de derivare, reținută într-o **stivă**
  - **Neterminalii** ce urmează a fi expandați
  - **Terminalii** ce urmează a fi confruntați cu *token*-ii
  - **Vârful** stivei = cel mai din stânga simbol de prelucrat
- **Acceptare**, la epuizarea intrării și a stivei
- **Respingere**, altfel, sau la accesarea unei celule goale din tabel

Algoritmul de analiză $LL(1)$ II

```
1  stiva ← ⟨S, $⟩
2  repetă
3      variante pop (stiva) :
4          neterminal X :
5              dacă tabel [X, peek ()] = Y_1 ... Y_n atunci
6                  push (Y_1 ... Y_n, stiva)
7              altfel eroare
8          terminal t :
9              dacă ¬ match (t) atunci eroare
10 până când stiva = ∅
```

Exemplu

E

- Șirul:

$$id + id * id\$$$

- Stiva:

$$E\$$$

- Tabelul:

	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	idT'			(E)		
T'		ε	$*T$		ε	ε

Exemplu

E

- Șirul:

$$id + id * id\$$$

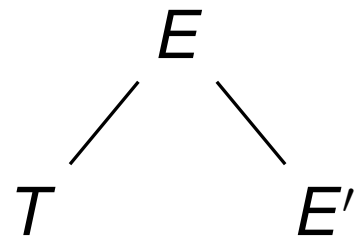
- Stiva:

$$E\$$$

- Tabelul:

	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε

Exemplu



- Șirul:

$$id + id * id\$$$

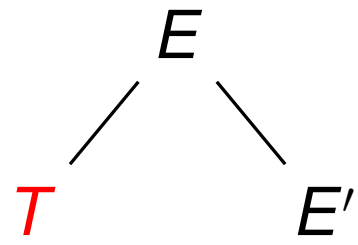
- Stiva:

$$T\,E'\,\$$$

- Tabelul:

	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε

Exemplu



- Șirul:

$$id + id * id\$$$

- Stiva:

$$T\,E'\,\$$$

- Tabelul:

	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε

Exemplu

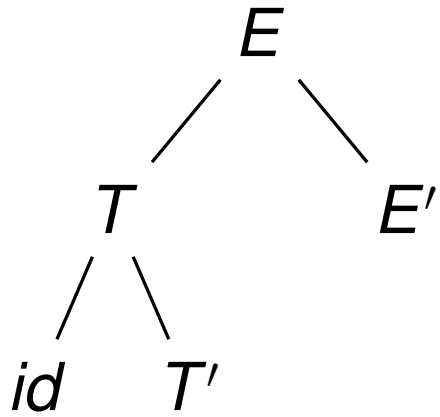
- Șirul:

$$id + id * id\$$$

- Stiva:

$$id\,T'\,E'\,\$$$

- Tabelul:



	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε

Exemplu

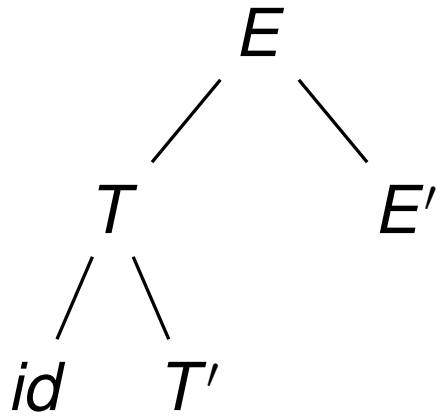
- Șirul:

$$id + id * id\$$$

- Stiva:

$$id\,T'\,E'\,\$$$

- Tabelul:



	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε

Exemplu

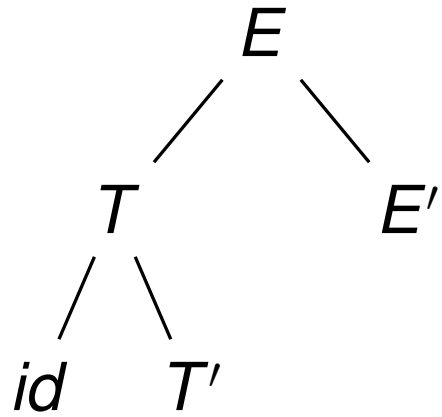
- Șirul:

$$id + id * id\$$$

- Stiva:

$$T'E'\$$$

- Tabelul:



	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	idT'			(E)		
T'		ε	$*T$		ε	ε

Exemplu

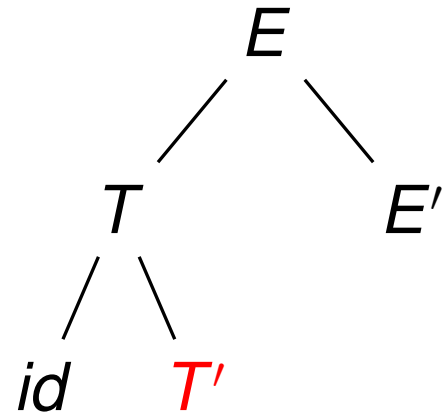
- Șirul:

$$id + id * id\$$$

- Stiva:

$$T' E'\$$$

- Tabelul:



	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε

Exemplu

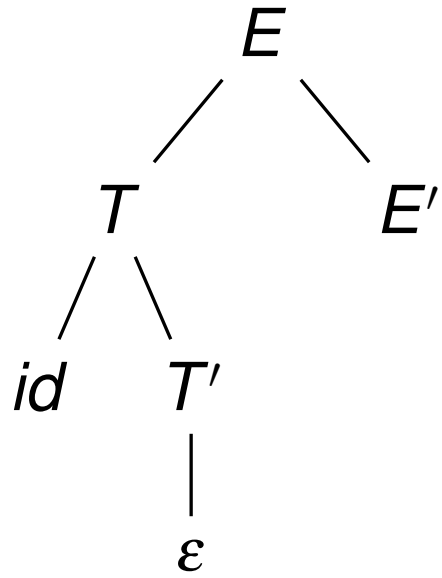
- Șirul:

$$id + id * id\$$$

- Stiva:

$$E'\$$$

- Tabelul:



	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε

Exemplu

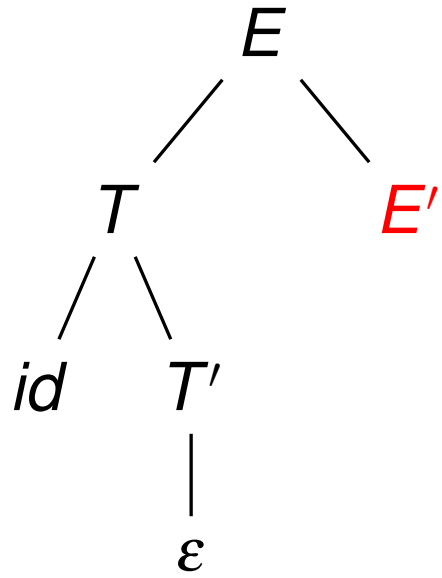
- Șirul:

$$id + id * id\$$$

- Stiva:

$$E'\$$$

- Tabelul:



	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε

Exemplu

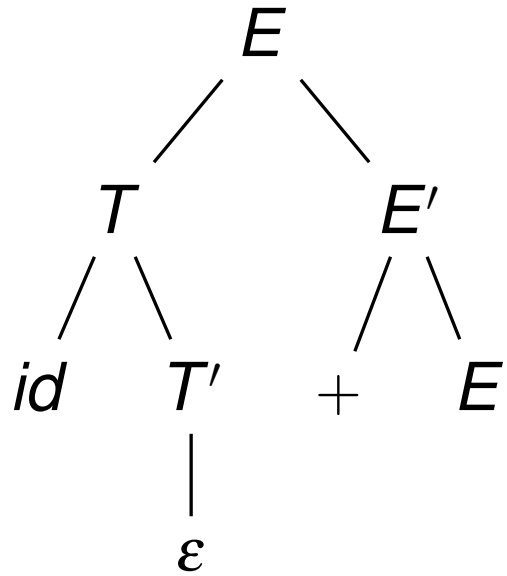
- Șirul:

$$id + id * id\$$$

- Stiva:

$$+E\$$$

- Tabelul:



	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε

Exemplu

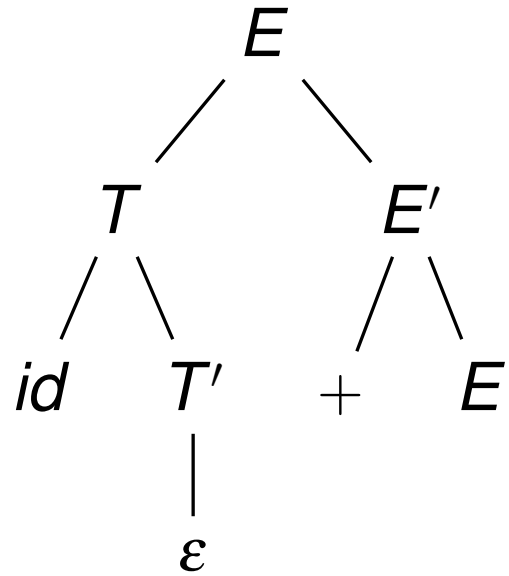
- Șirul:

$$id + id * id\$$$

- Stiva:

$$+E\$$$

- Tabelul:



	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	idT'			(E)		
T'		ε	$*T$		ε	ε

Exemplu

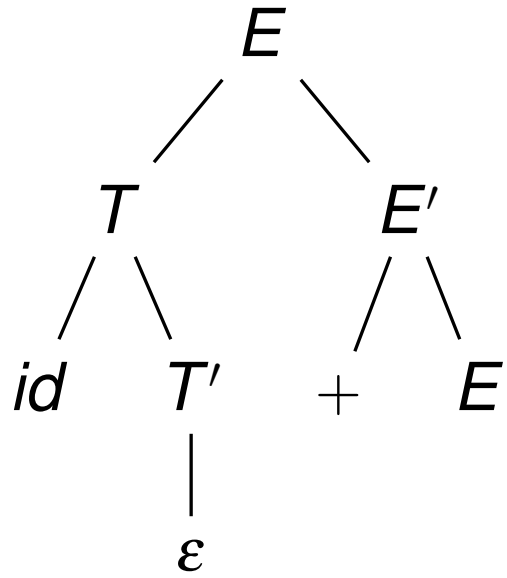
- Șirul:

$$id + id * id\$$$

- Stiva:

$$E\$$$

- Tabelul:



	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε

Exemplu

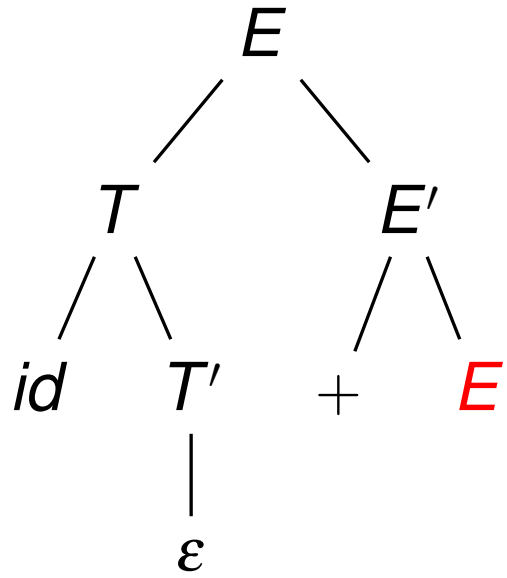
- Șirul:

$$id + id * id\$$$

- Stiva:

$$E\$$$

- Tabelul:



	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε

Exemplu

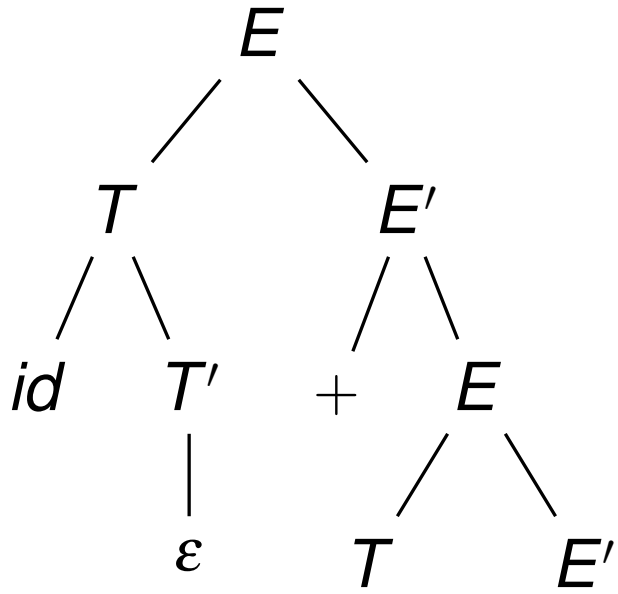
- Șirul:

$$id + id * id\$$$

- Stiva:

$$T\,E'\,\$$$

- Tabelul:



	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε

Exemplu

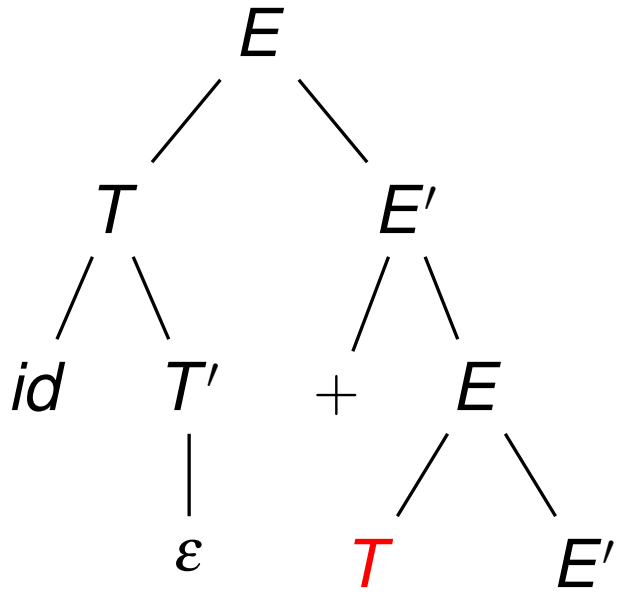
- Șirul:

$$id + id * id\$$$

- Stiva:

$$T\,E'\,\$$$

- Tabelul:



	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε

Exemplu

- Șirul:

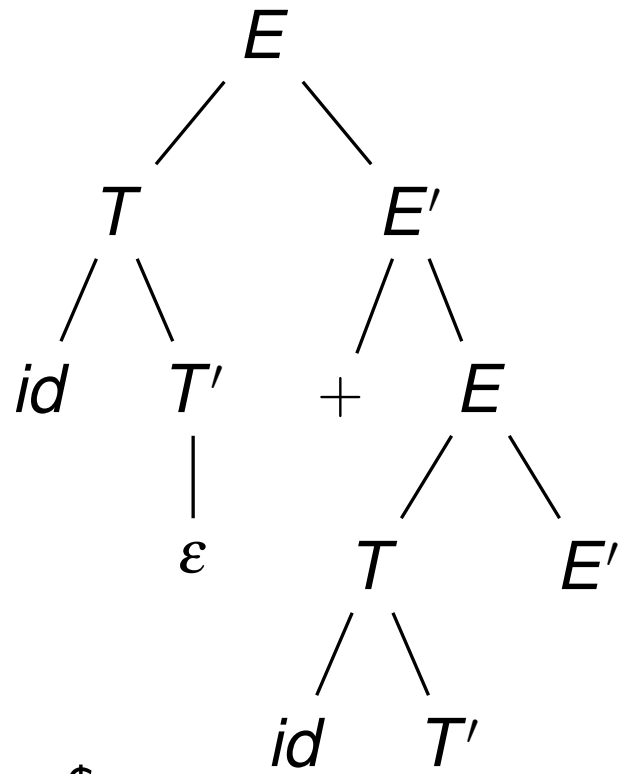
$$id + id * id\$$$

- Stiva:

$$id\,T'\,E'\,\$$$

- Tabelul:

	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε



Exemplu

- Șirul:

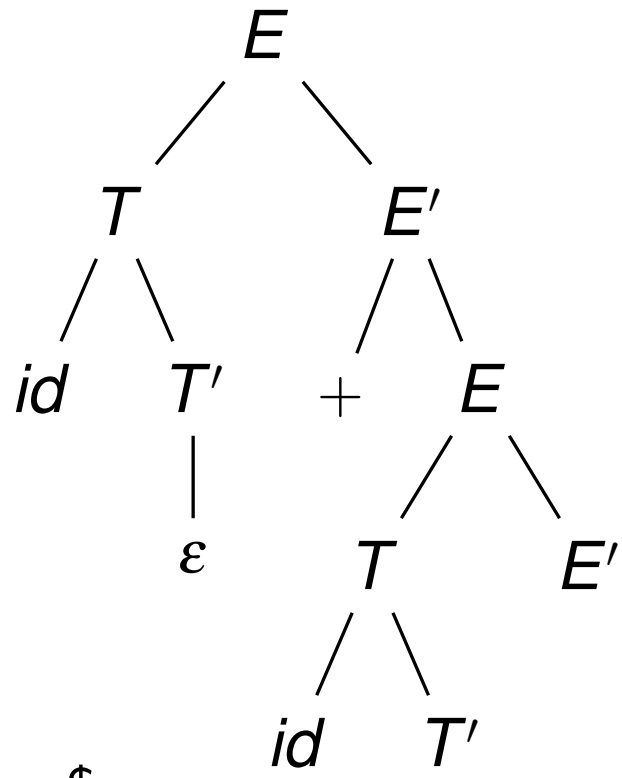
$$id + id * id\$$$

- Stiva:

$$id\,T'\,E'\,\$$$

- Tabelul:

	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε



Exemplu

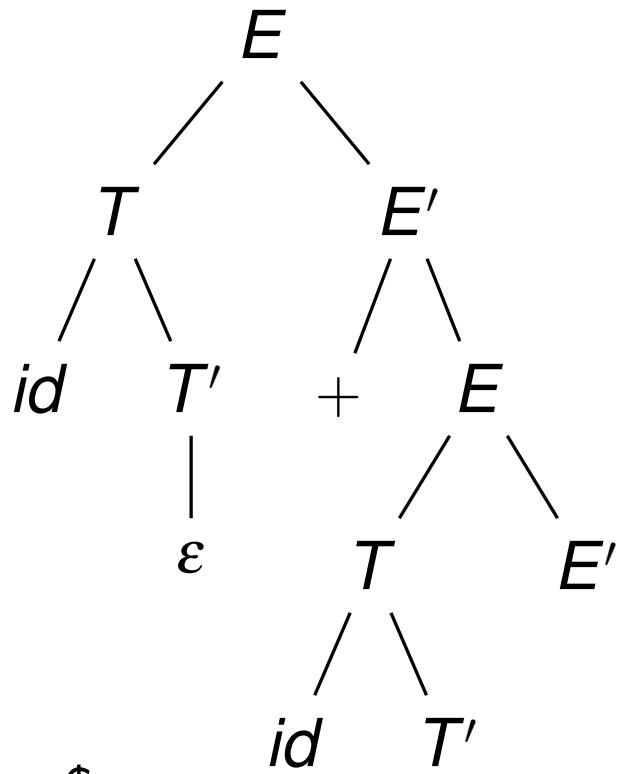
- Șirul:

$$id + id * id\$$$

- Stiva:

$$T'E'\$$$

- Tabelul:



	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	idT'			(E)		
T'		ε	$*T$		ε	ε

Exemplu

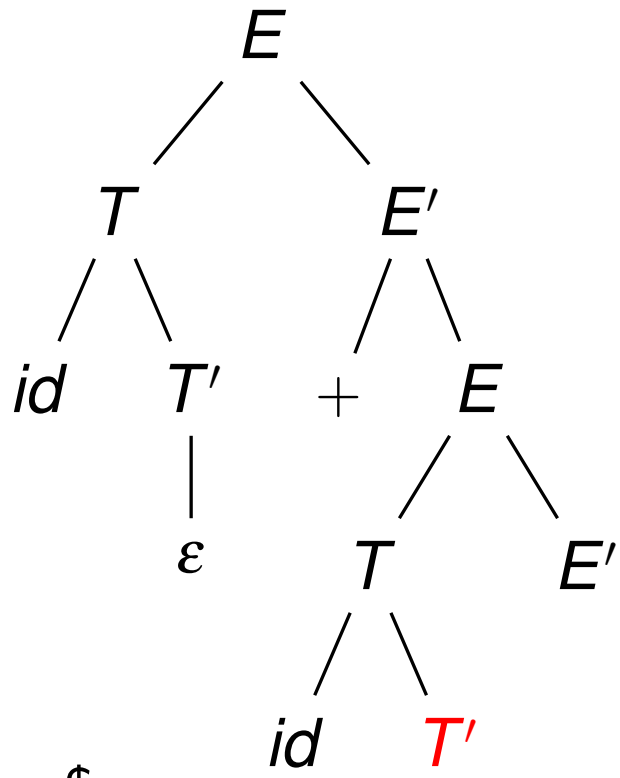
- Șirul:

$$id + id * id\$$$

- Stiva:

$$T'\,E'\,\$$$

- Tabelul:



	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε

Exemplu

- Șirul:

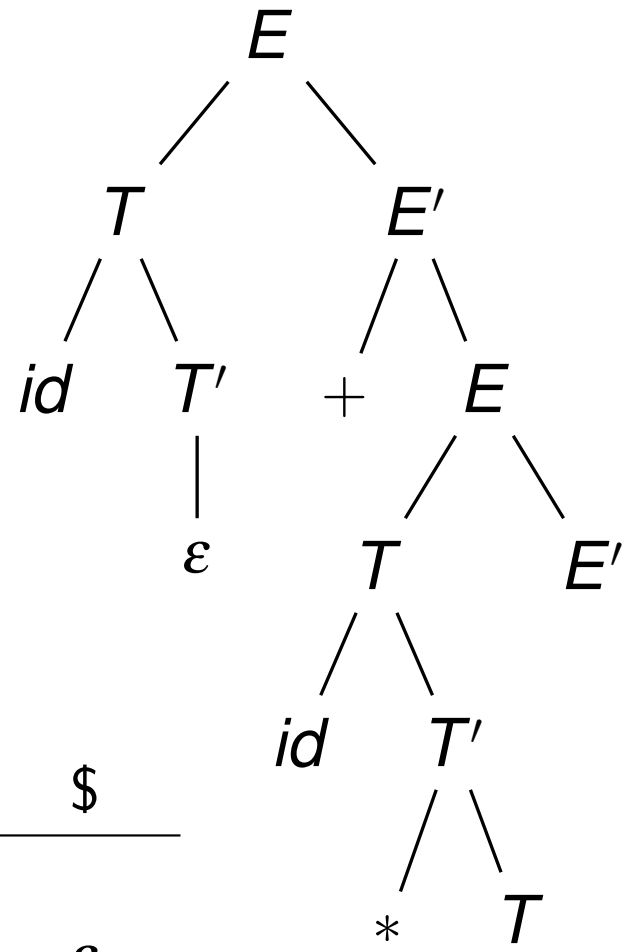
$$id + id * id\$$$

- Stiva:

$$* T E' \$$$

- Tabelul:

	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε



Exemplu

- Șirul:

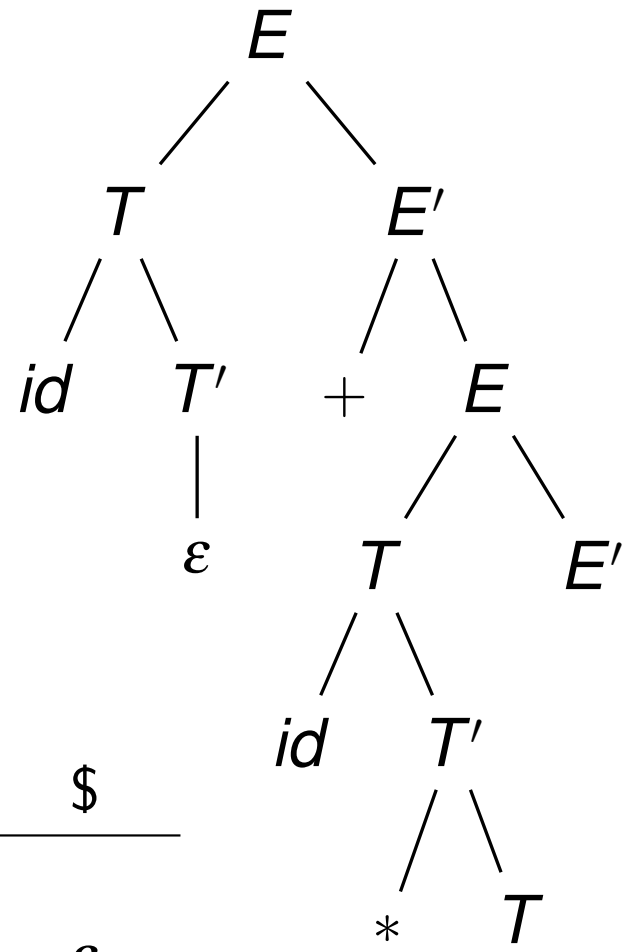
$$id + id * id\$$$

- Stiva:

$$* T E' \$$$

- Tabelul:

	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε



Exemplu

- Șirul:

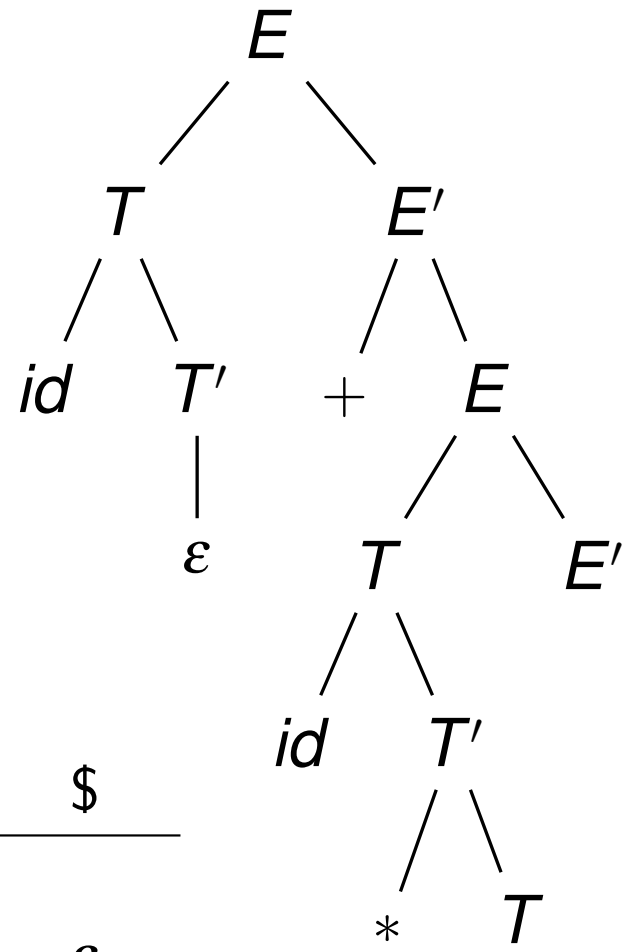
$$id + id * id\$$$

- Stiva:

$$T\,E'\,\$$$

- Tabelul:

	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε



Exemplu

- Șirul:

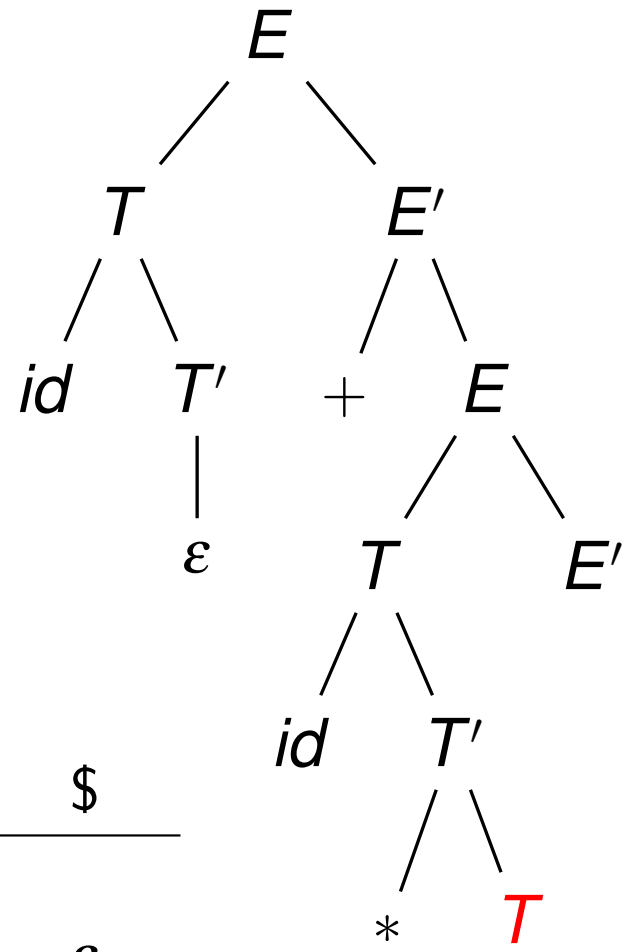
$$id + id * id\$$$

- Stiva:

$$T\,E'\,\$$$

- Tabelul:

	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε



Exemplu

- Șirul:

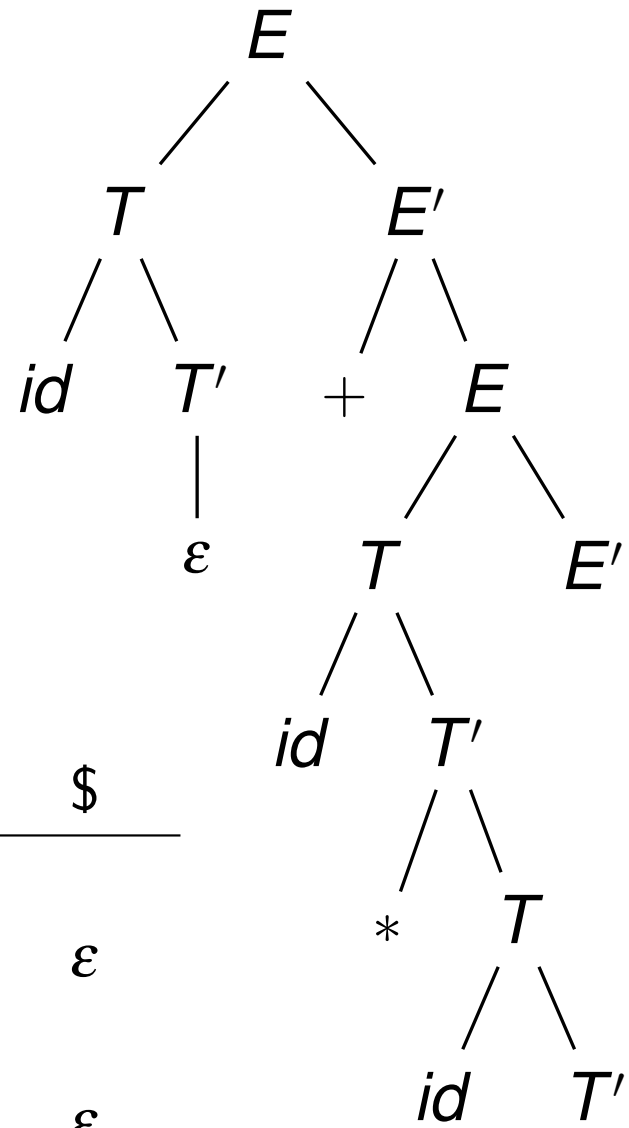
$$id + id * id\$$$

- Stiva:

$$id\,T'\,E'\,\$$$

- Tabelul:

	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε



Exemplu

- Șirul:

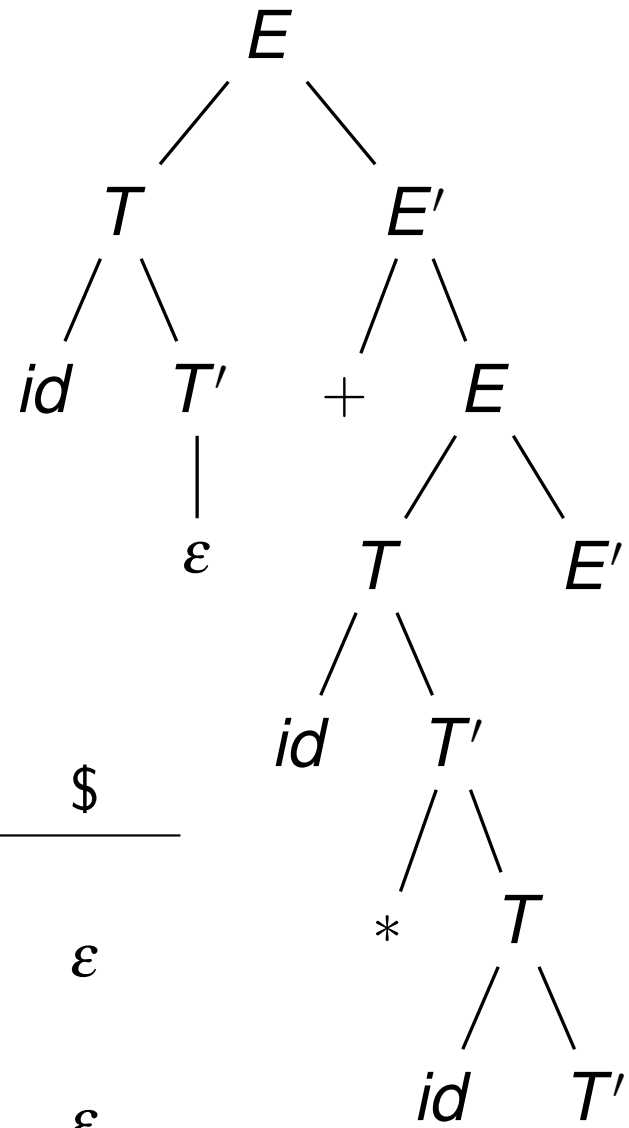
$$id + id * id\$$$

- Stiva:

$$id\, T'\, E'\, \$$$

- Tabelul:

	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε



Exemplu

- Șirul:

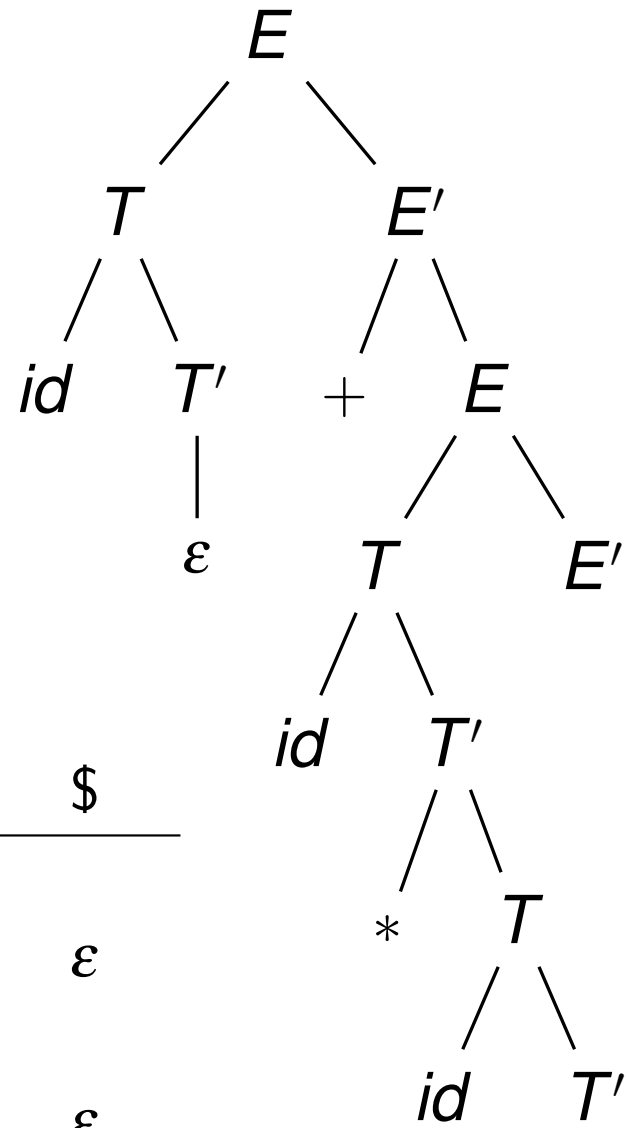
$$id + id * id\$$$

- Stiva:

$$T'E'\$$$

- Tabelul:

	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	idT'			(E)		
T'		ε	$*T$		ε	ε



Exemplu

- Șirul:

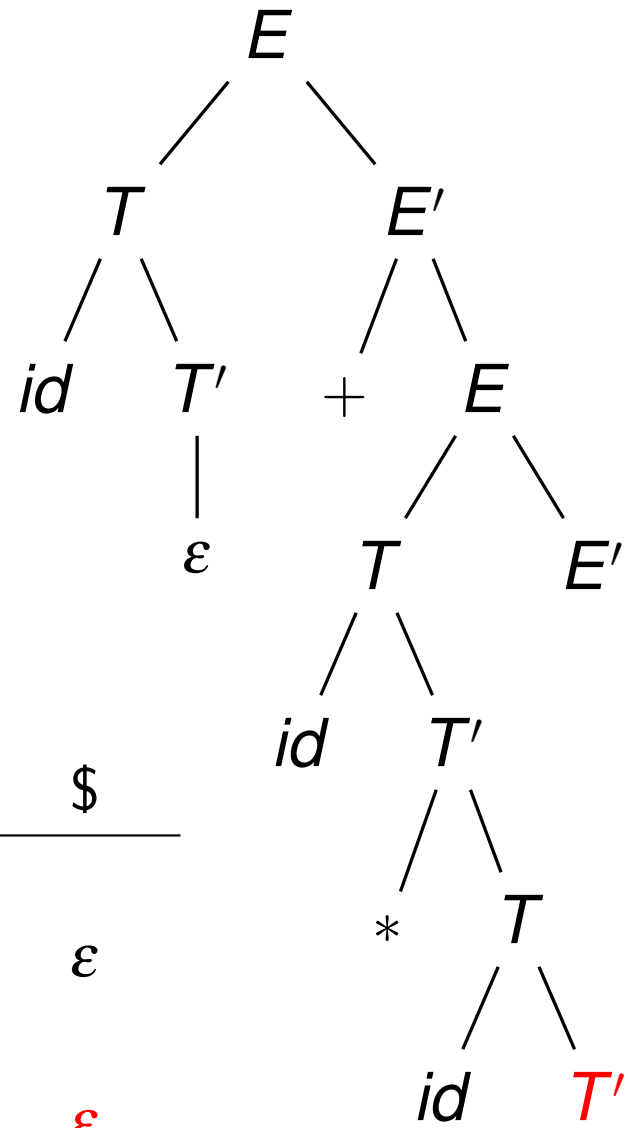
$$id + id * id\$$$

- Stiva:

$$T'\,E'\,\$$$

- Tabelul:

	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε



Exemplu

- Șirul:

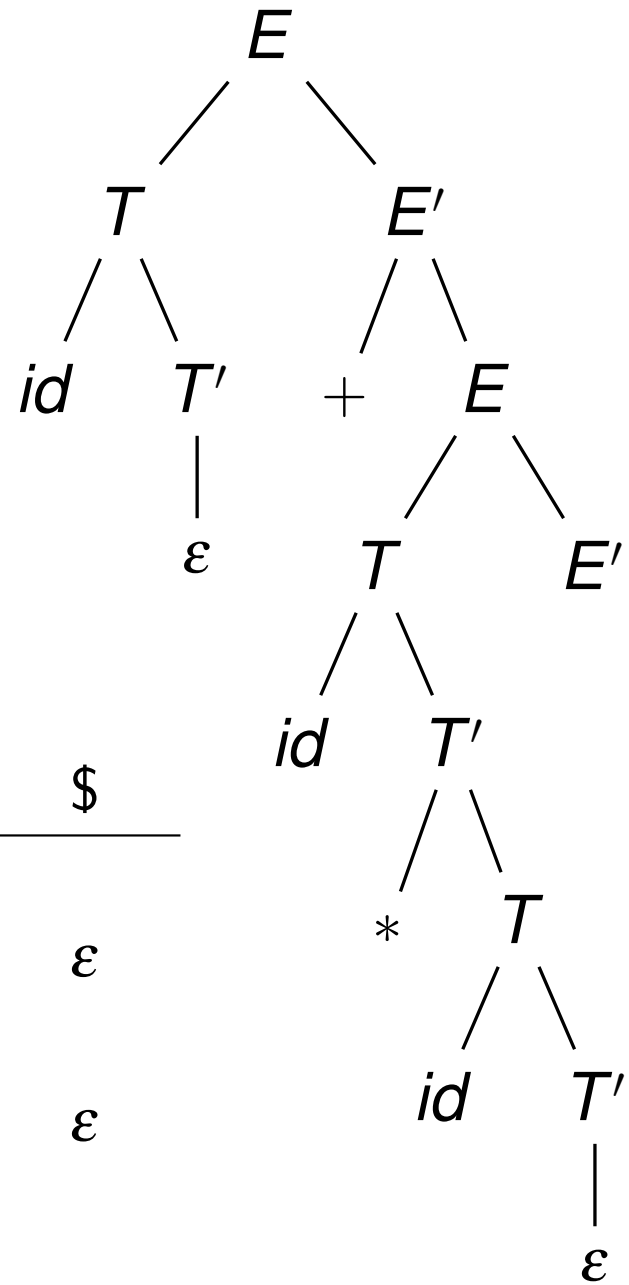
$$id + id * id\$$$

- Stiva:

$$E'\$$$

- Tabelul:

	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε



Exemplu

- Șirul:

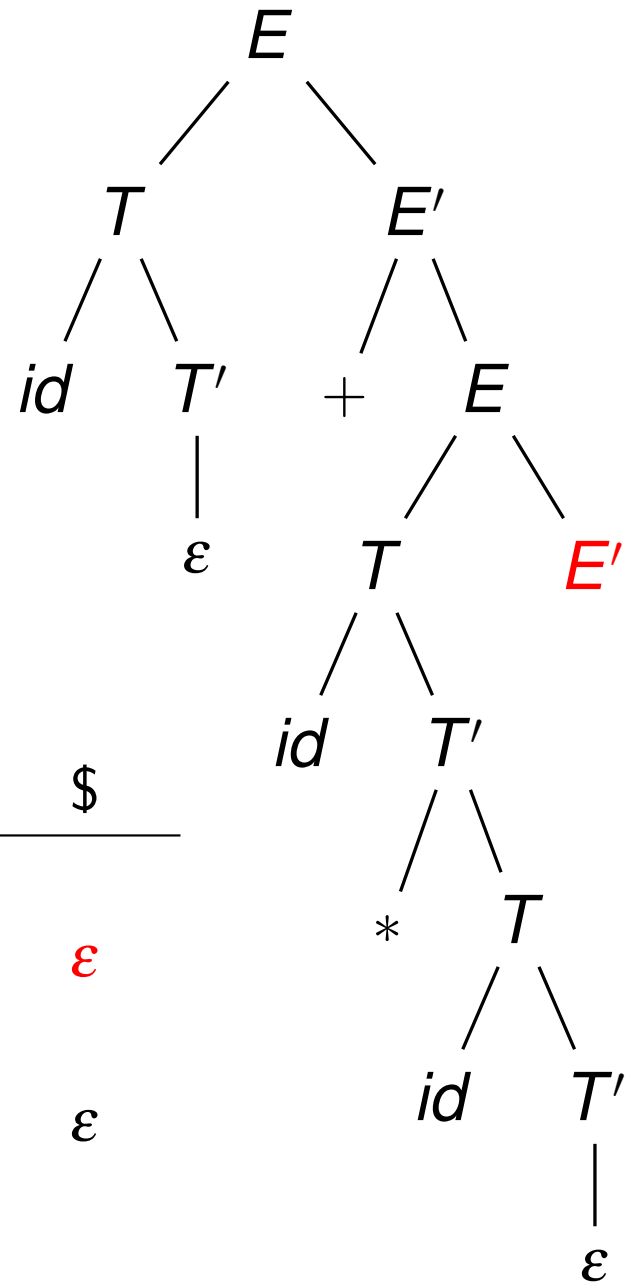
$$id + id * id \$$$

- Stiva:

$$E' \$$$

- Tabelul:

	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	$id\,T'$			(E)		
T'		ε	$*T$		ε	ε



Exemplu

- Șirul:

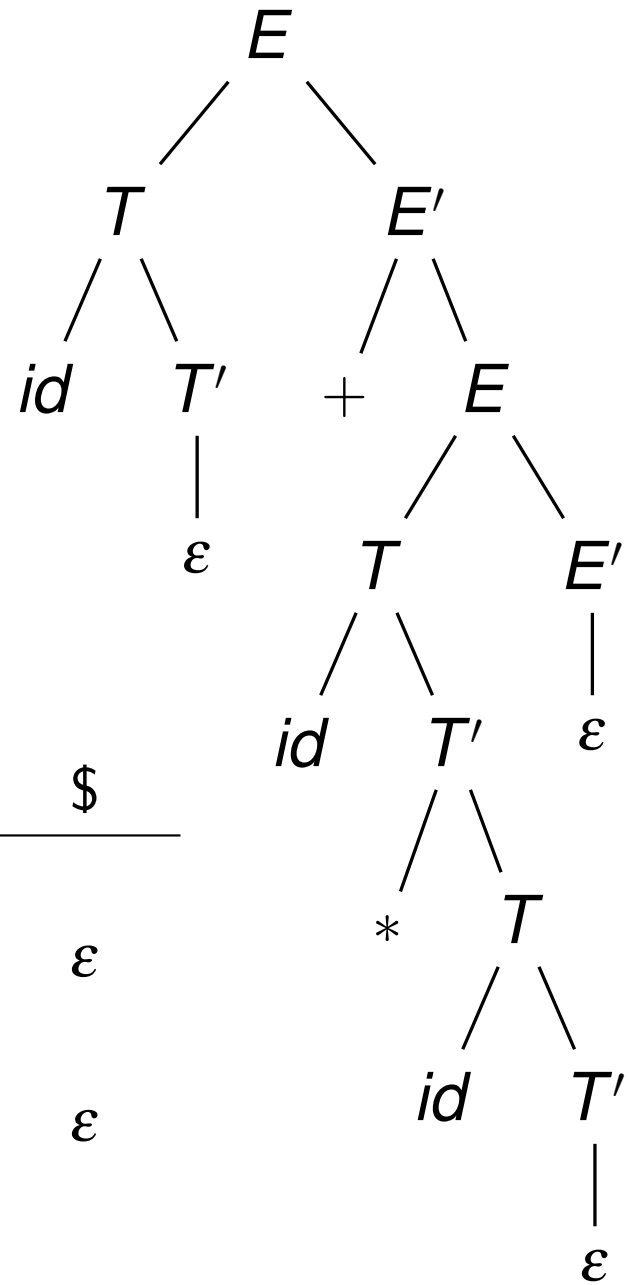
$$id + id * id\$$$

- Stiva:

$$\$$$

- Tabelul:

	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	idT'			(E)		
T'		ε	$*T$		ε	ε



Exemplu

- Șirul:

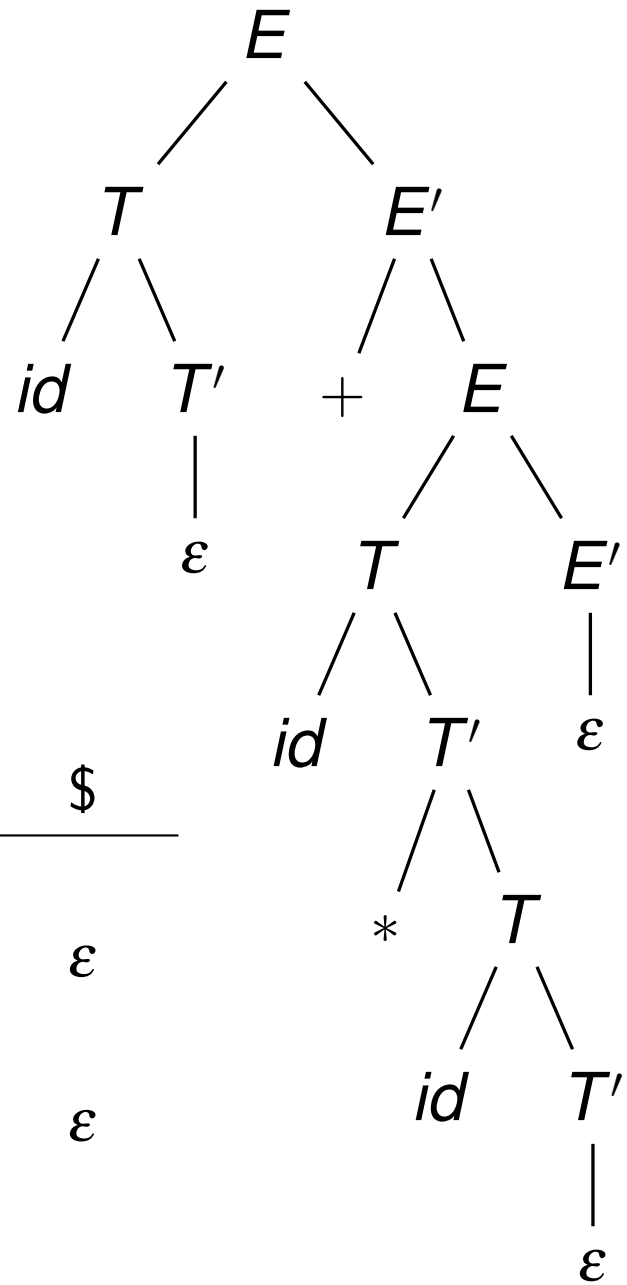
$$id + id * id\$$$

- Stiva:

$$\$$$

- Tabelul:

	id	$+$	$*$	$($	$)$	$\$$
E	TE'			TE'		
E'		$+E$			ε	ε
T	idT'			(E)		
T'		ε	$*T$		ε	ε



Intuiția din spatelele tabelelor

$$
\begin{aligned}
E &\to TE' \\
E' &\to +E \mid \varepsilon \\
T &\to (E) \mid idT' \\
T' &\to *T \mid \varepsilon
\end{aligned}
$$

- Pentru neterminalul curent E', cine prezice alternativa $+E$?

Intuiția din spatelele tabelelor

$$
\begin{aligned}
E &\to TE' \\
E' &\to +E \mid \varepsilon \\
T &\to (E) \mid idT' \\
T' &\to *T \mid \varepsilon
\end{aligned}
$$

- Pentru neterminalul curent E', cine prezice alternativa $+E$?
- Răspuns: *token*-ul $+ \in First(+E)$

Intuiția din spatelele tabelelor

$$
\begin{aligned}
E &\to TE' \\
E' &\to +E \mid \varepsilon \\
T &\to (E) \mid idT' \\
T' &\to *T \mid \varepsilon
\end{aligned}
$$

- Pentru neterminalul curent E, cine prezice alternativa TE'?

Intuiția din spatelele tabelelor

$$
\begin{aligned}
E &\to TE' \\
E' &\to +E \mid \varepsilon \\
T &\to (E) \mid idT' \\
T' &\to *T \mid \varepsilon
\end{aligned}
$$

- Pentru neterminalul curent E, cine prezice alternativa TE'?
- Răspuns: *token*-ii care îl prezic pe T: $\{(, id\} \subseteq First(TE')$

Intuiția din spatelele tabelelor

$$
\begin{aligned}
E &\to TE' \\
E' &\to +E \mid \varepsilon \\
T &\to (E) \mid idT' \\
T' &\to *T \mid \varepsilon
\end{aligned}
$$

- Pentru neterminalul curent E', cine prezice alternativa ε?

Intuiția din spatelele tabelelor

$$
\begin{aligned}
E &\to T\,E' \\
E' &\to +\,E \mid \varepsilon \\
T &\to (E) \mid id\,T' \\
T' &\to *\,T \mid \varepsilon
\end{aligned}
$$

- Pentru neterminalul curent E', cine prezice alternativa ε?
- Răspuns: Răspuns: *token*-ii care urmează referirile lui E' în alte producții: $E \to TE'$

Intuiția din spatelele tabelelor

$$
\begin{aligned}
E &\to T\,E' \\
E' &\to +\,E \mid \varepsilon \\
T &\to (E) \mid id\,T' \\
T' &\to *\,T \mid \varepsilon
\end{aligned}
$$

- Pentru neterminalul curent E', cine prezice alternativa ε?
- Răspuns: Răspuns: *token*-ii care urmează referirile lui E' în alte producții: $E \to TE'$, $T \to (E)$

Intuiția din spatelele tabelelor

$$
\begin{aligned}
E &\to T\,E' \\
E' &\to +\,E \mid \varepsilon \\
T &\to (E) \mid id\,T' \\
T' &\to *\,T \mid \varepsilon
\end{aligned}
$$

- Pentru neterminalul curent E', cine prezice alternativa ε?
- Răspuns: Răspuns: *token*-ii care urmează referirile lui E' în alte producții: $E \to TE'$, $T \to (E)$, deci $) \in Follow(E')$

Mulțimile *First*

- Definiție:

$$First(X) = \{t \in T \mid X \to^* t\alpha\} \cup \{\varepsilon \mid X \to^* \varepsilon\}$$

- $First(t) = \{t\}$
- $\varepsilon \in First(X)$ dacă:
  - $X \to \varepsilon$ SAU
  - $X \to \alpha$ și $\varepsilon \in First(\alpha)$
- $First(\alpha) \subseteq First(X)$, dacă $X \to \beta\alpha$ și $\varepsilon \in First(\beta)$
- $First(X_1X_2) = First(X_1)$, dacă $\varepsilon \notin First(X_1)$
- $First(X_1X_2) = (First(X_1) \setminus \{\varepsilon\}) \cup First(X_2)$, dacă $\varepsilon \in First(X_1)$

Exemplu

$$
\begin{aligned}
E &\rightarrow TE' \\
E' &\rightarrow +E \mid \varepsilon \\
T &\rightarrow (E) \mid id\,T' \\
T' &\rightarrow *T \mid \varepsilon
\end{aligned}
$$

$$
\begin{aligned}
First(id) &= \{id\} \\
First(+) &= \{+\} \\
First(*) &= \{*\} \\
First(() &= \{(\} \\
First()) &= \{)\} \\
First(E') &=
\end{aligned}
$$

Exemplu

$$
\begin{aligned}
E &\rightarrow TE' \\
E' &\rightarrow +E \mid \varepsilon \\
T &\rightarrow (E) \mid id\,T' \\
T' &\rightarrow *T \mid \varepsilon
\end{aligned}
$$

$$
\begin{aligned}
First(id) &= \{id\} \\
First(+) &= \{+\} \\
First(*) &= \{*\} \\
First(() &= \{(\} \\
First()) &= \{)\} \\
First(E') &= \{+,\ \varepsilon\} \\
First(T') &=
\end{aligned}
$$

Exemplu

$$
\begin{aligned}
E &\rightarrow TE' \\
E' &\rightarrow +E \mid \varepsilon \\
T &\rightarrow (E) \mid id\,T' \\
T' &\rightarrow *T \mid \varepsilon
\end{aligned}
$$

$$
\begin{aligned}
First(id) &= \{id\} \\
First(+) &= \{+\} \\
First(*) &= \{*\} \\
First(() &= \{(\} \\
First()) &= \{)\} \\
First(E') &= \{+,\ \varepsilon\} \\
First(T') &= \{*,\ \varepsilon\} \\
First(T) &=
\end{aligned}
$$

Exemplu

$$
\begin{aligned}
E &\rightarrow TE' \\
E' &\rightarrow +E \mid \varepsilon \\
T &\rightarrow (E) \mid id\,T' \\
T' &\rightarrow *T \mid \varepsilon
\end{aligned}
$$

$$
\begin{aligned}
First(id) &= \{id\} \\
First(+) &= \{+\} \\
First(*) &= \{*\} \\
First(() &= \{(\} \\
First()) &= \{)\} \\
First(E') &= \{+,\ \varepsilon\} \\
First(T') &= \{*,\ \varepsilon\} \\
First(T) &= \{(,\ id\} \\
First(E) &=
\end{aligned}
$$

Exemplu

$$
\begin{aligned}
E &\to TE' \\
E' &\to +E \mid \varepsilon \\
T &\to (E) \mid id\,T' \\
T' &\to *T \mid \varepsilon
\end{aligned}
$$

$$
\begin{aligned}
First(id) &= \{id\} \\
First(+) &= \{+\} \\
First(*) &= \{*\} \\
First(() &= \{(\} \\
First()) &= \{)\} \\
First(E') &= \{+,\ \varepsilon\} \\
First(T') &= \{*,\ \varepsilon\} \\
First(T) &= \{(,\ id\} \\
First(E) &= First(T) = \{(,\ id\}
\end{aligned}
$$

Mulțimile *Follow*

- Definiție:

$$Follow(X) = \{t \in T \mid Y \rightarrow^* \alpha X t \beta\}$$

- $\$ \in Follow(S)$ (simbolul de start)
- $First(\beta) \setminus \{\varepsilon\} \subseteq Follow(X)$ dacă $Y \rightarrow \alpha X \beta$
- $Follow(Y) \subseteq Follow(X)$, dacă $Y \rightarrow \alpha X \beta$ și $\varepsilon \in First(\beta)$

Exemplu

$\underline{Follow(+)}$

$$E \rightarrow TE'$$
$$E' \rightarrow +E \mid \varepsilon$$
$$T \rightarrow (E) \mid idT'$$
$$T' \rightarrow *T \mid \varepsilon$$

Exemplu

$$\underline{Follow(+)} \supseteq First(E) \setminus \{\varepsilon\} = \{(,\ id\}$$

$$\underline{Follow(*)}$$

$$E \rightarrow TE'$$
$$E' \rightarrow +E \mid \varepsilon$$
$$T \rightarrow (E) \mid idT'$$
$$T' \rightarrow *T \mid \varepsilon$$

Exemplu

$$\underline{Follow(+)} \supseteq First(E) \setminus \{\varepsilon\} = \{(,\ id\}$$
$$\underline{Follow(*)} \supseteq First(T) \setminus \{\varepsilon\} = \{(,\ id\}$$
$$\underline{Follow(()}$$

$$E \rightarrow TE'$$
$$E' \rightarrow +E \mid \varepsilon$$
$$T \rightarrow (E) \mid idT'$$
$$T' \rightarrow *T \mid \varepsilon$$

Exemplu

$$\underline{Follow(+)} \supseteq First(E) \setminus \{\varepsilon\} = \{(, id\}$$
$$\underline{Follow(*)} \supseteq First(T) \setminus \{\varepsilon\} = \{(, id\}$$
$$\underline{Follow(()} \supseteq First(E) \setminus \{\varepsilon\} = \{(, id\}$$
$$Follow(E)$$

$$E \rightarrow TE'$$
$$E' \rightarrow +E \mid \varepsilon$$
$$T \rightarrow (E) \mid idT'$$
$$T' \rightarrow *T \mid \varepsilon$$

Exemplu

$$
\begin{aligned}
E &\to TE' \\
E' &\to +E \mid \varepsilon \\
T &\to (E) \mid idT' \\
T' &\to *T \mid \varepsilon
\end{aligned}
$$

$$
\begin{aligned}
\underline{Follow(+)} &\supseteq First(E) \setminus \{\varepsilon\} = \{(, id\} \\
\underline{Follow(*)} &\supseteq First(T) \setminus \{\varepsilon\} = \{(, id\} \\
\underline{Follow(()} &\supseteq First(E) \setminus \{\varepsilon\} = \{(, id\} \\
Follow(E) &\supseteq \{\$\} \\
Follow(E)
\end{aligned}
$$

Exemplu

$$
\begin{aligned}
E &\to TE' \\
E' &\to +E \mid \varepsilon \\
T &\to (E) \mid idT' \\
T' &\to *T \mid \varepsilon
\end{aligned}
$$

$$
\begin{aligned}
\underline{Follow(+)} &\supseteq First(E) \setminus \{\varepsilon\} = \{(, id\} \\
\underline{Follow(*)} &\supseteq First(T) \setminus \{\varepsilon\} = \{(, id\} \\
\underline{Follow(()} &\supseteq First(E) \setminus \{\varepsilon\} = \{(, id\} \\
Follow(E) &\supseteq \{\$\} \\
Follow(E) &\supseteq Follow(E') \\
Follow(E)
\end{aligned}
$$

Exemplu

$$E \to TE'$$
$$E' \to +E \mid \varepsilon$$
$$T \to (E) \mid idT'$$
$$T' \to *T \mid \varepsilon$$

$$\underline{Follow(+)} \supseteq First(E) \setminus \{\varepsilon\} = \{(, id\}$$
$$\underline{Follow(*)} \supseteq First(T) \setminus \{\varepsilon\} = \{(, id\}$$
$$\underline{Follow(()} \supseteq First(E) \setminus \{\varepsilon\} = \{(, id\}$$
$$Follow(E) \supseteq \{\$\}$$
$$Follow(E) \supseteq Follow(E')$$
$$Follow(E) \supseteq First()) \setminus \{\varepsilon\} = \{)\}$$
$$Follow(E')$$

Exemplu

$$
\begin{aligned}
E &\to TE' \\
E' &\to +E \mid \varepsilon \\
T &\to (E) \mid idT' \\
T' &\to *T \mid \varepsilon
\end{aligned}
$$

$$
\begin{aligned}
\underline{Follow(+)} &\supseteq First(E) \setminus \{\varepsilon\} = \{(, id\} \\
\underline{Follow(*)} &\supseteq First(T) \setminus \{\varepsilon\} = \{(, id\} \\
\underline{Follow(()} &\supseteq First(E) \setminus \{\varepsilon\} = \{(, id\} \\
Follow(E) &\supseteq \{\$\} \\
Follow(E) &\supseteq Follow(E') \\
Follow(E) &\supseteq First()) \setminus \{\varepsilon\} = \{)\} \\
Follow(E') &\supseteq Follow(E) \\
\underline{Follow(E)} &
\end{aligned}
$$

Exemplu

$$
\begin{aligned}
E &\to T\,E' \\
E' &\to +\,E \mid \varepsilon \\
T &\to (E) \mid id\,T' \\
T' &\to *\,T \mid \varepsilon
\end{aligned}
$$

$$
\begin{aligned}
\underline{Follow(+)} &\supseteq First(E) \setminus \{\varepsilon\} = \{(,\ id\} \\
\underline{Follow(*)} &\supseteq First(T) \setminus \{\varepsilon\} = \{(,\ id\} \\
\underline{Follow(()} &\supseteq First(E) \setminus \{\varepsilon\} = \{(,\ id\} \\
Follow(E) &\supseteq \{\$\} \\
Follow(E) &\supseteq Follow(E') \\
Follow(E) &\supseteq First()) \setminus \{\varepsilon\} = \{)\} \\
Follow(E') &\supseteq Follow(E) \\
\underline{Follow(E)} &= \underline{Follow(E')} = \{),\ \$\} \\
Follow(T)
\end{aligned}
$$

Exemplu

$$
\begin{aligned}
E &\to T\,E' \\
E' &\to +\,E \mid \varepsilon \\
T &\to (E) \mid id\,T' \\
T' &\to *\,T \mid \varepsilon
\end{aligned}
$$

$$
\begin{aligned}
\underline{Follow(+)} &\supseteq First(E) \setminus \{\varepsilon\} = \{(,\ id\} \\
\underline{Follow(*)} &\supseteq First(T) \setminus \{\varepsilon\} = \{(,\ id\} \\
\underline{Follow(()} &\supseteq First(E) \setminus \{\varepsilon\} = \{(,\ id\} \\
Follow(E) &\supseteq \{\$\} \\
Follow(E) &\supseteq Follow(E') \\
Follow(E) &\supseteq First()) \setminus \{\varepsilon\} = \{)\} \\
Follow(E') &\supseteq Follow(E) \\
\underline{Follow(E)} &= \underline{Follow(E')} = \{),\ \$\} \\
Follow(T) &\supseteq First(E') \setminus \{\varepsilon\} = \{+\} \\
Follow(T)
\end{aligned}
$$

Exemplu

$$
\begin{aligned}
E &\to T\,E' \\
E' &\to +\,E \mid \varepsilon \\
T &\to (E) \mid id\,T' \\
T' &\to *\,T \mid \varepsilon
\end{aligned}
$$

$$
\begin{aligned}
\underline{Follow(+)} &\supseteq First(E) \setminus \{\varepsilon\} = \{(,\ id\} \\
\underline{Follow(*)} &\supseteq First(T) \setminus \{\varepsilon\} = \{(,\ id\} \\
\underline{Follow(()} &\supseteq First(E) \setminus \{\varepsilon\} = \{(,\ id\} \\
Follow(E) &\supseteq \{\$\} \\
Follow(E) &\supseteq Follow(E') \\
Follow(E) &\supseteq First()) \setminus \{\varepsilon\} = \{)\} \\
Follow(E') &\supseteq Follow(E) \\
\underline{Follow(E)} &= \underline{Follow(E')} = \{),\ \$\} \\
Follow(T) &\supseteq First(E') \setminus \{\varepsilon\} = \{+\} \\
Follow(T) &\supseteq Follow(E) \\
Follow(T)
\end{aligned}
$$

Exemplu

$$E \to TE'$$
$$E' \to +E \mid \varepsilon$$
$$T \to (E) \mid id\,T'$$
$$T' \to *\,T \mid \varepsilon$$

$$\underline{Follow(+)} \supseteq First(E) \setminus \{\varepsilon\} = \{(,\ id\}$$
$$\underline{Follow(*)} \supseteq First(T) \setminus \{\varepsilon\} = \{(,\ id\}$$
$$\underline{Follow(()} \supseteq First(E) \setminus \{\varepsilon\} = \{(,\ id\}$$
$$Follow(E) \supseteq \{\$\}$$
$$Follow(E) \supseteq Follow(E')$$
$$Follow(E) \supseteq First()) \setminus \{\varepsilon\} = \{)\}$$
$$Follow(E') \supseteq Follow(E)$$
$$\underline{Follow(E)} = \underline{Follow(E')} = \{),\ \$\}$$
$$Follow(T) \supseteq First(E') \setminus \{\varepsilon\} = \{+\}$$
$$Follow(T) \supseteq Follow(E)$$
$$Follow(T) \supseteq Follow(T')$$
$$Follow(T')$$

Exemplu

$$
\begin{aligned}
E &\to TE' \\
E' &\to +E \mid \varepsilon \\
T &\to (E) \mid idT' \\
T' &\to *T \mid \varepsilon
\end{aligned}
$$

$$
\begin{aligned}
\underline{Follow(+)} &\supseteq First(E) \setminus \{\varepsilon\} = \{(, id\} \\
\underline{Follow(*)} &\supseteq First(T) \setminus \{\varepsilon\} = \{(, id\} \\
\underline{Follow(()} &\supseteq First(E) \setminus \{\varepsilon\} = \{(, id\} \\
Follow(E) &\supseteq \{\$\} \\
Follow(E) &\supseteq Follow(E') \\
Follow(E) &\supseteq First()) \setminus \{\varepsilon\} = \{)\} \\
Follow(E') &\supseteq Follow(E) \\
\underline{Follow(E)} &= \underline{Follow(E')} = \{), \$\} \\
Follow(T) &\supseteq First(E') \setminus \{\varepsilon\} = \{+\} \\
Follow(T) &\supseteq Follow(E) \\
Follow(T) &\supseteq Follow(T') \\
Follow(T') &\supseteq Follow(T) \\
\underline{Follow(T)}
\end{aligned}
$$

Exemplu

$$
\begin{aligned}
E &\to TE' \\
E' &\to +E \mid \varepsilon \\
T &\to (E) \mid idT' \\
T' &\to *T \mid \varepsilon
\end{aligned}
$$

$$
\begin{aligned}
\underline{Follow(+)} &\supseteq First(E) \setminus \{\varepsilon\} = \{(, id\} \\
\underline{Follow(*)} &\supseteq First(T) \setminus \{\varepsilon\} = \{(, id\} \\
\underline{Follow(()} &\supseteq First(E) \setminus \{\varepsilon\} = \{(, id\} \\
Follow(E) &\supseteq \{\$\} \\
Follow(E) &\supseteq Follow(E') \\
Follow(E) &\supseteq First()) \setminus \{\varepsilon\} = \{)\} \\
Follow(E') &\supseteq Follow(E) \\
\underline{Follow(E)} &= \underline{Follow(E')} = \{), \$\} \\
Follow(T) &\supseteq First(E') \setminus \{\varepsilon\} = \{+\} \\
Follow(T) &\supseteq Follow(E) \\
Follow(T) &\supseteq Follow(T') \\
Follow(T') &\supseteq Follow(T) \\
\underline{Follow(T)} &= \underline{Follow(T')} = \{+,), \$\}
\end{aligned}
$$

Exemplu

$$Follow(id)$$

$$E \rightarrow TE'$$
$$E' \rightarrow +E \mid \varepsilon$$
$$T \rightarrow (E) \mid id\,T'$$
$$T' \rightarrow *T \mid \varepsilon$$

Exemplu

$$Follow(id) \supseteq First(T') \setminus \{\varepsilon\} = \{*\}$$

$Follow(id)$

$$E \rightarrow TE'$$
$$E' \rightarrow +E \mid \varepsilon$$
$$T \rightarrow (E) \mid id\,T'$$
$$T' \rightarrow *T \mid \varepsilon$$

Exemplu

$$\mathit{Follow}(id) \supseteq \mathit{First}(T') \setminus \{\varepsilon\} = \{*\}$$
$$\mathit{Follow}(id) \supseteq \mathit{Follow}(T)$$
$$\underline{\mathit{Follow}(id)}$$

$$E \rightarrow TE'$$
$$E' \rightarrow +E \mid \varepsilon$$
$$T \rightarrow (E) \mid idT'$$
$$T' \rightarrow *T \mid \varepsilon$$

Exemplu

$$E \to TE'$$
$$E' \to +E \mid \varepsilon$$
$$T \to (E) \mid idT'$$
$$T' \to *T \mid \varepsilon$$

$$Follow(id) \supseteq First(T') \setminus \{\varepsilon\} = \{*\}$$
$$Follow(id) \supseteq Follow(T)$$
$$\underline{Follow(id)} = \{+, *,), \$\}$$
$$\underline{Follow())}$$

Exemplu

$$E \to TE'$$
$$E' \to +E \mid \varepsilon$$
$$T \to (E) \mid idT'$$
$$T' \to *T \mid \varepsilon$$

$$Follow(id) \supseteq First(T') \setminus \{\varepsilon\} = \{*\}$$
$$Follow(id) \supseteq Follow(T)$$
$$\underline{Follow(id)} = \{+, *,), \$\}$$
$$\underline{Follow())} \supseteq Follow(T) = \{+,), \$\}$$

Construirea tabelelor

Pentru fiecare producție $X \to \alpha$ și terminal (sau \$) t, avem că $tabel[X, t] = \alpha$, dacă:

- $t \in First(\alpha)$ SAU
- $\varepsilon \in First(\alpha)$ și $t \in Follow(X)$

Gramatici non-$LL(1)$

- Valori multiple ale intrărilor din tabel în caz de gramatică:
  - Ambiguă
  - Recursivă la stânga
  - Nefactorizată la stânga
  - Precum și în alte situații
- Construcții din programare non-$LL(1)$ (scăderea asociativă la dreapta?!)

Partea V

Analiza sintactică *bottom-up*

Cuprins

Cuprins

Principii

- Arbore de derivare construit de jos în sus și de la stânga la dreapta
- **Reducerea** intrării la simbolul de start, prin inversarea producțiilor
- ***Reducție*** = *producție* aplicată invers

Caracteristici

- Generalitate mai mare decât analiza *top-down* predictivă
- Eficiență comparabilă
- Absența necesității de eliminare a recursivității la stânga și de factorizare la stânga a gramaticilor!

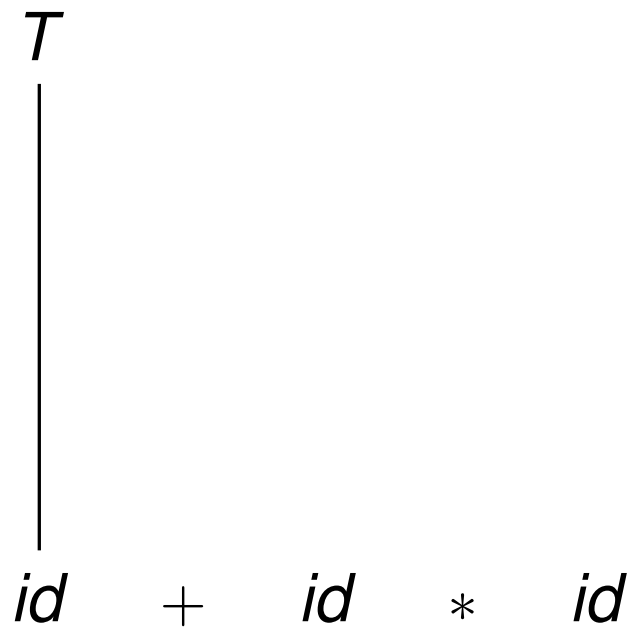
Exemplu

$$E \to T + E \mid T$$
$$T \to id * T \mid id \mid (E)$$

$$id \quad + \quad id \quad * \quad id$$

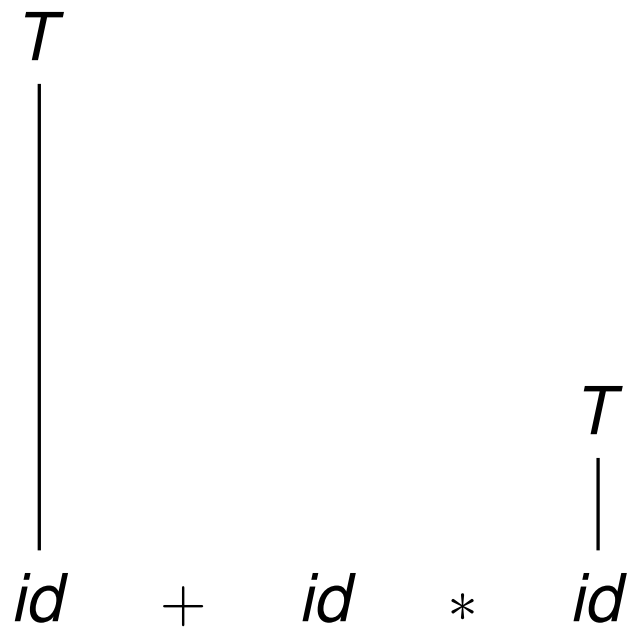
Exemplu

$$E \to T + E \mid T$$
$$T \to id * T \mid id \mid (E)$$



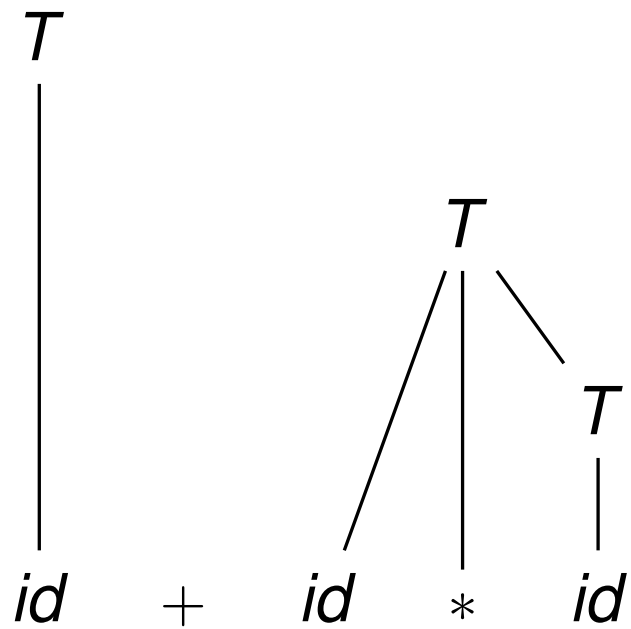
Exemplu

$$E \to T + E \mid T$$
$$T \to id * T \mid id \mid (E)$$



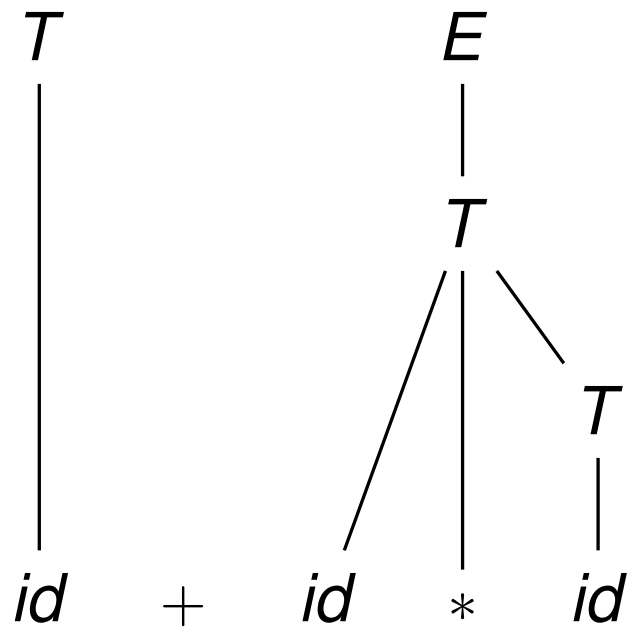
Exemplu

$$E \to T + E \mid T$$
$$T \to id * T \mid id \mid (E)$$



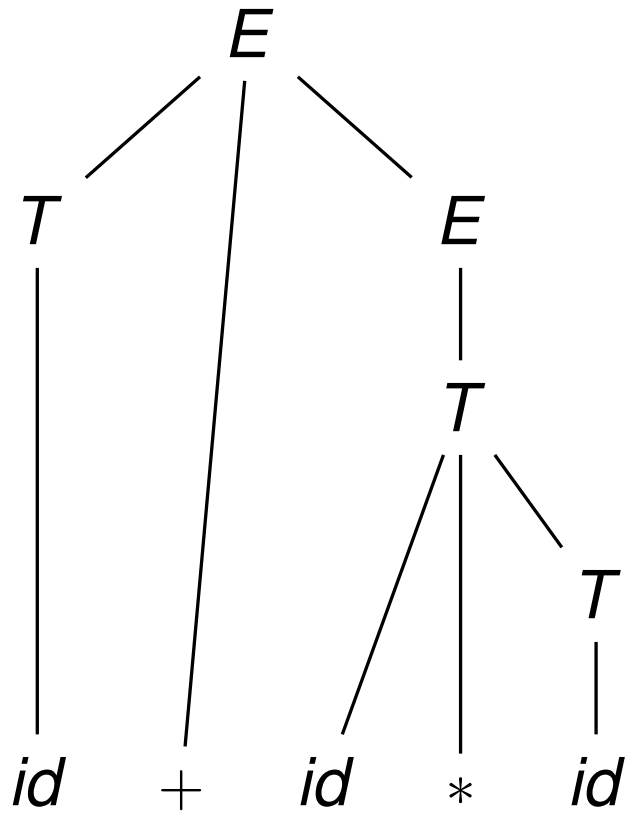
Exemplu

$$E \to T + E \mid T$$
$$T \to id * T \mid id \mid (E)$$



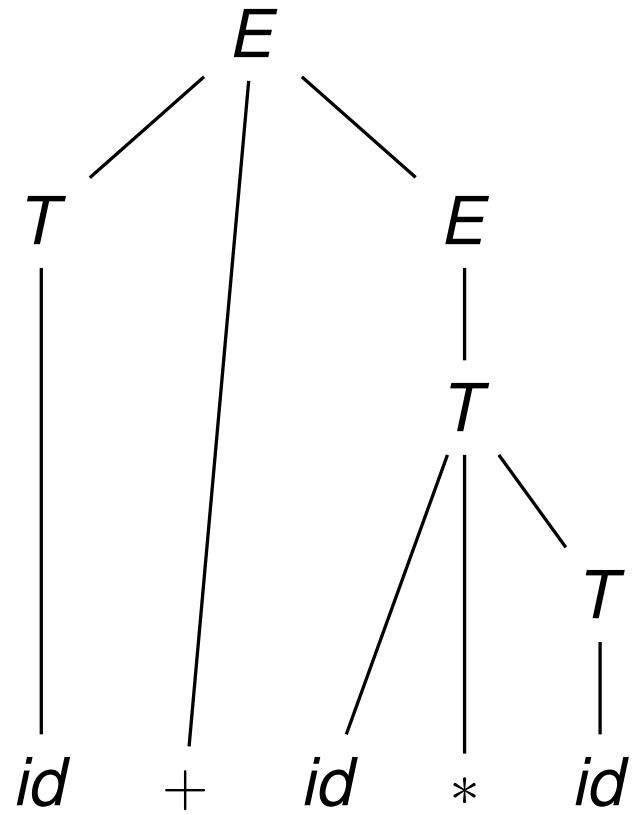
Exemplu

$$E \to T + E \mid T$$
$$T \to id * T \mid id \mid (E)$$



Exemplu

$$E \to T + E \mid T$$
$$T \to id * T \mid id \mid (E)$$



Derivare **la dreapta**, construită în ordine inversă!

Cuprins

Observație

- Starea curentă a șirului: $\alpha\beta\gamma$
- Următoarea reducție, conform $X \to \beta$
- Atunci, γ, format exclusiv din terminali

Observație

- Starea curentă a șirului: $\alpha\beta\gamma$
- Următoarea reducție, conform $X \to \beta$
- Atunci, γ, format exclusiv din terminali
- Explicație: existența pasului $\alpha X\gamma \to \alpha\beta\gamma$ în cadrul unei derivări **la dreapta**

Idee

- Divizarea șirului în două secvențe, separate de „|”: $\alpha\beta \mid \gamma$
- Secvența din **dreapta**, neprelucrată încă, formată exclusiv din terminali
- Secvența din **stânga**, formată din terminali și neterminali
- Inițial, întreagul șir, neprelucrat: $\mid x_1 \ldots x_n$

Acțiuni

- ***Shift***: deplasarea lui „|” o poziție spre dreapta:

$$Y_1 \dots Y_m \mid x_1 x_2 \dots x_n \;\rightarrow\; Y_1 \dots Y_m x_1 \mid x_2 \dots x_n$$

- ***Reduce***: aplicarea inversă a unei producții $X \rightarrow Y_i \dots Y_m$, la stânga lui „|”:

$$Y_1 \dots Y_{i-1} Y_i \dots Y_m \mid x_1 x_2 \dots x_n \;\rightarrow\; Y_1 \dots Y_{i-1} X \mid x_1 x_2 \dots x_n$$

Exemplu

$$E \rightarrow T + E \mid T$$
$$T \rightarrow id * T \mid id \mid (E)$$

$$\mid id \quad + \quad id \quad * \quad id$$

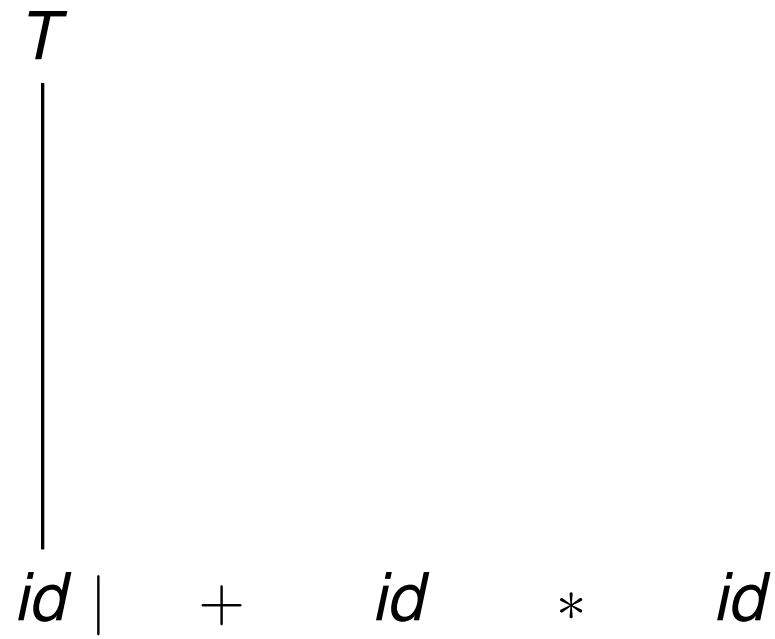
Exemplu

$$E \rightarrow T + E \mid T$$
$$T \rightarrow id * T \mid id \mid (E)$$

$$id \mid \quad + \quad id \quad * \quad id$$

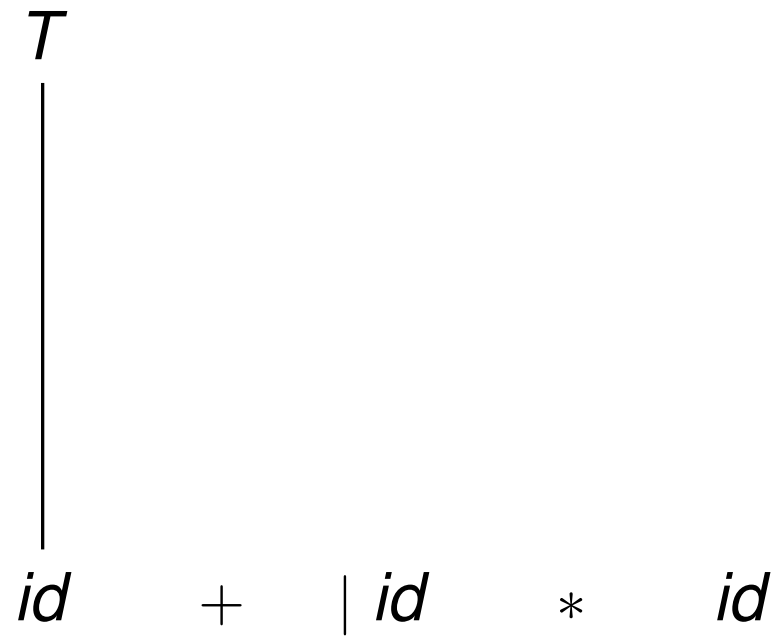
Exemplu

$$E \rightarrow T + E \mid T$$
$$T \rightarrow id * T \mid id \mid (E)$$



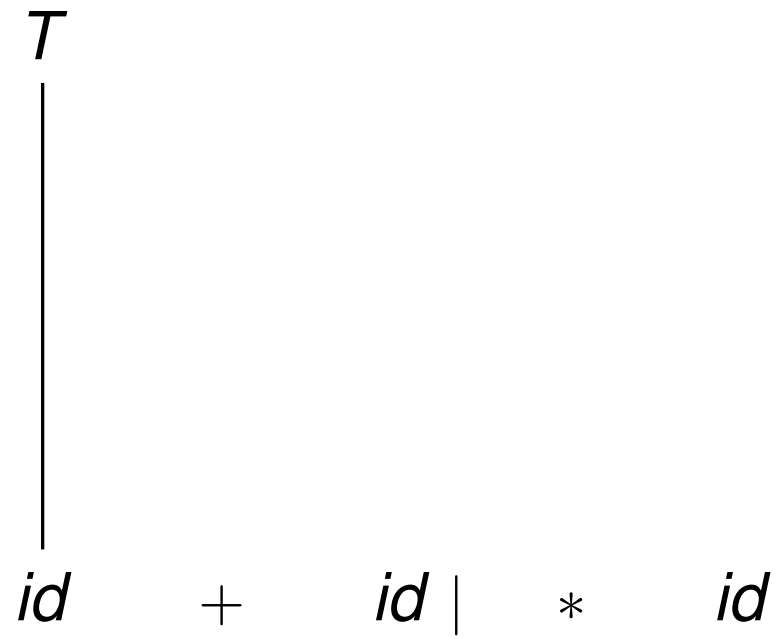
Exemplu

$$E \to T + E \mid T$$
$$T \to id * T \mid id \mid (E)$$



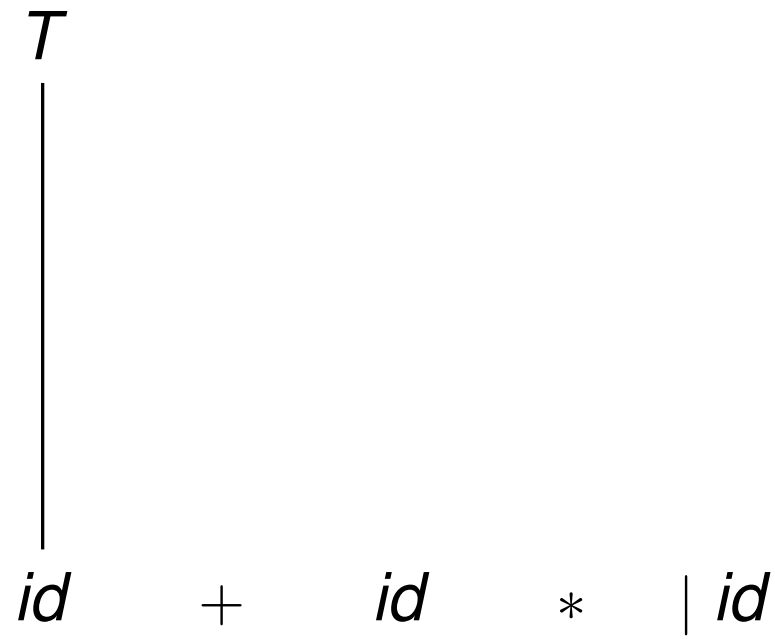
Exemplu

$$E \to T + E \mid T$$
$$T \to id * T \mid id \mid (E)$$



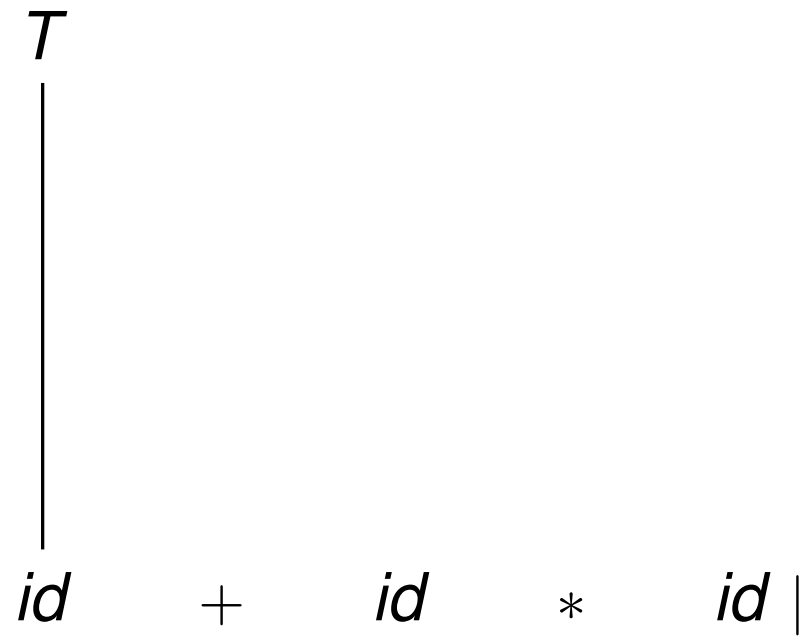
Exemplu

$$E \to T + E \mid T$$
$$T \to id * T \mid id \mid (E)$$



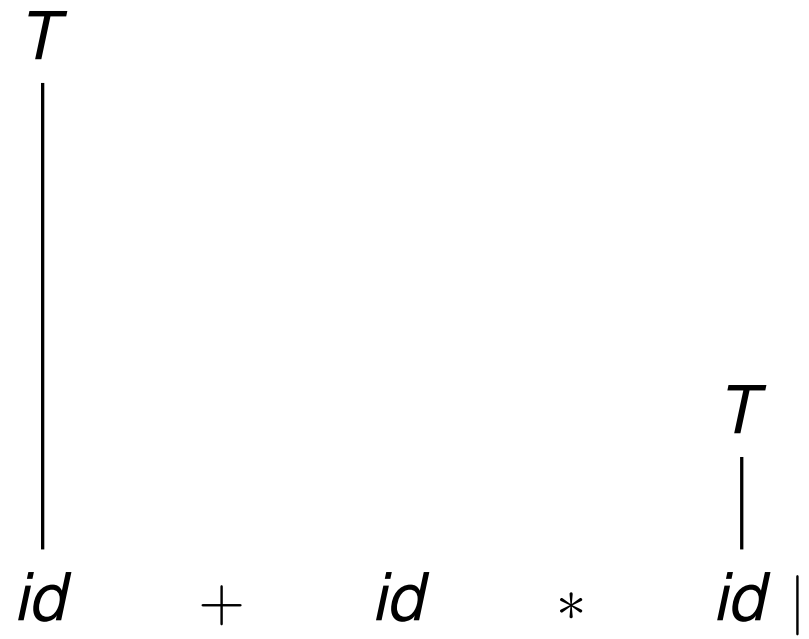
Exemplu

$$E \rightarrow T + E \mid T$$
$$T \rightarrow id * T \mid id \mid (E)$$



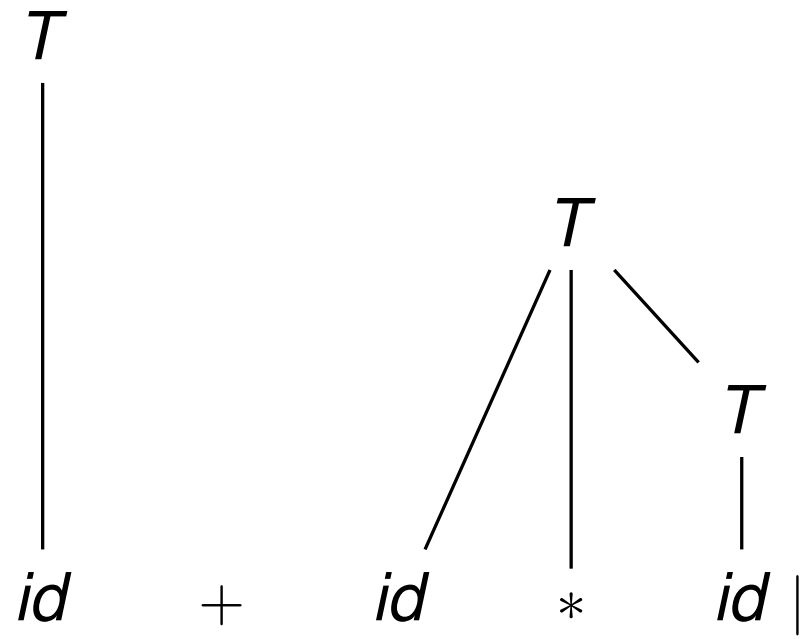
Exemplu

$$E \rightarrow T + E \mid T$$
$$T \rightarrow id * T \mid id \mid (E)$$



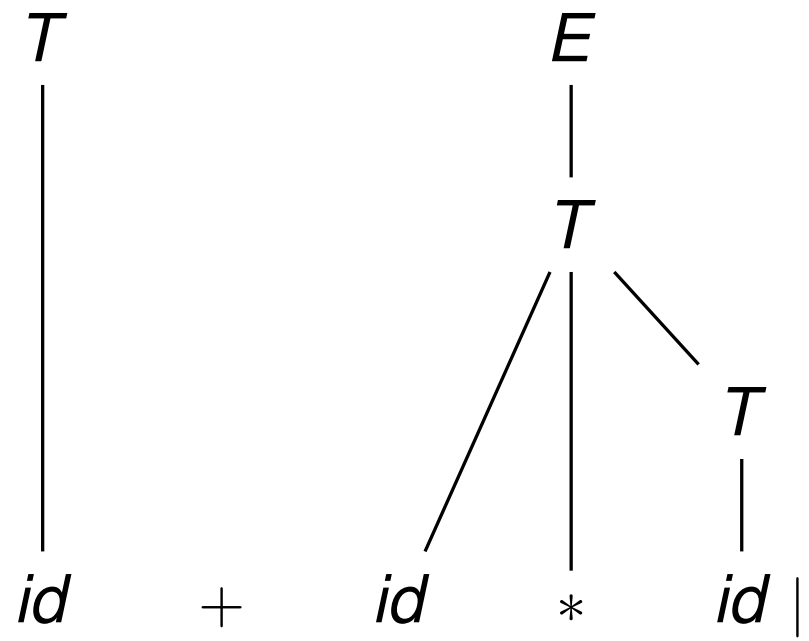
Exemplu

$$E \to T + E \mid T$$
$$T \to id * T \mid id \mid (E)$$



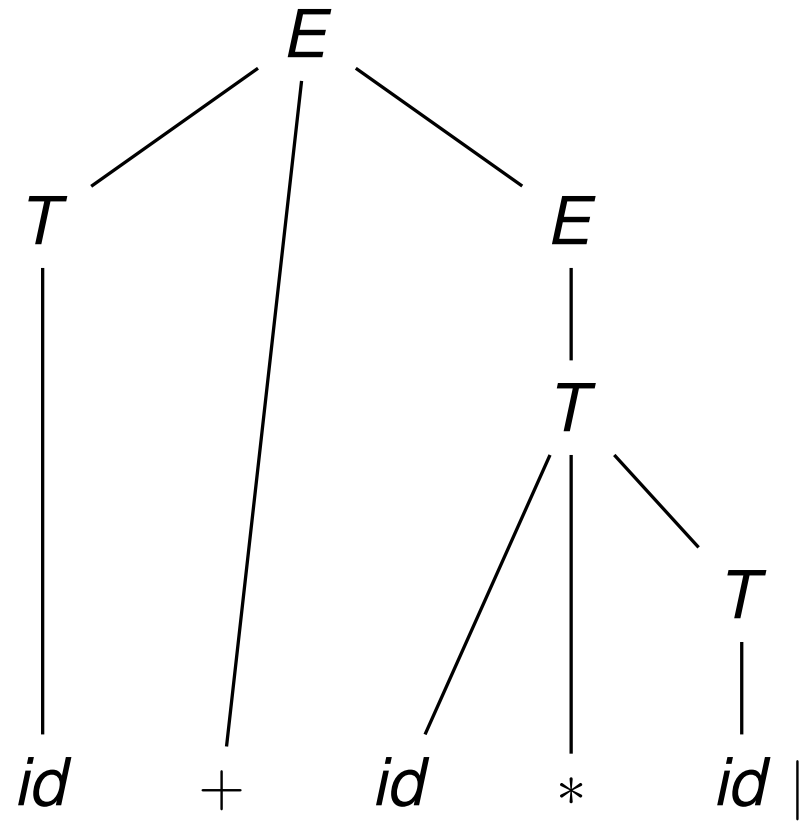
Exemplu

$$E \to T + E \mid T$$
$$T \to id * T \mid id \mid (E)$$



Exemplu

$$E \rightarrow T + E \mid T$$
$$T \rightarrow id * T \mid id \mid (E)$$



Implementare

- Posibilitatea reprezentării secvenței din stânga printr-o **stivă**, cu vârful reprezentat de „|”
- ***Shift***: adăugarea în stivă a unui terminal
- ***Reduce***: extragerea din stivă a zero sau mai multor simboluri corespunzătoare părții drepte a unei producții și adăugarea unui neterminal, aferent părții stângi

Conflicte

- În starea $id \mid + id * id$, cum alegem între *shift* și *reduce* cu $T \to id$?
- Conflict ***shift-reduce***: posibilitatea aplicării simultane a celor două acțiuni
- Conflict ***reduce-reduce***: posibilitatea aplicării acțiunii *reduce* conform cu cel puțin două producții (grav!)

Repere (*handles*)

- În starea $T + id \mid *id$, imposibilitatea reducerii la E în cazul aplicării *reduce* cu $T \to id$ (absența secvenței $T*$ din gramatică)
- Scop: aplicarea *reduce* doar în cazul în care reducerea poate continua până la simbolul de start
- **Reper** (*handle*): producția $X \to \beta$, alături de poziția de la dreapta lui α, astfel încât aplicarea ei constituie un pas al unei derivări **la dreapta**:

$$S \to^* \alpha X \gamma \to \alpha \beta \gamma$$

- Prezența reperelor exclusiv în vârful stivei, la stânga lui „|” (inducție după numărul de acțiuni *reduce*)

Recunoașterea reperelor

- Algoritmi eficienți, necunoscuți
- În practică, utilizarea **euristicilor**, care, pentru anumite gramatici restricționate, obțin întotdeauna răspunsul corect
- Ierarhie a GIC, de la general la particular:
  - Neambigue $\supset$
  - *LR*(k) (*left to right*, *rightmost derivation*) $\supset$
  - *LALR*(k) (*lookahead LR*) $\supset$
  - *SLR*(k) (*simple LR*)

Prefixe viabile

- **Prefix viabil** α, dacă $\alpha \mid \gamma$ poate fi o stare a analizorului, i.e. dacă $\alpha\gamma$ poate constitui o etapă intermediară a derivării **la dreapta**
- Întinderea prefixului viabil, cel mult până la limita din dreapta a unui **reper**
- Existența prefixelor viabile $\Rightarrow$ absența erorilor sintactice până în acel moment
- Mulțimea prefixelor viabile: limbaj regulat, i.e. prefixe viabile, acceptabile de către un AF

Organizarea stivei

- Exemplu: pentru șirul (id), $(E \mid)$ este o stare a analizorului
- (E, prefix al părții drepte a producției $T \to (E)$
- Cum precizăm observarea acestui prefix?
- Rescrierea regulii, în forma $T \to (E\bullet)$ (item)

Itemi

- **Item**: producție conținând „•” în orice poziție posibilă a părții drepte

- Pentru producția $T \to (E)$:

$$T \to \bullet(E)$$
$$T \to (\bullet E)$$
$$T \to (E\bullet)$$
$$T \to (E)\bullet$$

- Pentru producția $X \to \varepsilon$: $X \to \bullet$

Organizarea stivei (cont.)

- Posibilitatea existenței pe stivă a mai multor prefixe aferente părților drepte ale unor producții diferite:

$$\pi_1 \pi_2 \ldots \pi_{n-1} \pi_n \mid \gamma$$

- π_i = prefixul părții drepte a producției $X_i \to \beta_i$, i.e. $\beta_i = \pi_i \ldots$

- X_i, continuare a prefixului π_{i-1} a părții drepte a producției $X_{i-1} \to \beta_{i-1}$, i.e. $X_{i-1} \to \pi_{i-1} X_i \ldots$

Exemplu

- Gramatica:

$$E \to T + E \mid T$$
$$T \to id * T \mid id \mid (E)$$

- Șirul:

$$T + \quad id * \mid id$$

- Prefixele și itemii:

Exemplu

- Gramatica:

$$E \to T + E \mid T$$
$$T \to id * T \mid id \mid (E)$$

- Șirul:

$$T + \quad id * \mid id$$

- Prefixele și itemii:

$$T \to id * \bullet T$$

Exemplu

- Gramatica:

$$E \to T + E \mid T$$
$$T \to id * T \mid id \mid (E)$$

- Șirul:

$$T + \varepsilon id * \mid id$$

- Prefixele și itemii:

$$T \to id * \bullet T$$
$$E \to \bullet T$$

Exemplu

- Gramatica:

$$E \to T + E \mid T$$
$$T \to id * T \mid id \mid (E)$$

- Șirul:

$$T + \quad id * \mid id$$

- Prefixele și itemii:

$$T \to id * \bullet T$$
$$E \to \bullet T$$
$$E \to T + \bullet E$$

Exemplu

- Gramatica:

$$E \to T + E \mid T$$
$$T \to id * T \mid id \mid (E)$$

- Șirul:

$$T + \quad id * \mid id$$

- Prefixele și itemii:

$$T \to id * \bullet T$$
$$E \to \bullet T$$
$$E \to T + \bullet E$$

Recunoașterea prefixelor viabile cu AFN

1. Creează noul **simbol de start** S' și adaugă **producția** $S' \to S$ la gramatică
2. **Stările** AFN: itemii gramaticii
3. Pentru fiecare item $Y \to \alpha \bullet X\gamma$, cu X terminal sau neterminal, adaugă **tranziția**

$$Y \to \alpha \bullet X\gamma \quad \to^X \quad Y \to \alpha X \bullet \gamma$$

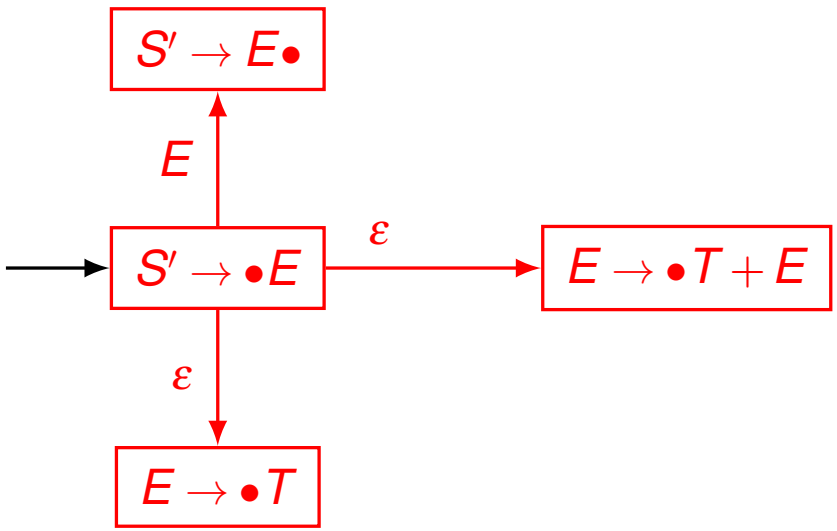
4. Pentru fiecare item $Y \to \alpha \bullet X\gamma$, cu X neterminal, și fiecare producție $X \to \beta$, adaugă **tranziția**

$$Y \to \alpha \bullet X\gamma \quad \to^\varepsilon \quad X \to \bullet\beta$$

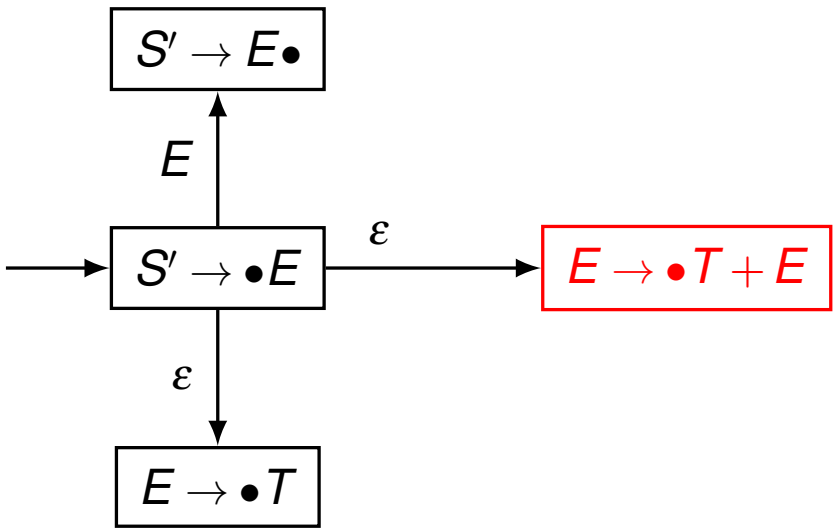
5. Starea **inițială**: $S' \to \bullet S$
6. Stările **finale**: toate stările!

AFN

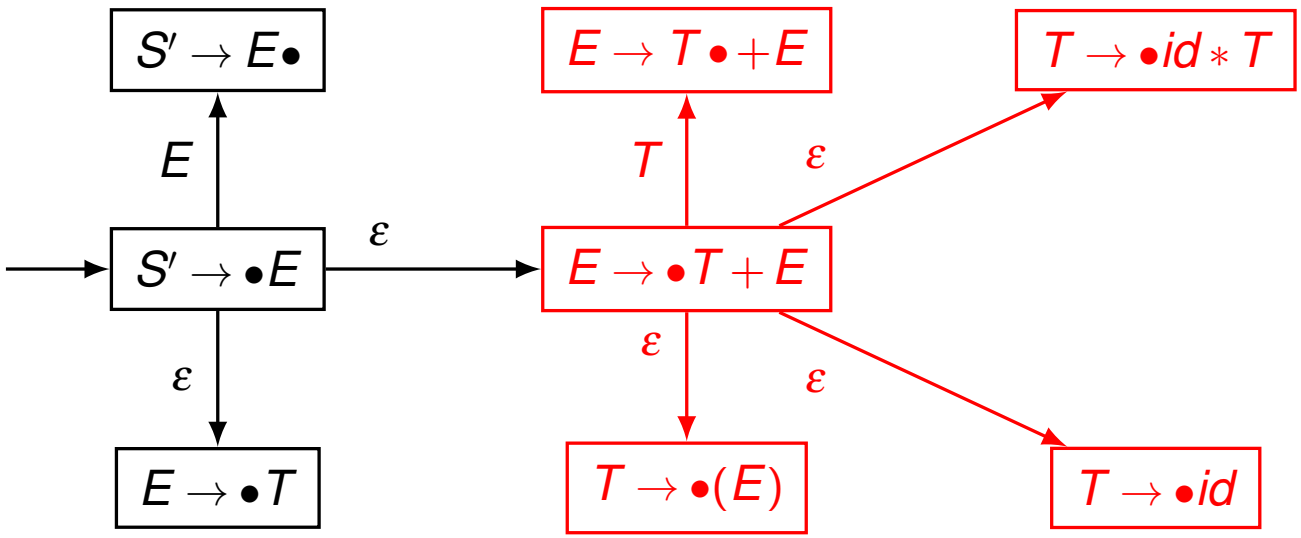
AFN


$S' \to E\bullet$
E
$S' \to \bullet E$
ε
$E \to \bullet T + E$
ε
$E \to \bullet T$

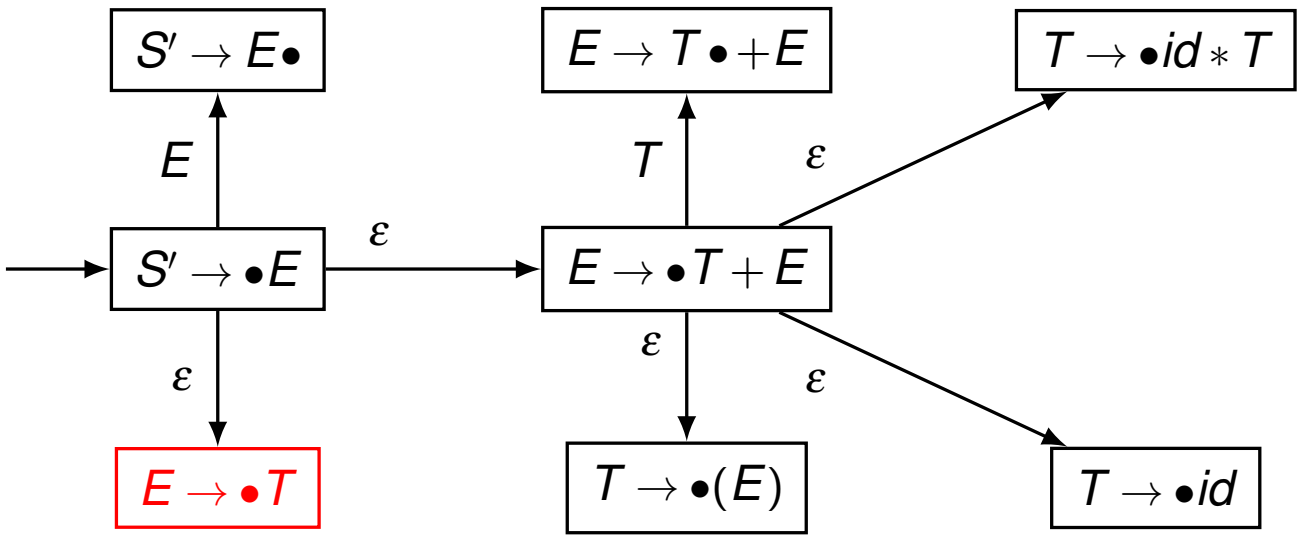
AFN


$S' \to E\bullet$
E
$S' \to \bullet E$
ε
$E \to \bullet T + E$
ε
$E \to \bullet T$

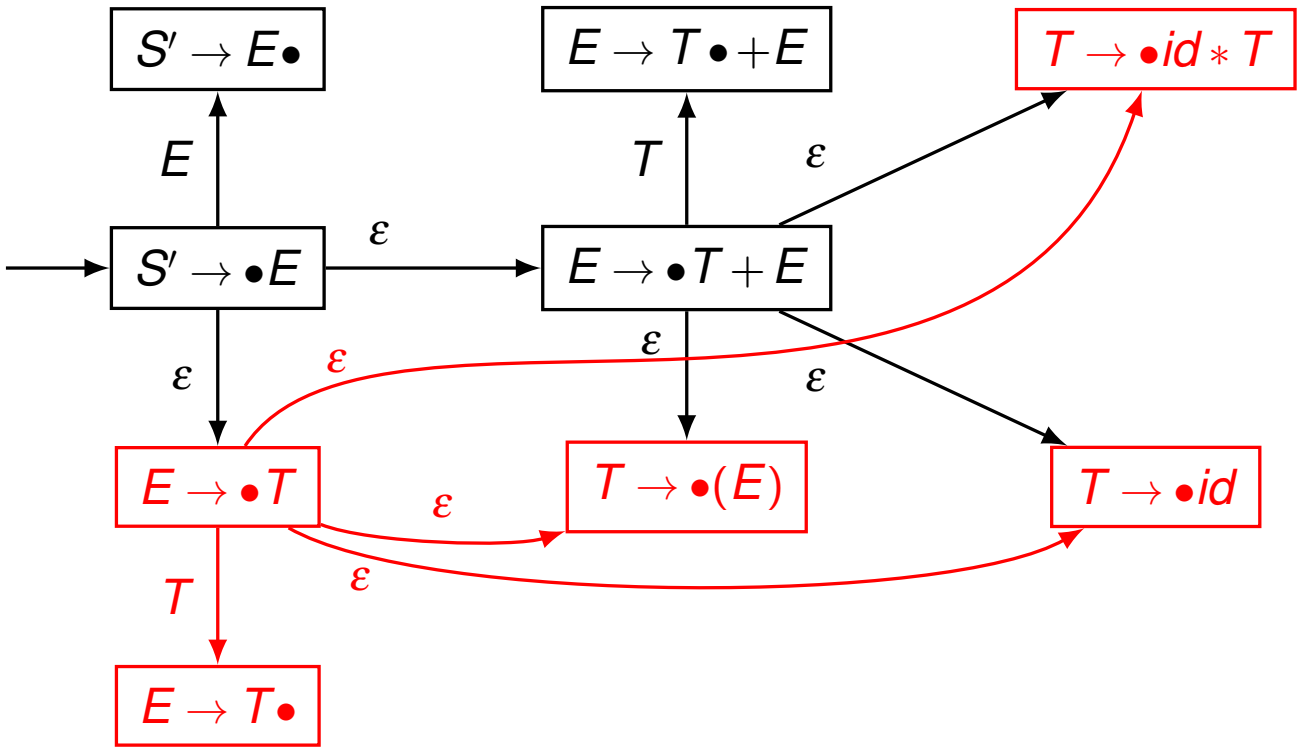
AFN


$S' \to E\bullet$
E
$S' \to \bullet E$
ε
$E \to \bullet T$
ε
$E \to T \bullet + E$
T
ε
$E \to \bullet T + E$
ε
ε
$T \to \bullet(E)$
$T \to \bullet id * T$
$T \to \bullet id$

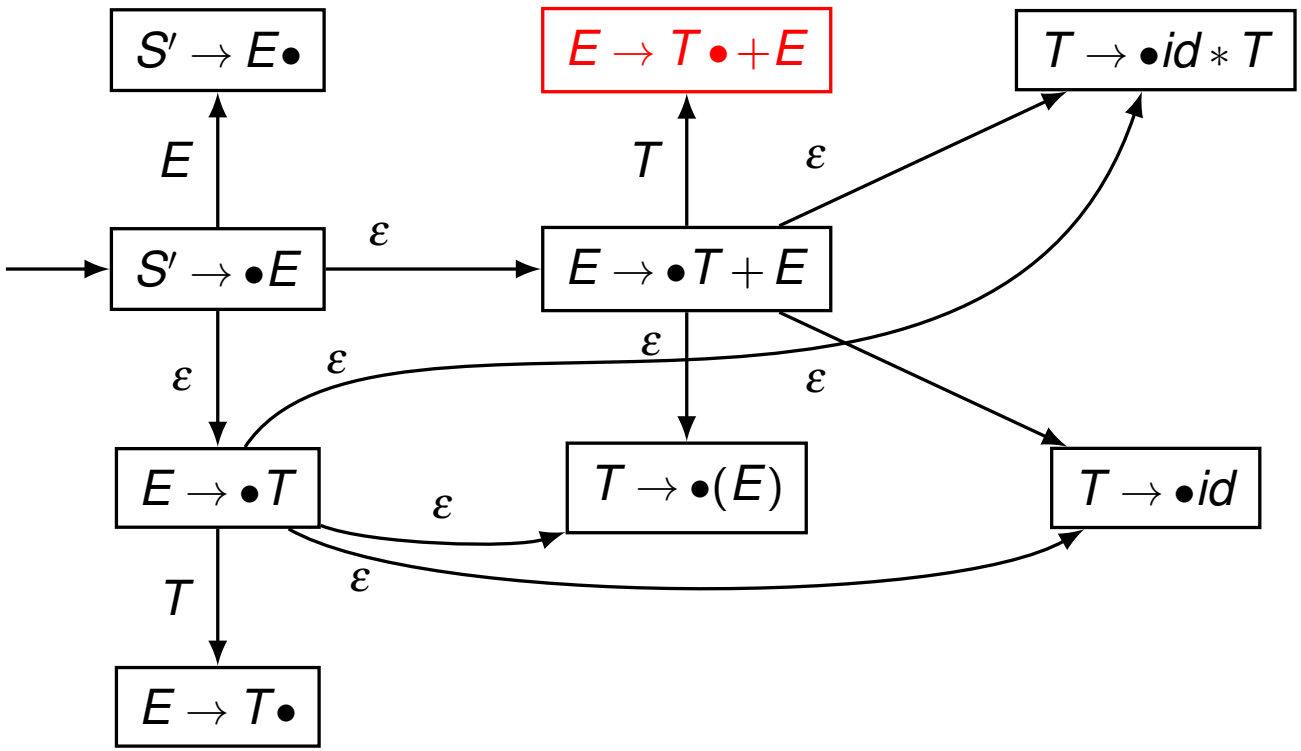
AFN


$S' \to E\bullet$
$E \to T \bullet + E$
$T \to \bullet id * T$
E
T
ε
$S' \to \bullet E$
ε
$E \to \bullet T + E$
ε
ε
ε
$E \to \bullet T$
$T \to \bullet (E)$
$T \to \bullet id$

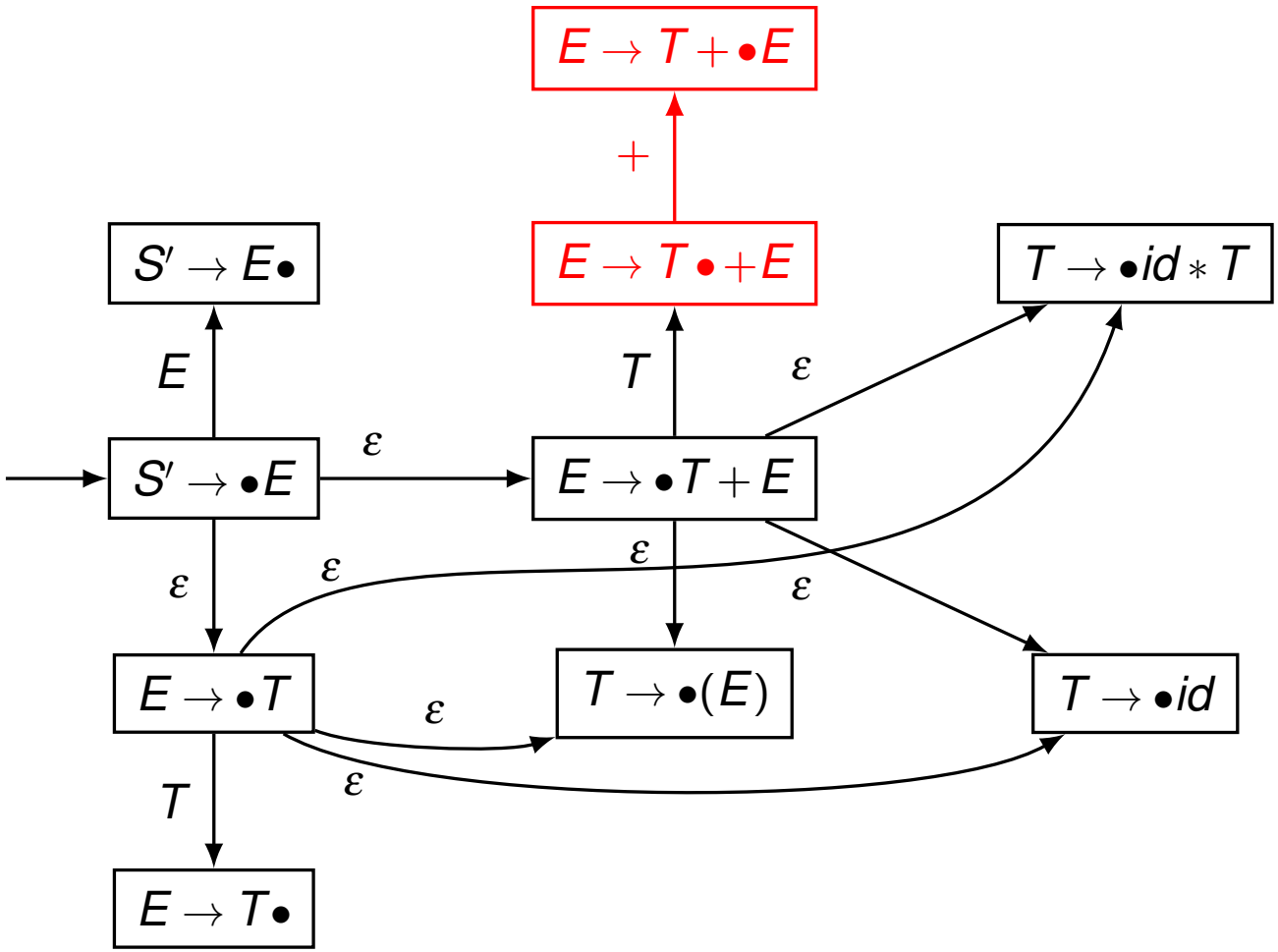
AFN


$S' \to E\bullet$
$E \to T\bullet + E$
$T \to \bullet id * T$
E
T
ε
$S' \to \bullet E$
ε
$E \to \bullet T + E$
ε
ε
ε
ε
$E \to \bullet T$
ε
$T \to \bullet(E)$
$T \to \bullet id$
ε
T
$E \to T\bullet$

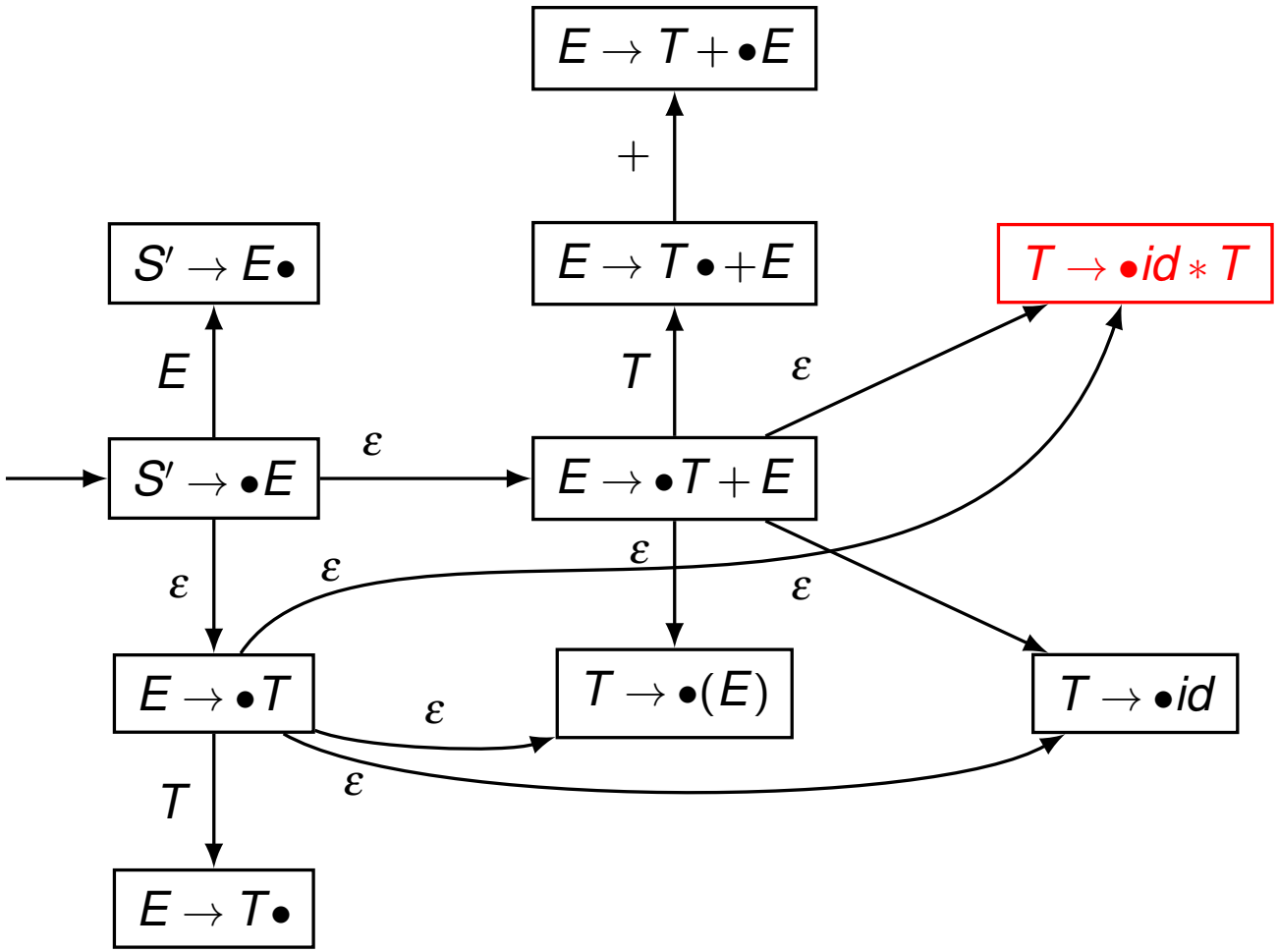
AFN


$S' \to E\bullet$
$E \to T \bullet + E$
$T \to \bullet id * T$
E
T
ε
$S' \to \bullet E$
ε
$E \to \bullet T + E$
ε
ε
ε
ε
$E \to \bullet T$
$T \to \bullet (E)$
$T \to \bullet id$
ε
ε
T
$E \to T\bullet$

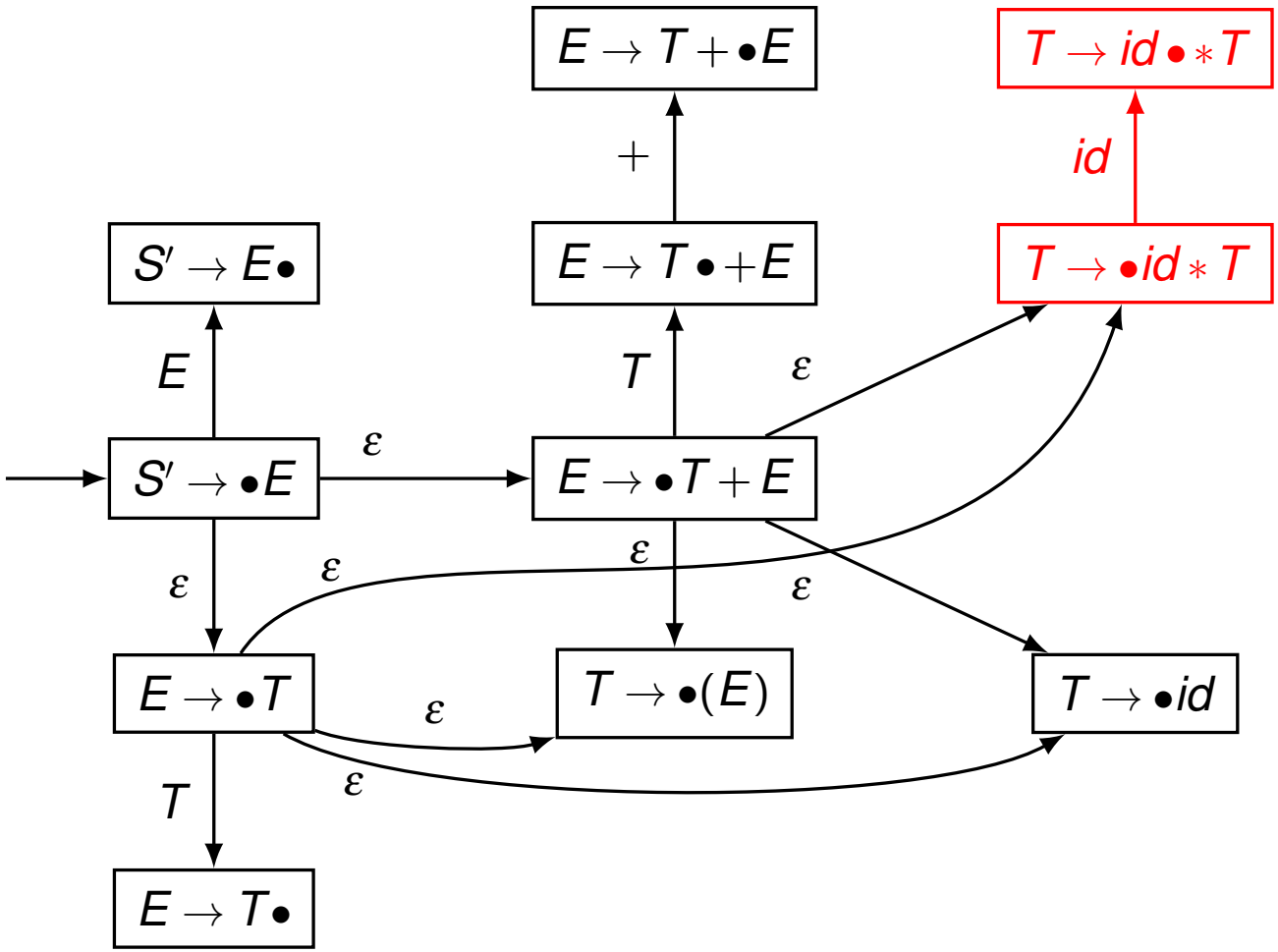
AFN


$E \to T + \bullet E$
+
$E \to T \bullet + E$
$S' \to E\bullet$
$T \to \bullet id * T$
E
T
ε
ε
$S' \to \bullet E$
$E \to \bullet T + E$
ε
ε
ε
$E \to \bullet T$
$T \to \bullet(E)$
$T \to \bullet id$
ε
ε
T
$E \to T\bullet$

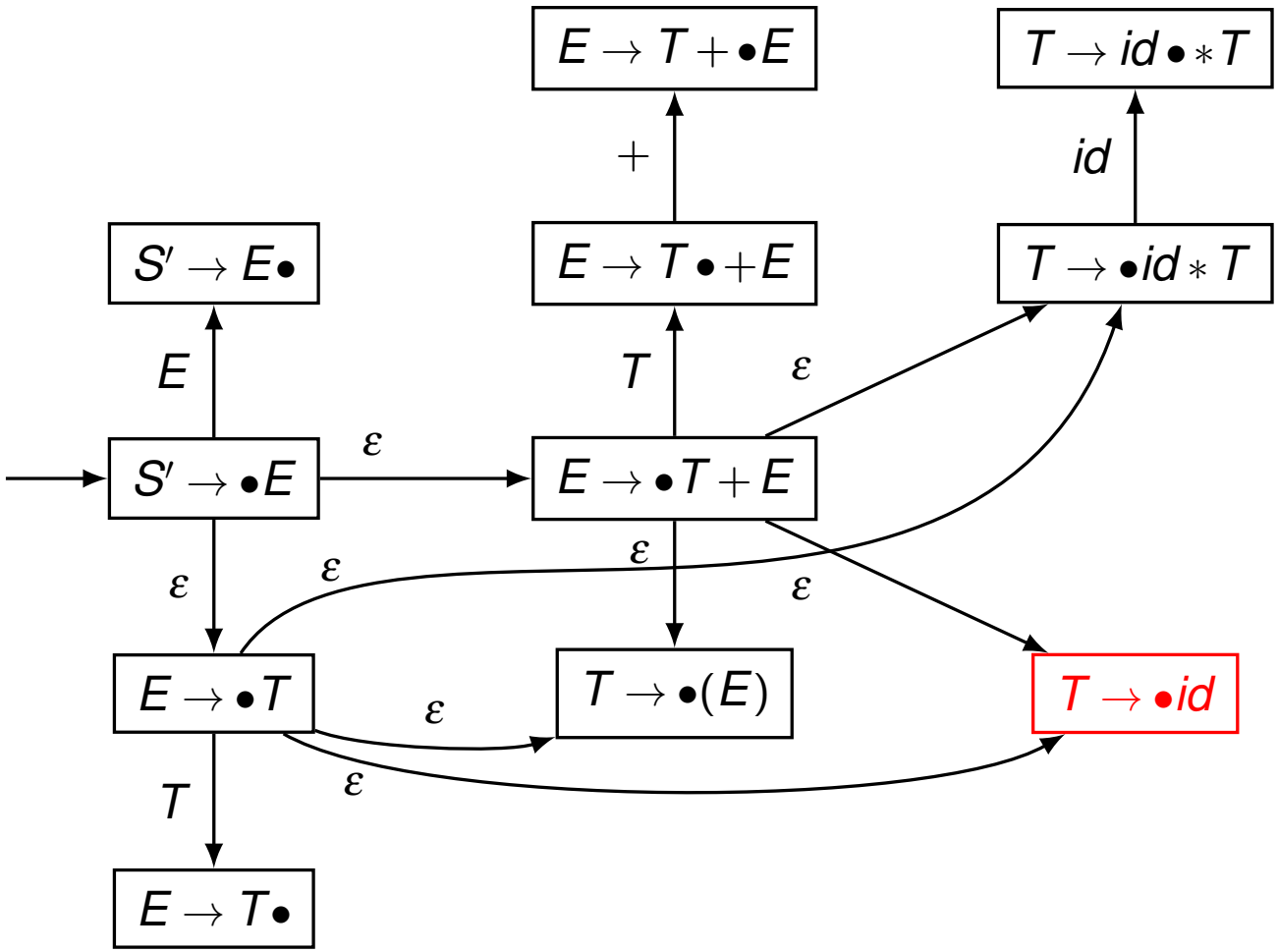
AFN


$E \to T + \bullet E$
$+$
$S' \to E\bullet$
$E \to T \bullet + E$
$T \to \bullet id * T$
E
T
ε
$S' \to \bullet E$
ε
$E \to \bullet T + E$
ε
ε
ε
ε
$E \to \bullet T$
$T \to \bullet(E)$
$T \to \bullet id$
ε
ε
T
$E \to T\bullet$

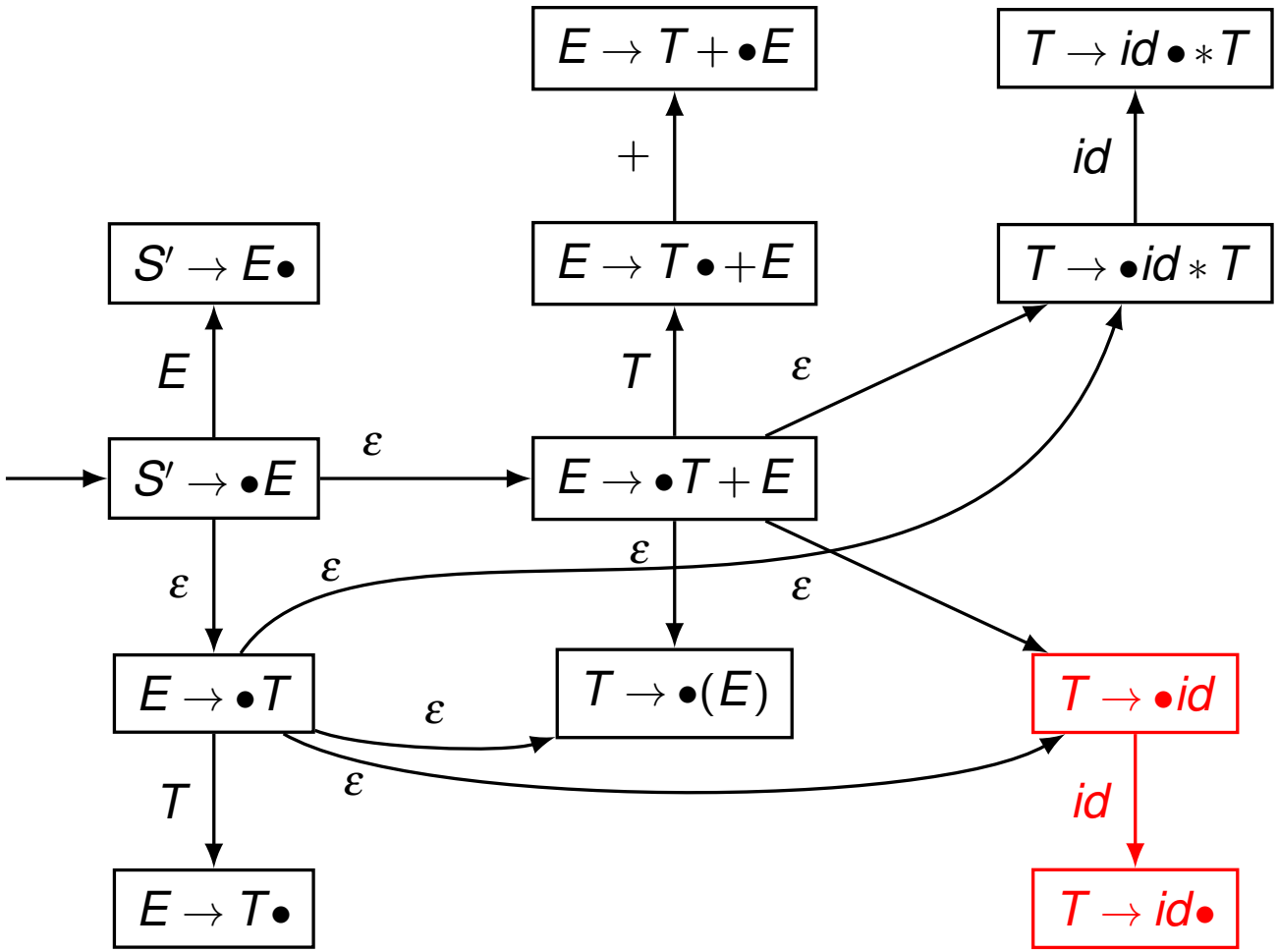
AFN


$E \to T + \bullet E$
$T \to id \bullet * T$
$+$
id
$S' \to E\bullet$
$E \to T \bullet + E$
$T \to \bullet id * T$
E
T
ε
$S' \to \bullet E$
ε
$E \to \bullet T + E$
ε
ε
ε
ε
$E \to \bullet T$
$T \to \bullet (E)$
$T \to \bullet id$
ε
ε
T
$E \to T\bullet$

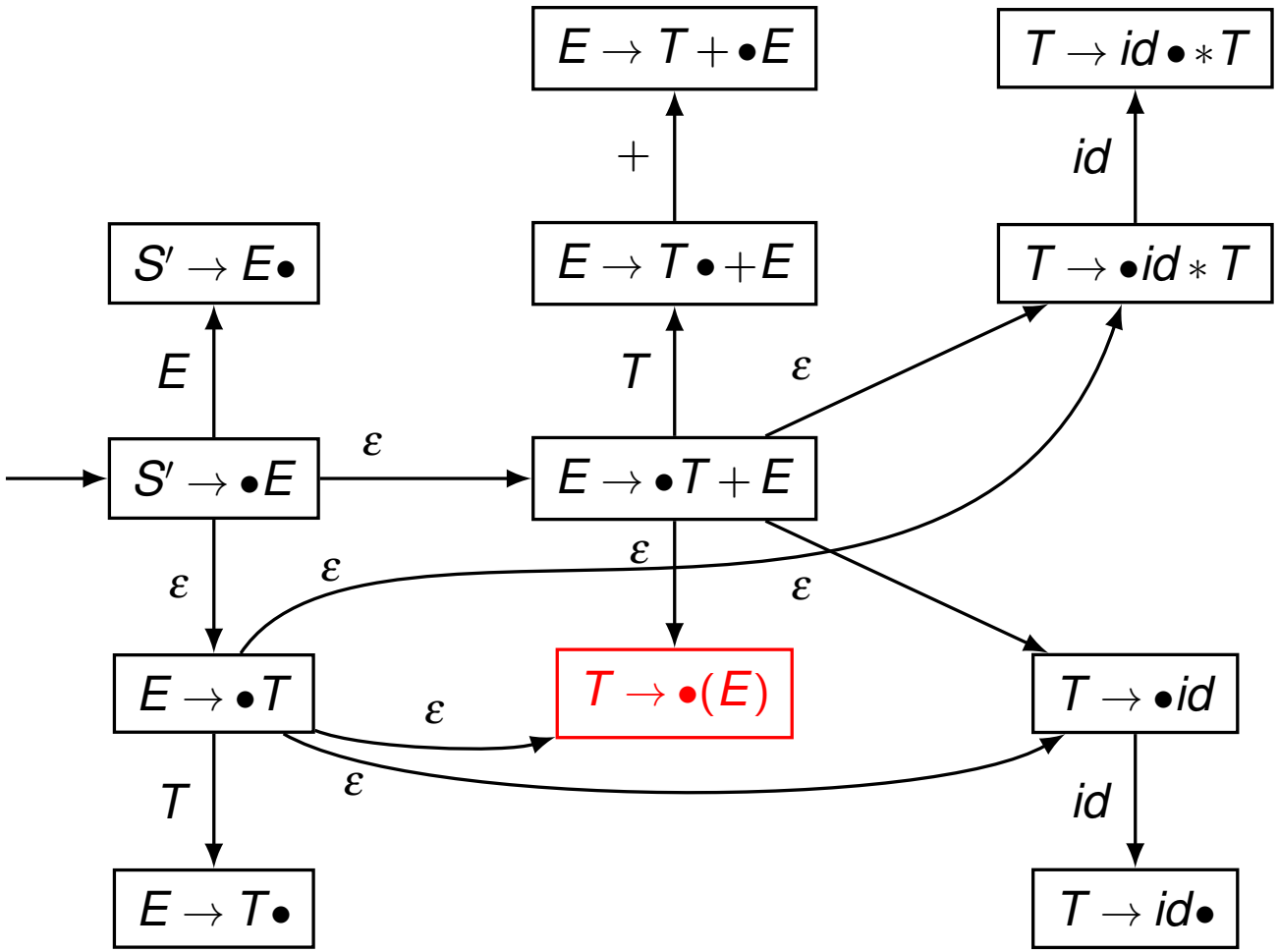
AFN


$E \to T + \bullet E$
$T \to id \bullet * T$
$+$
id
$S' \to E\bullet$
$E \to T \bullet + E$
$T \to \bullet id * T$
E
T
ε
$S' \to \bullet E$
ε
$E \to \bullet T + E$
ε
ε
ε
ε
$E \to \bullet T$
$T \to \bullet (E)$
$T \to \bullet id$
ε
ε
T
$E \to T\bullet$

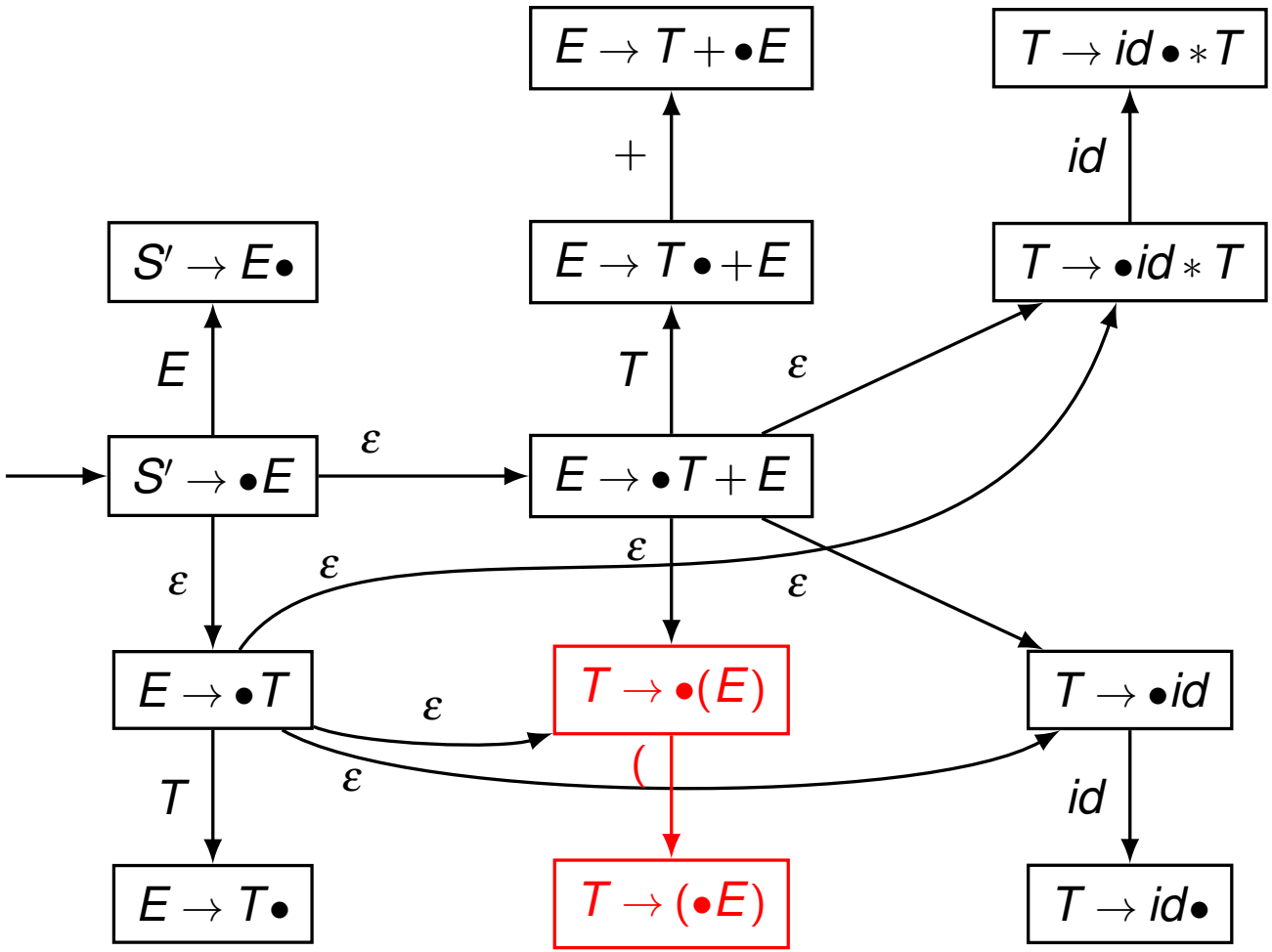
AFN


$E \to T + \bullet E$
$T \to id \bullet * T$
$+$
id
$S' \to E\bullet$
$E \to T \bullet + E$
$T \to \bullet id * T$
E
T
ε
$S' \to \bullet E$
ε
$E \to \bullet T + E$
ε
ε
ε
ε
$E \to \bullet T$
$T \to \bullet(E)$
$T \to \bullet id$
ε
ε
T
id
$E \to T\bullet$
$T \to id\bullet$

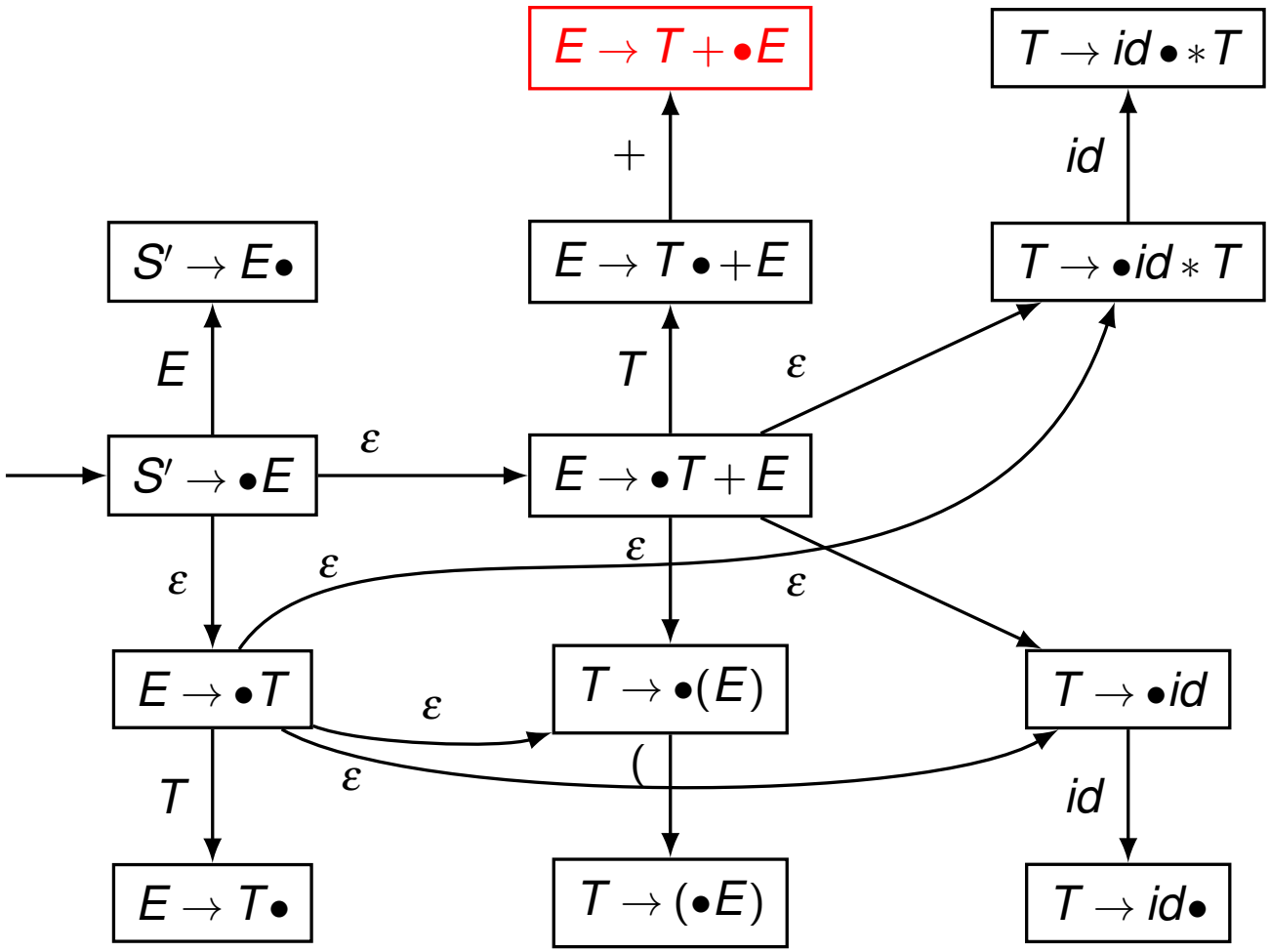
AFN


$E \to T + \bullet E$
$T \to id \bullet * T$
$+$
id
$S' \to E\bullet$
$E \to T \bullet + E$
$T \to \bullet id * T$
E
T
ε
$S' \to \bullet E$
ε
$E \to \bullet T + E$
ε
ε
ε
ε
$E \to \bullet T$
$T \to \bullet(E)$
$T \to \bullet id$
ε
ε
T
id
$E \to T\bullet$
$T \to id\bullet$

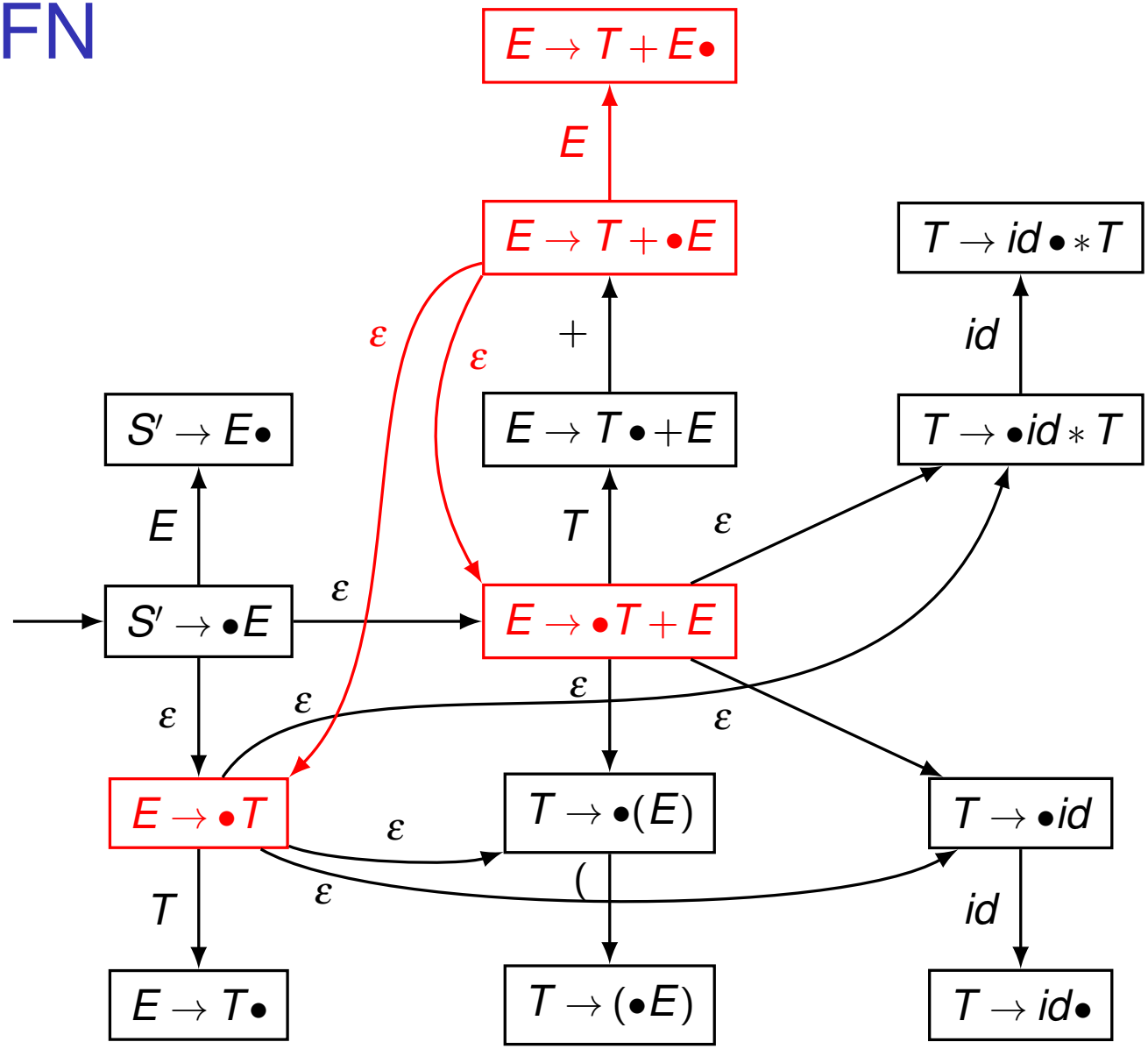
AFN


$E \to T + \bullet E$
$T \to id \bullet * T$
$+$
id
$S' \to E\bullet$
$E \to T \bullet + E$
$T \to \bullet id * T$
E
T
ε
$S' \to \bullet E$
ε
$E \to \bullet T + E$
ε
ε
ε
ε
$E \to \bullet T$
ε
$T \to \bullet (E)$
$T \to \bullet id$
$($
ε
T
id
$E \to T\bullet$
$T \to (\bullet E)$
$T \to id\bullet$

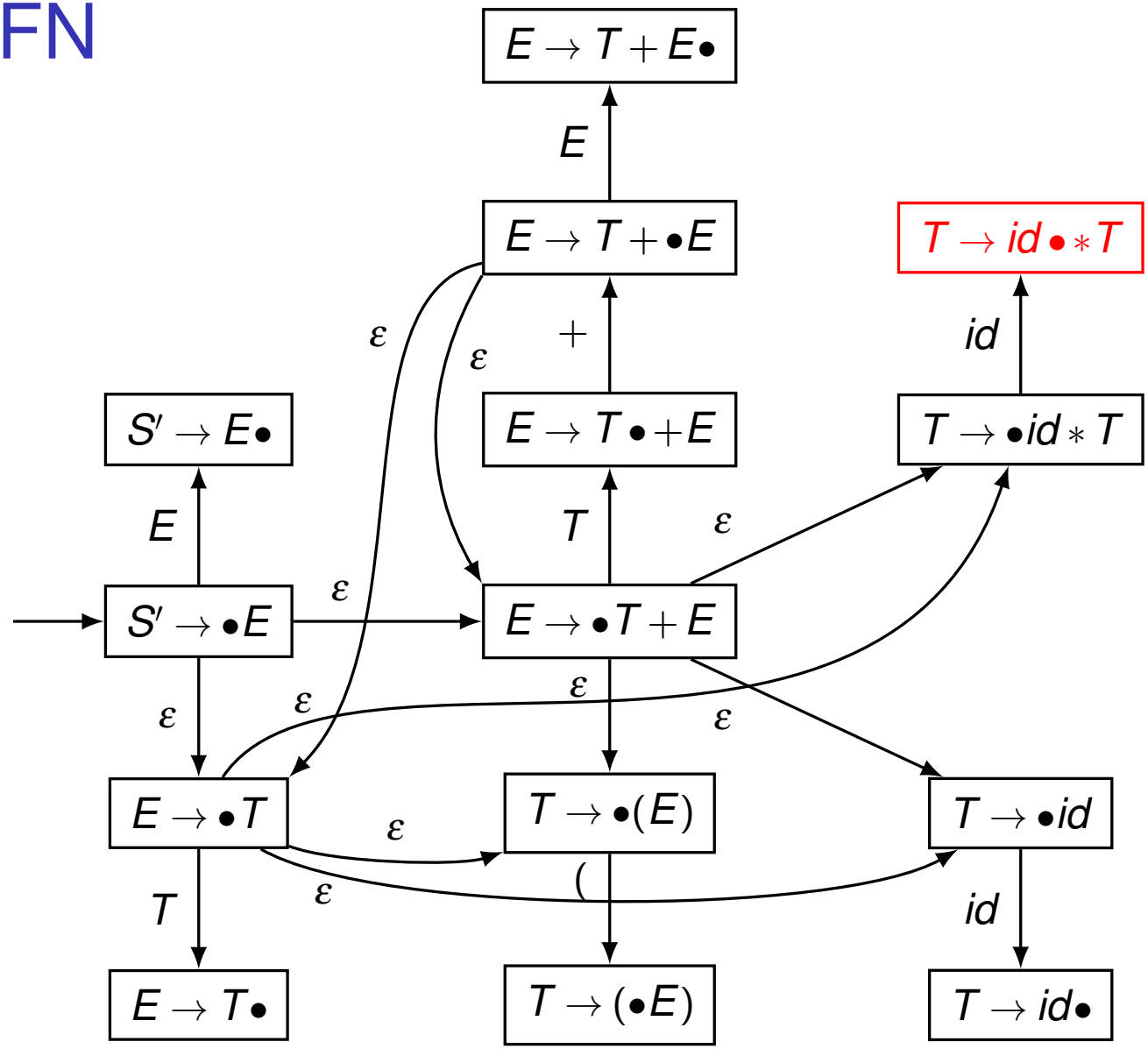
AFN


$E \to T + \bullet E$
$T \to id \bullet * T$
$+$
id
$S' \to E\bullet$
$E \to T \bullet + E$
$T \to \bullet id * T$
E
T
ε
$S' \to \bullet E$
ε
$E \to \bullet T + E$
ε
ε
ε
ε
$E \to \bullet T$
ε
$T \to \bullet (E)$
$T \to \bullet id$
$($
ε
T
id
$E \to T\bullet$
$T \to (\bullet E)$
$T \to id\bullet$

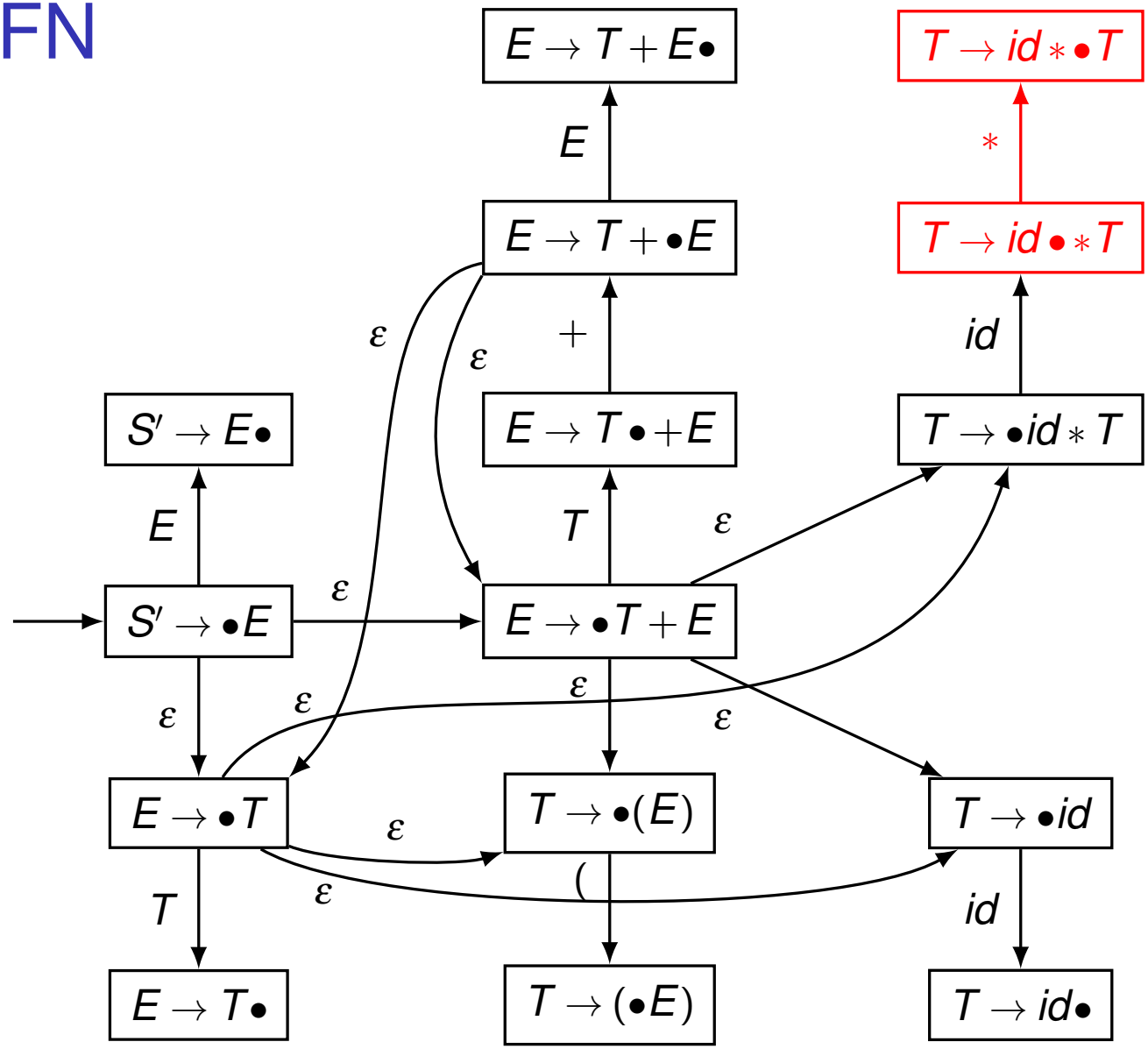
AFN


$E \to T + E\bullet$
E
$E \to T + \bullet E$
$T \to id \bullet * T$
ε
ε
$+$
id
$S' \to E\bullet$
$E \to T \bullet + E$
$T \to \bullet id * T$
E
T
ε
$S' \to \bullet E$
ε
$E \to \bullet T + E$
ε
ε
ε
ε
$E \to \bullet T$
$T \to \bullet (E)$
$T \to \bullet id$
ε
$($
T
ε
id
$E \to T\bullet$
$T \to (\bullet E)$
$T \to id\bullet$

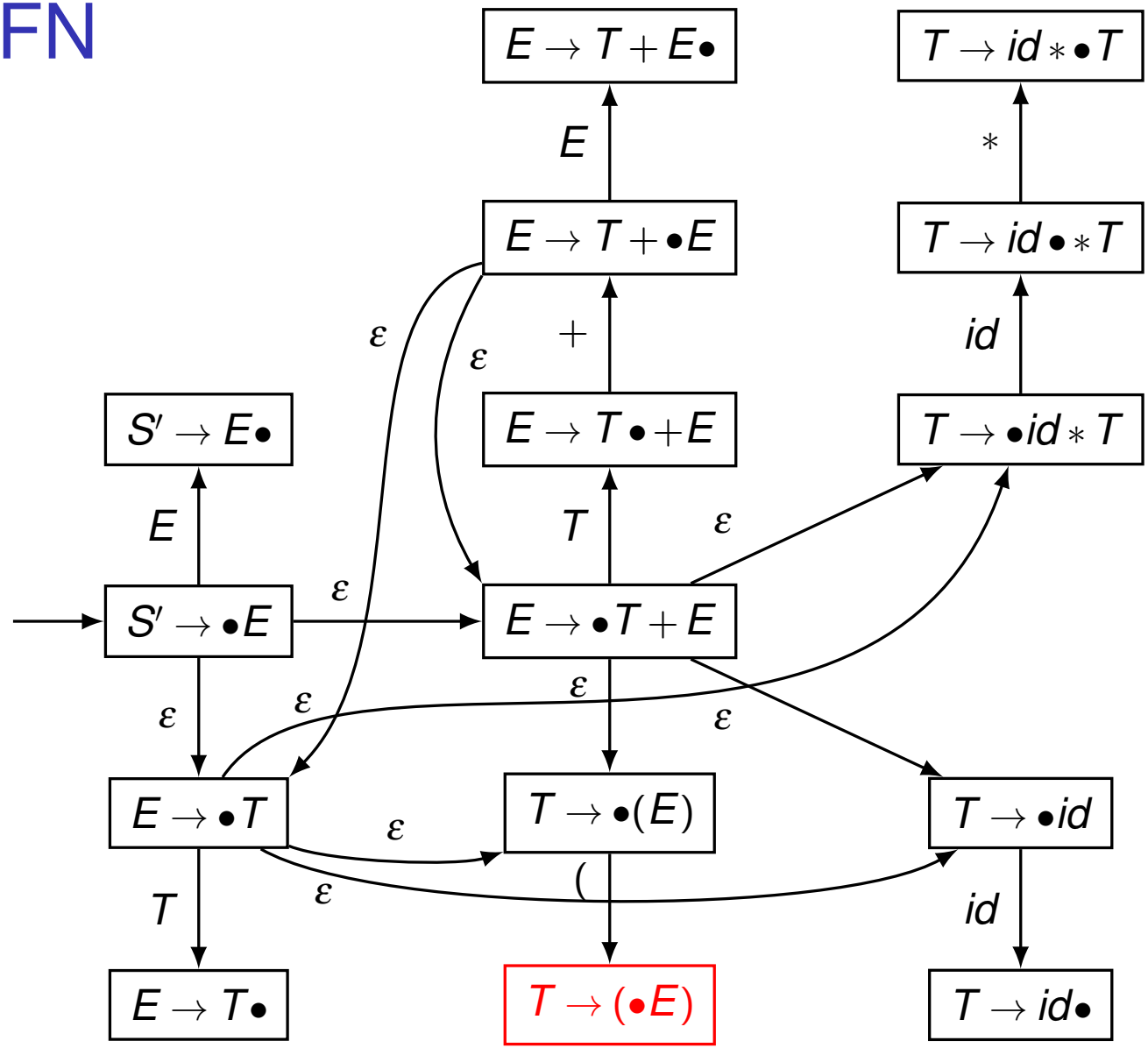
AFN


$E \to T + E\bullet$
E
$E \to T + \bullet E$
$T \to id \bullet * T$
ε
ε
$+$
id
$S' \to E\bullet$
$E \to T \bullet + E$
$T \to \bullet id * T$
E
T
ε
$S' \to \bullet E$
ε
$E \to \bullet T + E$
ε
ε
ε
ε
$E \to \bullet T$
$T \to \bullet (E)$
$T \to \bullet id$
ε
$($
T
ε
id
$E \to T\bullet$
$T \to (\bullet E)$
$T \to id\bullet$

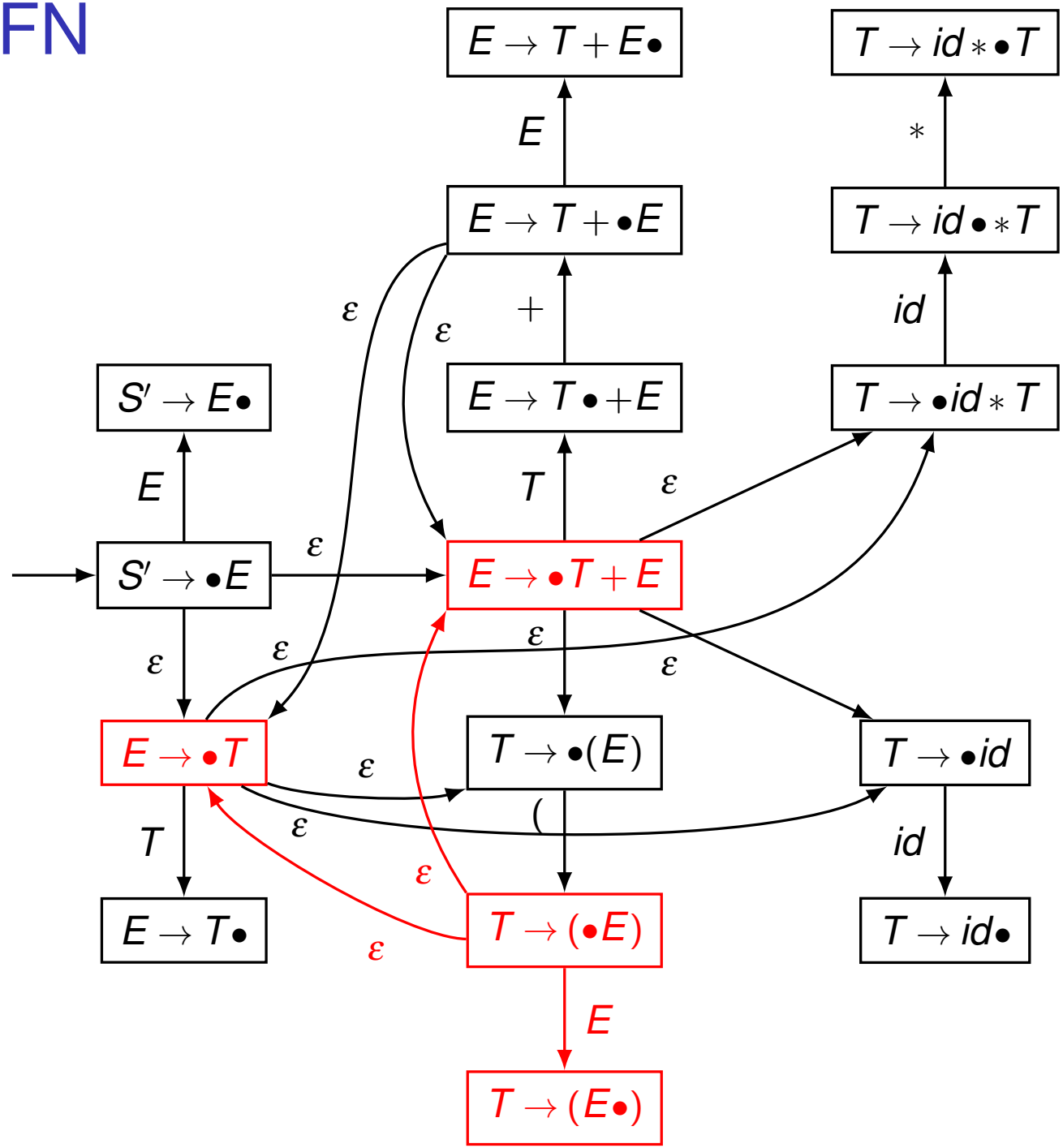
AFN


$E \to T + E\bullet$
E
$E \to T + \bullet E$
$+$
$E \to T \bullet + E$
T
$E \to \bullet T + E$
$T \to id * \bullet T$
$*$
$T \to id \bullet * T$
id
$T \to \bullet id * T$
$S' \to E\bullet$
E
$S' \to \bullet E$
$E \to \bullet T$
$T \to \bullet(E)$
$T \to \bullet id$
T
$E \to T\bullet$
$($
$T \to (\bullet E)$
id
$T \to id\bullet$
ε

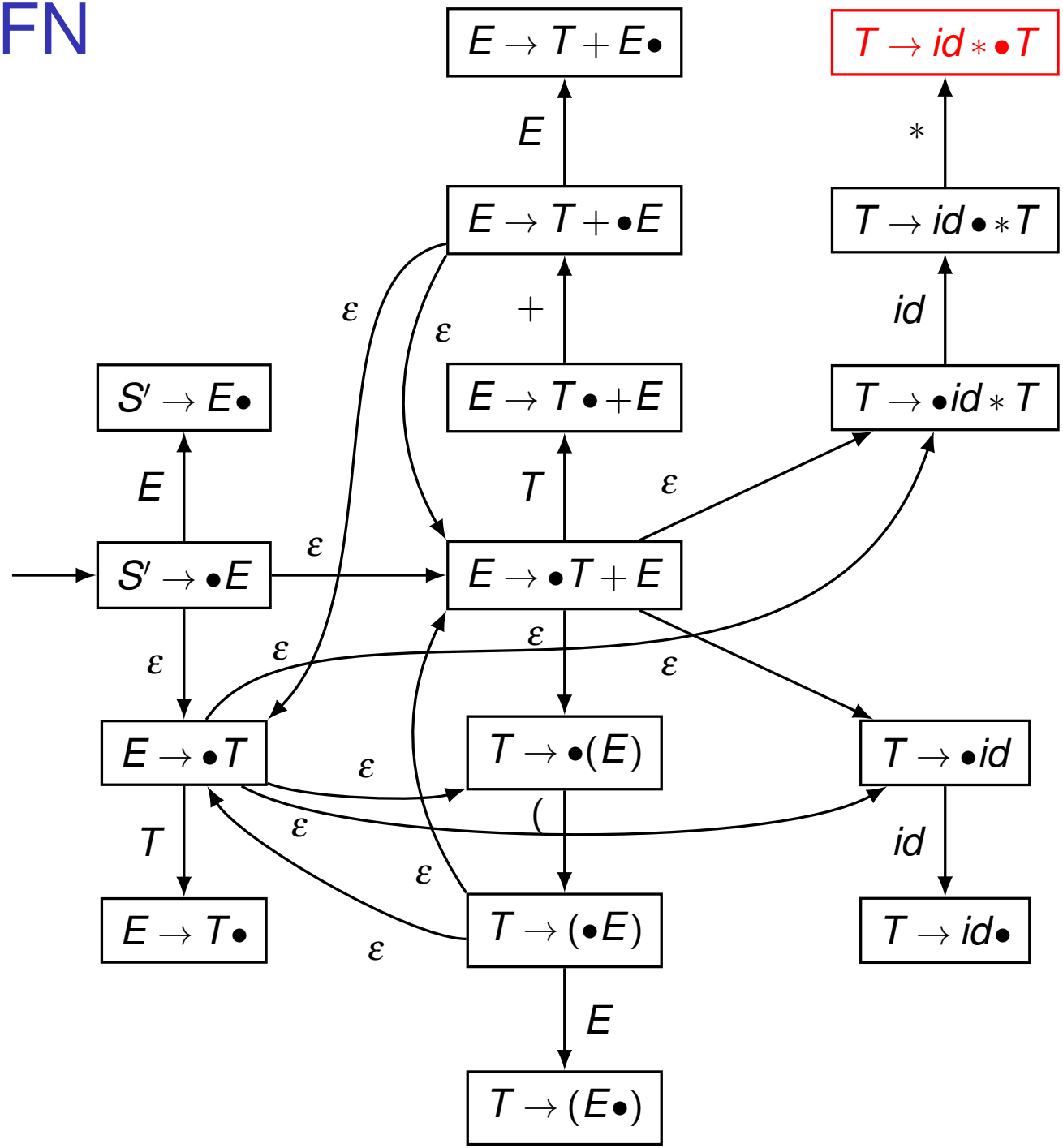
AFN


$E \to T + E\bullet$
$T \to id * \bullet T$
E
$*$
$E \to T + \bullet E$
$T \to id \bullet * T$
ε
ε
$+$
id
$S' \to E\bullet$
$E \to T \bullet + E$
$T \to \bullet id * T$
E
T
ε
$S' \to \bullet E$
ε
$E \to \bullet T + E$
ε
ε
ε
ε
$E \to \bullet T$
ε
$T \to \bullet (E)$
$T \to \bullet id$
$($
ε
T
id
$E \to T\bullet$
$T \to (\bullet E)$
$T \to id\bullet$

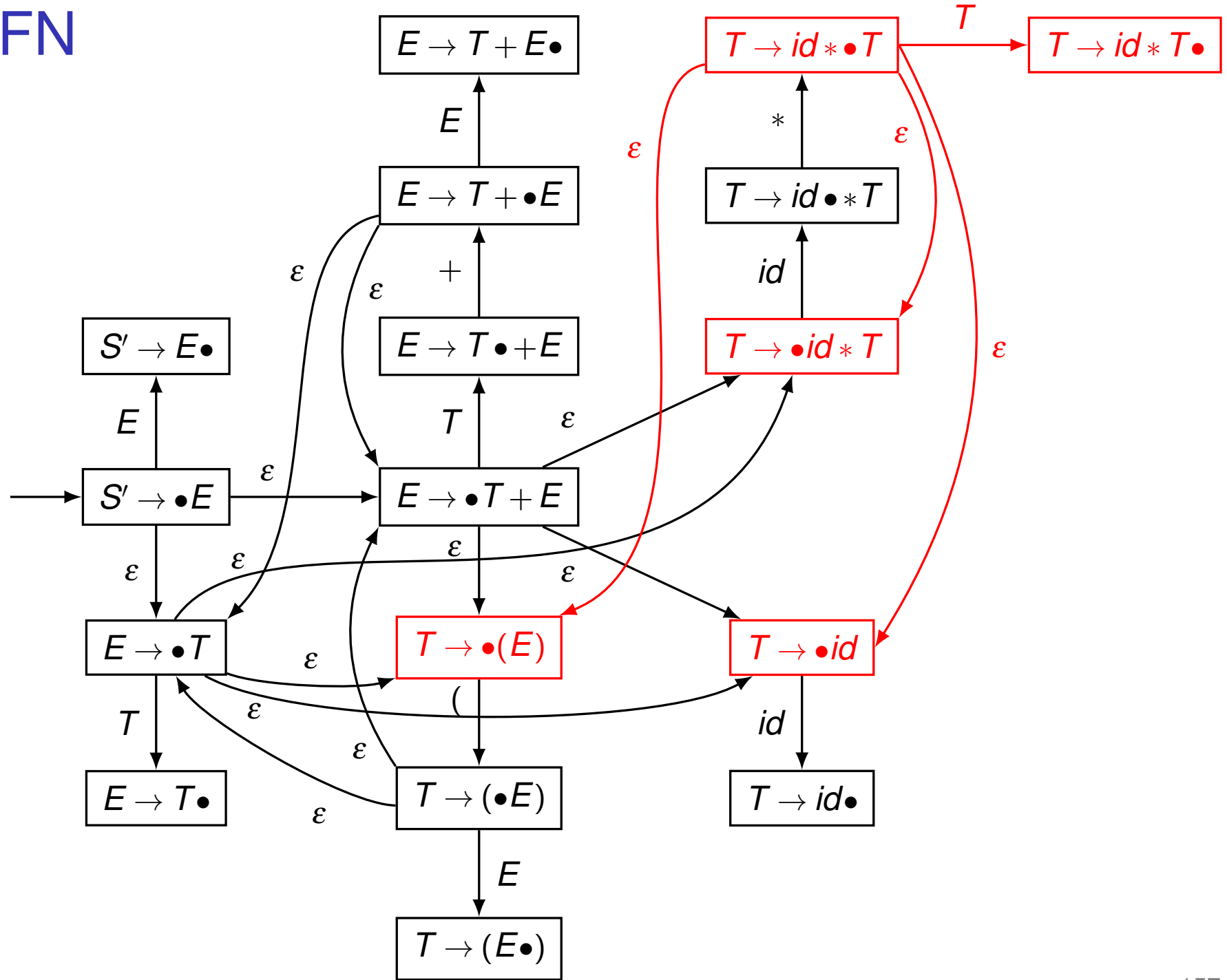
AFN


$E \to T + E\bullet$
$T \to id * \bullet T$
E
$*$
$E \to T + \bullet E$
$T \to id \bullet * T$
ε
ε
$+$
id
$S' \to E\bullet$
$E \to T \bullet + E$
$T \to \bullet id * T$
E
T
ε
$S' \to \bullet E$
ε
$E \to \bullet T + E$
ε
ε
ε
ε
$E \to \bullet T$
$T \to \bullet (E)$
$T \to \bullet id$
ε
$($
T
ε
id
ε
$E \to T\bullet$
$T \to (\bullet E)$
$T \to id\bullet$
ε
E
$T \to (E\bullet)$

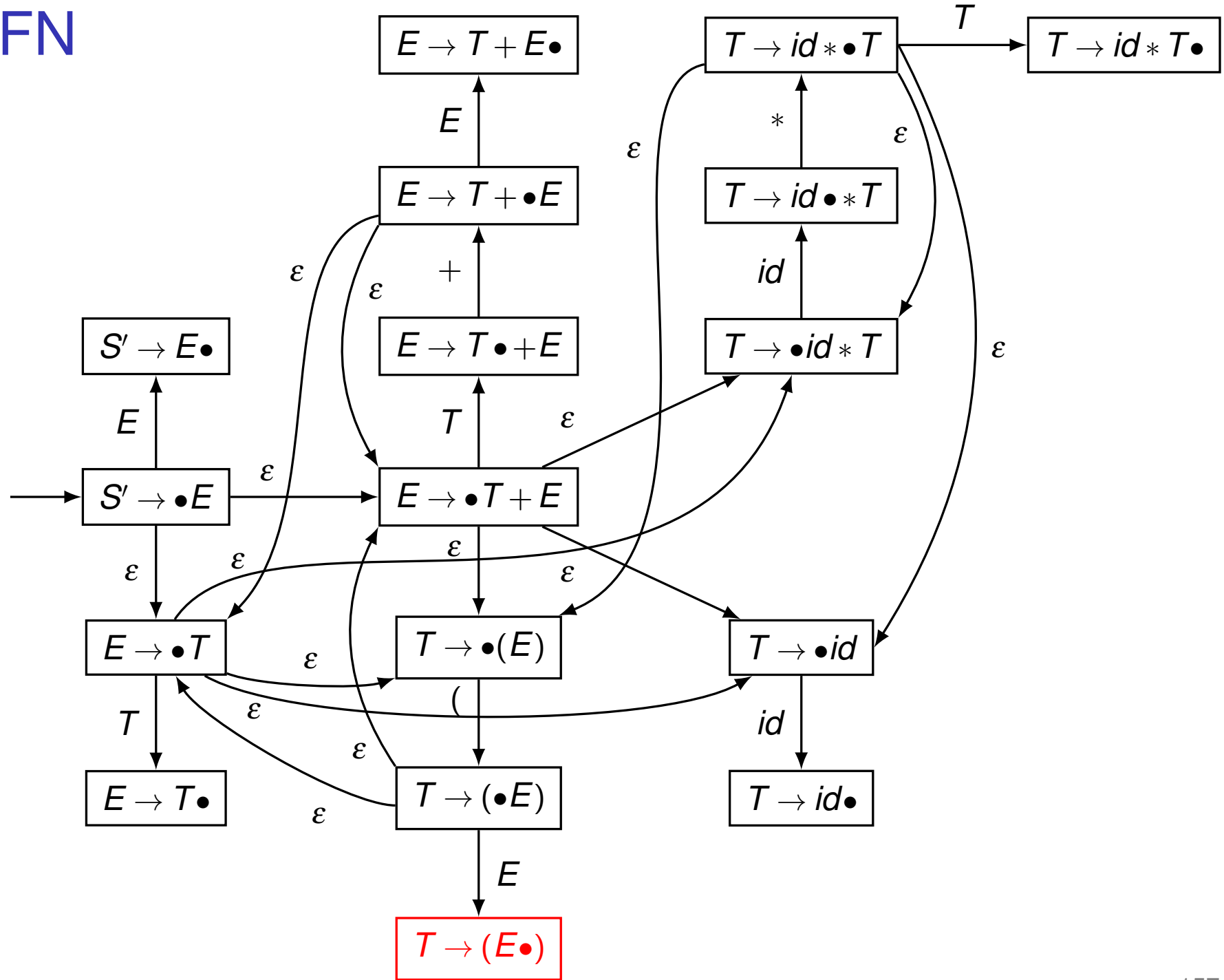
AFN


$E \to T + E\bullet$
$T \to id * \bullet T$
E
$*$
$E \to T + \bullet E$
$T \to id \bullet * T$
ε
ε
$+$
id
$S' \to E\bullet$
$E \to T \bullet + E$
$T \to \bullet id * T$
E
T
ε
$S' \to \bullet E$
ε
$E \to \bullet T + E$
ε
ε
ε
ε
$E \to \bullet T$
ε
$T \to \bullet (E)$
$T \to \bullet id$
$($
T
ε
id
ε
$E \to T\bullet$
ε
$T \to (\bullet E)$
$T \to id\bullet$
E
$T \to (E\bullet)$

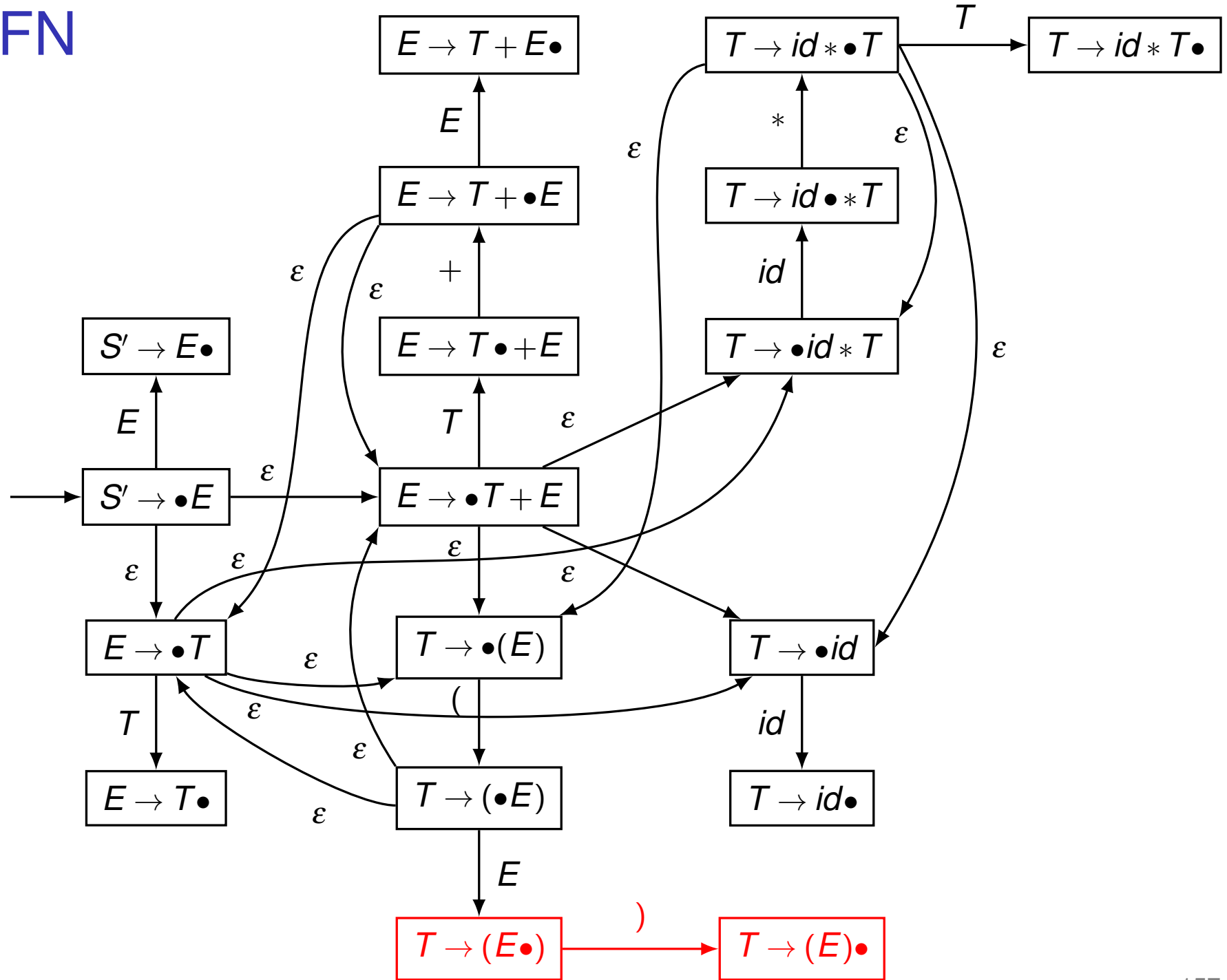
AFN


$E \to T + E\bullet$
$T \to id * \bullet T$
T
$T \to id * T\bullet$
E
$*$
ε
ε
$E \to T + \bullet E$
$T \to id \bullet * T$
ε
ε
$+$
id
$S' \to E\bullet$
$E \to T \bullet + E$
$T \to \bullet id * T$
ε
E
T
ε
ε
$S' \to \bullet E$
$E \to \bullet T + E$
ε
ε
ε
ε
$E \to \bullet T$
$T \to \bullet (E)$
$T \to \bullet id$
ε
$($
T
ε
id
ε
$E \to T\bullet$
$T \to (\bullet E)$
$T \to id\bullet$
ε
E
$T \to (E\bullet)$

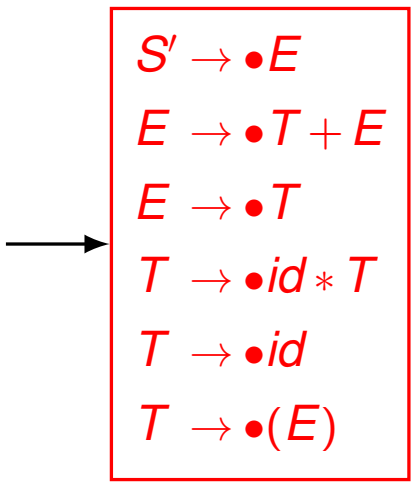
AFN


$E \to T + E\bullet$
$T \to id * \bullet T$
T
$T \to id * T\bullet$
E
$*$
ε
ε
$E \to T + \bullet E$
$T \to id \bullet * T$
ε
ε
$+$
id
$S' \to E\bullet$
$E \to T \bullet + E$
$T \to \bullet id * T$
ε
E
T
ε
ε
$S' \to \bullet E$
$E \to \bullet T + E$
ε
ε
ε
ε
$E \to \bullet T$
$T \to \bullet (E)$
$T \to \bullet id$
ε
$($
T
ε
id
ε
$E \to T\bullet$
$T \to (\bullet E)$
$T \to id\bullet$
ε
E
$T \to (E\bullet)$

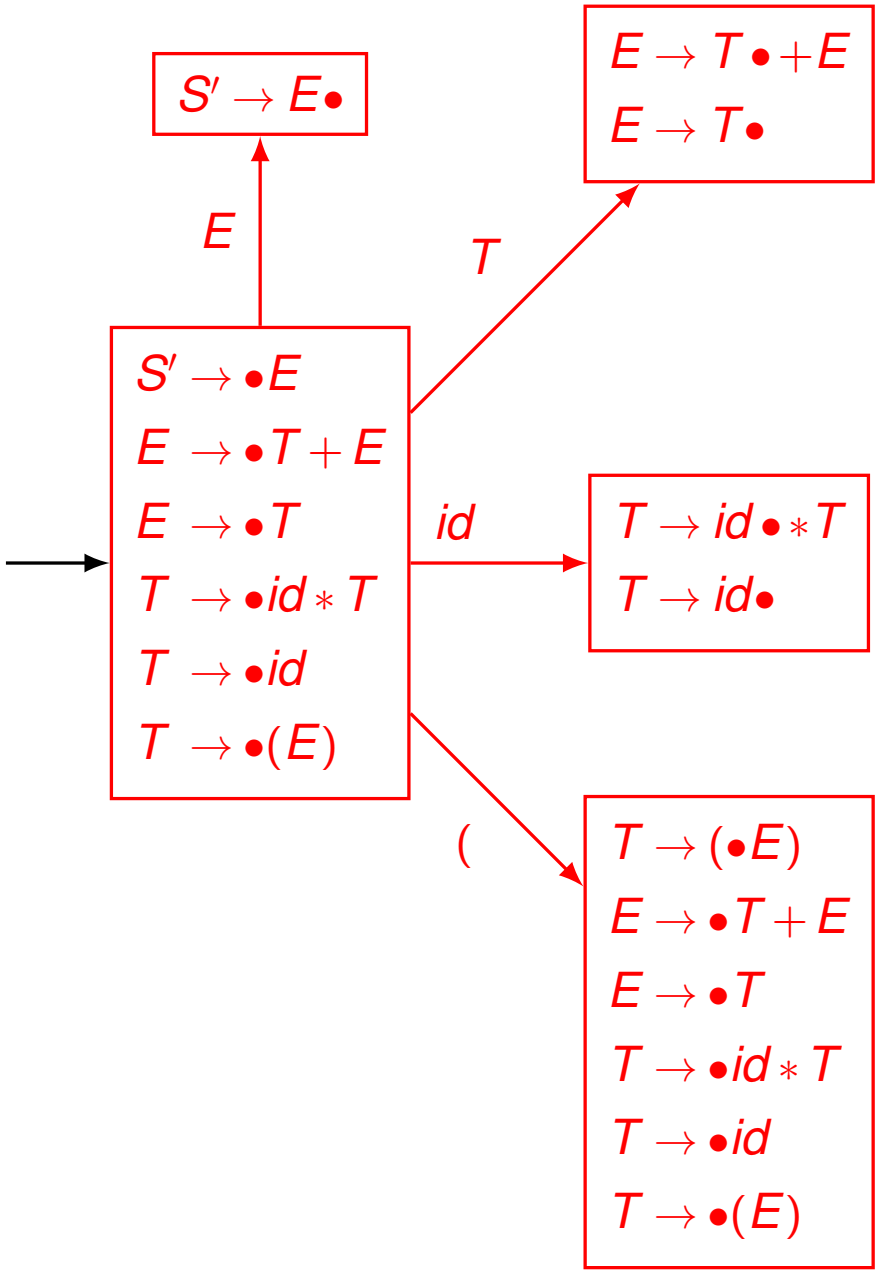
AFN


$E \to T + E\bullet$
$T \to id * \bullet T$
T
$T \to id * T\bullet$
E
$*$
ε
ε
$E \to T + \bullet E$
$T \to id \bullet * T$
ε
$+$
id
ε
$S' \to E\bullet$
$E \to T \bullet + E$
$T \to \bullet id * T$
ε
E
T
ε
ε
$S' \to \bullet E$
$E \to \bullet T + E$
ε
ε
ε
ε
$E \to \bullet T$
$T \to \bullet (E)$
$T \to \bullet id$
ε
(
T
ε
id
ε
$E \to T\bullet$
$T \to (\bullet E)$
$T \to id\bullet$
ε
E
$T \to (E\bullet)$
)
$T \to (E)\bullet$

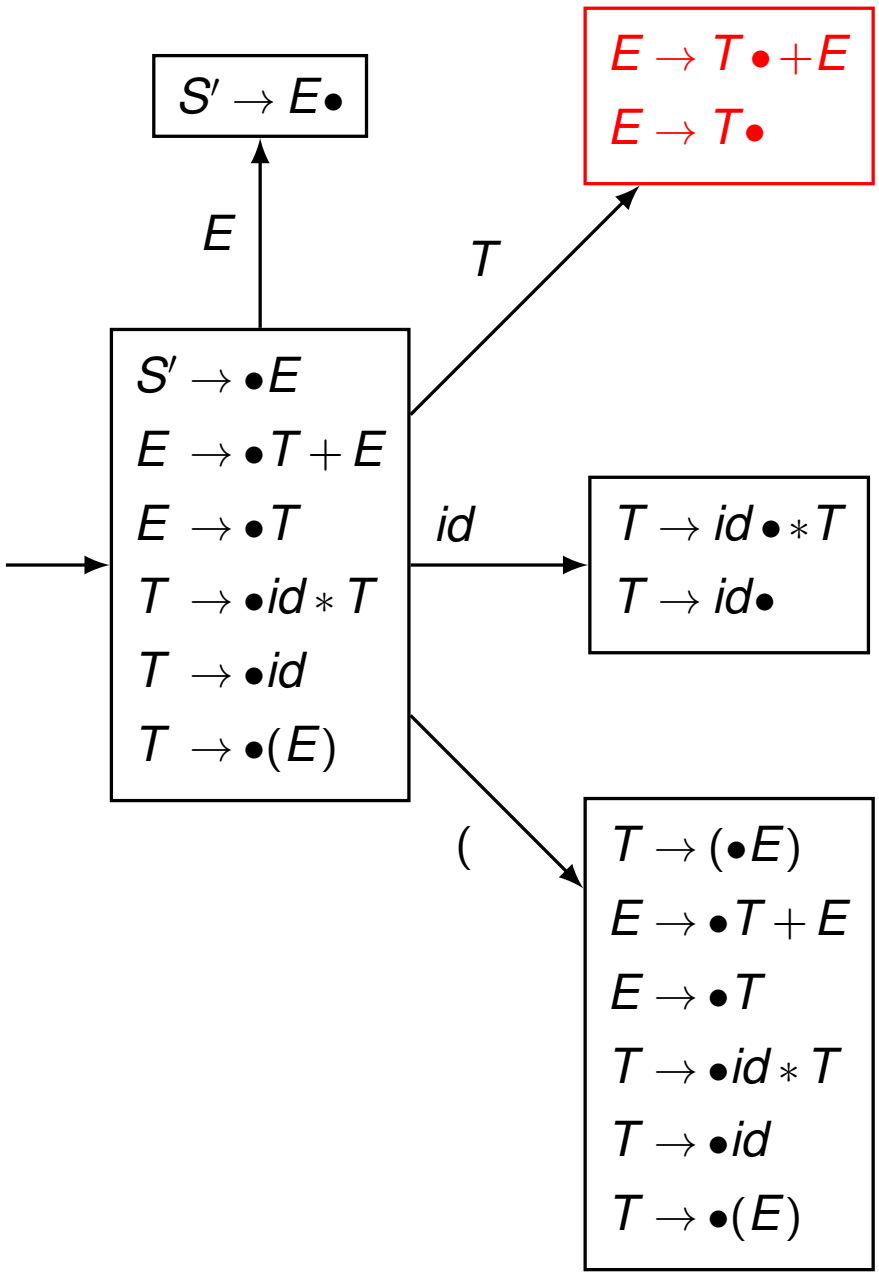
AFD


$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$

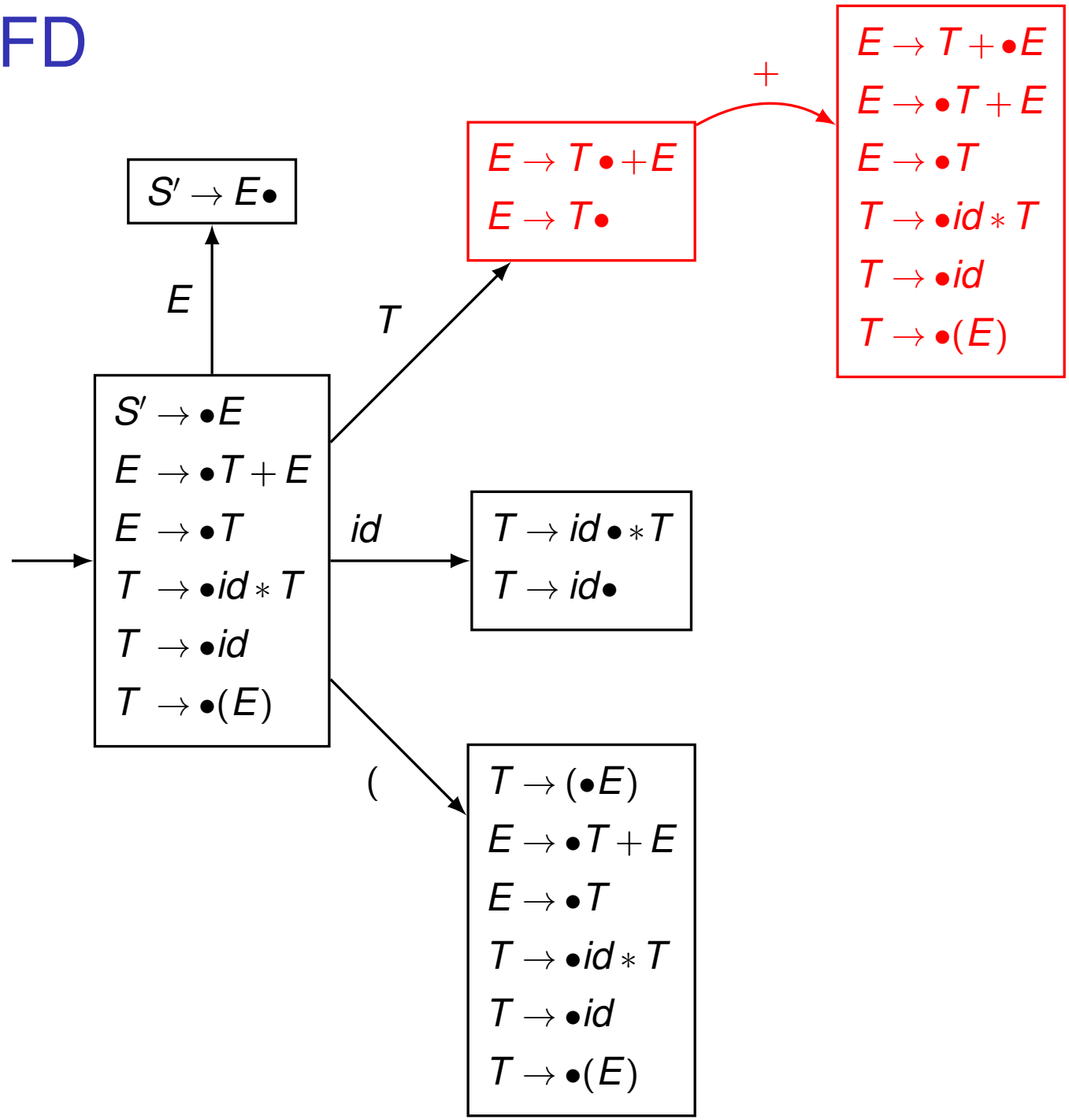
AFD


$S' \to E\bullet$
$E \to T\bullet + E$
$E \to T\bullet$
E
T
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
id
$T \to id\bullet * T$
$T \to id\bullet$
(
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$

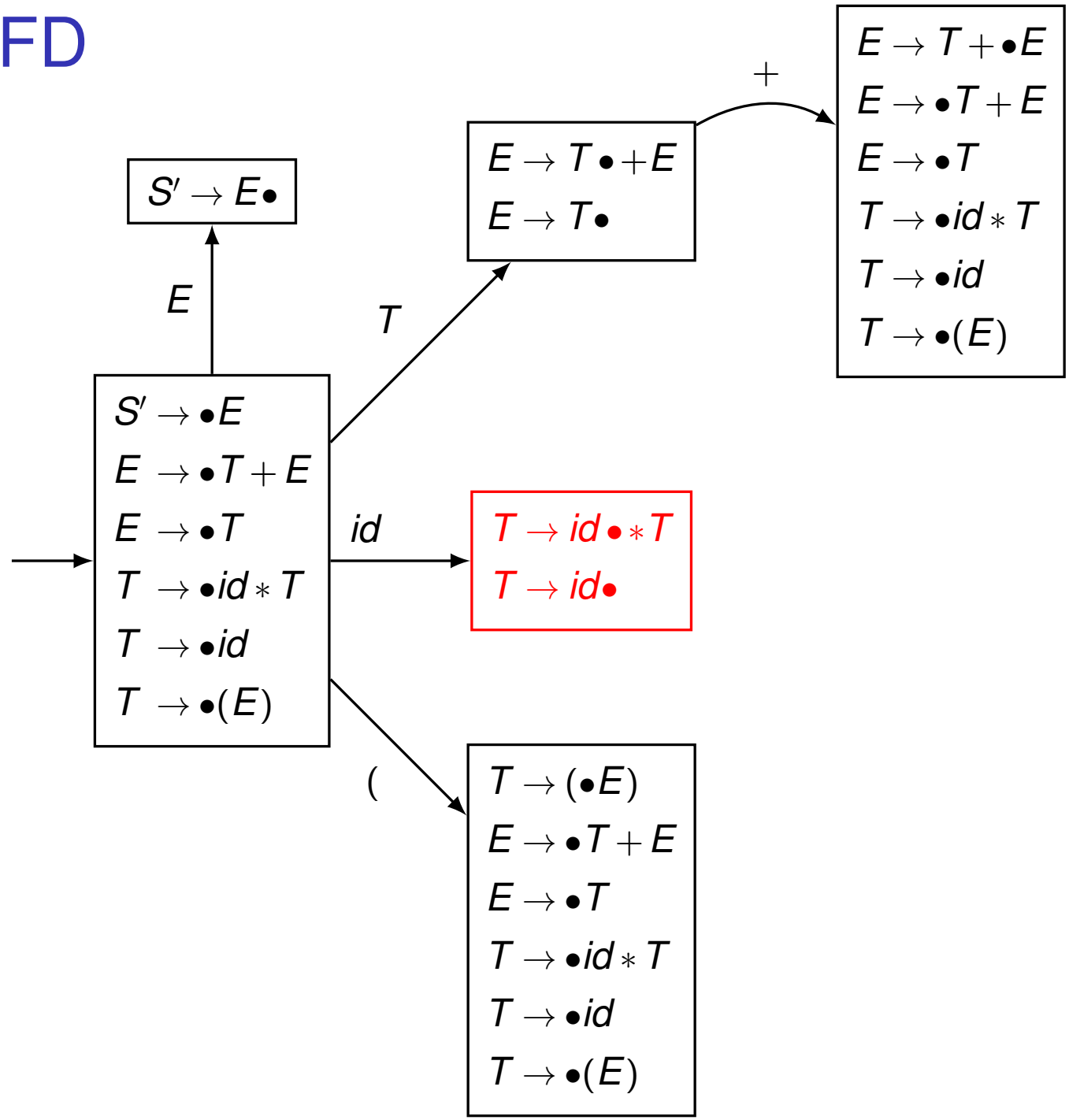
AFD


$S' \to E\bullet$
$E \to T\bullet + E$
$E \to T\bullet$
E
T
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
id
$T \to id\bullet * T$
$T \to id\bullet$
$($
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$

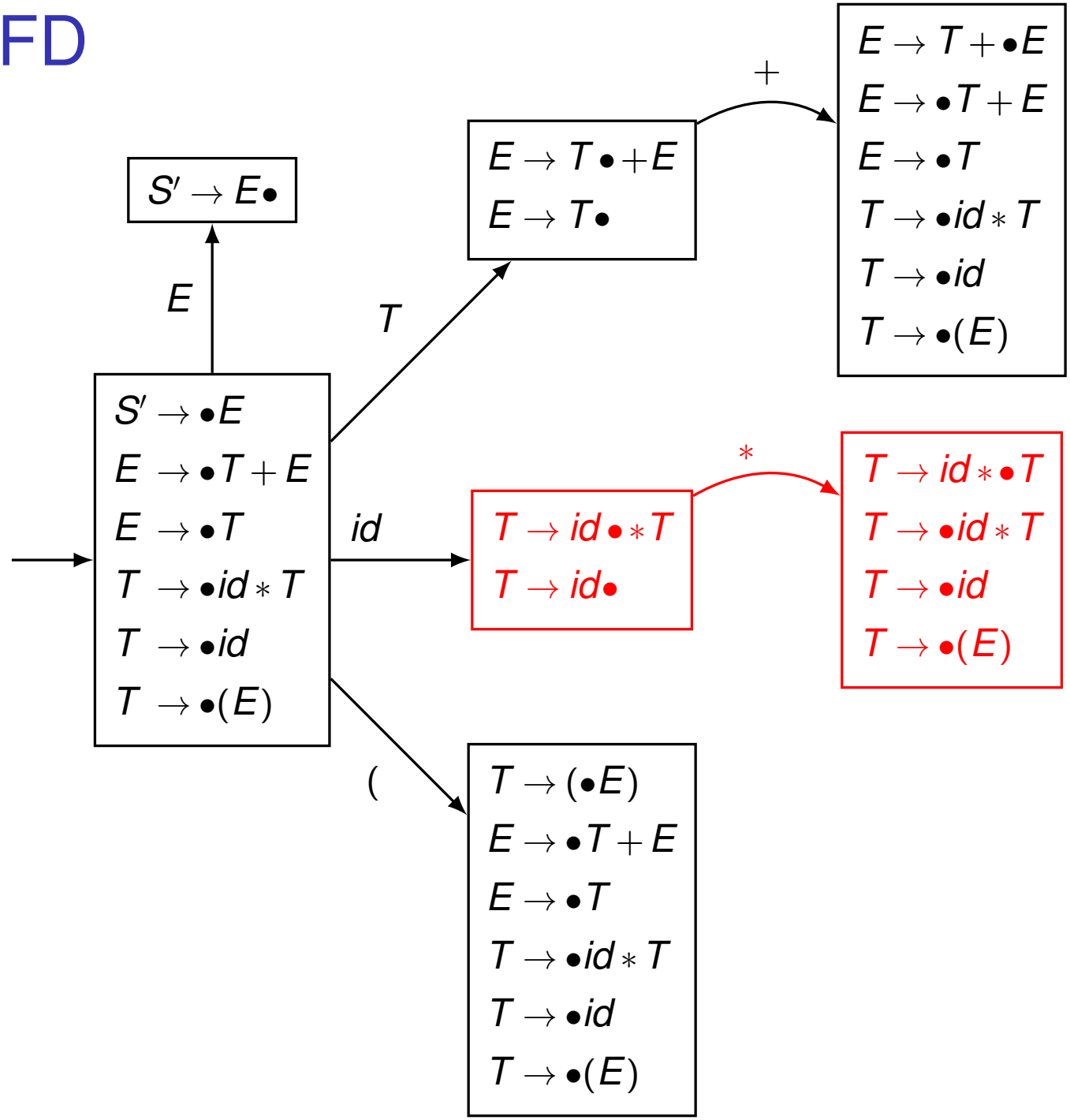
AFD


$S' \to E\bullet$
E
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
T
$E \to T \bullet + E$
$E \to T\bullet$
$+$
$E \to T + \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
id
$T \to id \bullet * T$
$T \to id\bullet$
$($
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$

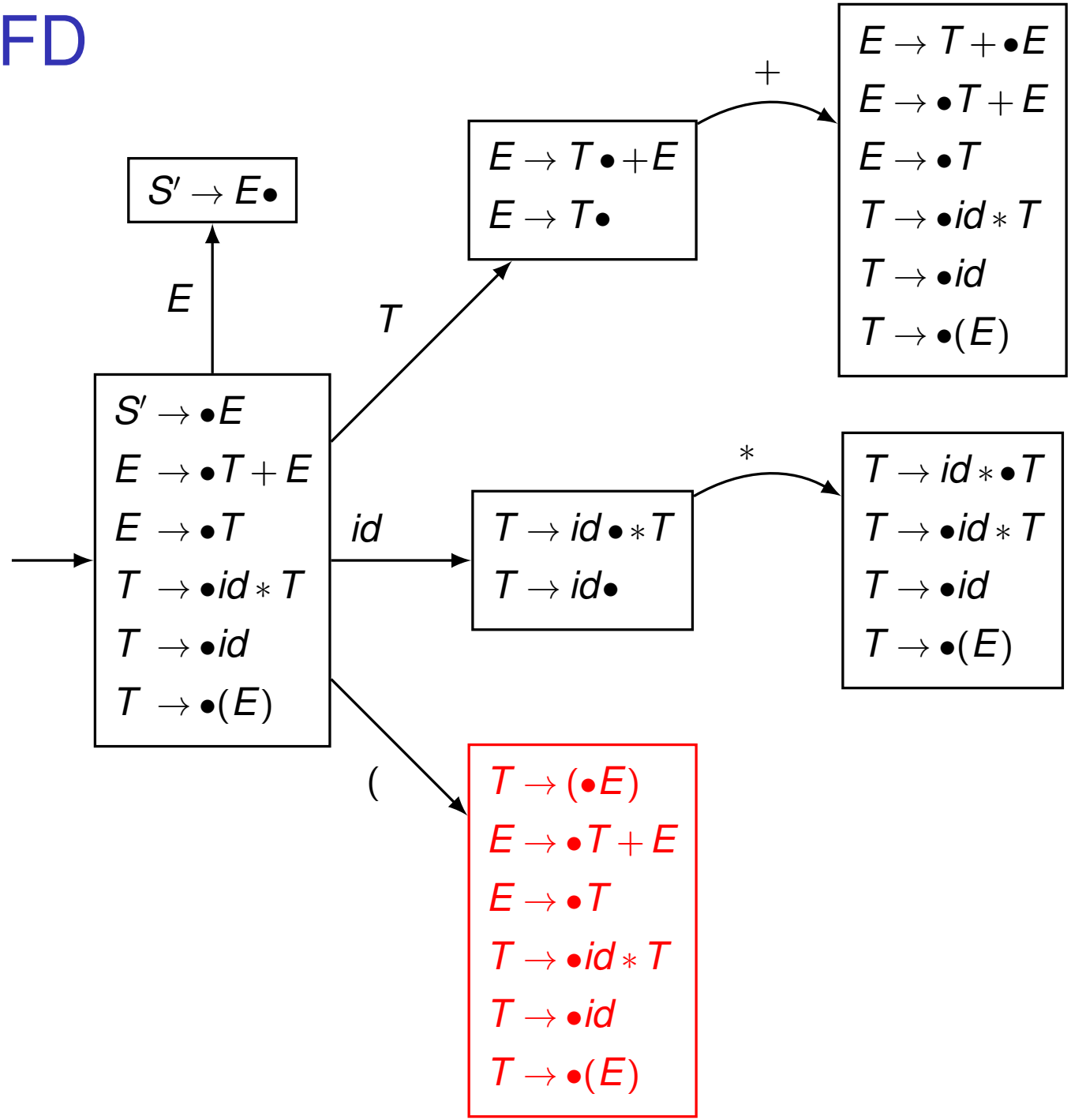
AFD


$S' \to E\bullet$
E
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
T
$E \to T \bullet + E$
$E \to T\bullet$
$+$
$E \to T + \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
id
$T \to id \bullet * T$
$T \to id\bullet$
$($
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$

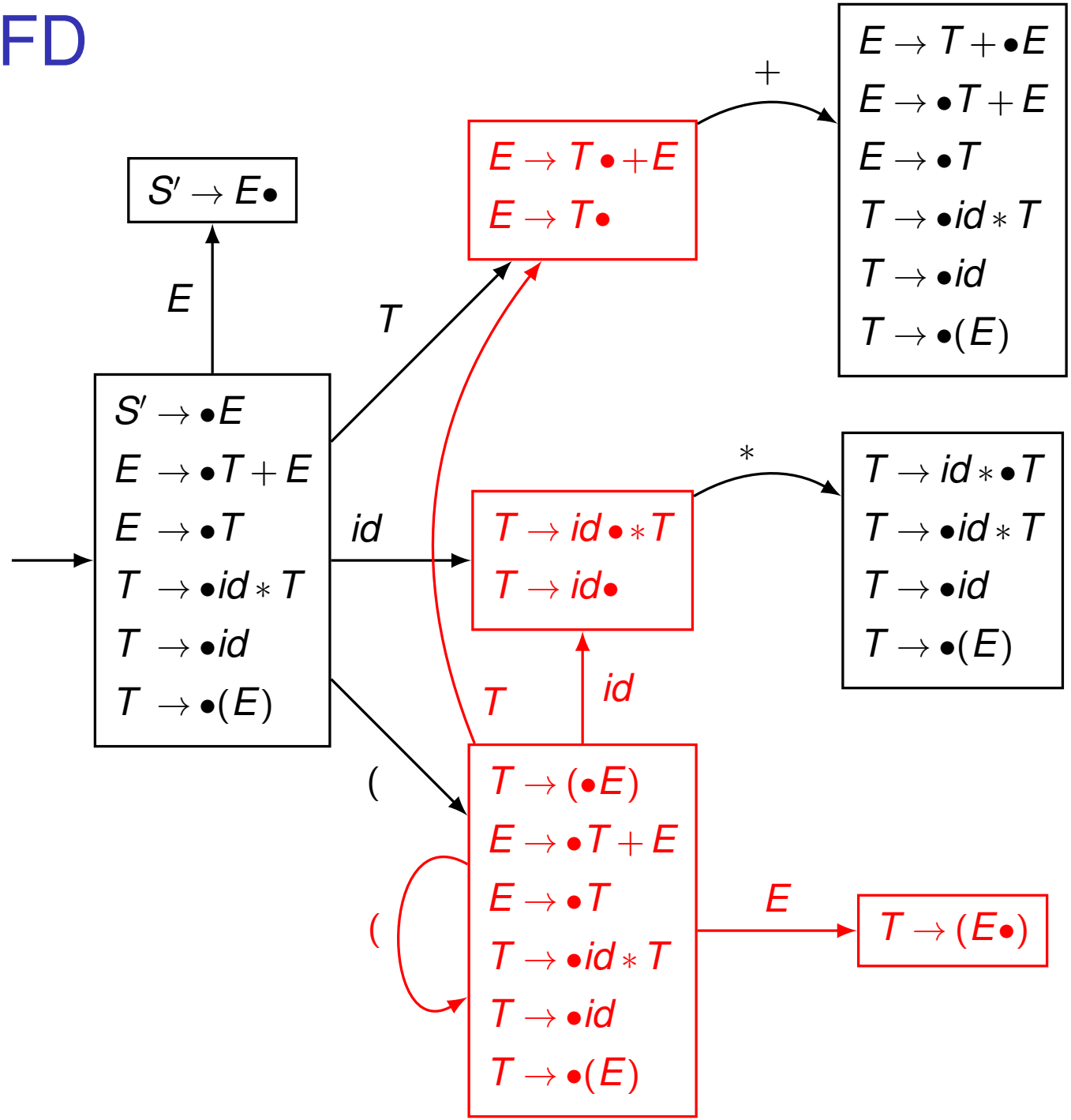
AFD


$S' \to E\bullet$
E
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
T
$E \to T \bullet + E$
$E \to T\bullet$
$+$
$E \to T + \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
id
$T \to id \bullet * T$
$T \to id\bullet$
$*$
$T \to id * \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$($
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$

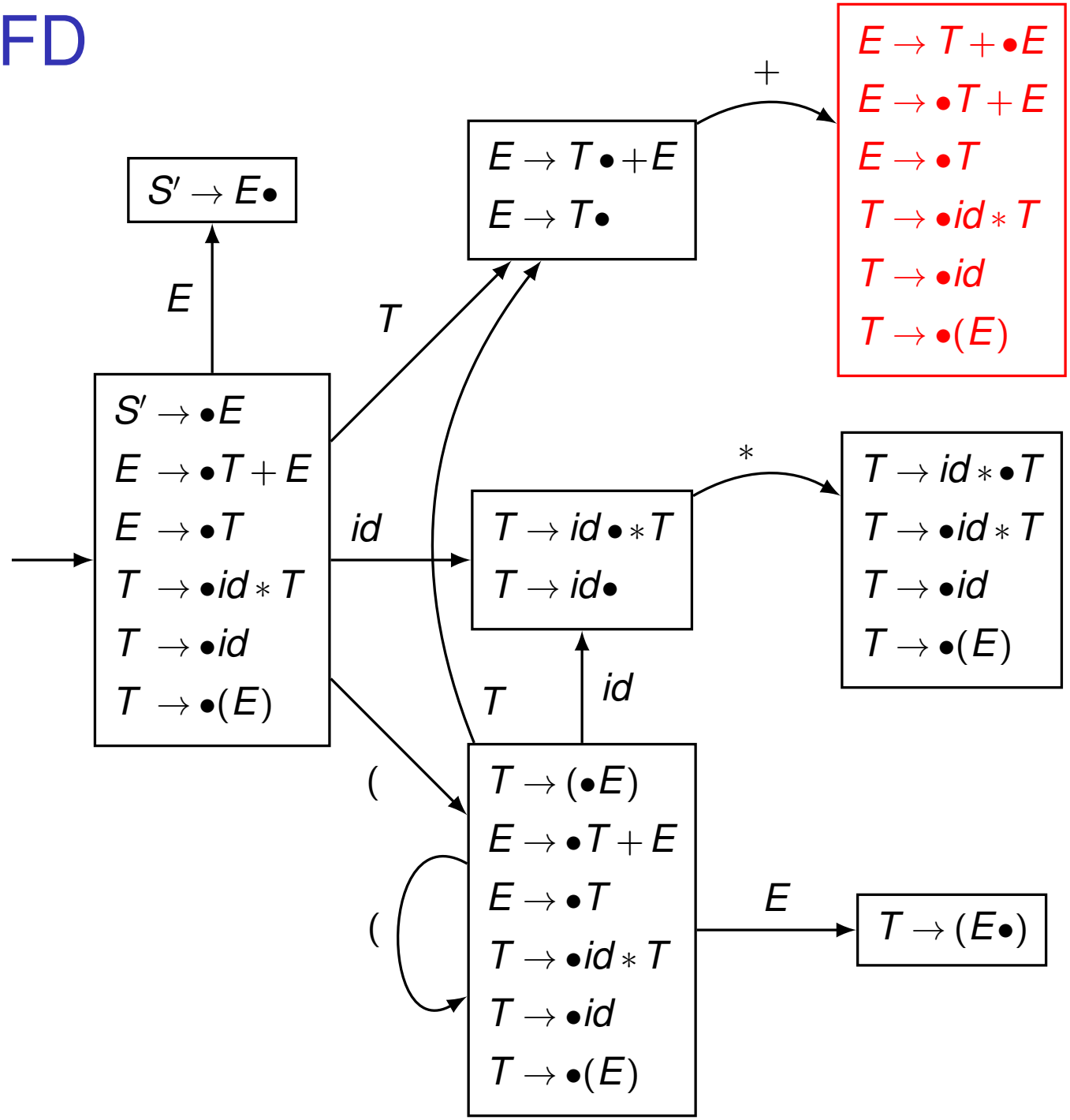
AFD


$S' \to E\bullet$
E
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
T
$E \to T \bullet + E$
$E \to T\bullet$
+
$E \to T + \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
id
$T \to id \bullet * T$
$T \to id\bullet$
*
$T \to id * \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
(
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$

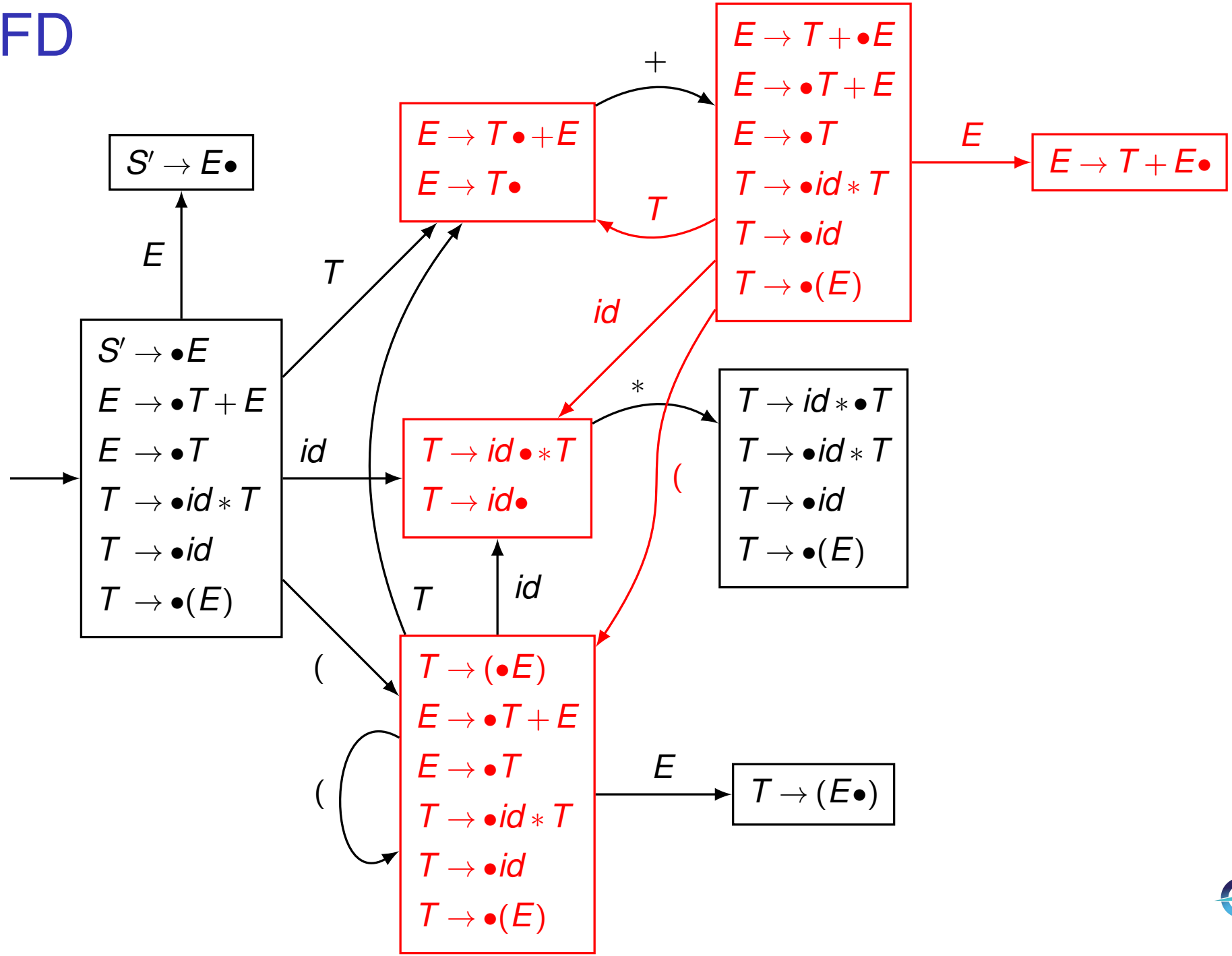
AFD


$S' \to E\bullet$
E
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
T
$E \to T \bullet + E$
$E \to T\bullet$
$+$
$E \to T + \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
id
$T \to id \bullet * T$
$T \to id\bullet$
$*$
$T \to id * \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$($
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
T
id
$($
E
$T \to (E\bullet)$

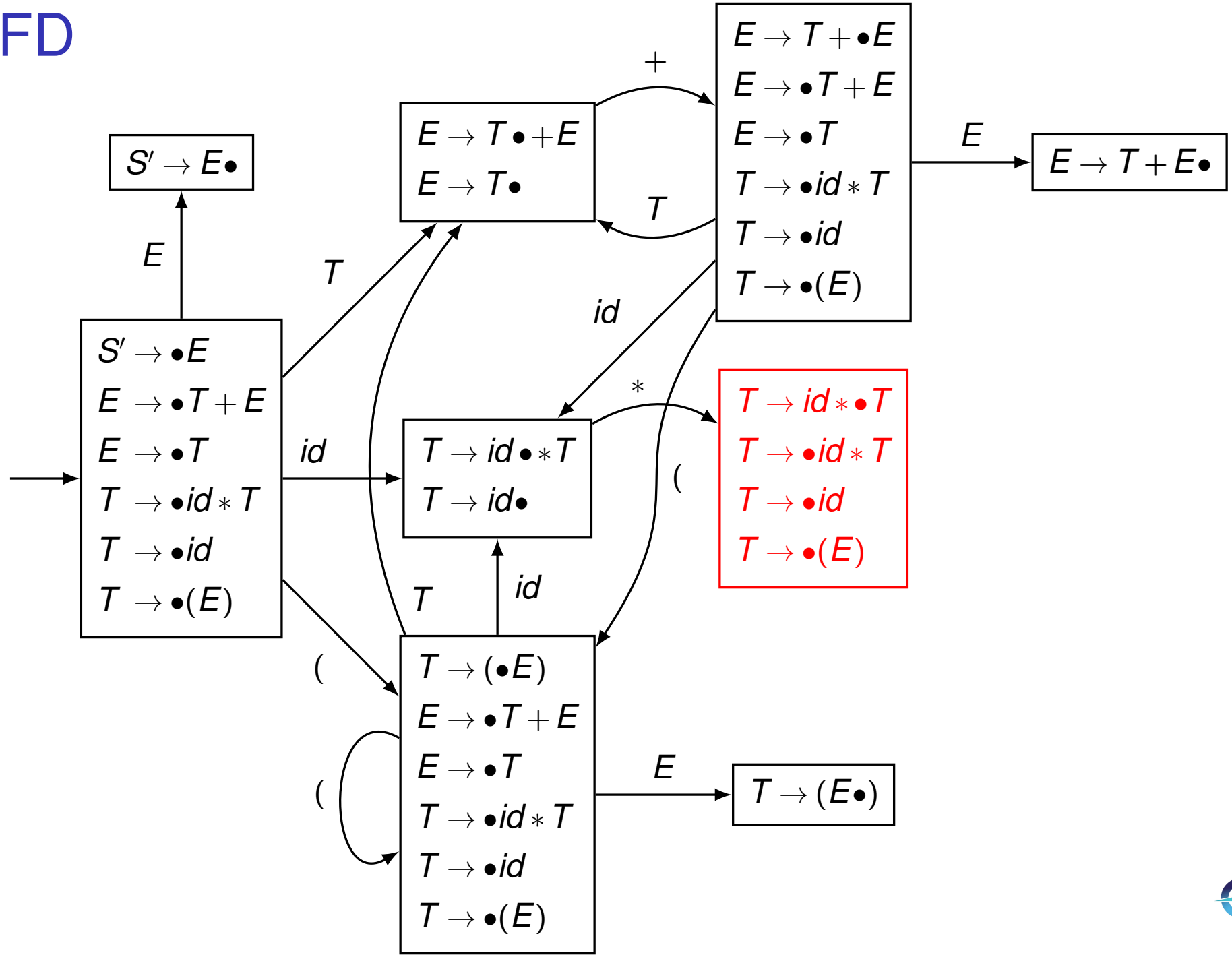
AFD


$S' \to E\bullet$
$E \to T\bullet + E$
$E \to T\bullet$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to id\bullet * T$
$T \to id\bullet$
$T \to id * \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to (E\bullet)$
E
T
+
id
*
T
id
(
(
E

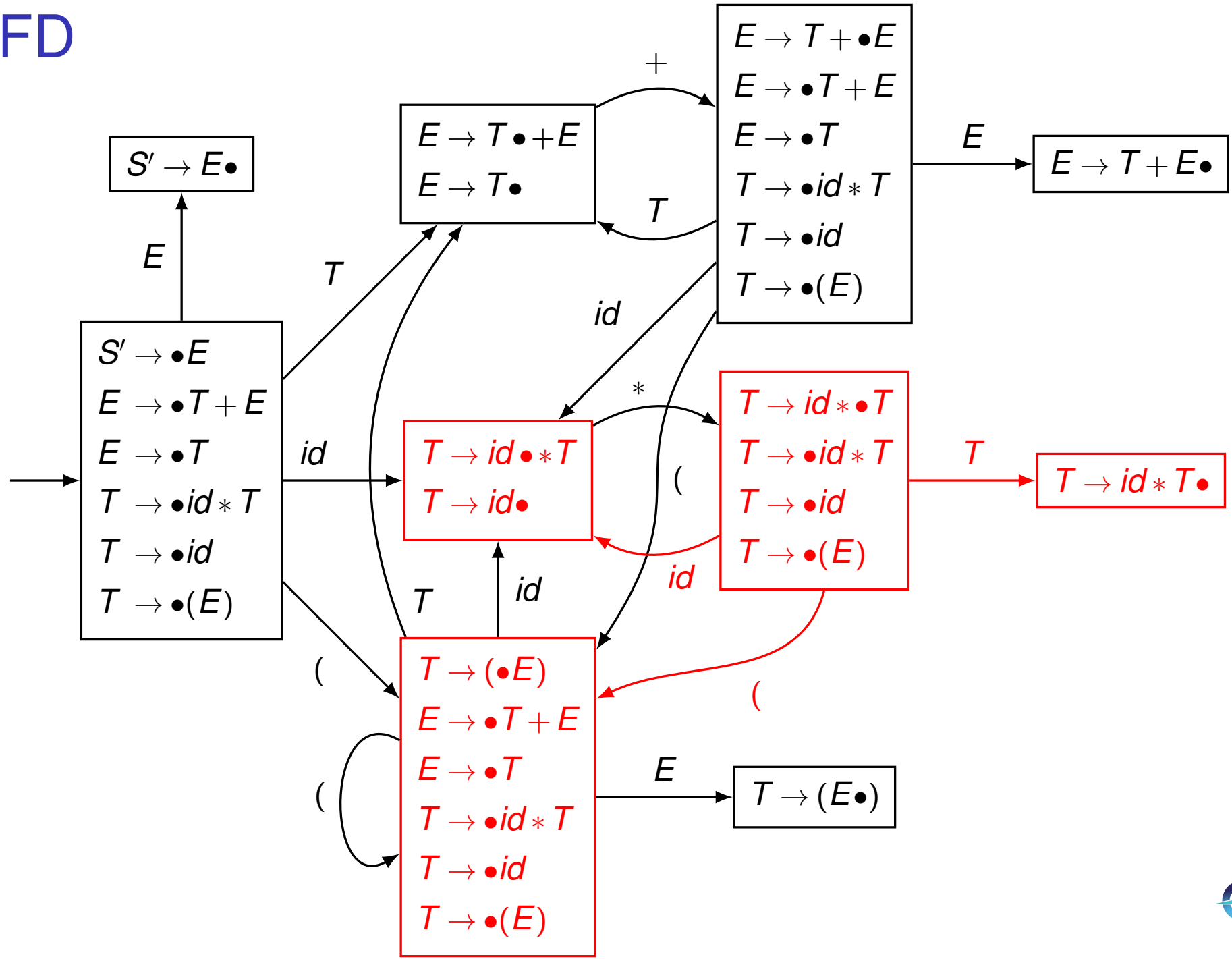
AFD


$S' \to E\bullet$
E
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
T
$E \to T \bullet + E$
$E \to T\bullet$
$+$
$E \to T + \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
E
$E \to T + E\bullet$
id
$T \to id \bullet * T$
$T \to id\bullet$
$*$
$T \to id * \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$($
T
id
$($
$($
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
E
$T \to (E\bullet)$

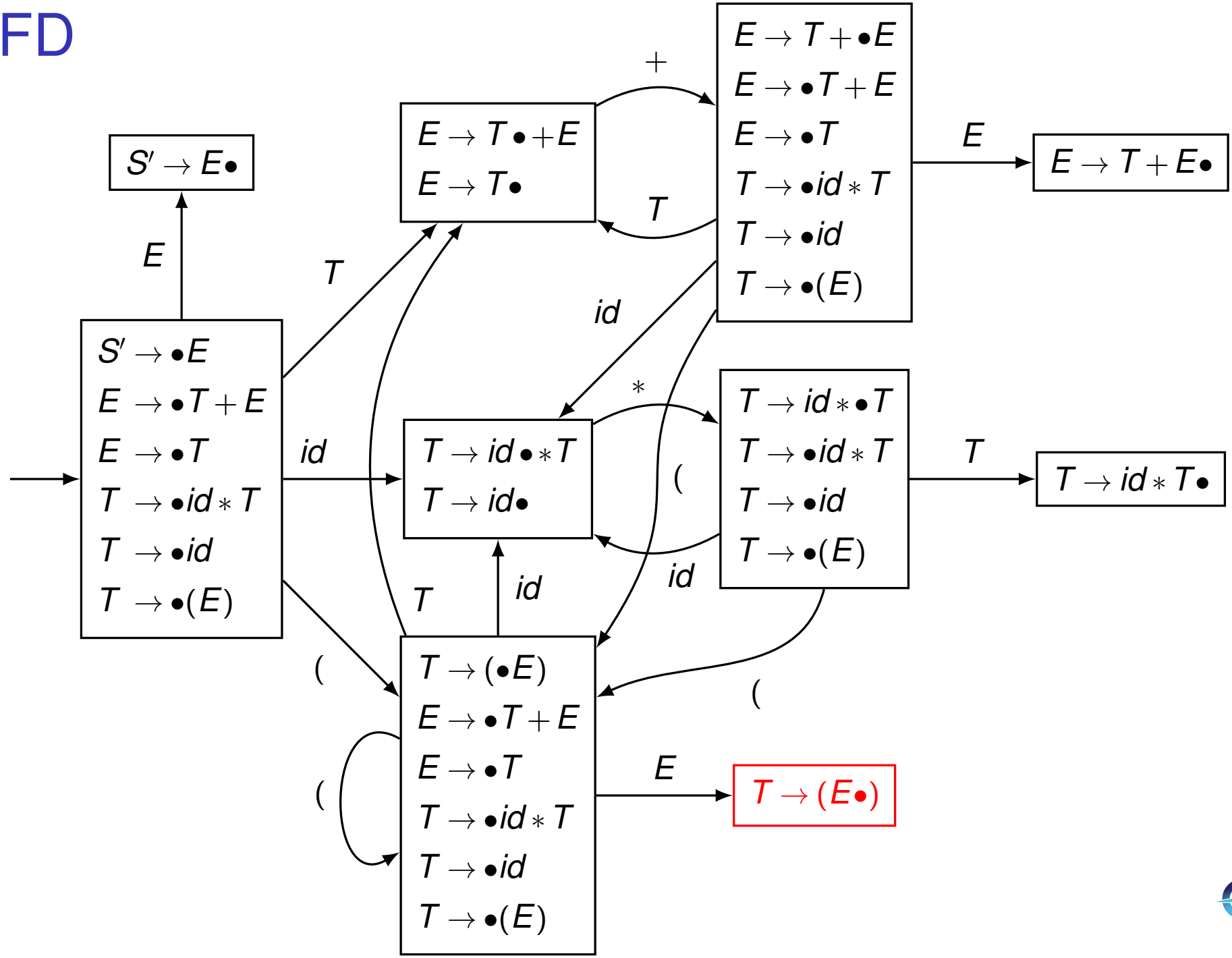
AFD


$S' \to E\bullet$
$E \to T\bullet + E$
$E \to T\bullet$
$E \to T + \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$E \to T + E\bullet$
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to id\bullet * T$
$T \to id\bullet$
$T \to id * \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to (E\bullet)$
+
T
E
E
T
id
*
(
id
T
(
(
E

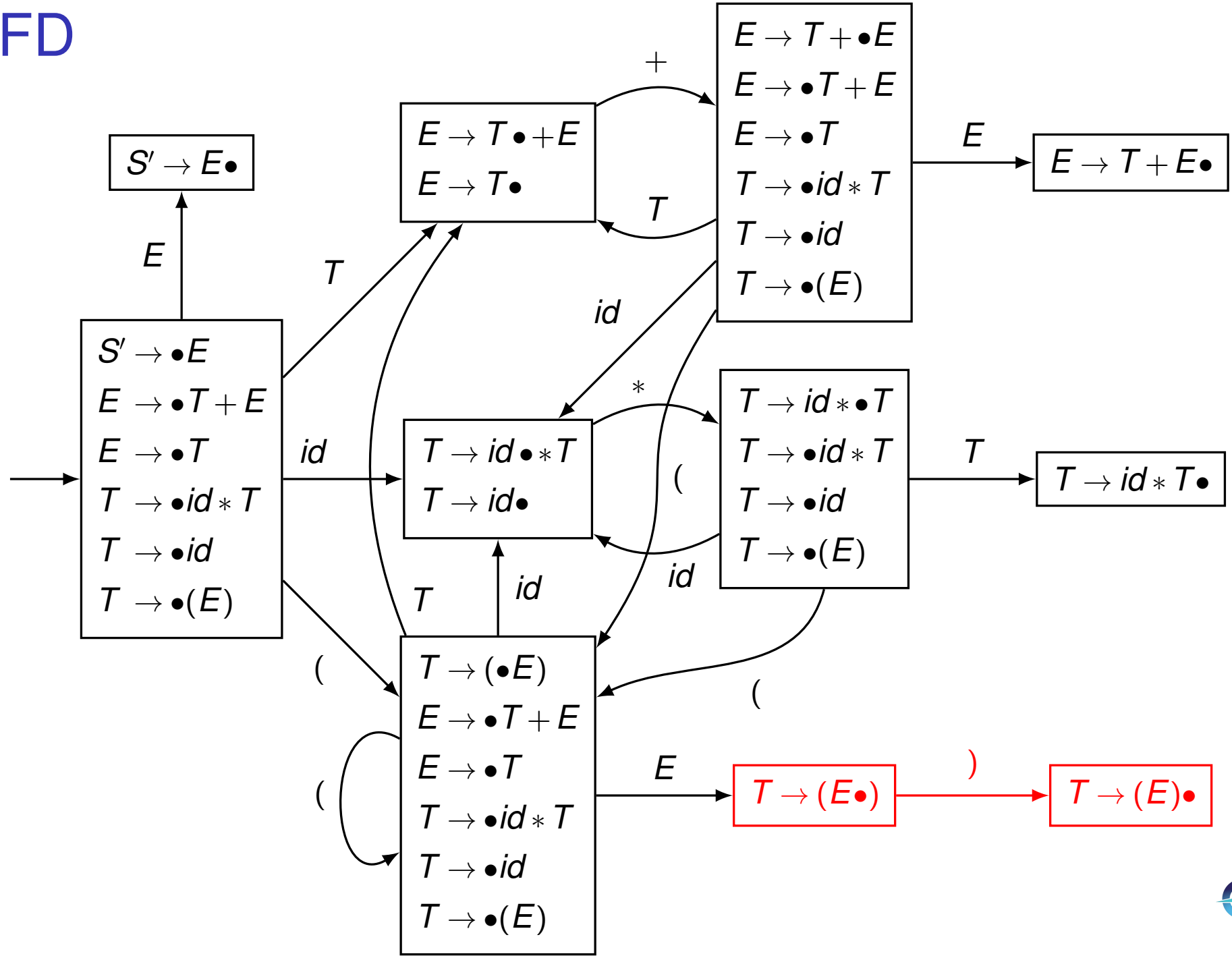
AFD


$S' \to E\bullet$
$E \to T\bullet + E$
$E \to T\bullet$
$E \to T + \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$E \to T + E\bullet$
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to id\bullet * T$
$T \to id\bullet$
$T \to id * \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to id * T\bullet$
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to (E\bullet)$
+
E
T
E
T
id
*
(
id
T
id
(
(
E

AFD


$S' \to E\bullet$
$E \to T\bullet + E$
$E \to T\bullet$
$E \to T + \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$E \to T + E\bullet$
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to id\bullet * T$
$T \to id\bullet$
$T \to id * \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to id * T\bullet$
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to (E\bullet)$
E
T
id
+
*
(

AFD


$S' \to E\bullet$
$E \to T\bullet + E$
$E \to T\bullet$
$E \to T + \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$E \to T + E\bullet$
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to id\bullet * T$
$T \to id\bullet$
$T \to id * \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to id * T\bullet$
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to (E\bullet)$
$T \to (E)\bullet$
E
T
id
*
+
(
)

Itemi valizi

- Itemul $X \to \beta \bullet \gamma$, **valid** pentru **prefixul viabil** $\alpha\beta$, dacă secvența următoare poate apărea într-o derivare **la dreapta**:

$$S' \to^* \alpha X \delta \to \alpha\beta\gamma\delta$$

- În starea $\alpha\beta \mid \gamma\delta$, **reper** încă neevidențiat reprezentat de producția $X \to \beta\gamma$, și poziția de la dreapta lui α

- Ulterior, în starea $\alpha\beta\gamma \mid \delta$, posibilitatea reducerii cu producția $X \to \beta\gamma$

- Alternativ, item valid *I* pentru prefixul α, dacă execuția AFD pe șirul α se încheie într-o stare care îl conține pe *I*

Cuprins

Analiză $LR(0)$

- $LR(\mathbf{0})$: absența *lookahead*-ului în alegerea producțiilor (reducțiilor)
- Starea **analizorului**: $\alpha\beta \mid t\ldots$, cu t terminal
- Starea finală a **AFD** rulat pe prefixul $\alpha\beta$: s
- ***Reduce*** cu $X \to \beta$, dacă s conține $X \to \beta\bullet$
- ***Shift***, dacă s conține $X \to \beta \bullet t\gamma$

Conflicte $LR(0)$

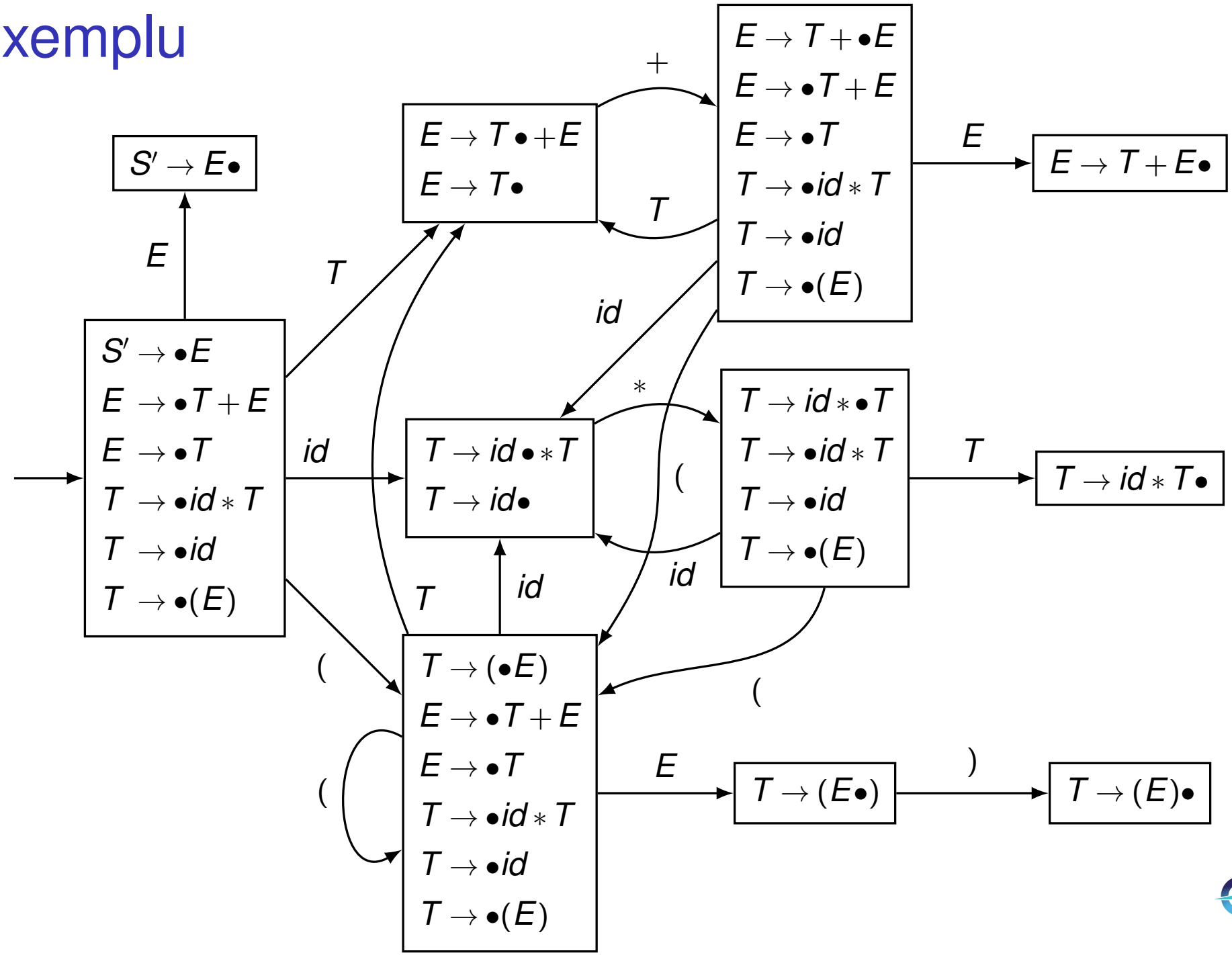
- Conflict ***shift-reduce***, în cazul existenței unei stări cu câte un item de fiecare tip:

$$X_1 \to \beta_1 \bullet$$
$$X_2 \to \beta_2 \bullet t\gamma$$

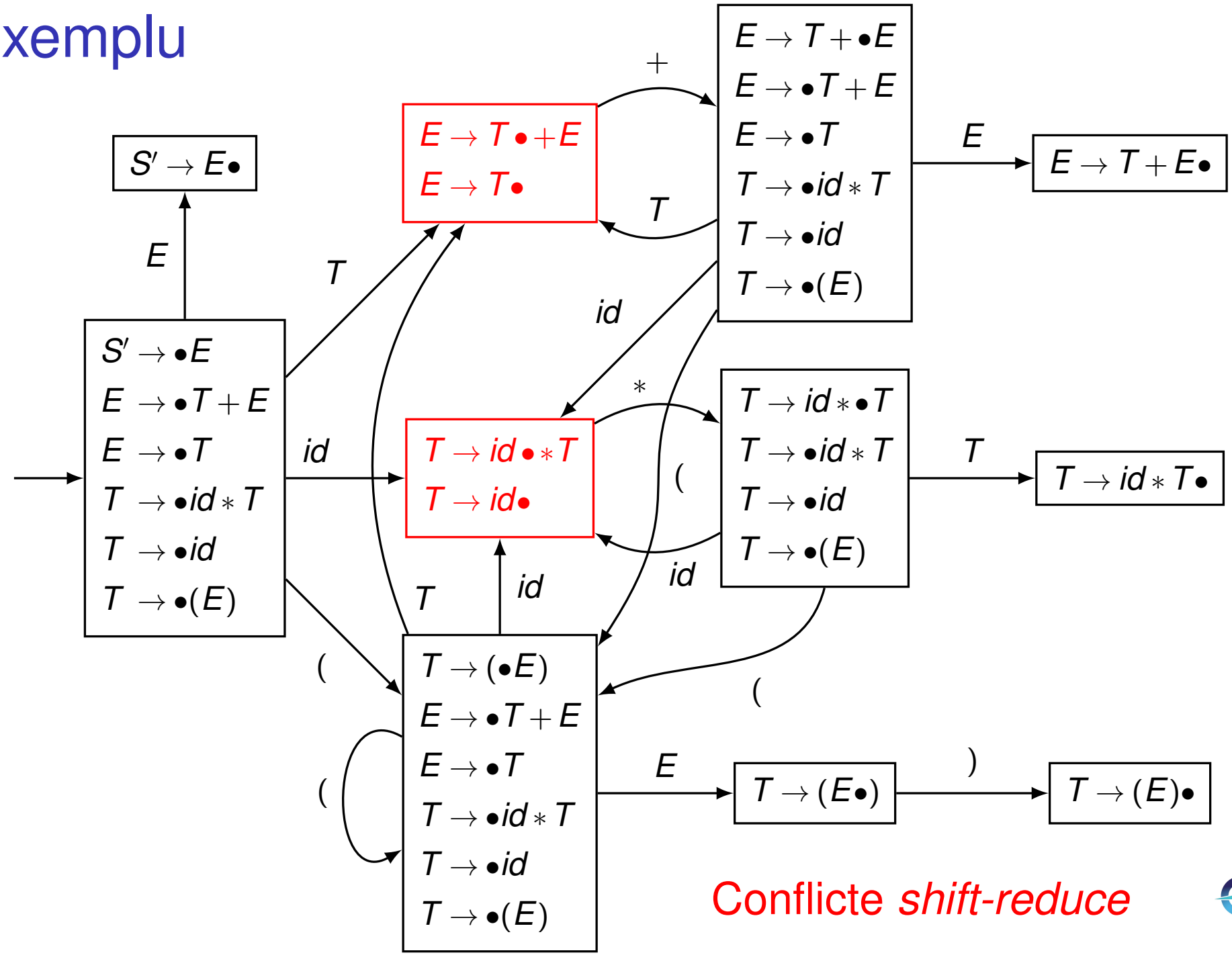
- Conflict ***reduce-reduce***, în cazul existenței unei stări cu cel puțin doi itemi *reduce*:

$$X_1 \to \beta_1 \bullet$$
$$X_2 \to \beta_2 \bullet$$

Exemplu


$S' \to E\bullet$
$E \to T\bullet + E$
$E \to T\bullet$
$E \to T + \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$E \to T + E\bullet$
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to id\bullet * T$
$T \to id\bullet$
$T \to id * \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to id * T\bullet$
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to (E\bullet)$
$T \to (E)\bullet$
E
T
id
*
+
(
)

Exemplu

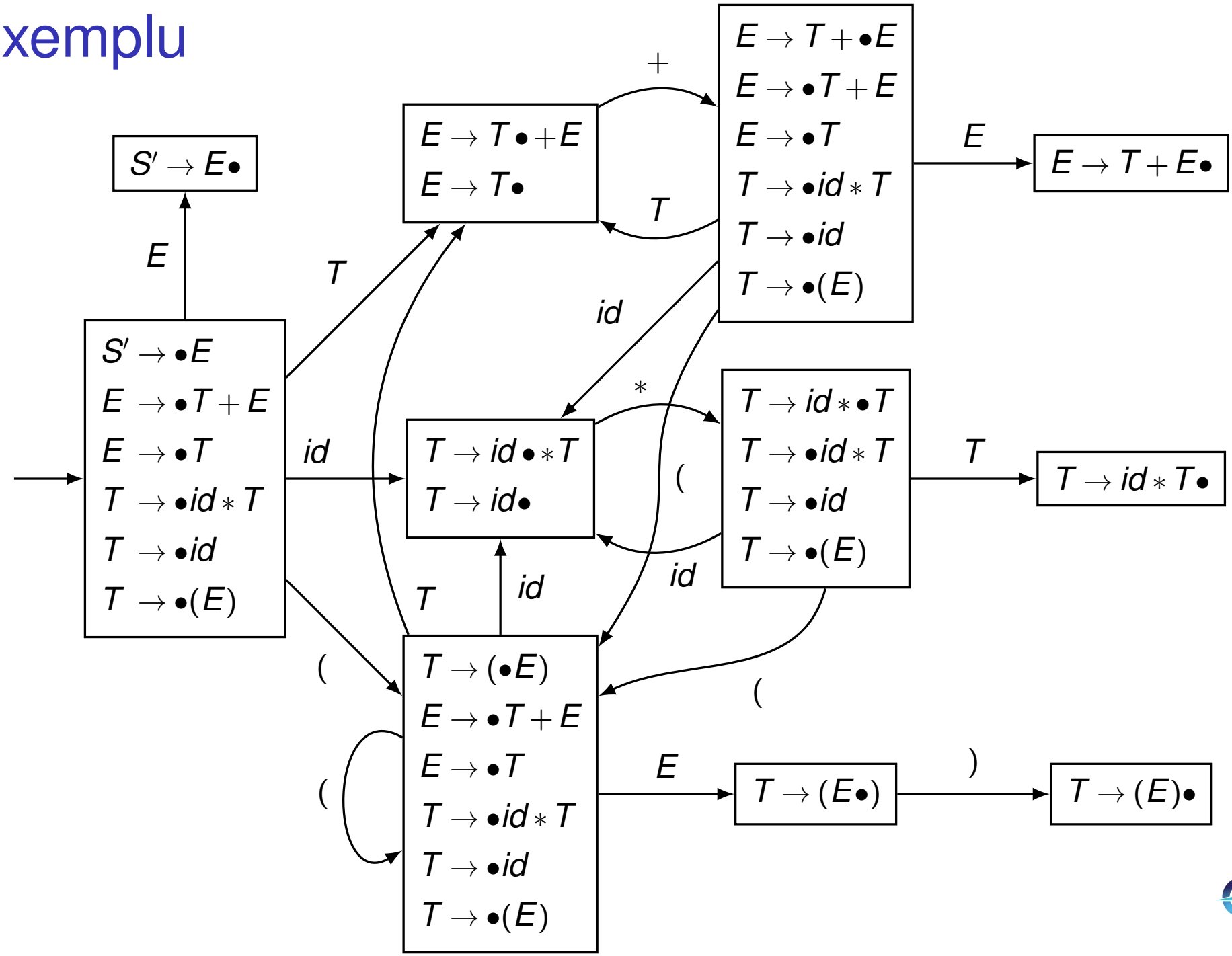


Conflicte *shift-reduce*

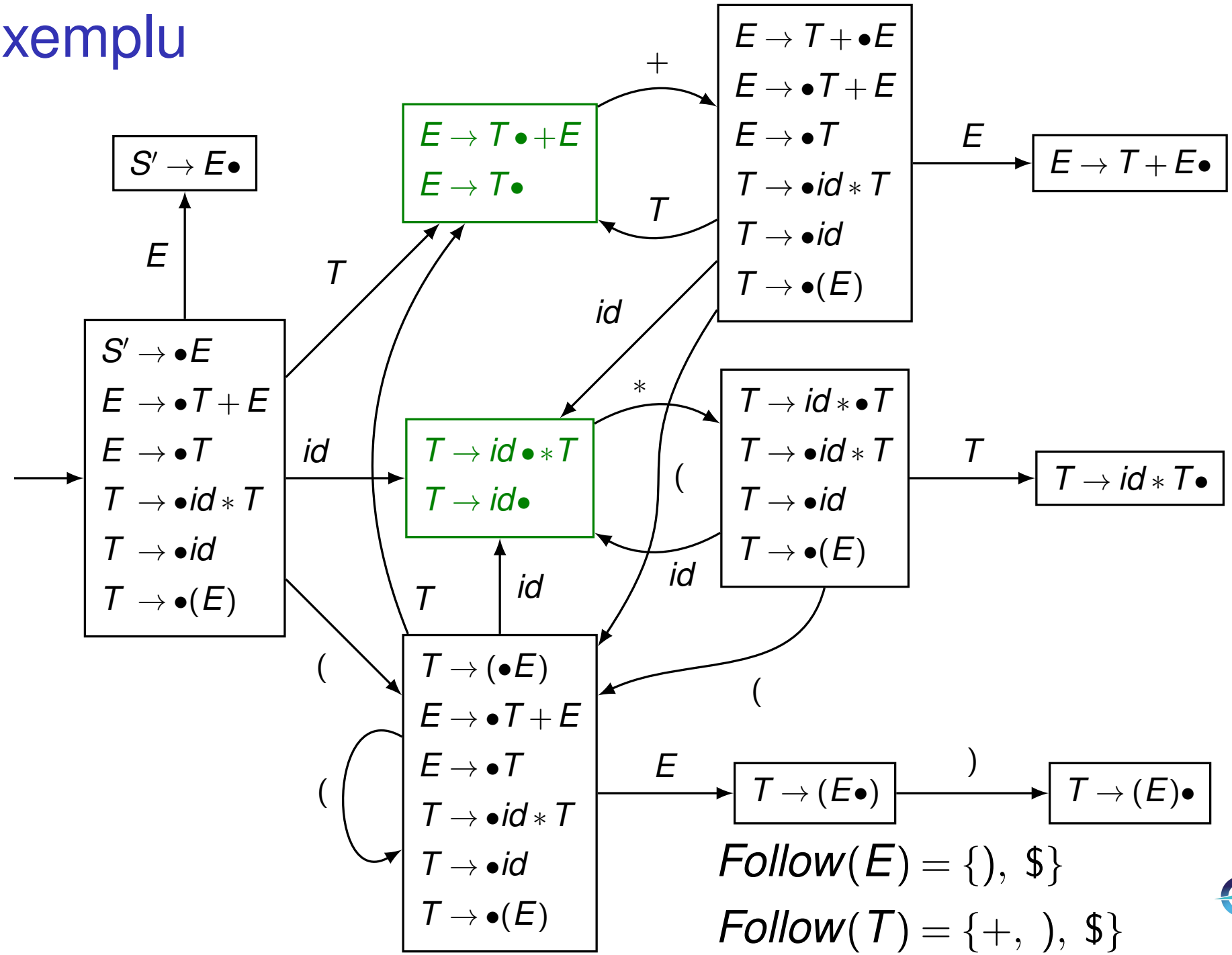
Analiză *SLR*(1)

- ***S****LR*(1): *Simple LR*
- *SLR*(**1**): utilizarea unui *token* de ***lookahead*** pentru reducerea numărului stărilor conflictuale
- Îmbunătățire: ***Reduce*** cu $X \to \beta$, dacă s conține $X \to \beta\bullet$ și $t \in Follow(X)$

Exemplu


$S' \to E\bullet$
$E \to T\bullet + E$
$E \to T\bullet$
$E \to T + \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$E \to T + E\bullet$
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to id\bullet * T$
$T \to id\bullet$
$T \to id * \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to id * T\bullet$
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet(E)$
$T \to (E\bullet)$
$T \to (E)\bullet$
E
T
id
*
+
(
)

Exemplu



$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Gramatica aritmetică ambiguă

- Gramatica:

$$\begin{aligned} E &\to E + E \\ &\mid \quad E * E \\ &\mid \quad (E) \\ &\mid \quad id \end{aligned}$$

- Existența unei stări cu conflict ***shift-reduce***:

$$E \to E * E\bullet$$
$$E \to E \bullet + E$$

- Necesitatea declarațiilor de **precedență**!

Algoritmul $SLR(1)$ naiv

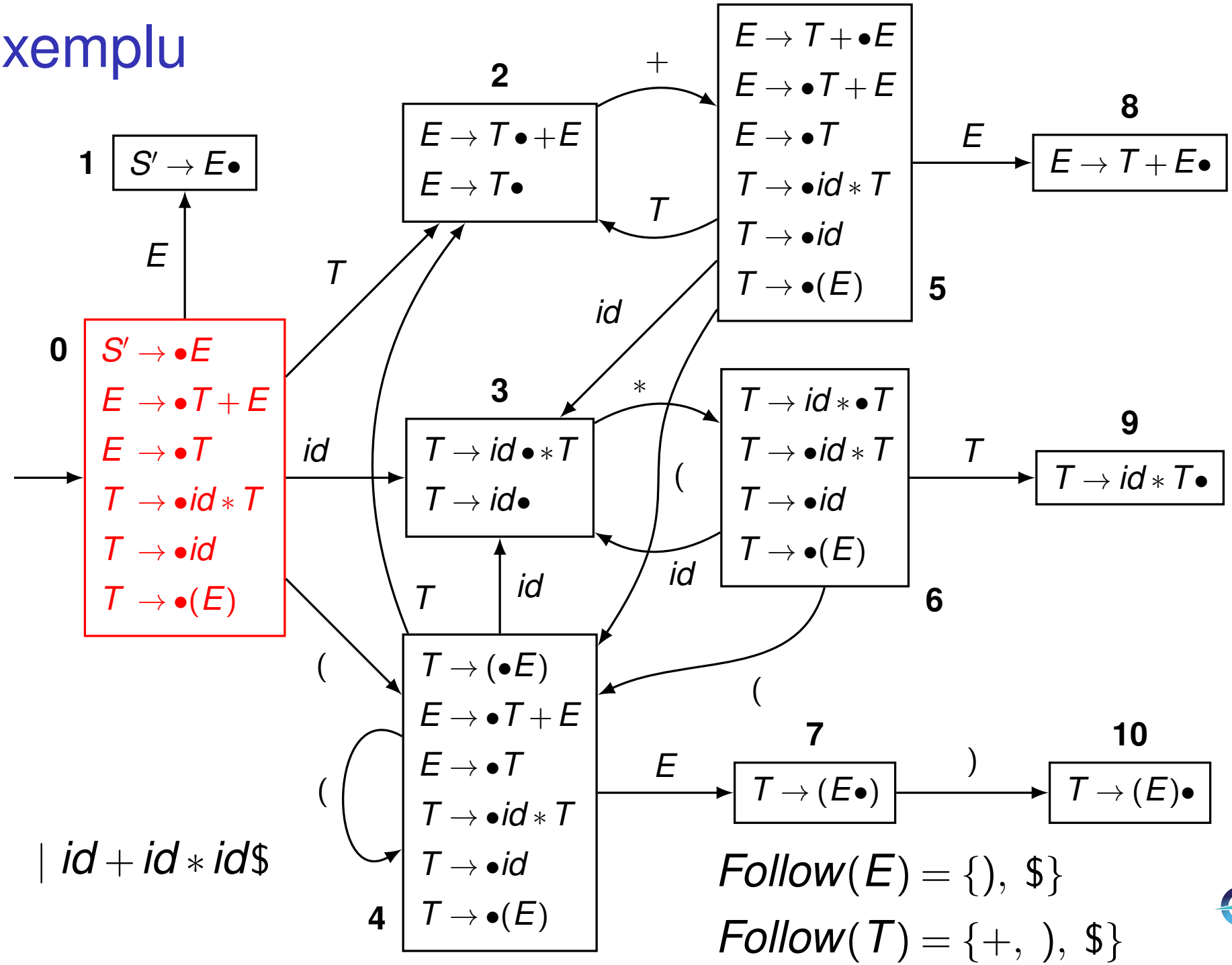
1. Starea **inițială**: $\mid x_1 \ldots x_n\$$

2. Pentru starea curentă a analizorului $\alpha \mid \gamma$, rulează **AFD** pe prefixul α, cu încheierea execuției în starea s.

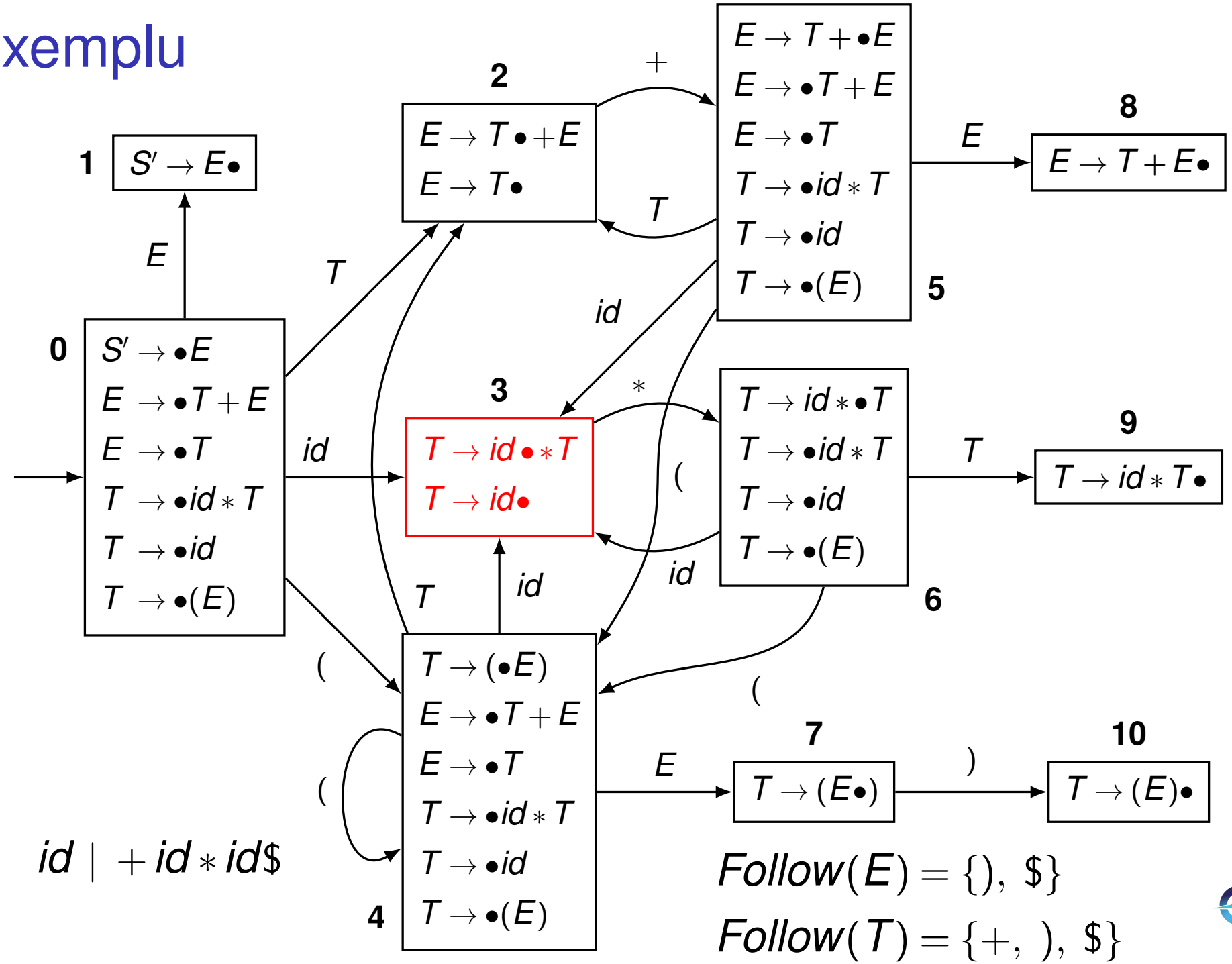
3. Aplică **acțiunile** *shift* sau *reduce* conform strategiei $SLR(1)$, sau semnalează eroare dacă nu există acțiuni aplicabile.

4. Dacă starea analizorului este $S' \mid \$$, **acceptă**. Altfel, reia de la pasul 2.

Exemplu


0
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
1
$S' \to E\bullet$
2
$E \to T \bullet + E$
$E \to T\bullet$
3
$T \to id \bullet * T$
$T \to id\bullet$
4
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
5
$E \to T + \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
6
$T \to id * \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
7
$T \to (E\bullet)$
8
$E \to T + E\bullet$
9
$T \to id * T\bullet$
10
$T \to (E)\bullet$
E
T
id
(
+
*
)
$| \; id + id * id\$$
$Follow(E) = \{), \; \$\}$
$Follow(T) = \{+, \;), \; \$\}$

Exemplu

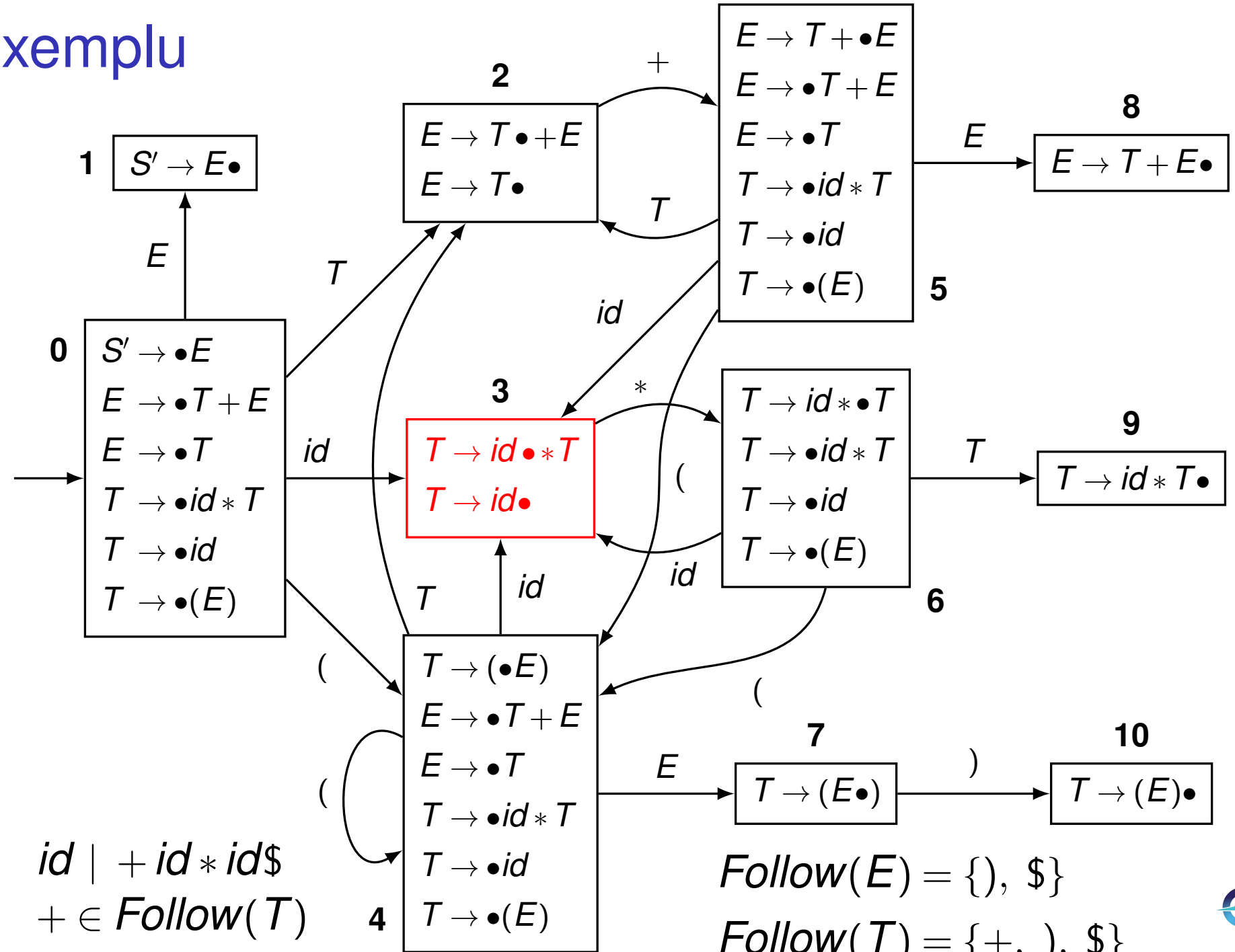


$id \mid +id * id\$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu



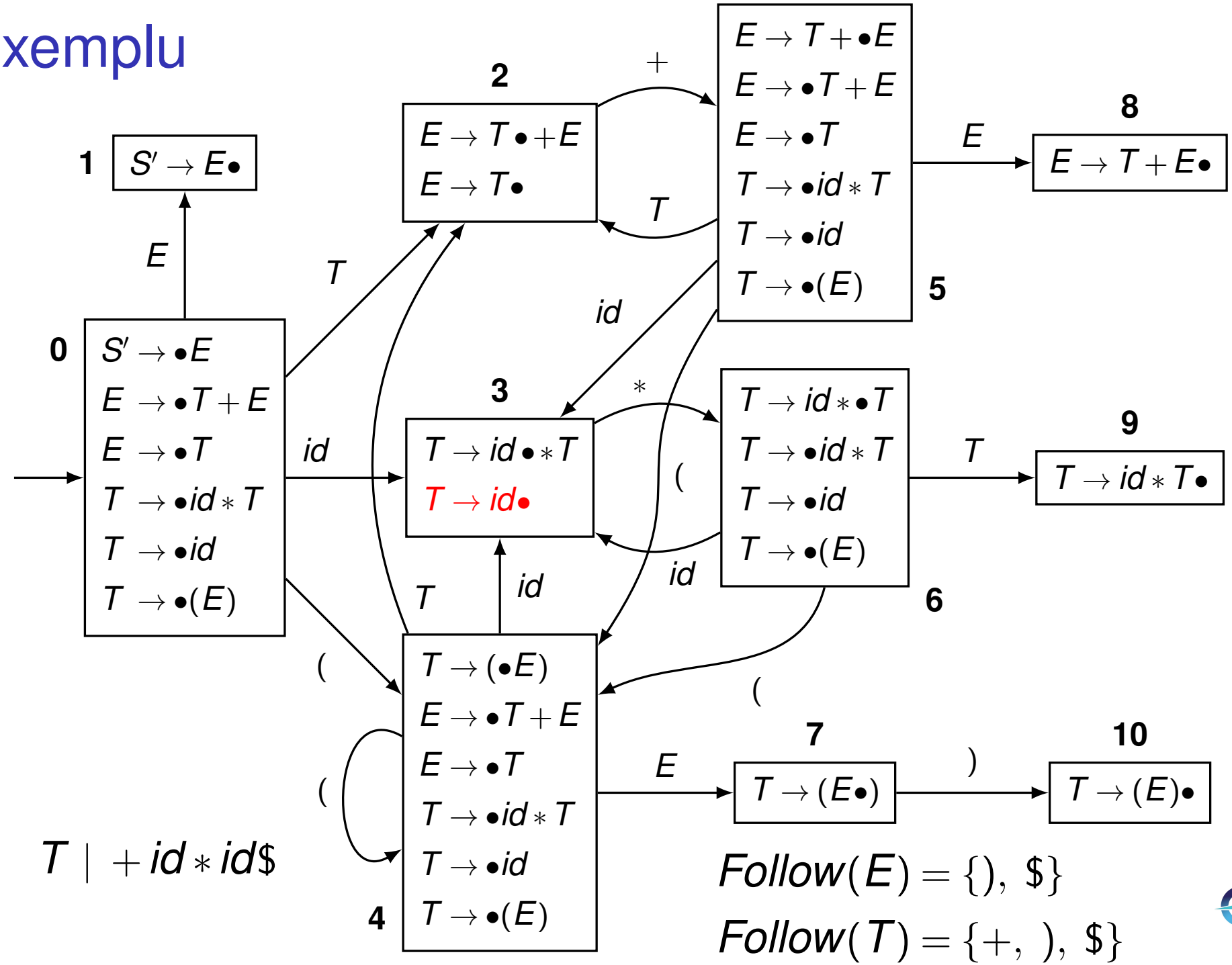
$id \mid + id * id\$$

$+ \in Follow(T)$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu

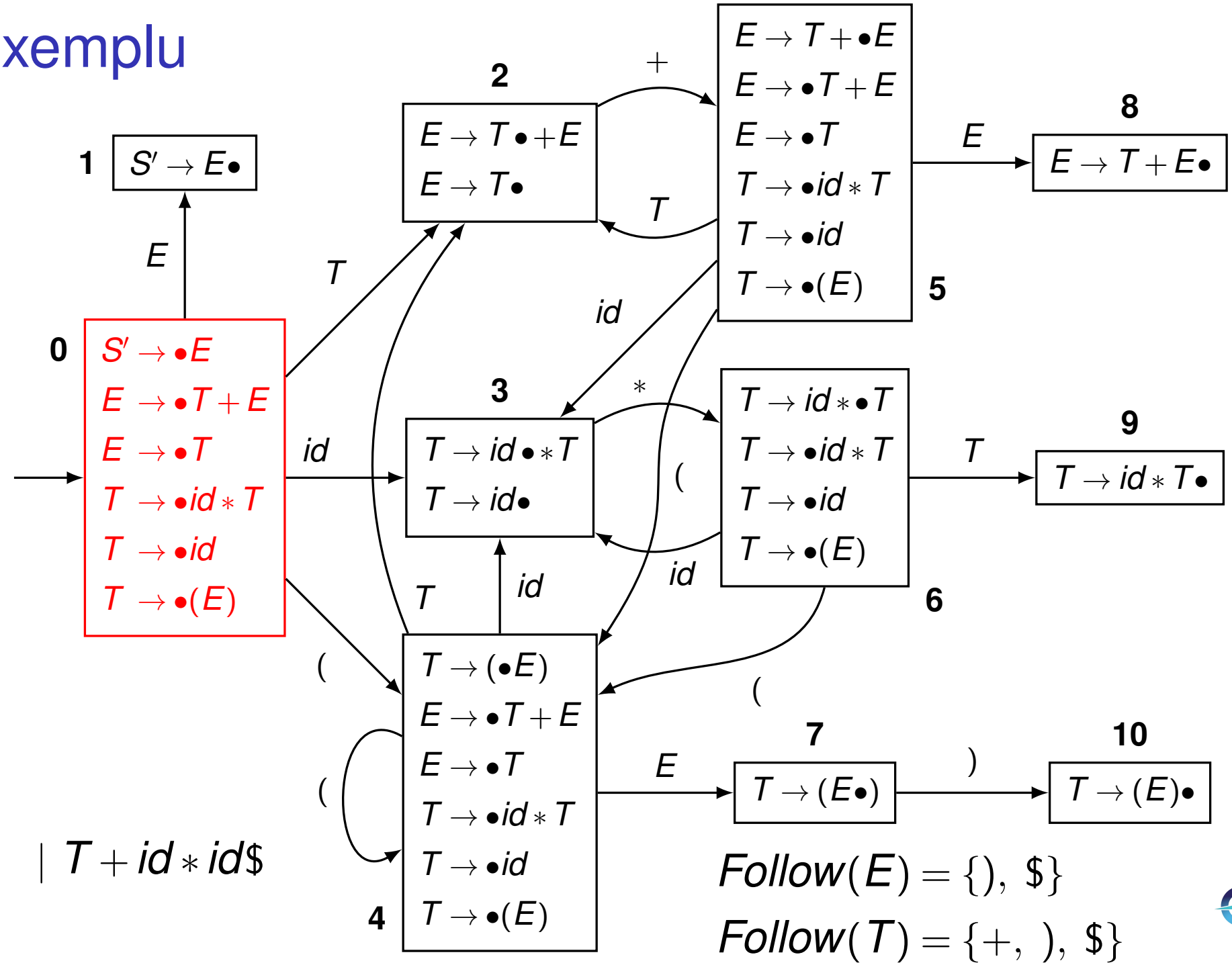


$T \mid +id * id\$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu

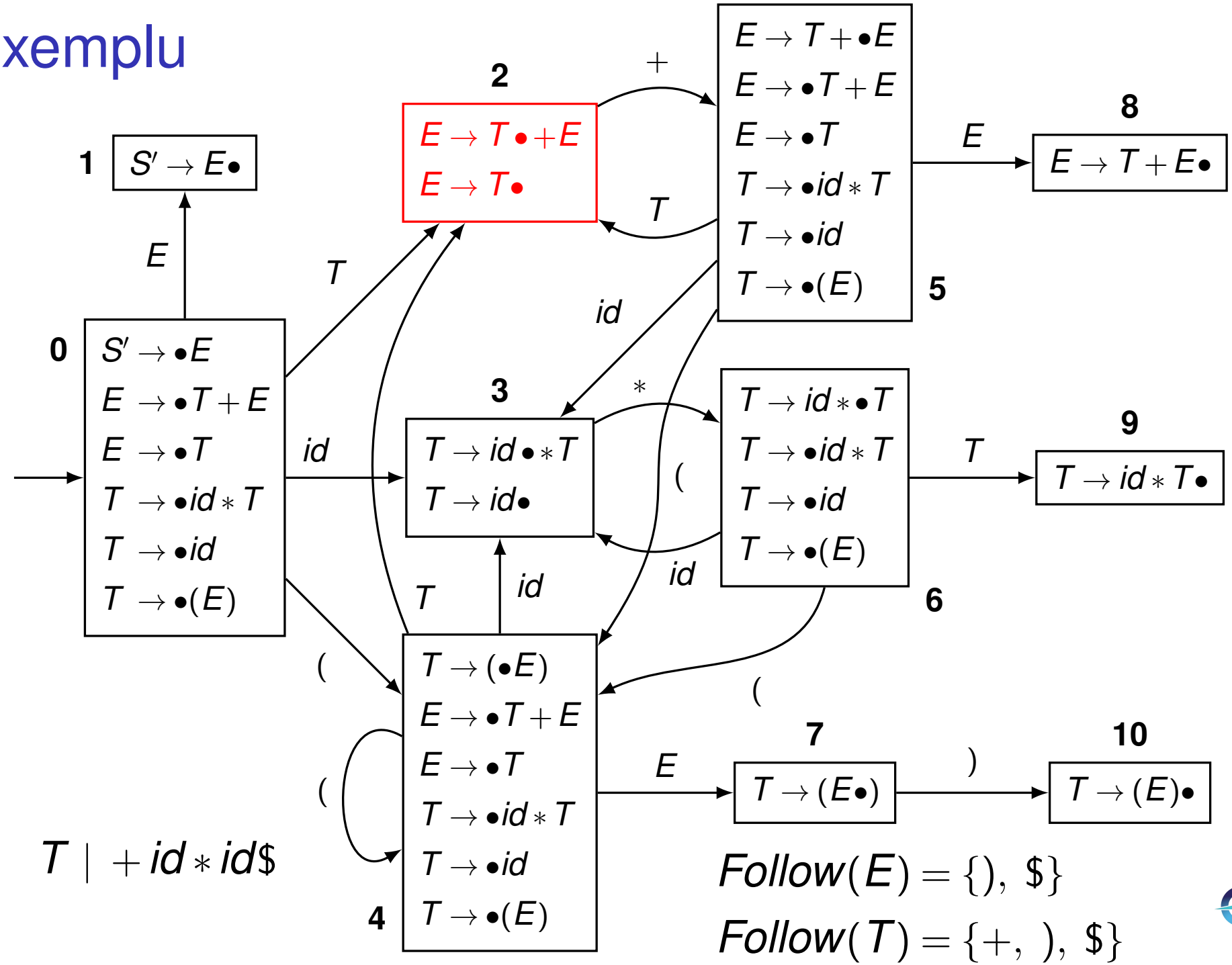


$| \; T + id * id\$$

$Follow(E) = \{), \; \$\}$

$Follow(T) = \{+, \;), \; \$\}$

Exemplu

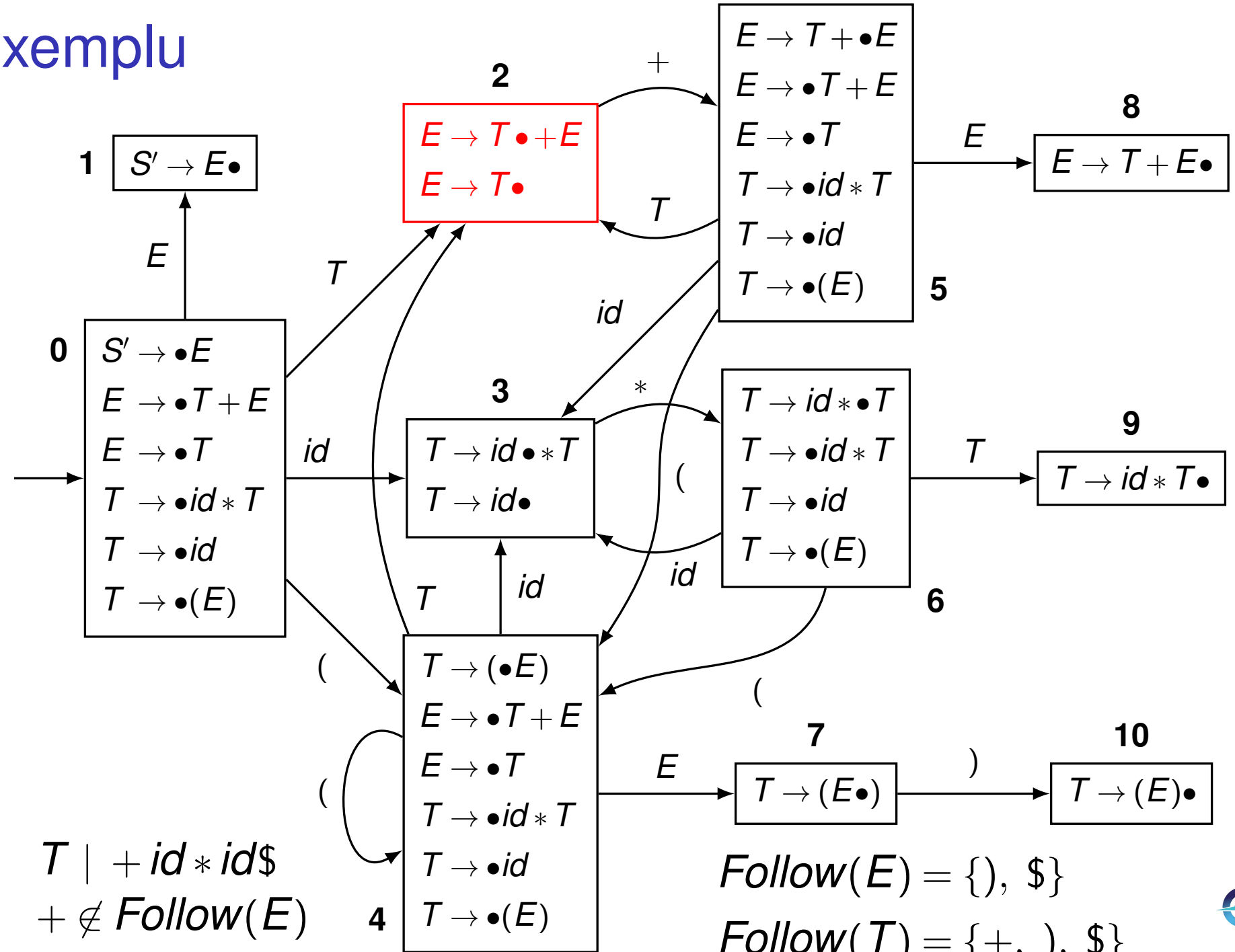


$T \mid +id * id\$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu



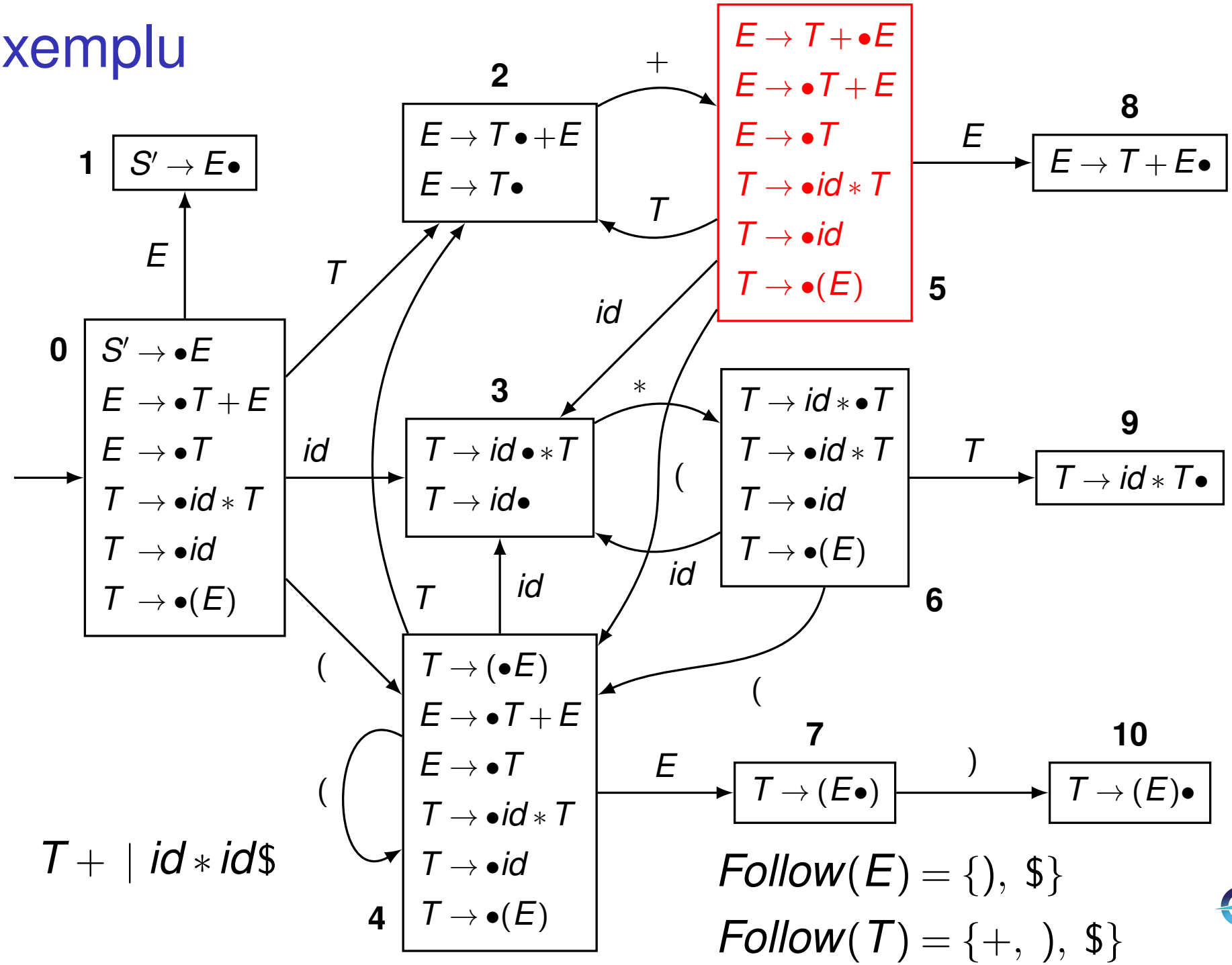
$T \mid +id * id\$$

$+ \notin Follow(E)$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu

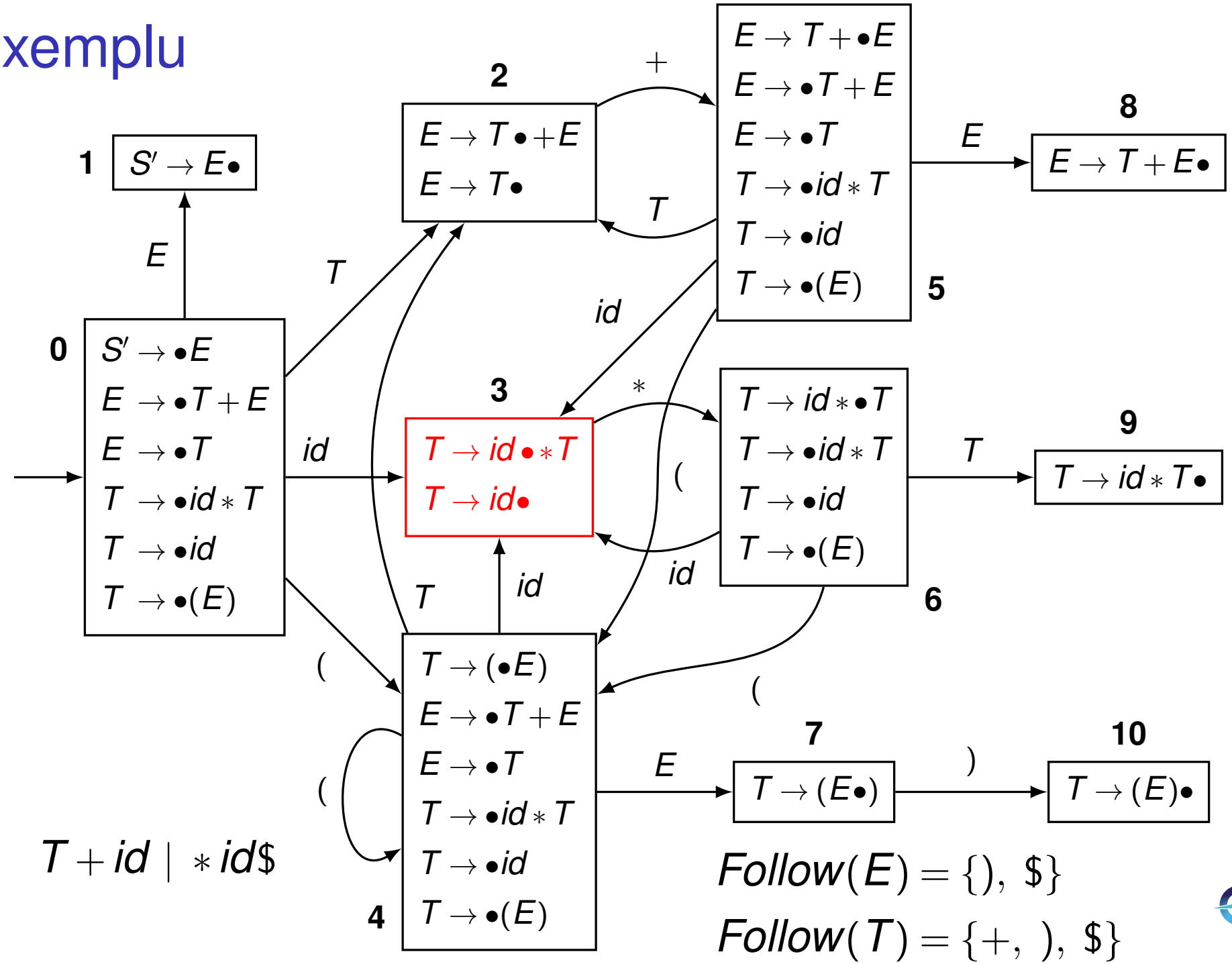


$T + \mid id * id\$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu

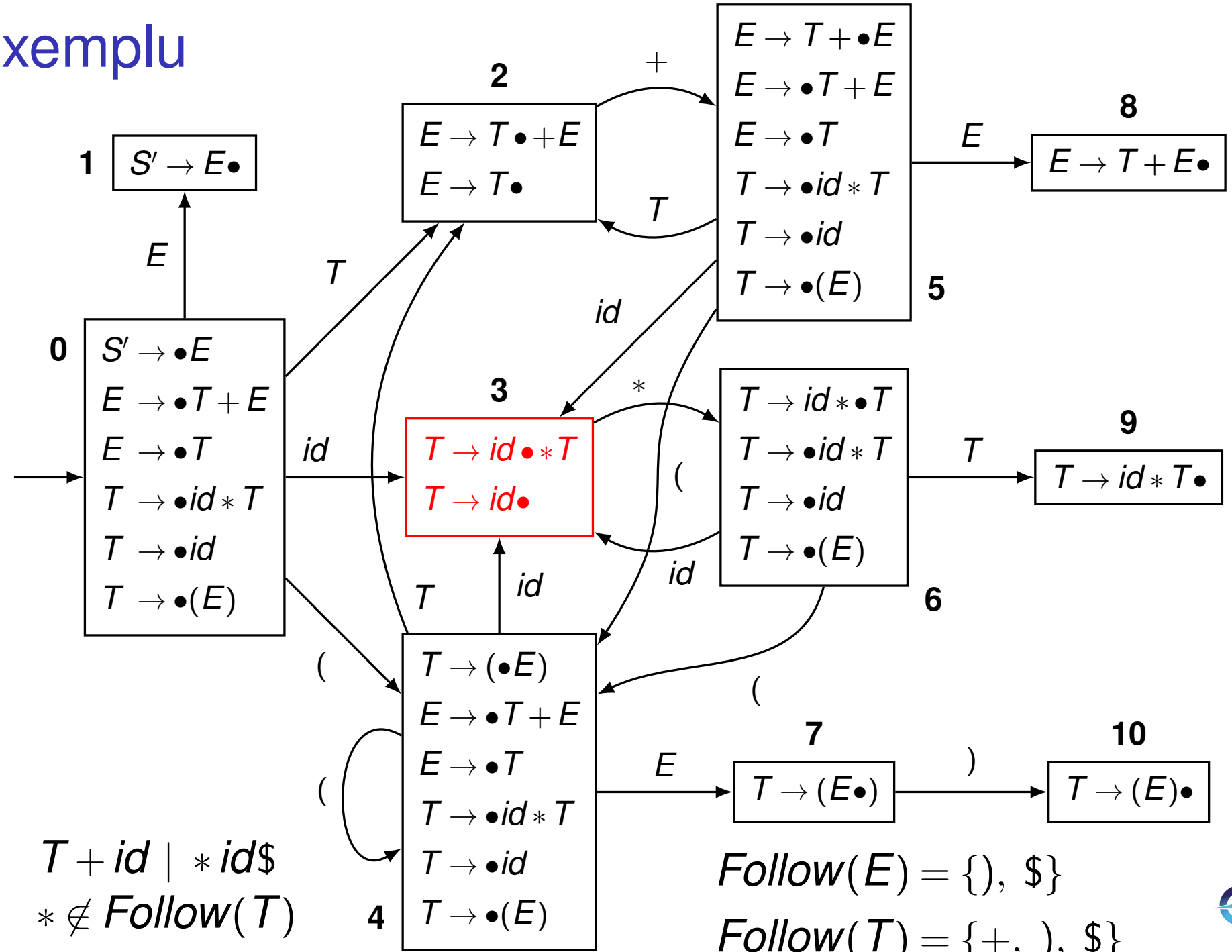


$T + id \mid * id\$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu



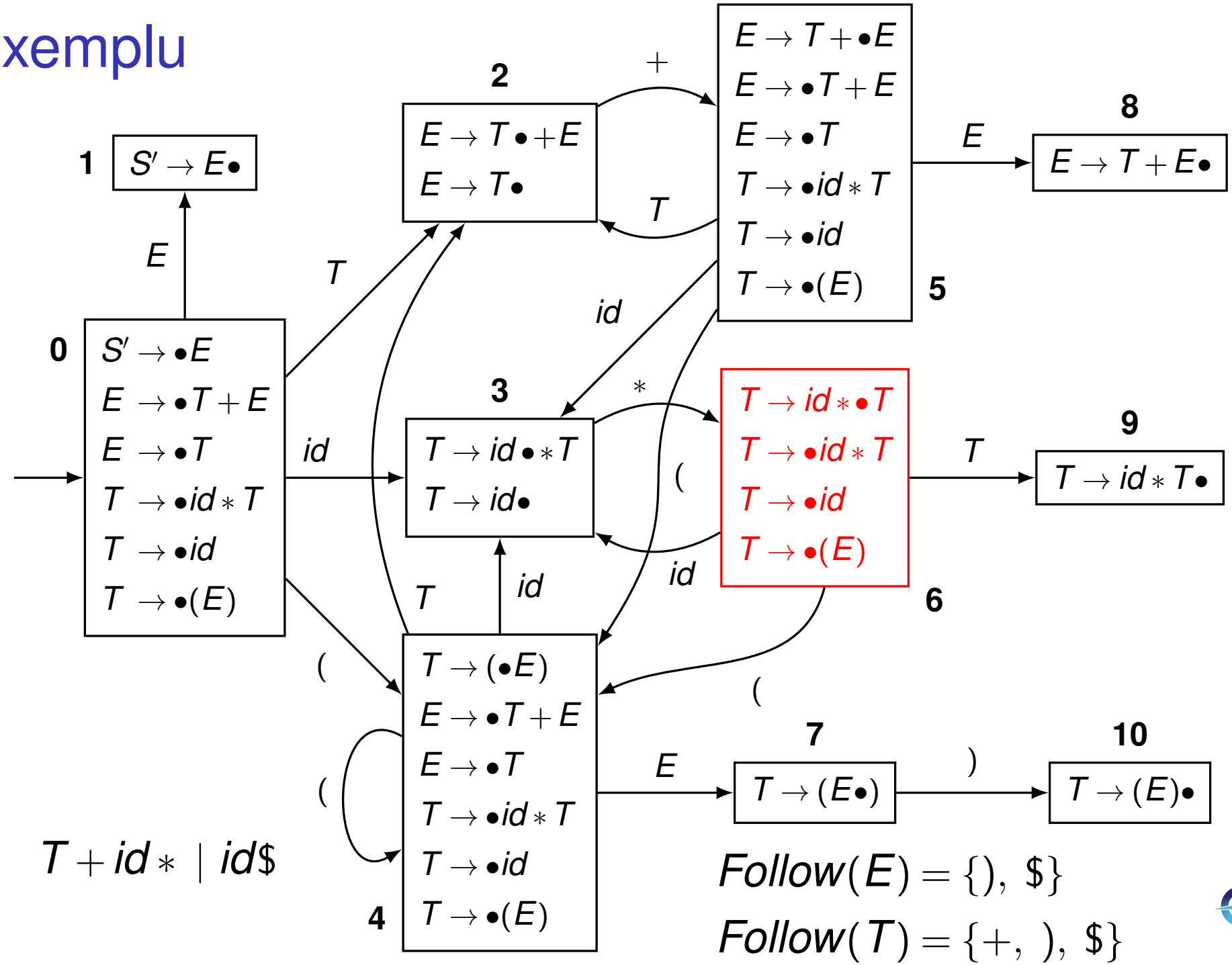
$T + id \mid * id\$$

$* \notin Follow(T)$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu

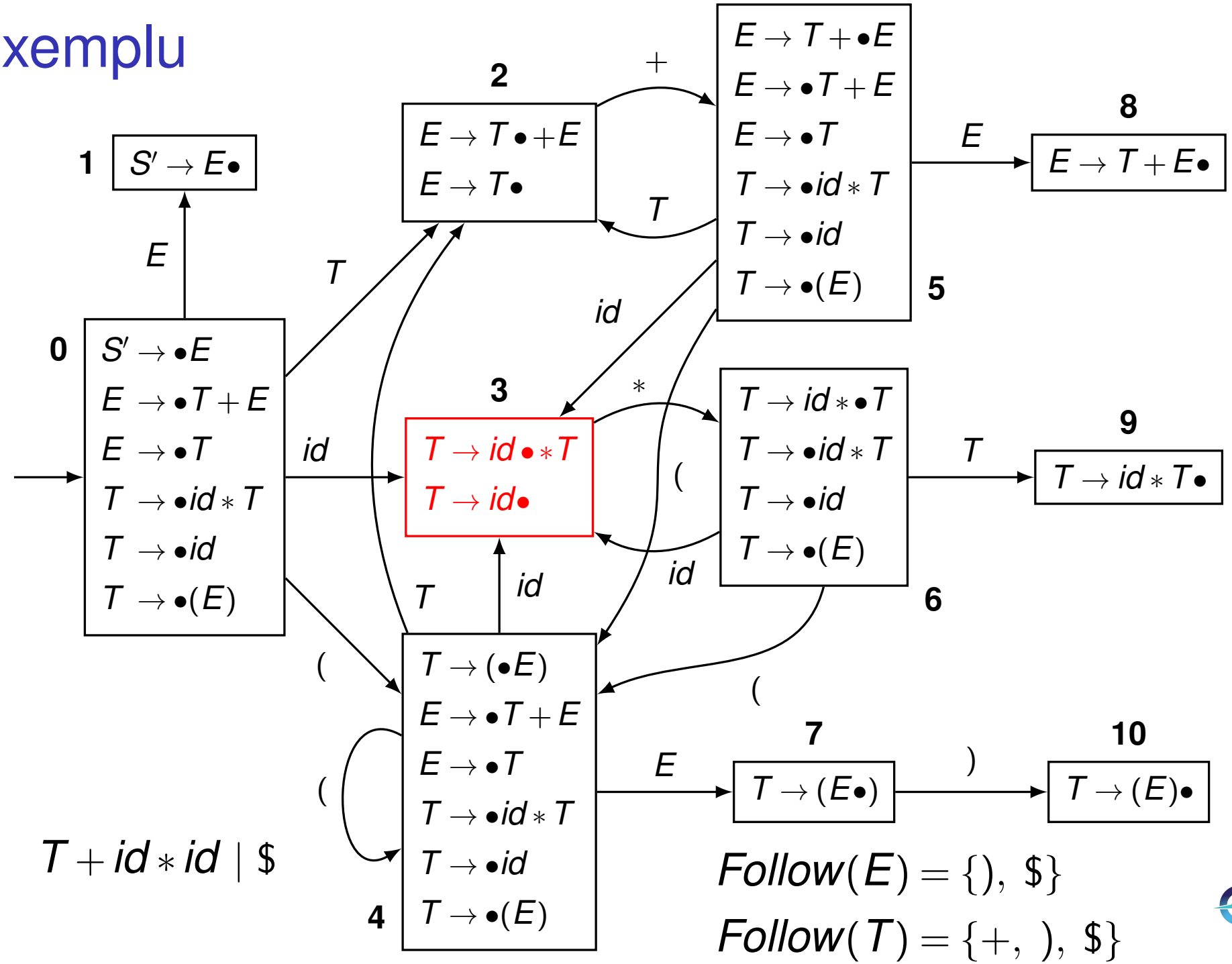


$T + id * \mid id\$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu

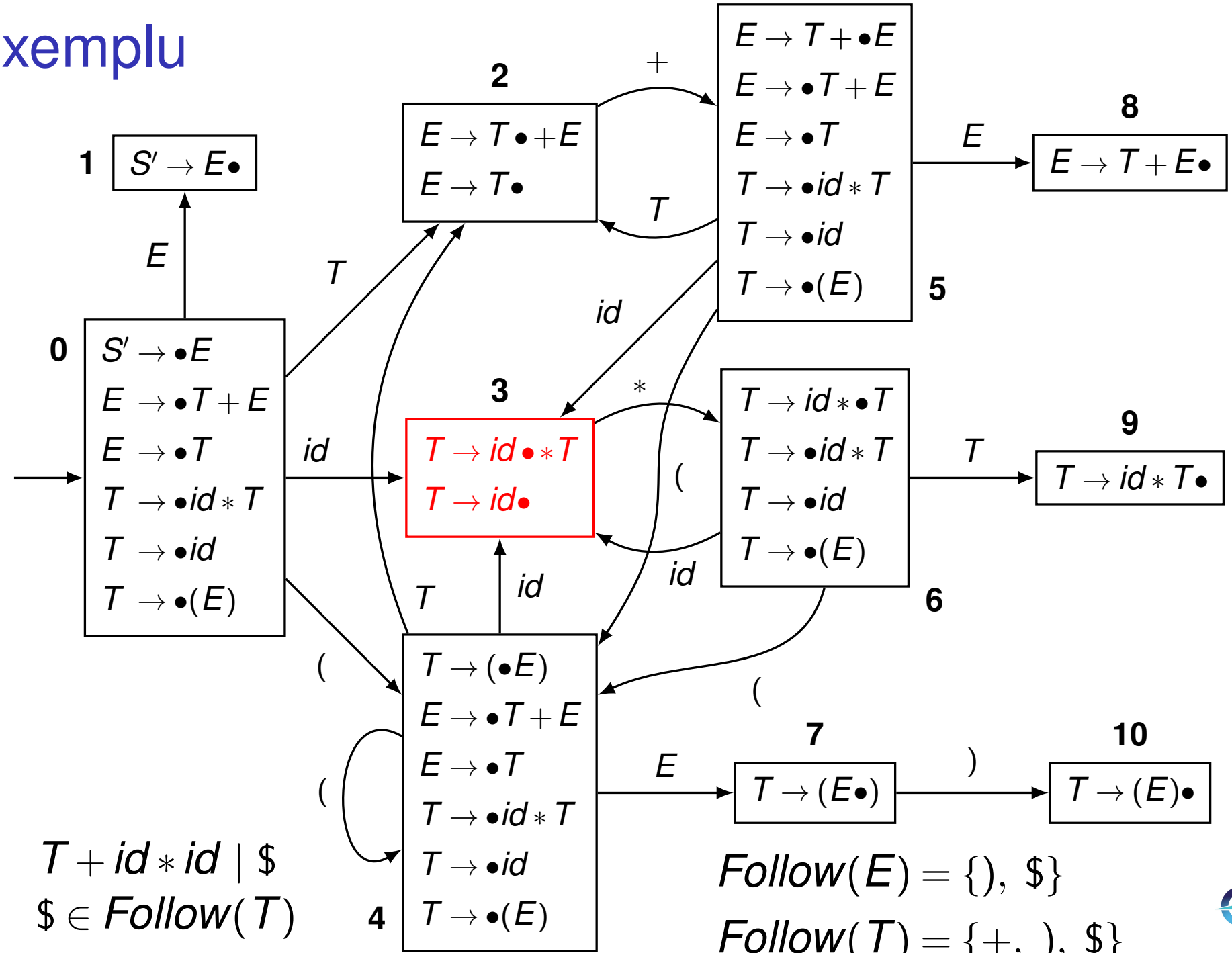


$T + id * id \mid \$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu



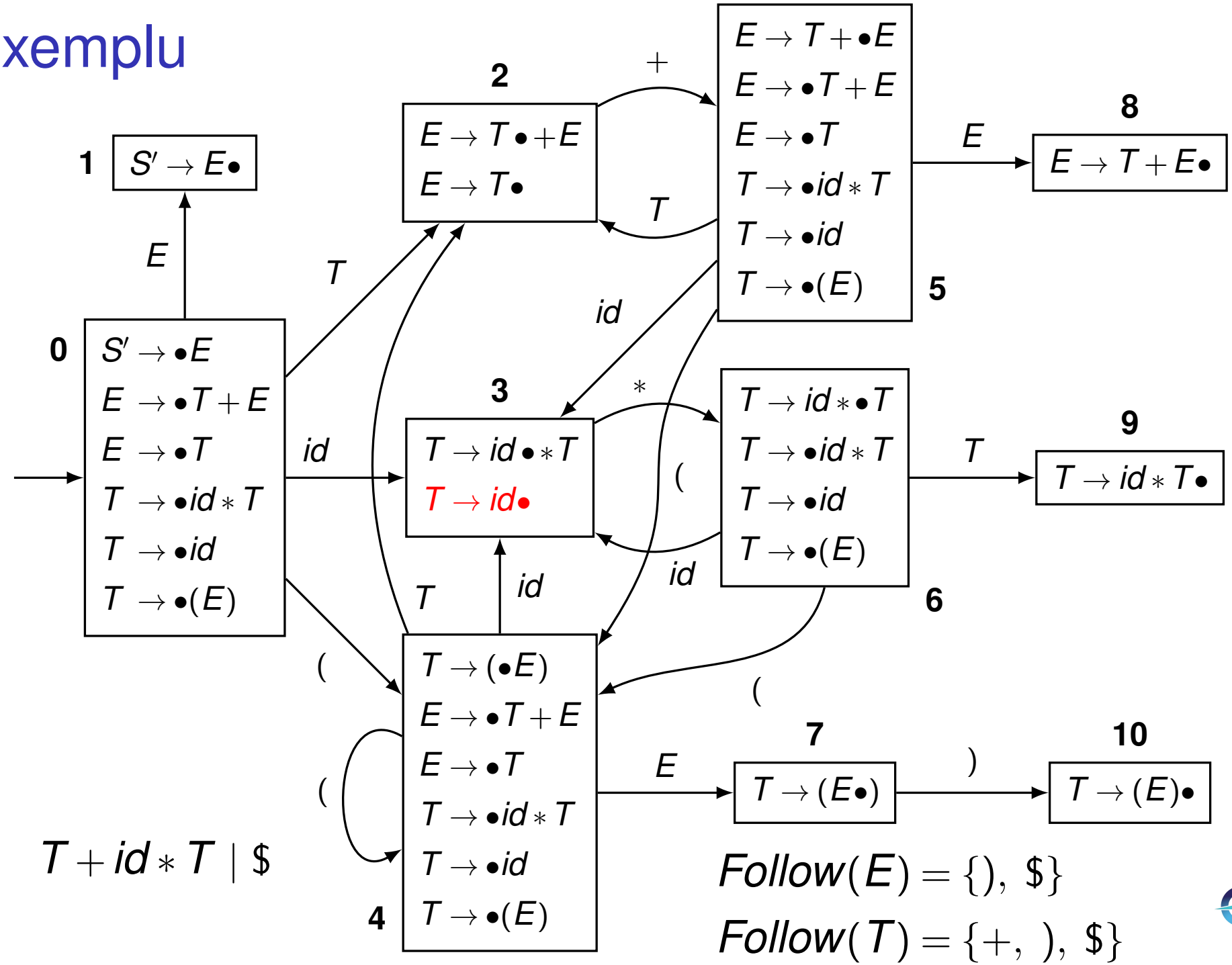
$T + id * id \mid \$$

$\$ \in Follow(T)$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu

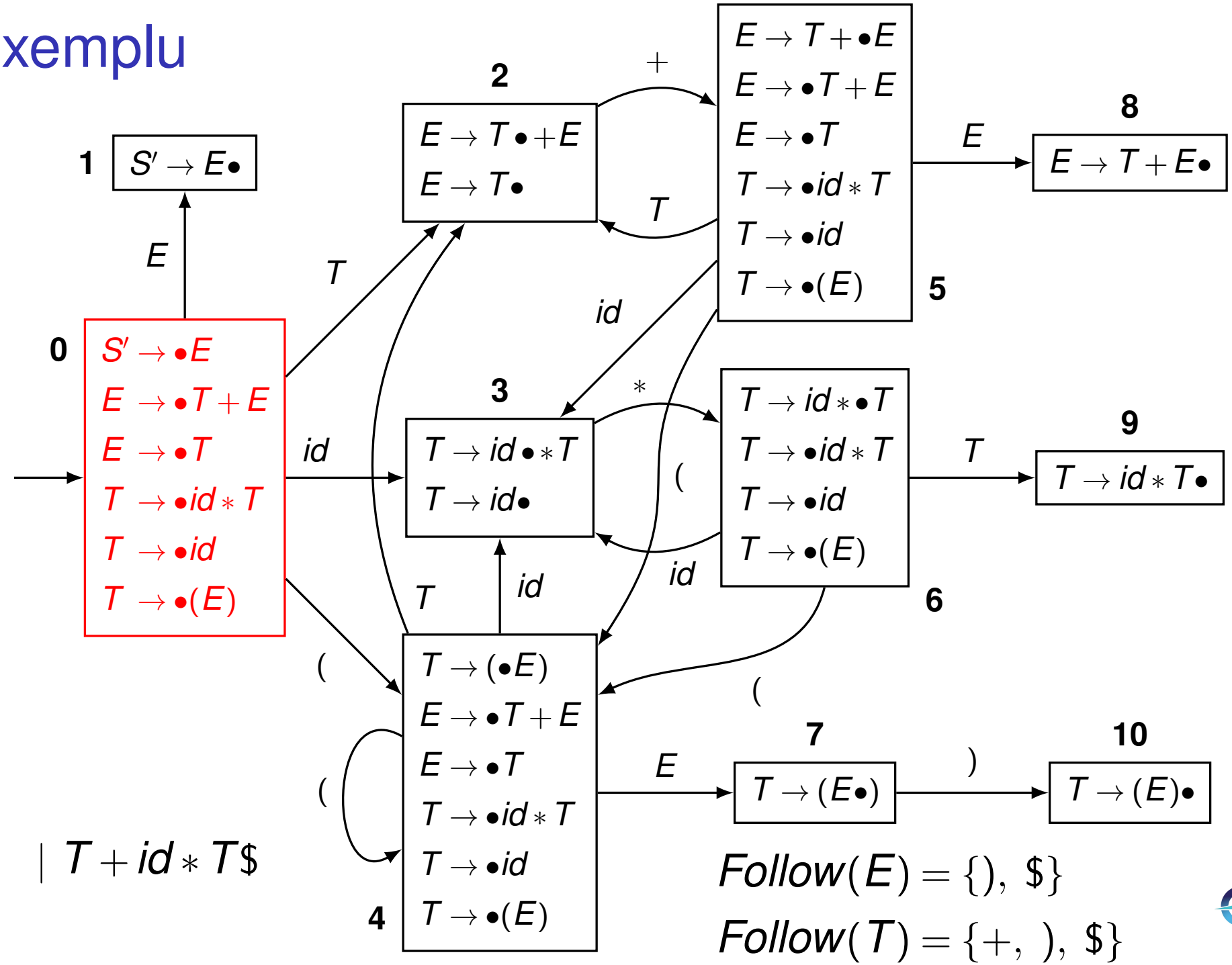


$T + id * T \mid \$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu

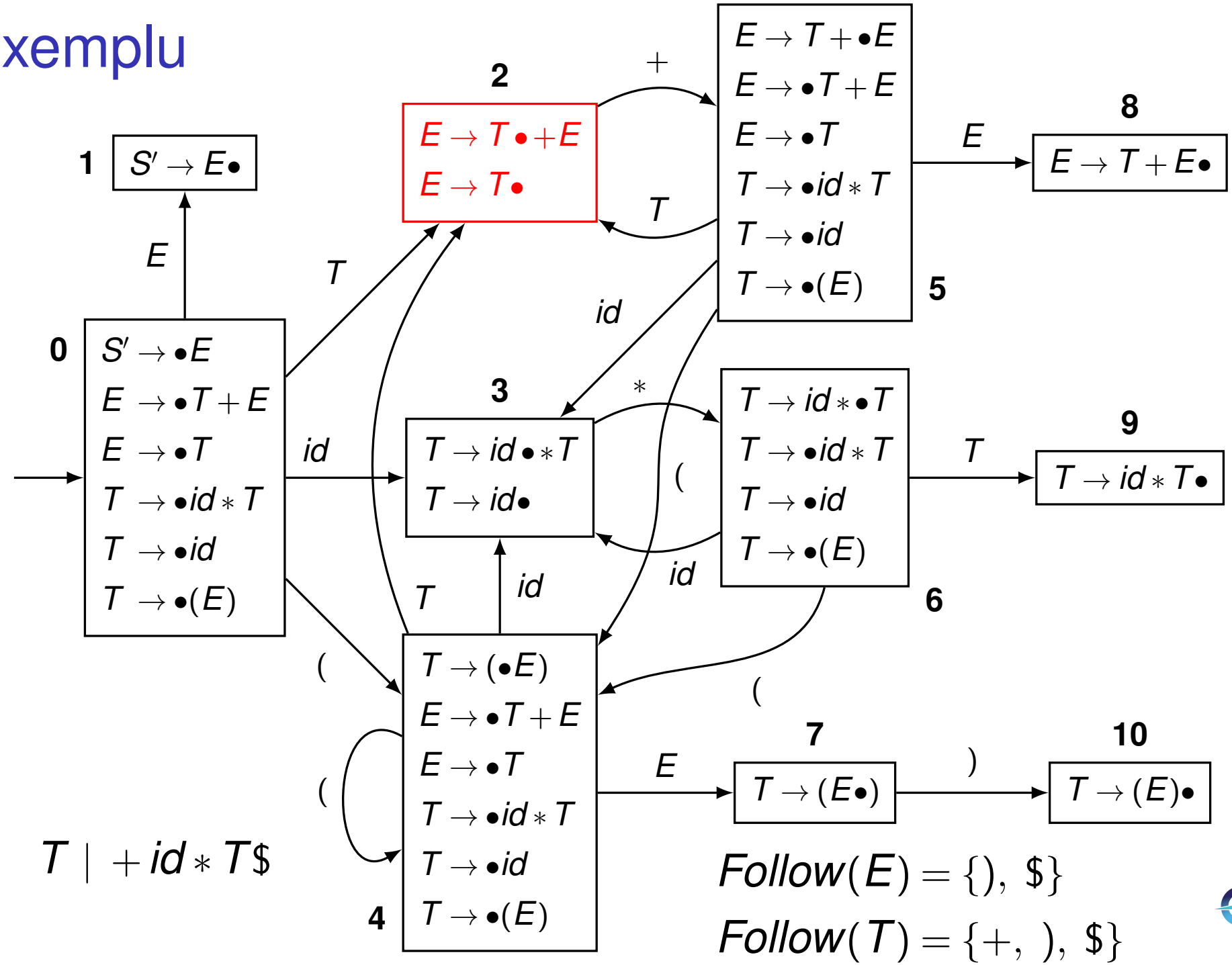


$| \; T + id * T\$$

$Follow(E) = \{), \; \$\}$

$Follow(T) = \{+, \;), \; \$\}$

Exemplu

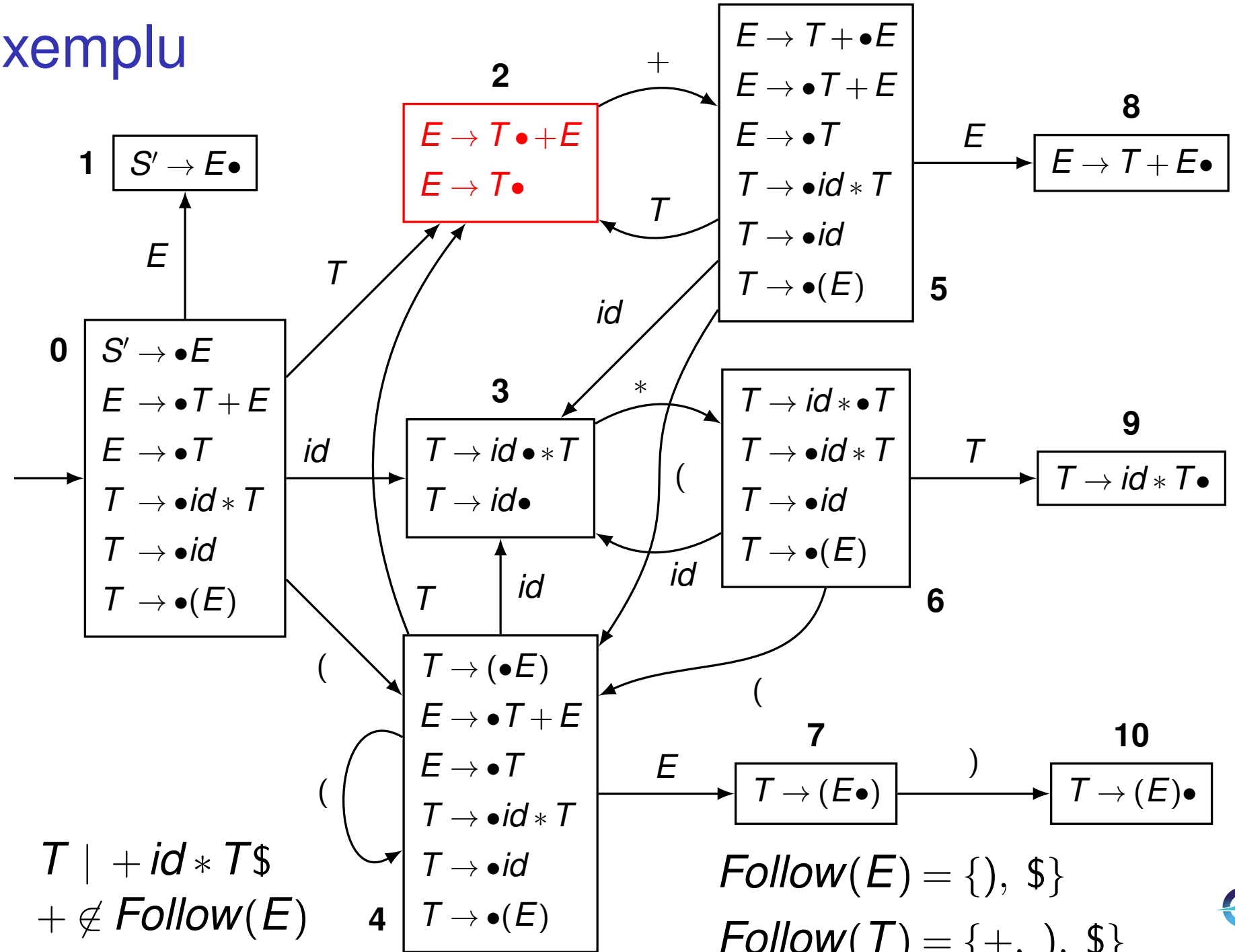


$T \mid +id * T\$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu



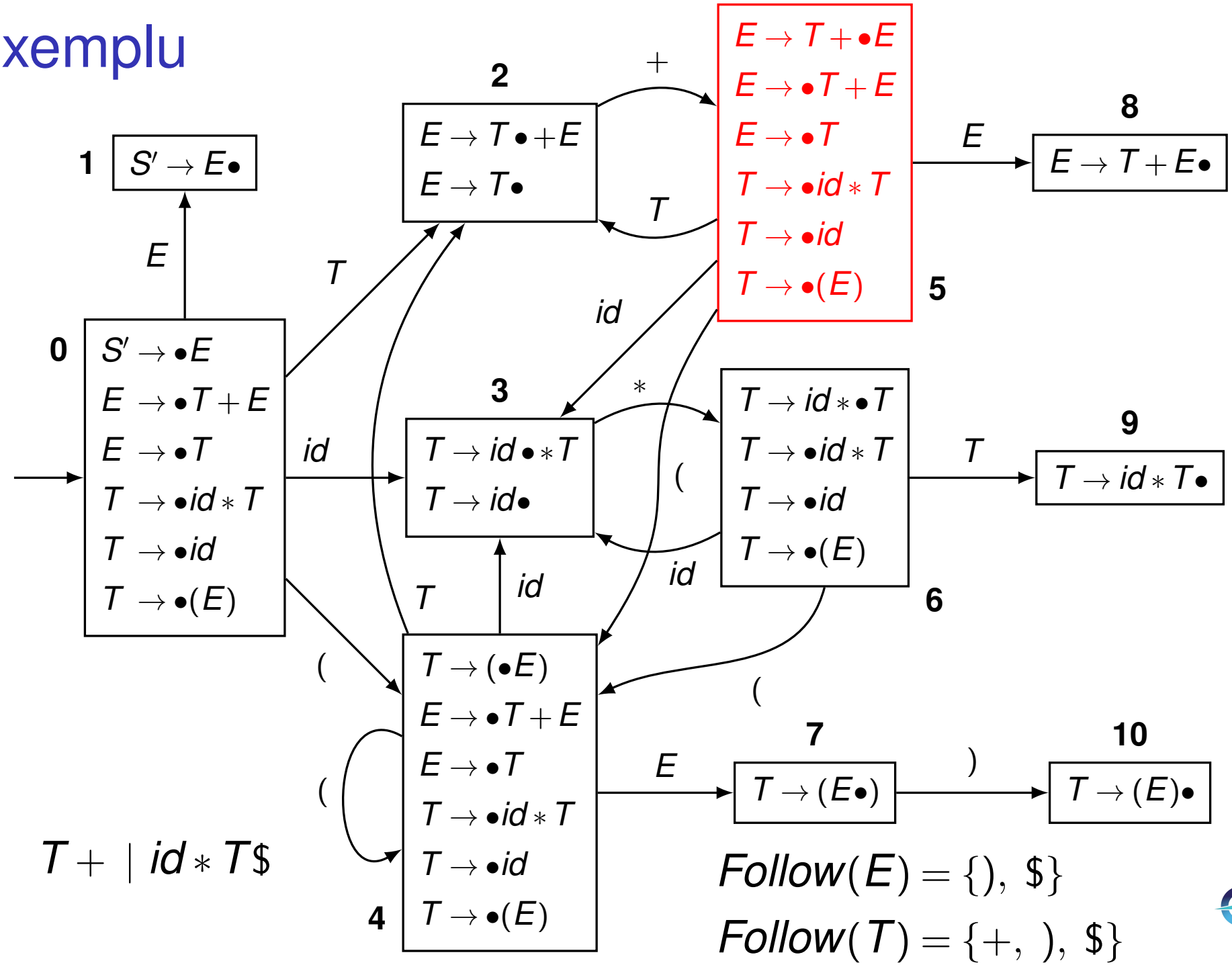
$T \mid +id * T\$$

$+ \notin Follow(E)$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu

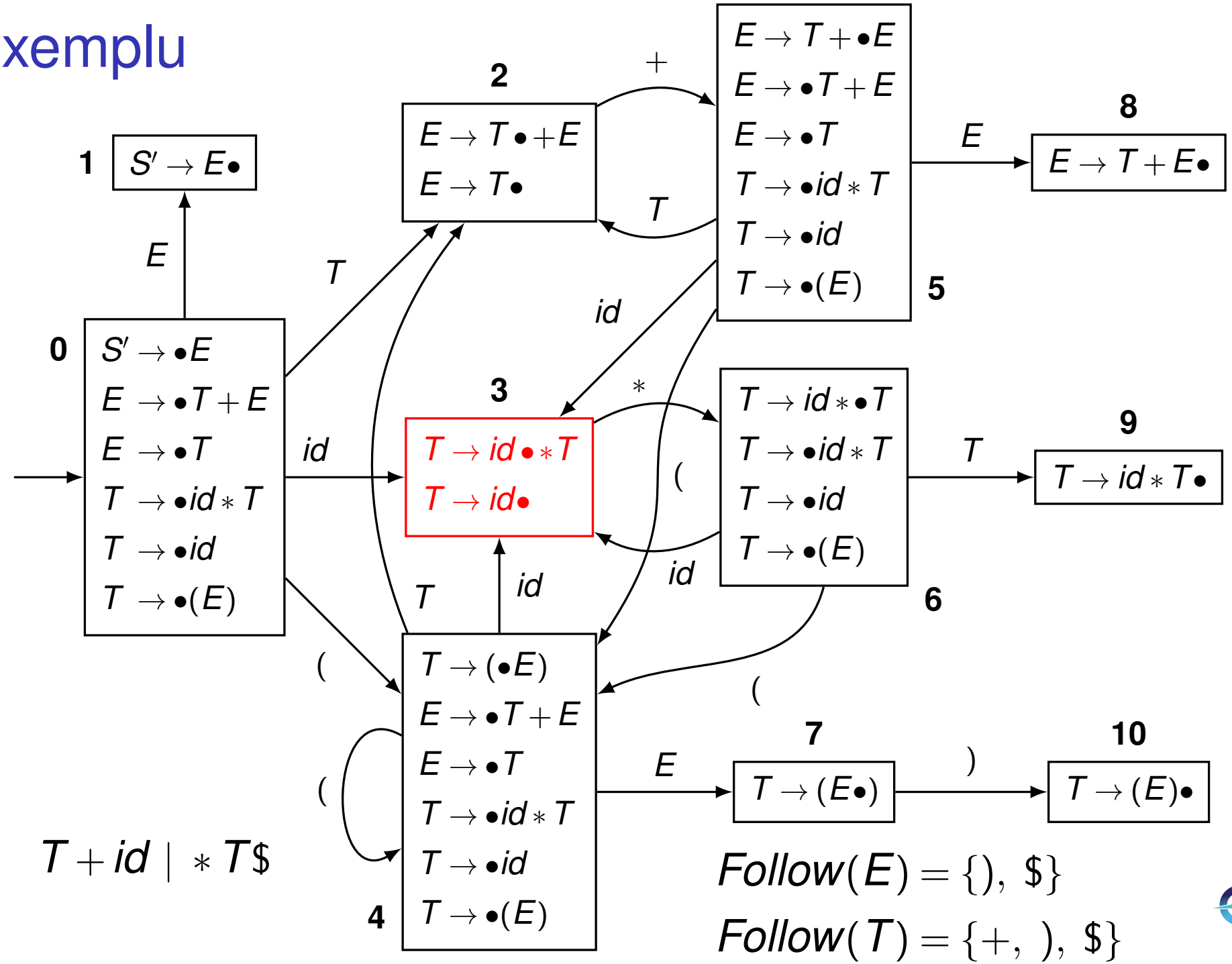


$T + \mid id * T\$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu

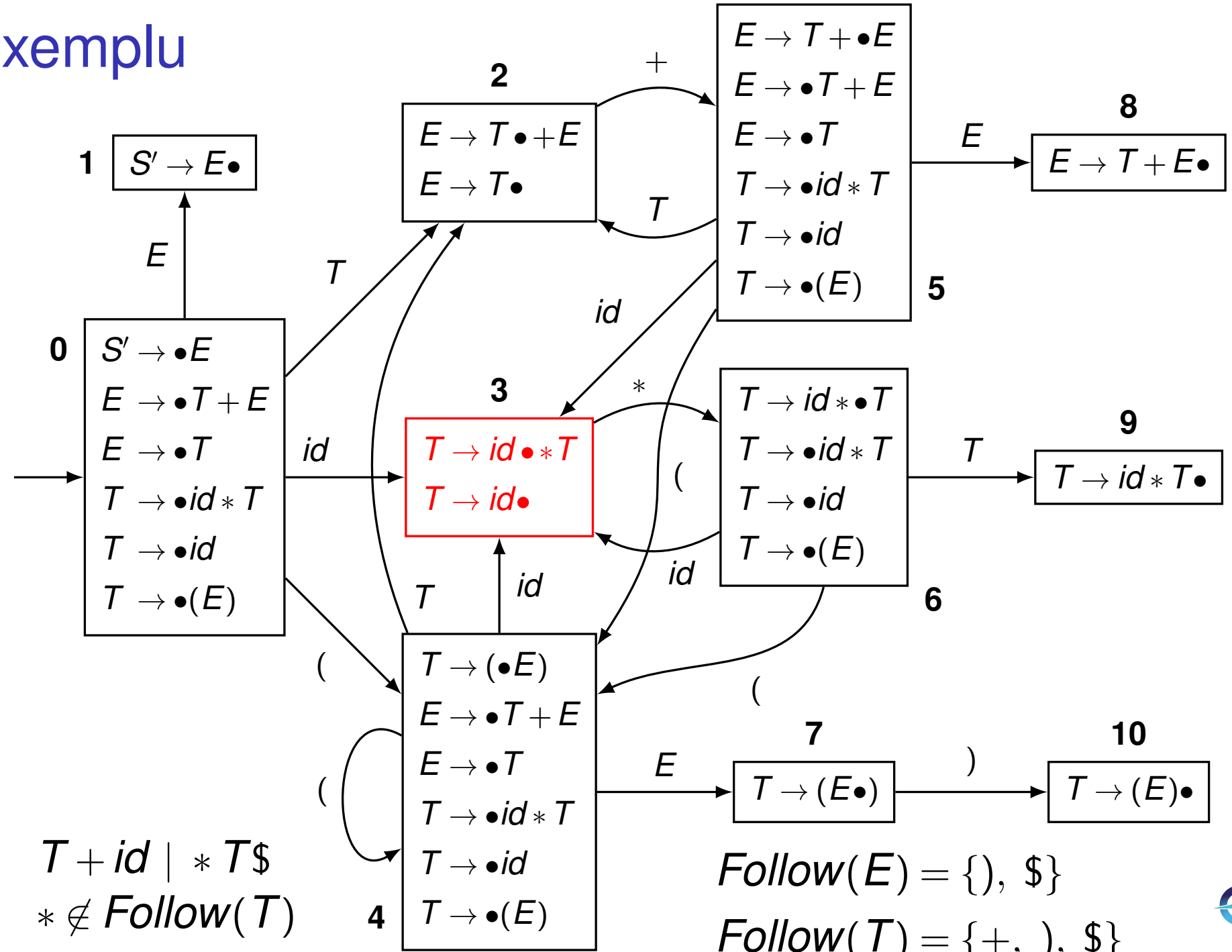


$T + id \mid * T\$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu



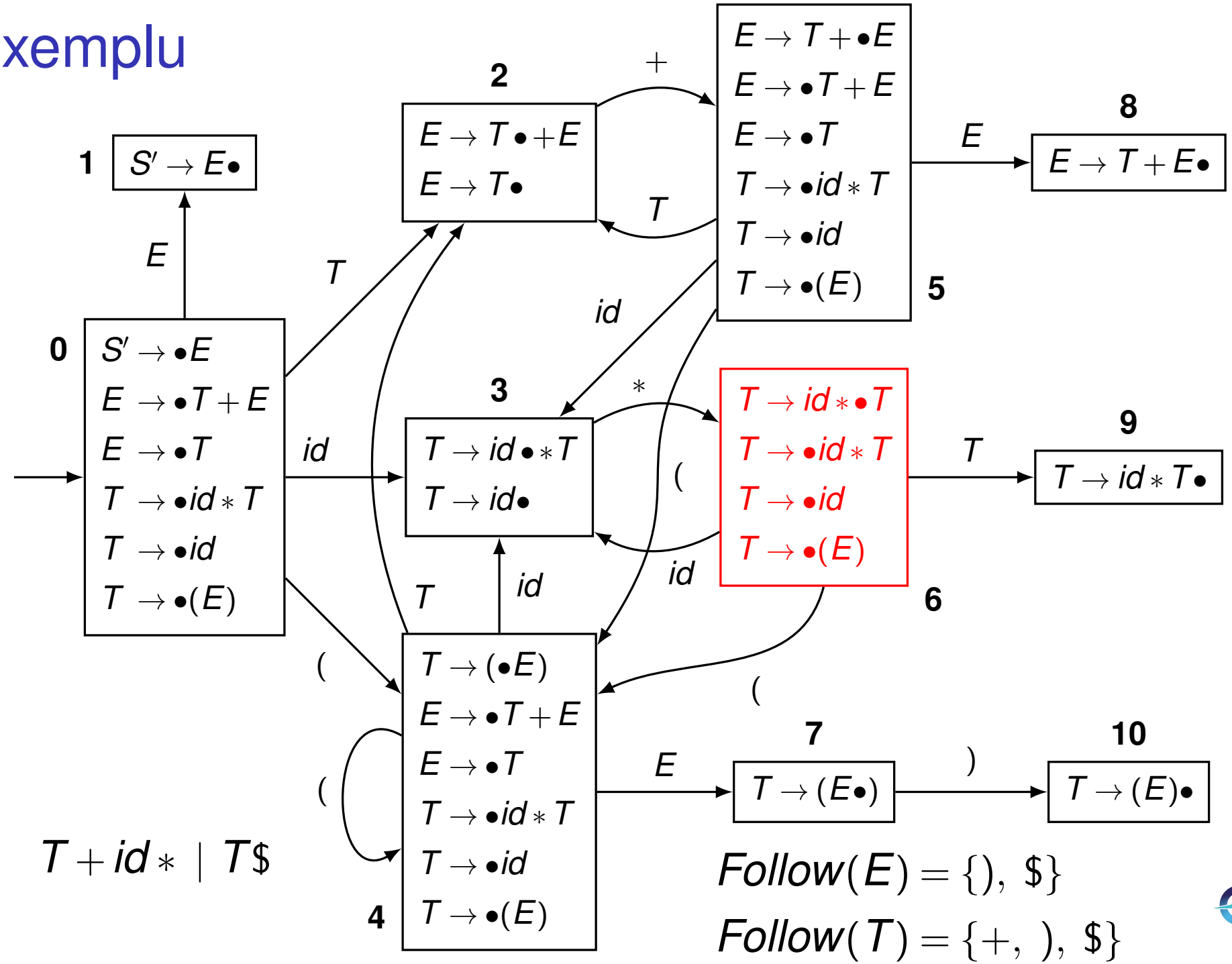
$T + id \mid * T\$$

$* \notin Follow(T)$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu

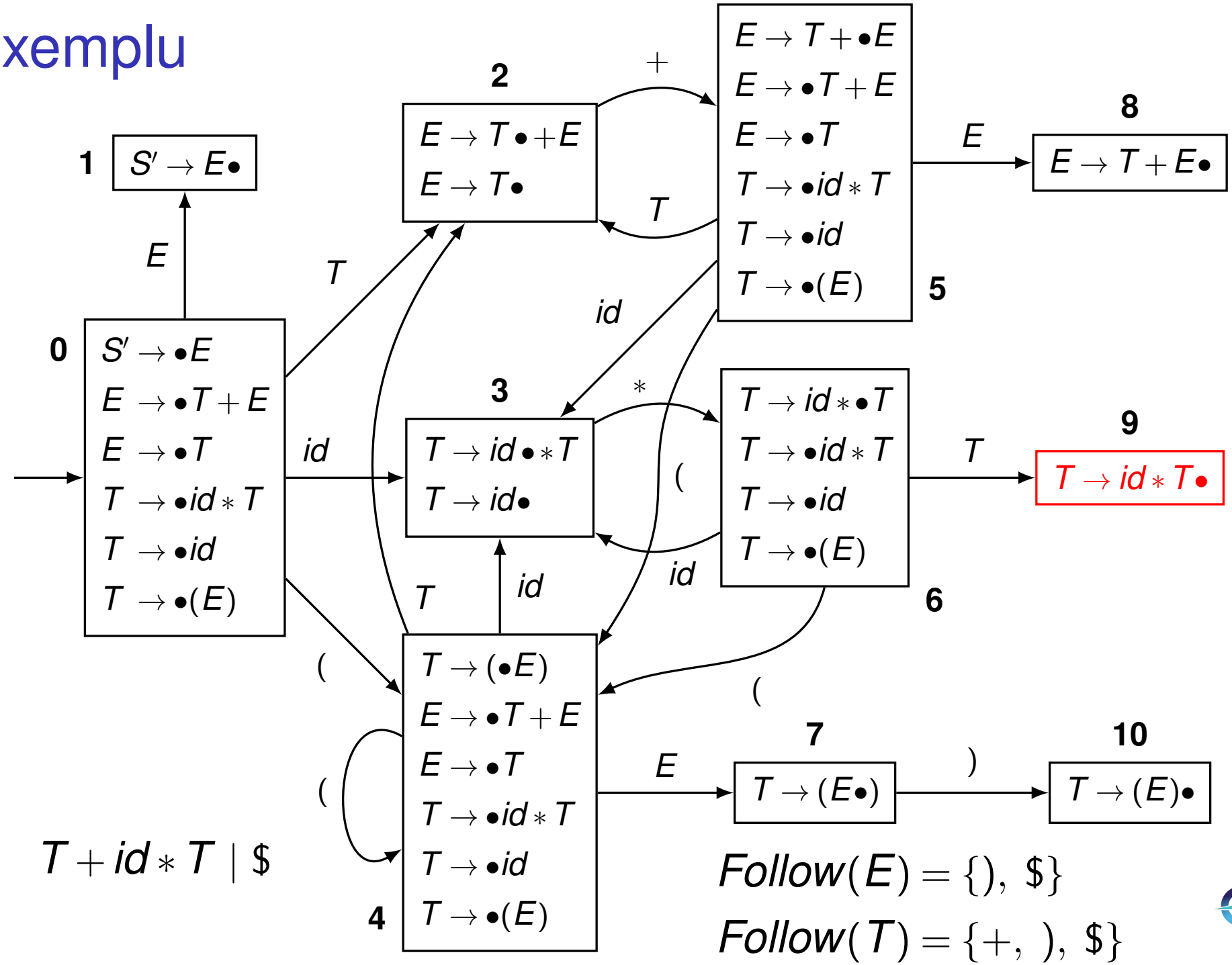


$T + id * \mid T\$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu

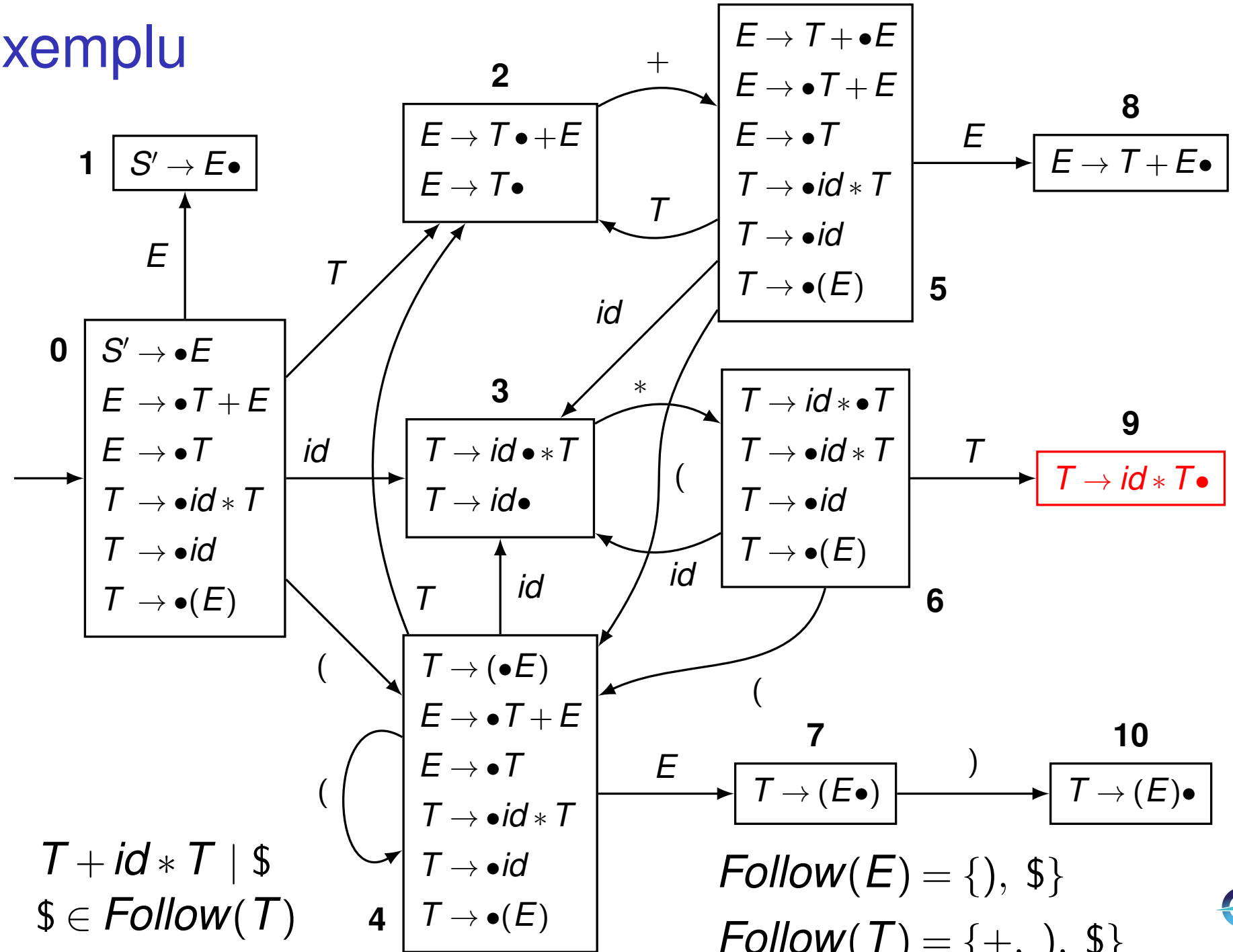


$T + id * T \mid \$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu



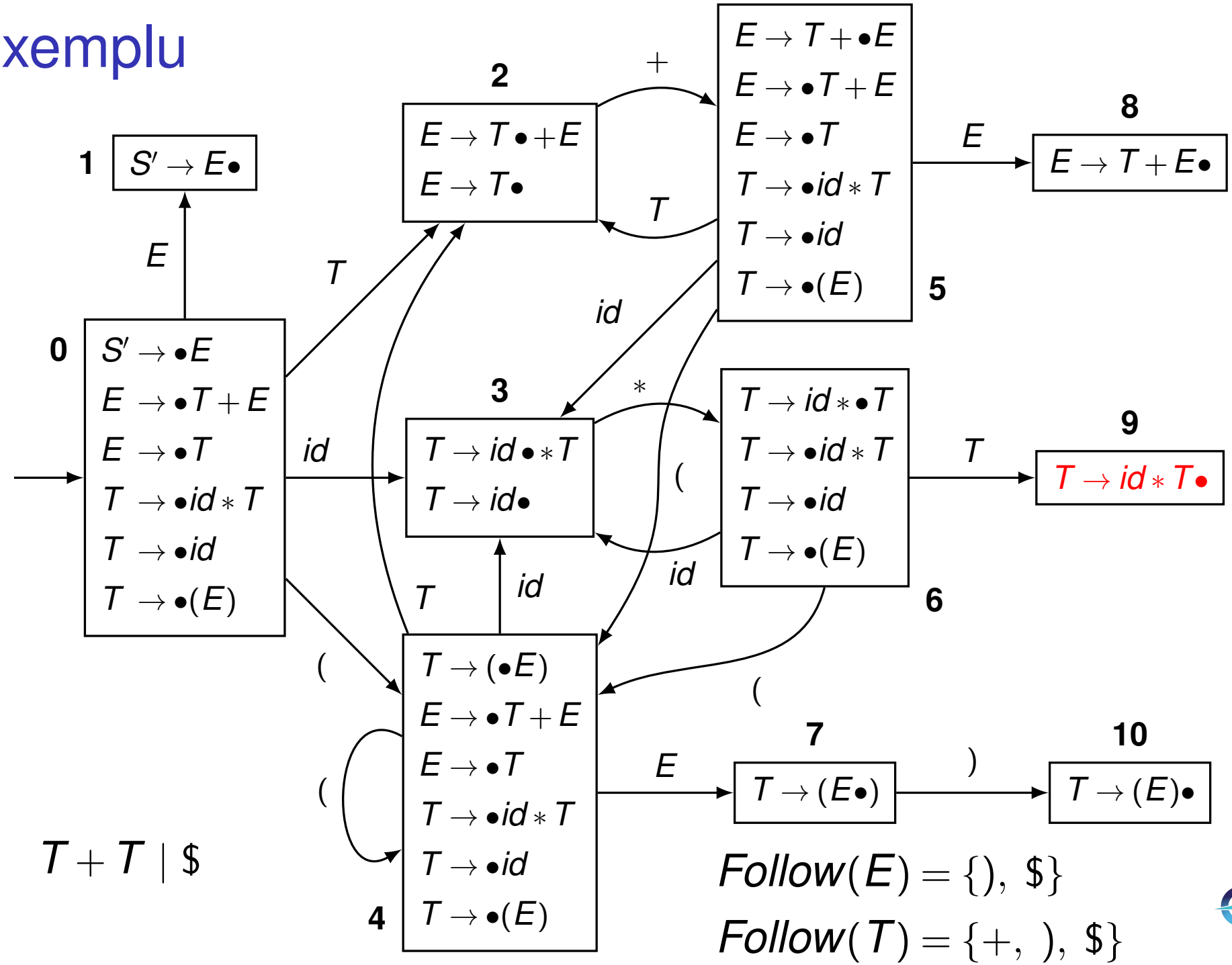
$T + id * T \mid \$$

$\$ \in Follow(T)$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu

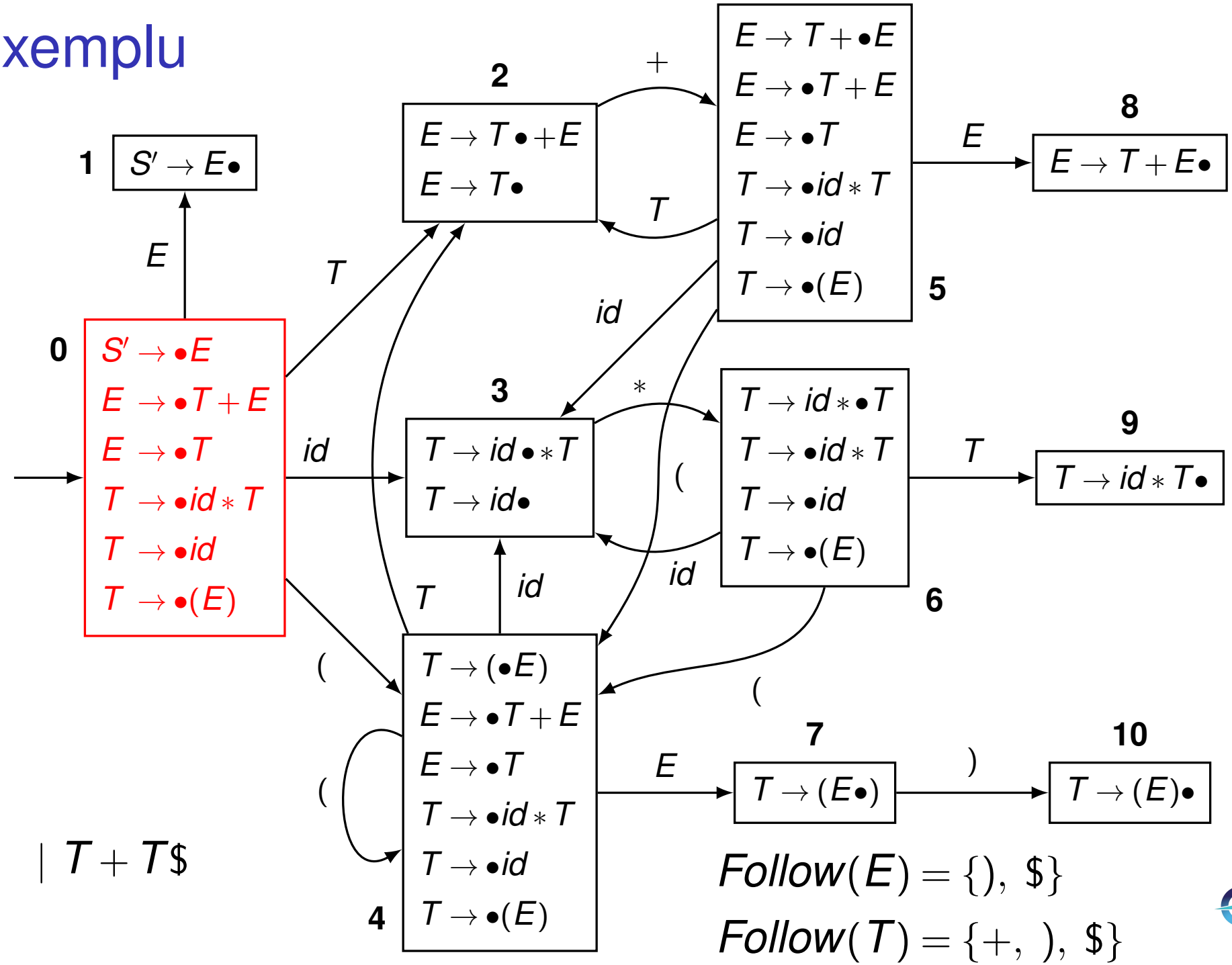


$T + T \mid \$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu

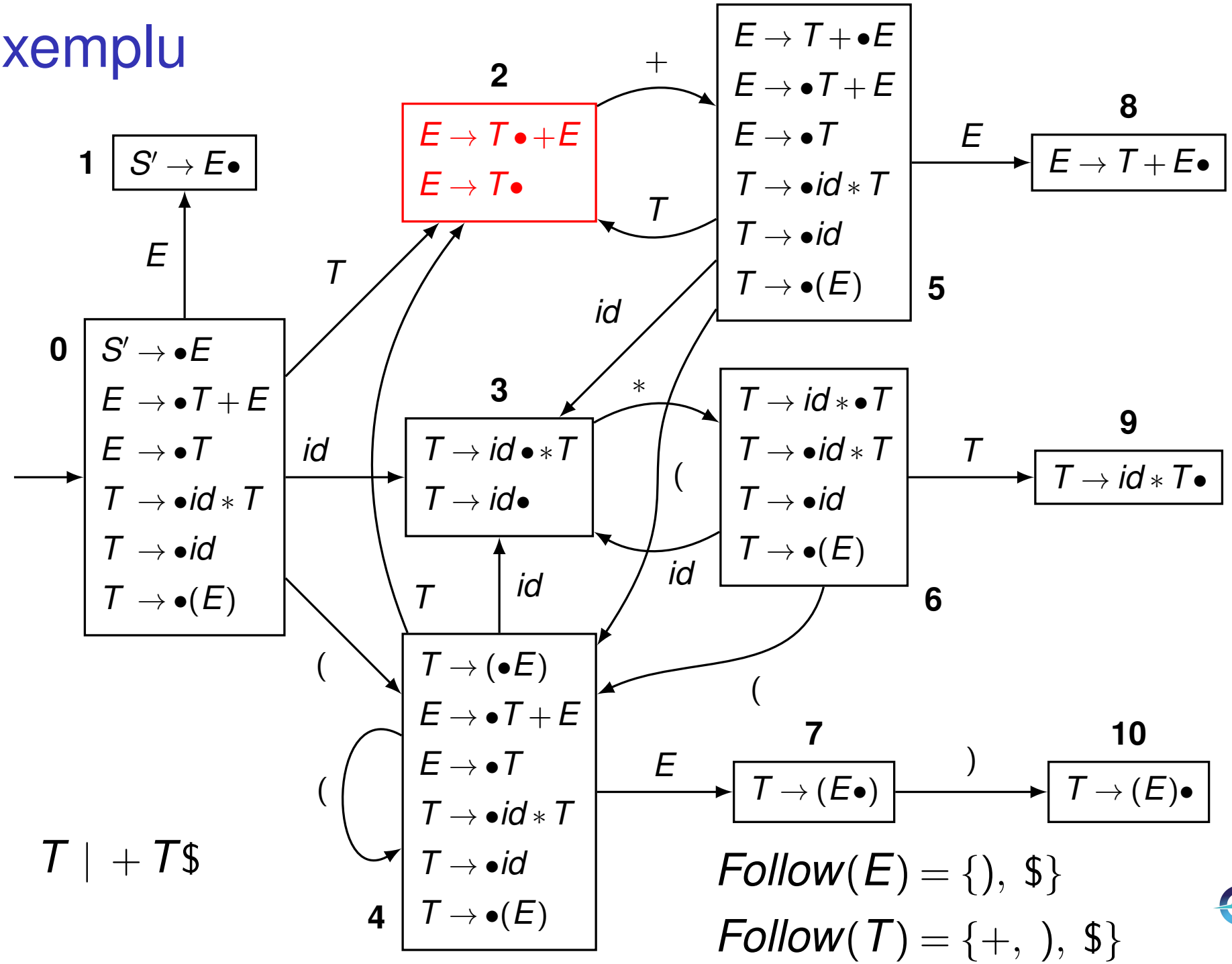


$| \; T + T\$$

$Follow(E) = \{), \; \$\}$

$Follow(T) = \{+, \;), \; \$\}$

Exemplu

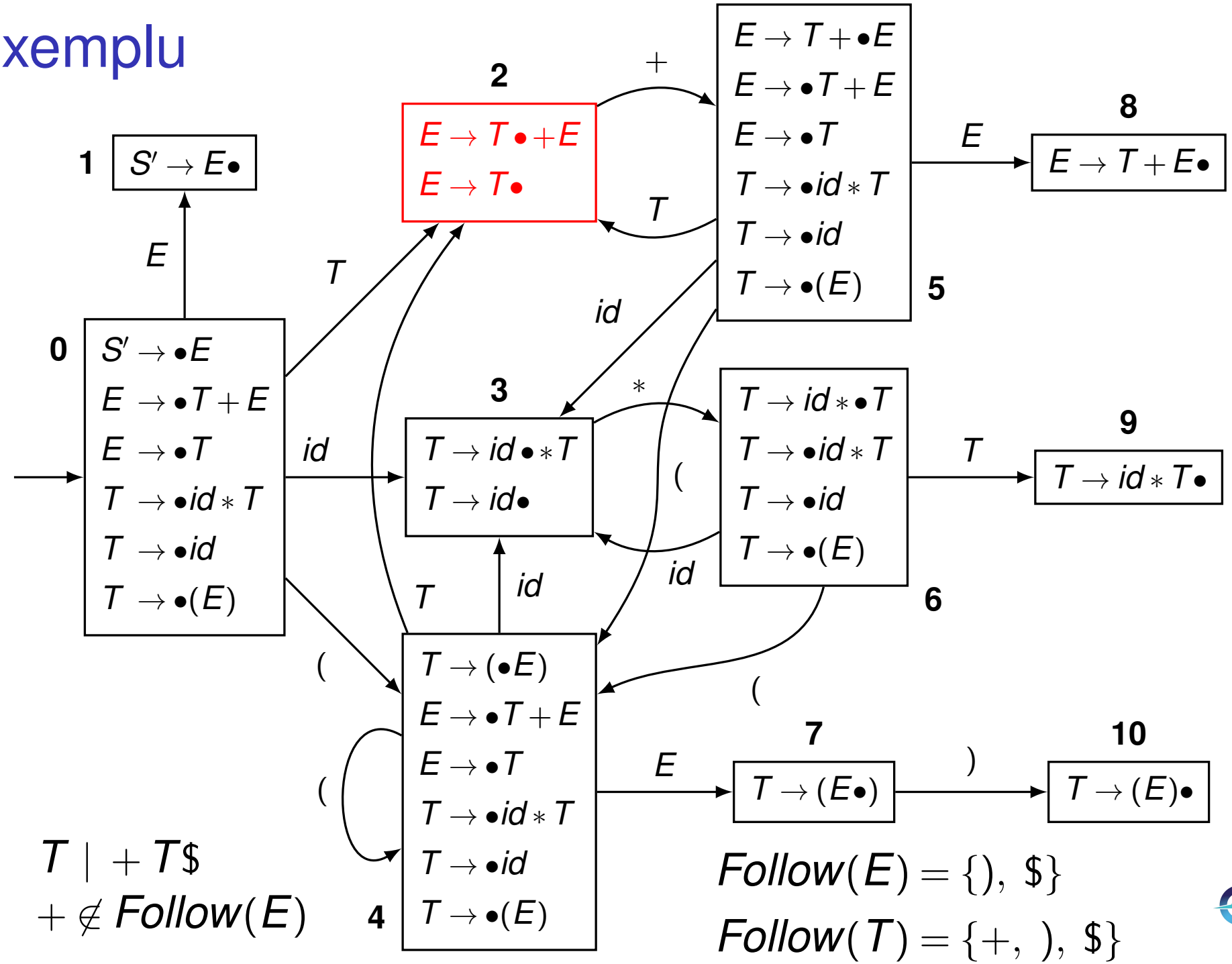


$T \mid +T\$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu



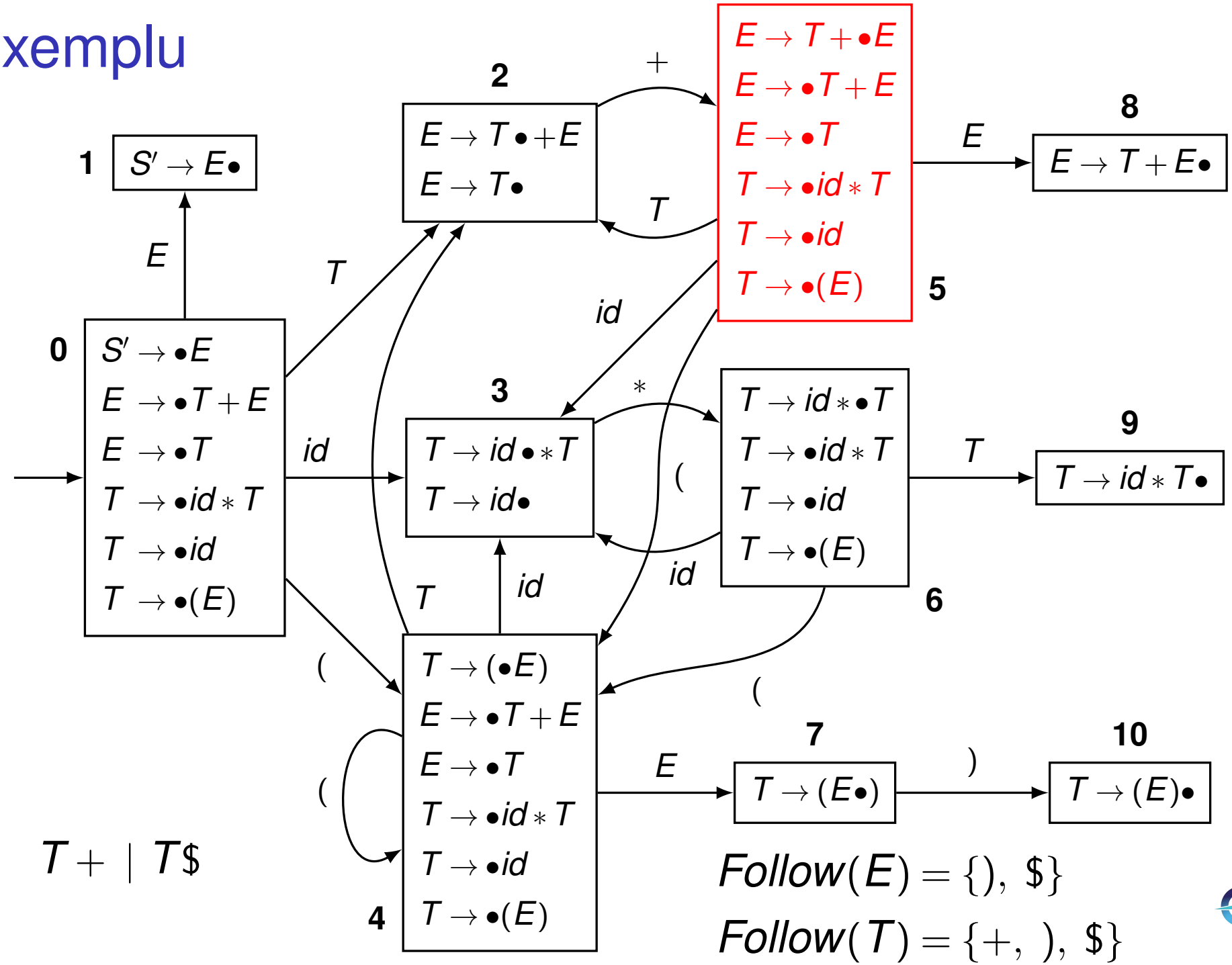
$T \mid +T\$$

$+ \notin Follow(E)$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu

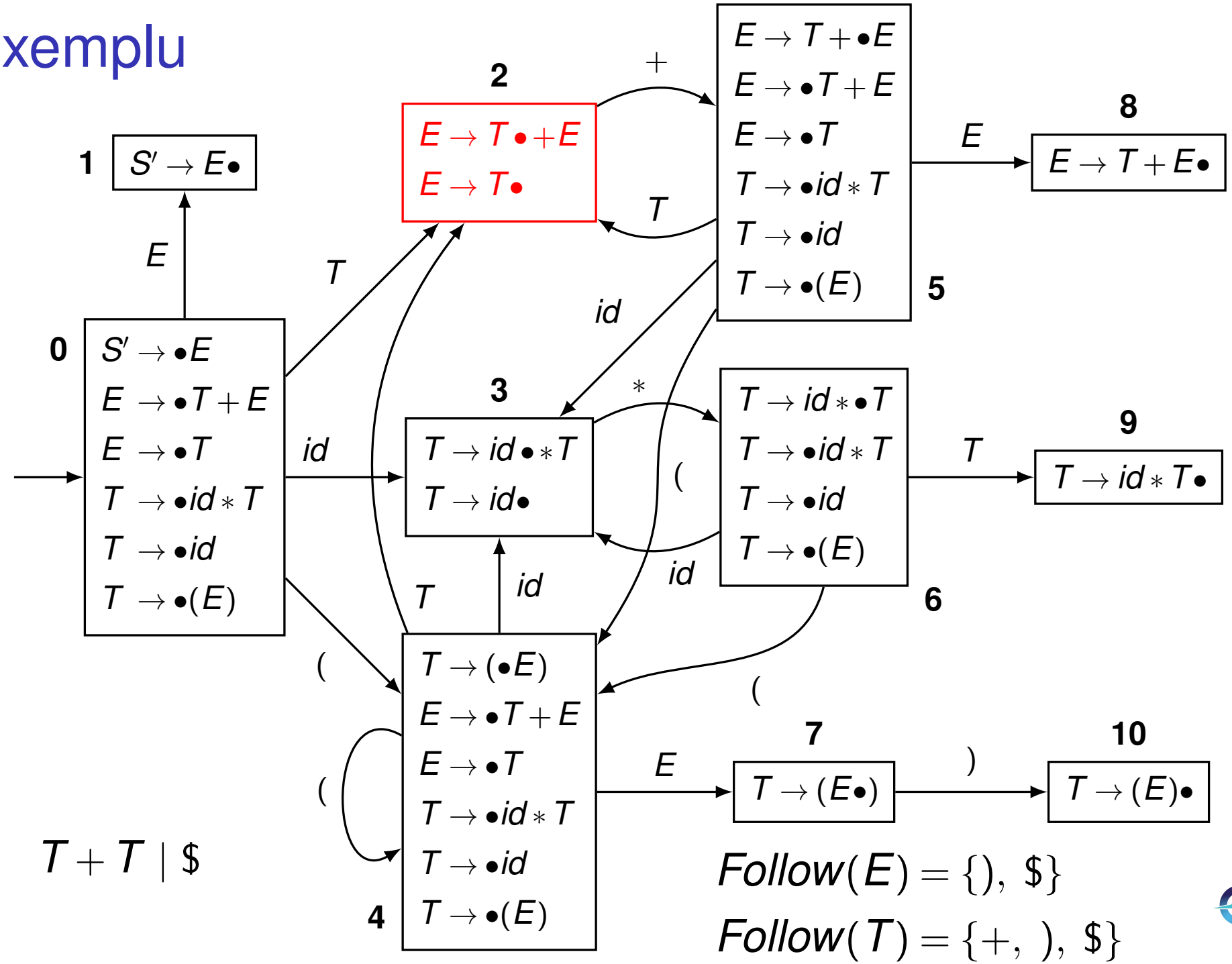


$T+ \mid T\$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu

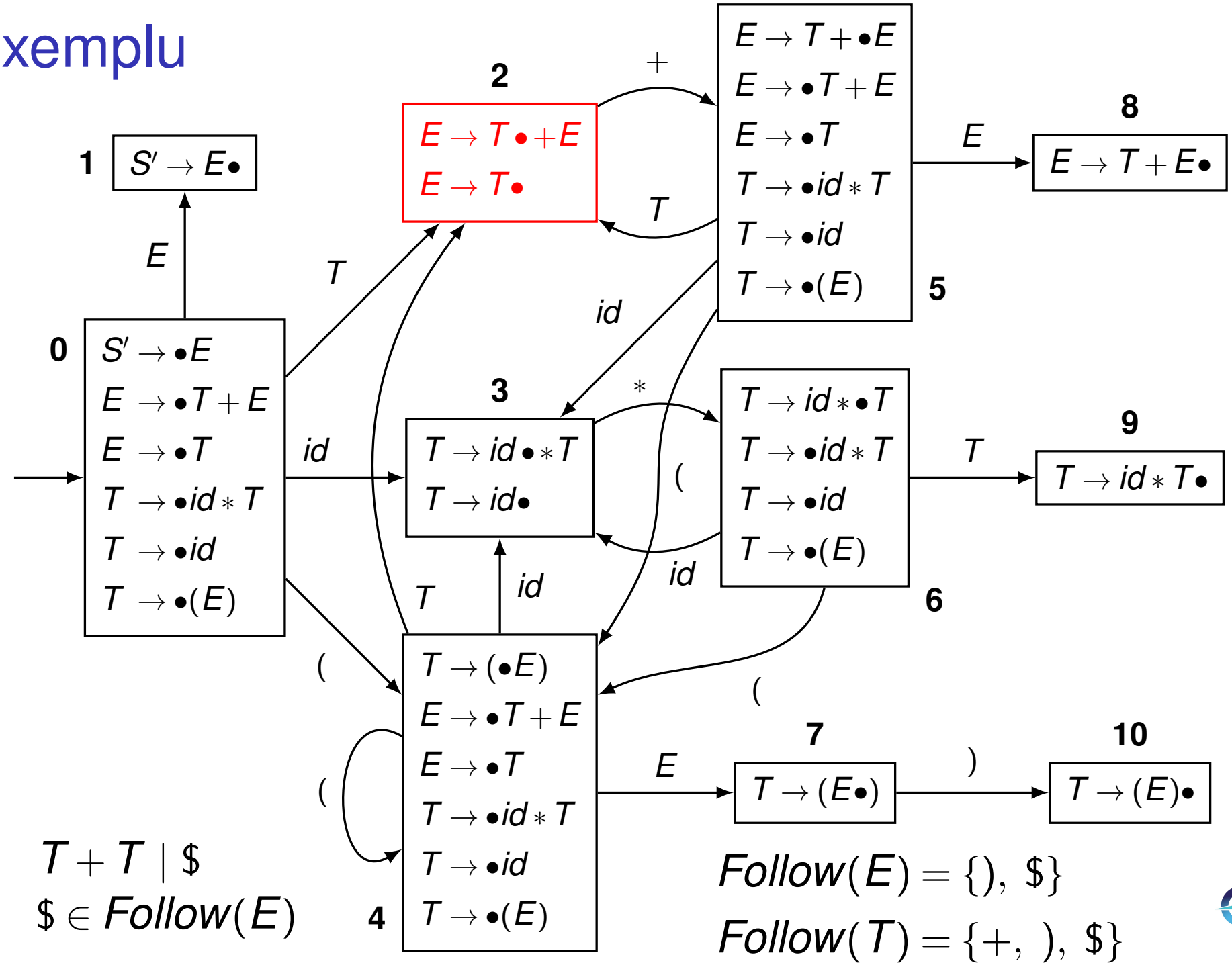


$T + T \mid \$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu



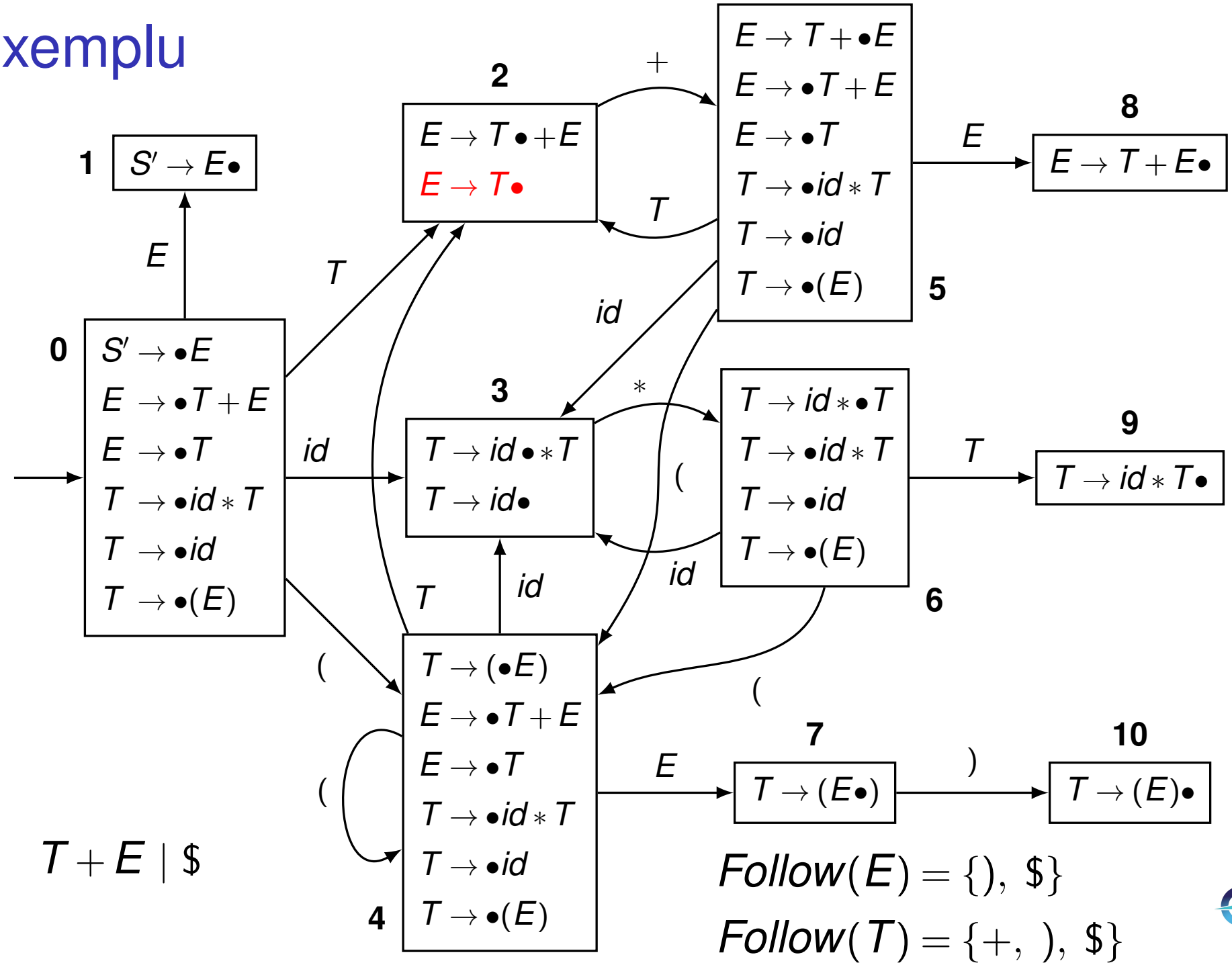
$T + T \mid \$$

$\$ \in Follow(E)$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu

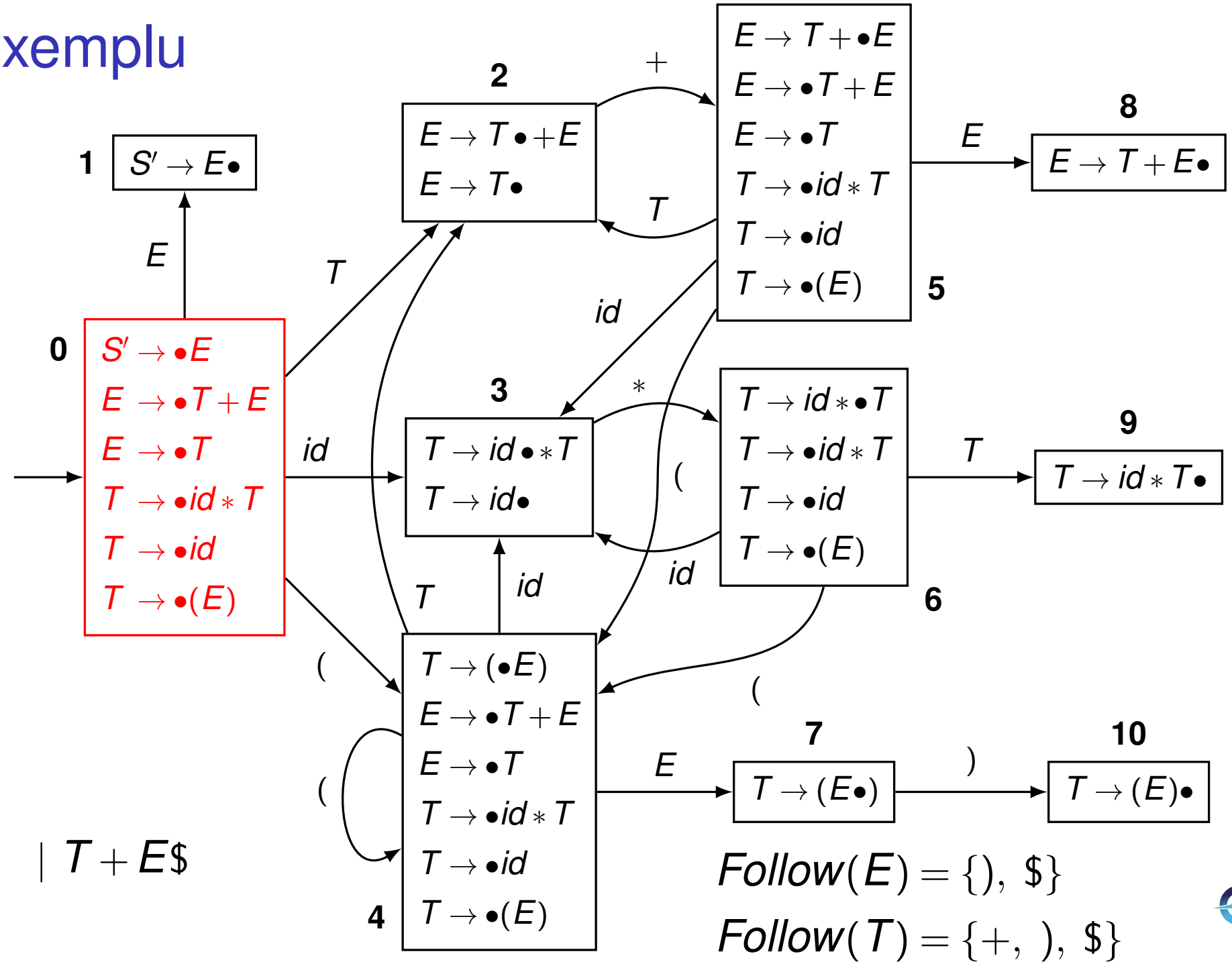


$T + E \mid \$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu

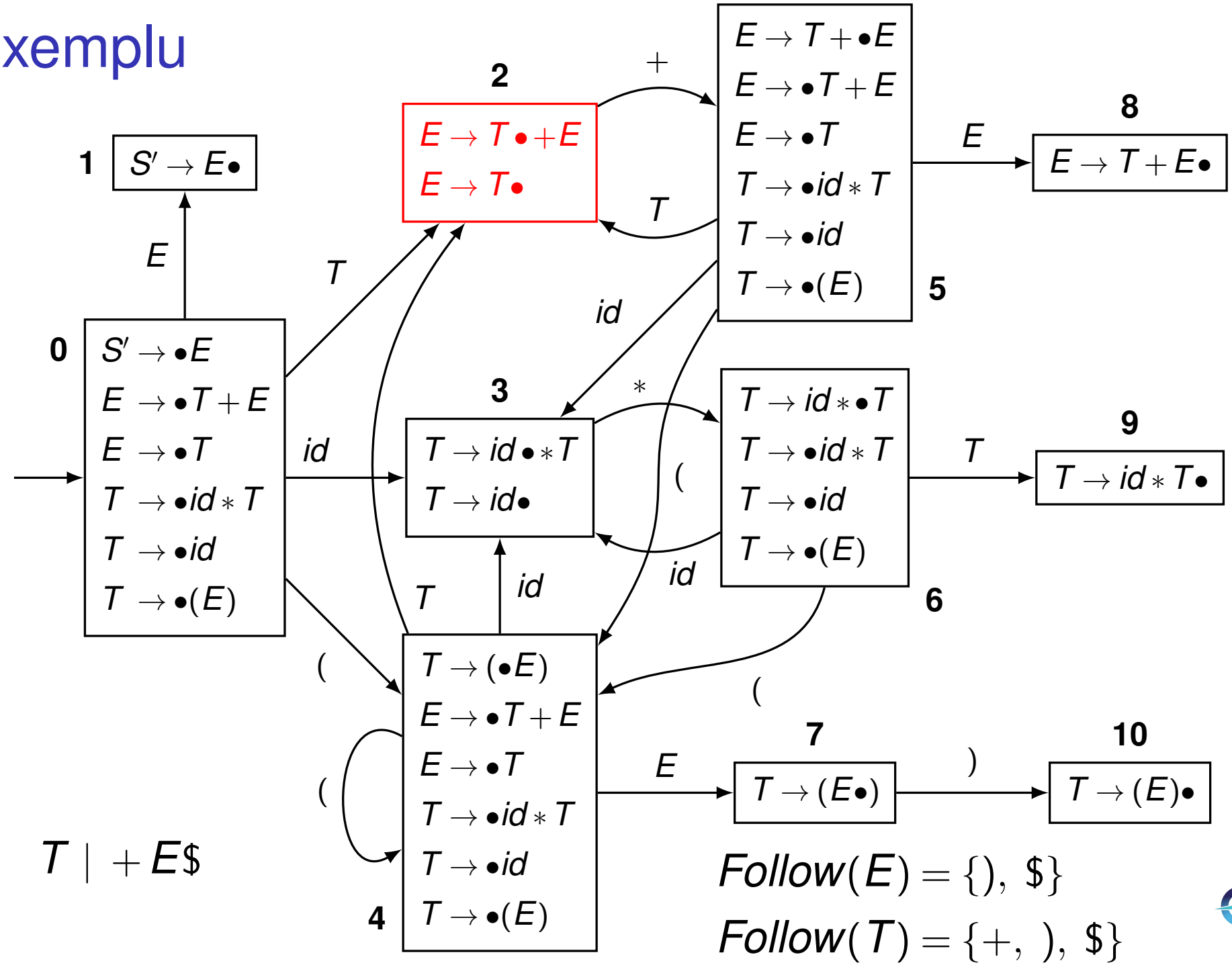


$| \; T + E\$$

$Follow(E) = \{), \; \$\}$

$Follow(T) = \{+, \;), \; \$\}$

Exemplu

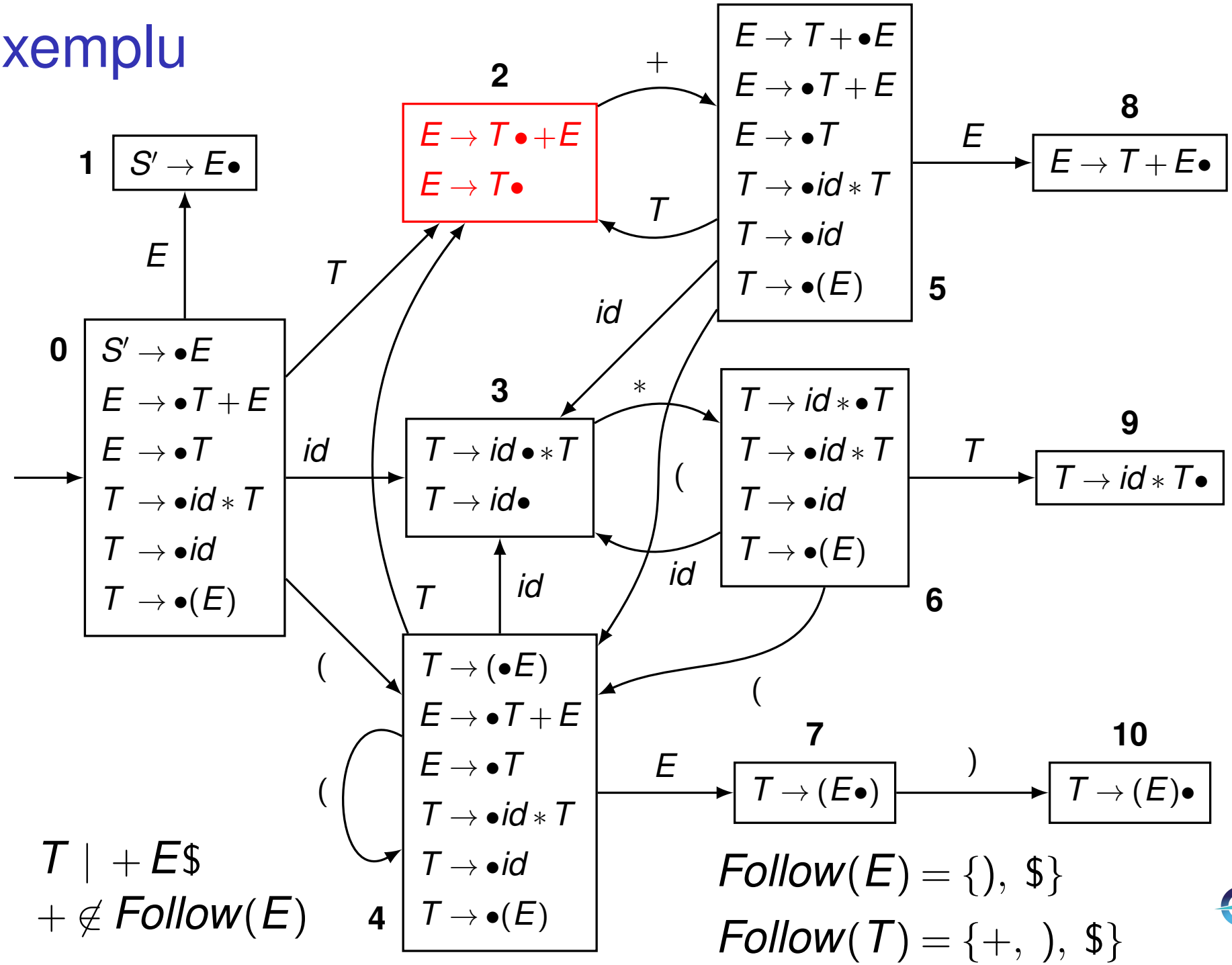


$T \mid +E\$$

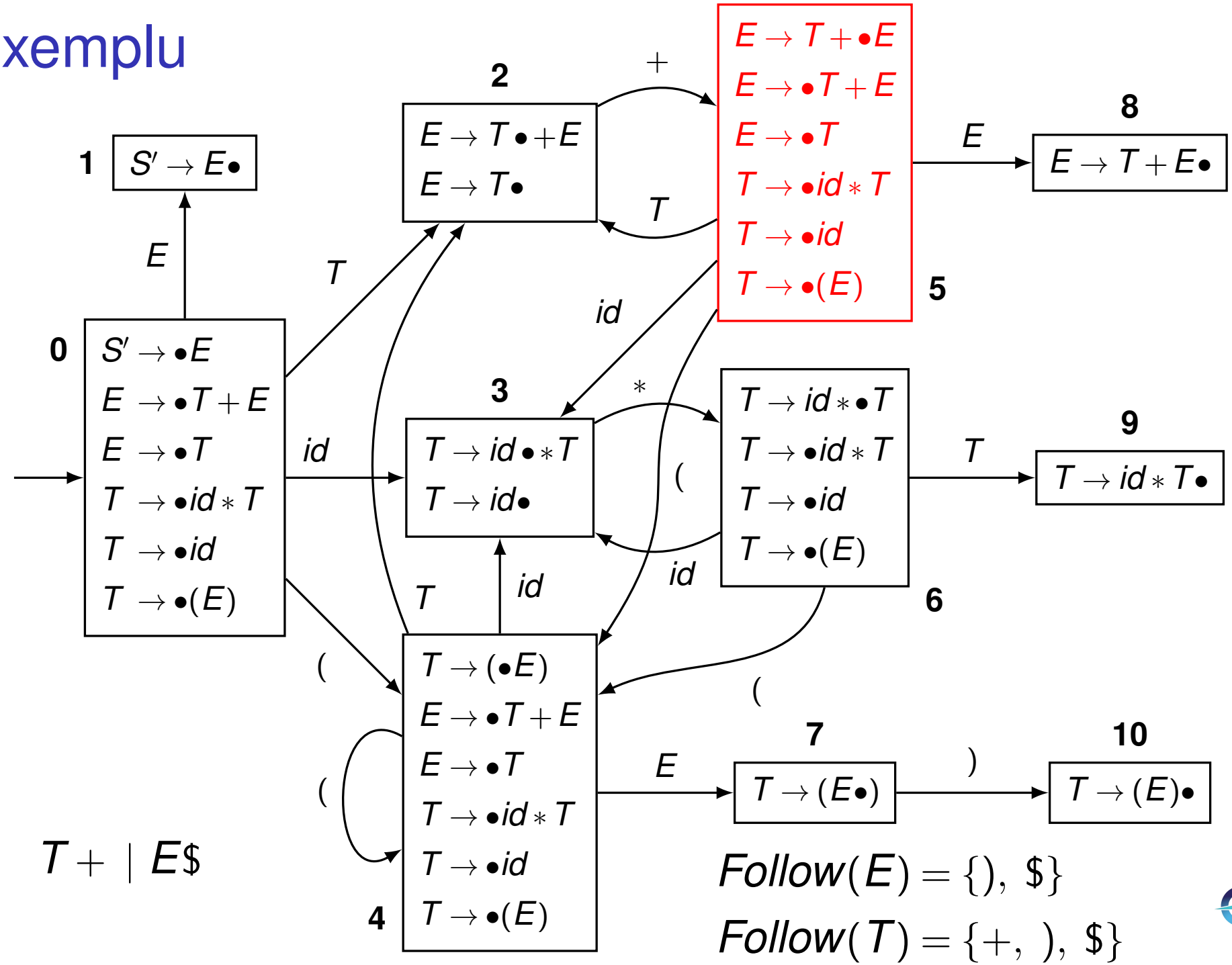
$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu


0
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
1
$S' \to E\bullet$
2
$E \to T \bullet + E$
$E \to T\bullet$
3
$T \to id \bullet * T$
$T \to id\bullet$
4
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
5
$E \to T + \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
6
$T \to id * \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
7
$T \to (E\bullet)$
8
$E \to T + E\bullet$
9
$T \to id * T\bullet$
10
$T \to (E)\bullet$
E
T
id
(
+
*
)
$T \mid +E\$$
$+ \notin Follow(E)$
$Follow(E) = \{), \$\}$
$Follow(T) = \{+,), \$\}$

Exemplu

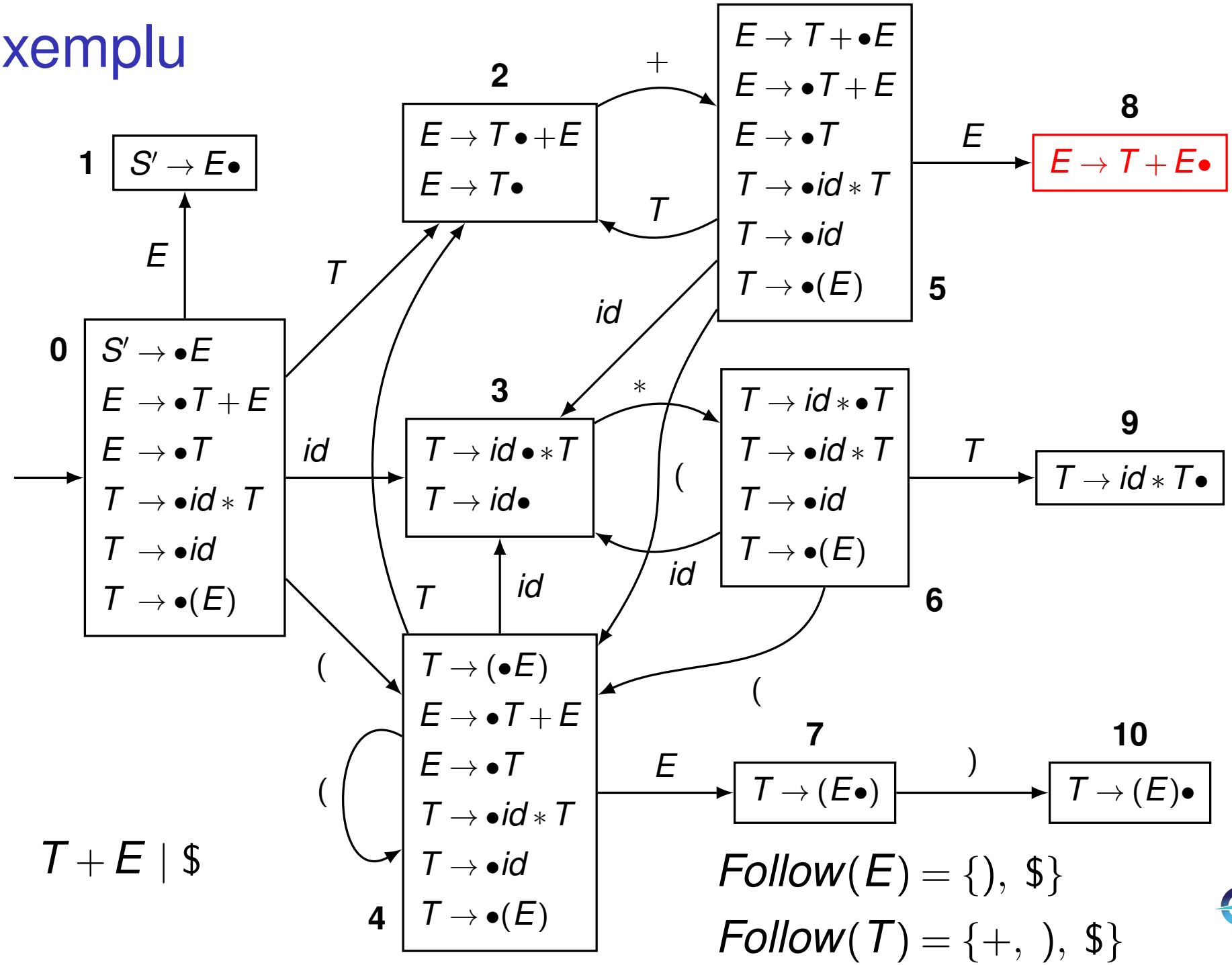


$T + \mid E\$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu

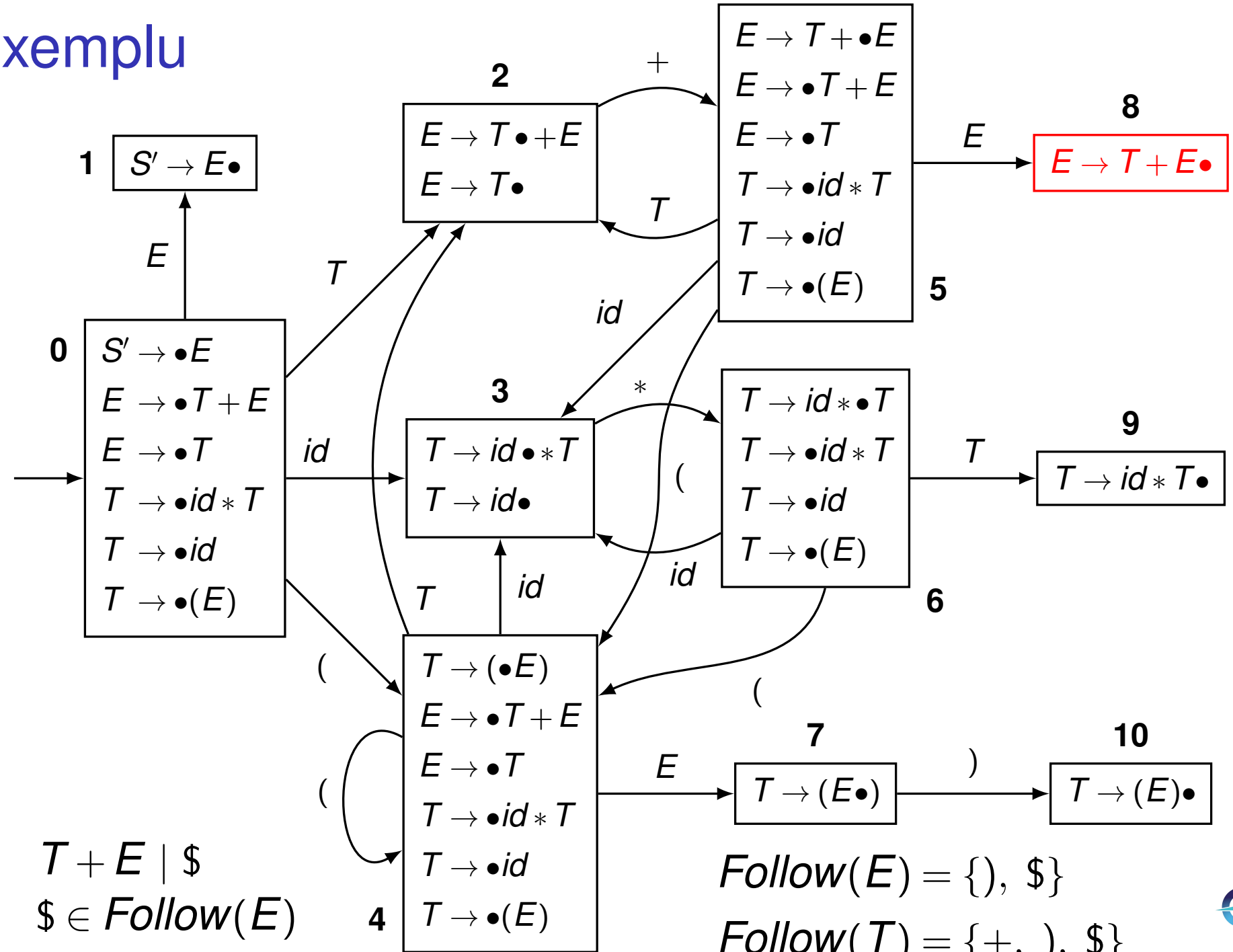


$T + E \mid \$$

$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

Exemplu



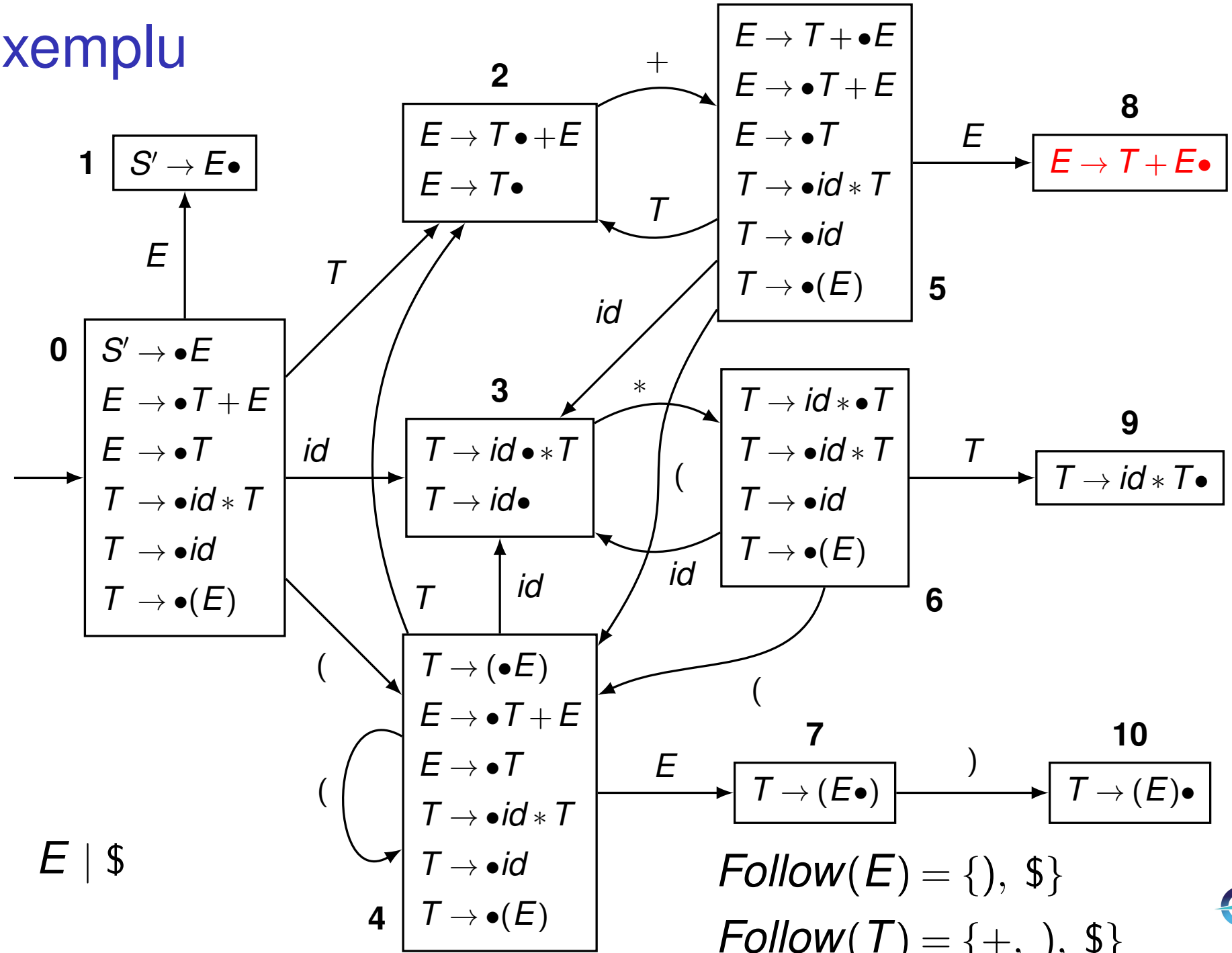
$T + E \mid \$$

$\$ \in Follow(E)$

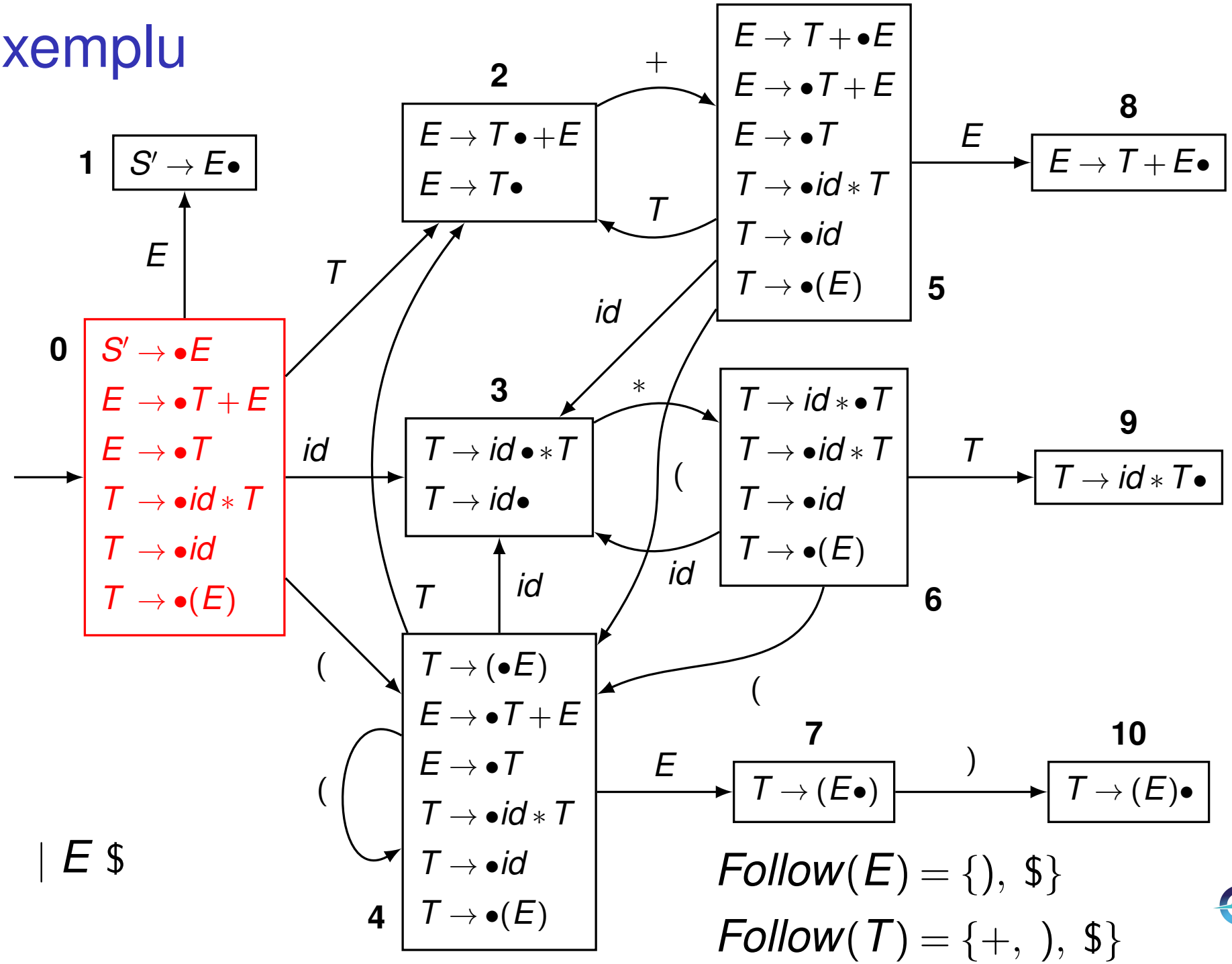
$Follow(E) = \{), \$\}$

$Follow(T) = \{+,), \$\}$

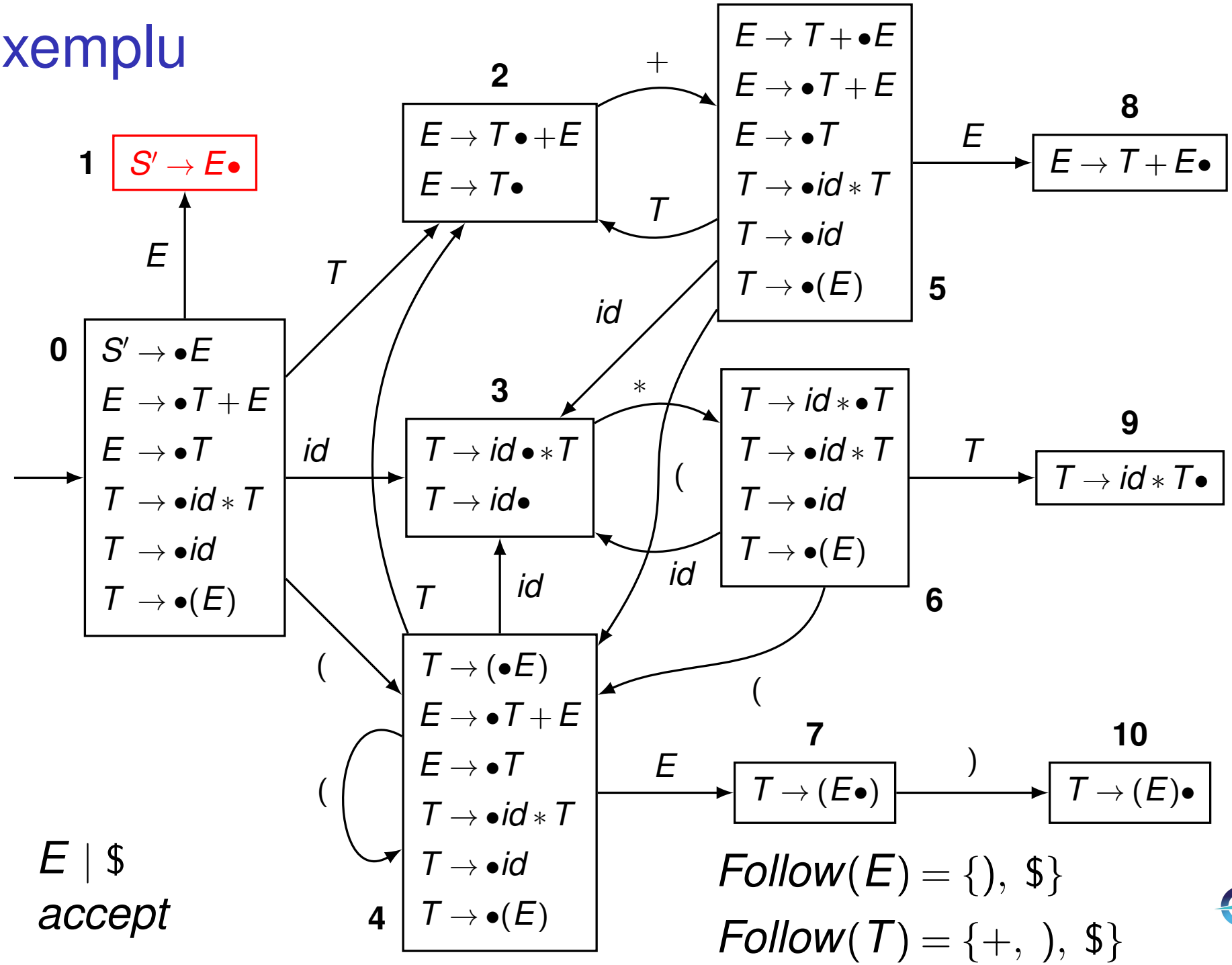
Exemplu


0
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
1
$S' \to E\bullet$
2
$E \to T \bullet + E$
$E \to T\bullet$
3
$T \to id \bullet * T$
$T \to id\bullet$
4
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
5
$E \to T + \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
6
$T \to id * \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
7
$T \to (E\bullet)$
8
$E \to T + E\bullet$
9
$T \to id * T\bullet$
10
$T \to (E)\bullet$
E
T
id
+
*
(
)
$E \mid \$$
$Follow(E) = \{), \$\}$
$Follow(T) = \{+,), \$\}$

Exemplu


0
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
1
$S' \to E\bullet$
2
$E \to T \bullet + E$
$E \to T\bullet$
3
$T \to id \bullet * T$
$T \to id\bullet$
4
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
5
$E \to T + \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
6
$T \to id * \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
7
$T \to (E\bullet)$
8
$E \to T + E\bullet$
9
$T \to id * T\bullet$
10
$T \to (E)\bullet$
E
T
id
*
+
(
)
$| E \$$
$Follow(E) = \{), \$\}$
$Follow(T) = \{+,), \$\}$

Exemplu


0
$S' \to \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
1
$S' \to E\bullet$
2
$E \to T \bullet + E$
$E \to T\bullet$
3
$T \to id \bullet * T$
$T \to id\bullet$
4
$T \to (\bullet E)$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
5
$E \to T + \bullet E$
$E \to \bullet T + E$
$E \to \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
6
$T \to id * \bullet T$
$T \to \bullet id * T$
$T \to \bullet id$
$T \to \bullet (E)$
7
$T \to (E\bullet)$
8
$E \to T + E\bullet$
9
$T \to id * T\bullet$
10
$T \to (E)\bullet$
E
T
id
(
+
*
)
$E \mid \$$
accept
$Follow(E) = \{), \$\}$
$Follow(T) = \{+,), \$\}$

Îmbunătățire

- Dezavntajul algoritmului naiv: repetarea execuției AFD pe întregul prefix, în pofida modificării doar a câtorva simboluri din vârful stivei
- Soluție: împerecherea simbolurilor din stivă cu **starea** AFD atinsă până la simbolul respectiv
- Noua **stivă**: $\langle X_1, s_1 \rangle \ldots \langle X_n, s_n \rangle$, unde s_i = starea AFD atinsă după parcurgerea $X_1 \ldots X_i$
- **Baza** stivei: $\langle dummy, start \rangle$

Tabelul *goto*

- **Funcția de tranziție** a AFD

- $goto[i, X] = j$, dacă $s_i \rightarrow^X s_j$

Noile acțiuni

- ***Shift*** s: adaugă $\langle t, s \rangle$ în stivă, unde t = simbolul curent de la intrare, iar s = starea AFD
- ***Reduce*** $X \to \beta$
- ***Accept***
- ***Error***

Tabelul *action*

Pentru fiecare stare s_i și terminal t:

- $action[i, t] =$ ***shift*** j, dacă s_i conține itemul $X \to \beta \bullet t\gamma$ și $goto[i, t] = j$
- $action[i, t] =$ ***reduce*** $X \to \beta$, dacă s_i conține itemul $X \to \beta\bullet$, $t \in Follow(X)$ și $X \neq S'$
- $action[i, \$] =$ ***accept***, dacă s_i conține itemul $S' \to S\bullet$
- $action[i, t] =$ ***error***, altfel

Tabelele din exemplu

$E \to T + E$ (1)
$E \to T$ (2)
$T \to id * T$ (3)
$T \to id$ (4)
$T \to (E)$ (5)

Stare	*action*						*goto*	
	id	$+$	$*$	(	)	$	*E*	*T*
0	s3			s4			1	2
1						a		
2		s5			r2	r2		
3		r4	s6		r4	r4		
4	s3			s4			7	2
5	s3			s4			8	2
6	s3			s4				9
7					s10			
8					r1	r1		
9		r3			r3	r3		
10		r2			r2	r2		

s = *shift* către stare, r = *reduce* cu producție, a = *accept*

Algoritmul *SLR*(1)

```
1  șir ← t_1 ... t_n$
2  stiva ← ⟨dummy, 0⟩
3  repetă
4      variante action [top (stiva), peek ()] :
5          shift j :  push (⟨consume(), j⟩, stiva)
6          reduce X → β :
7              pop (stiva) × |β|
8              push (⟨X, goto[top(stiva), X]⟩, stiva)
9          accept :  stop
10         error :  eroare ()
11 până când
```

Partea VI

Analiza semantică

Cuprins

Cuprins

Analiza semantică I

- Determinarea sensului propozițiilor
- Rezolvarea referințelor: ce desemnează un anumit simbol?

Exemplu în limba română:

Andrei are o carte. Cartea lui este albastră.

La cine se referă *lui*?

Exemplu în programare:

```
1  void f(int x) {
2        int x;
3        return x;
4  }
```

La cine se referă `x`?

Analiza semantică II

- Verificarea compatibilității între entități

Exemplu în limba română:

Cartea și-a luat zborul.

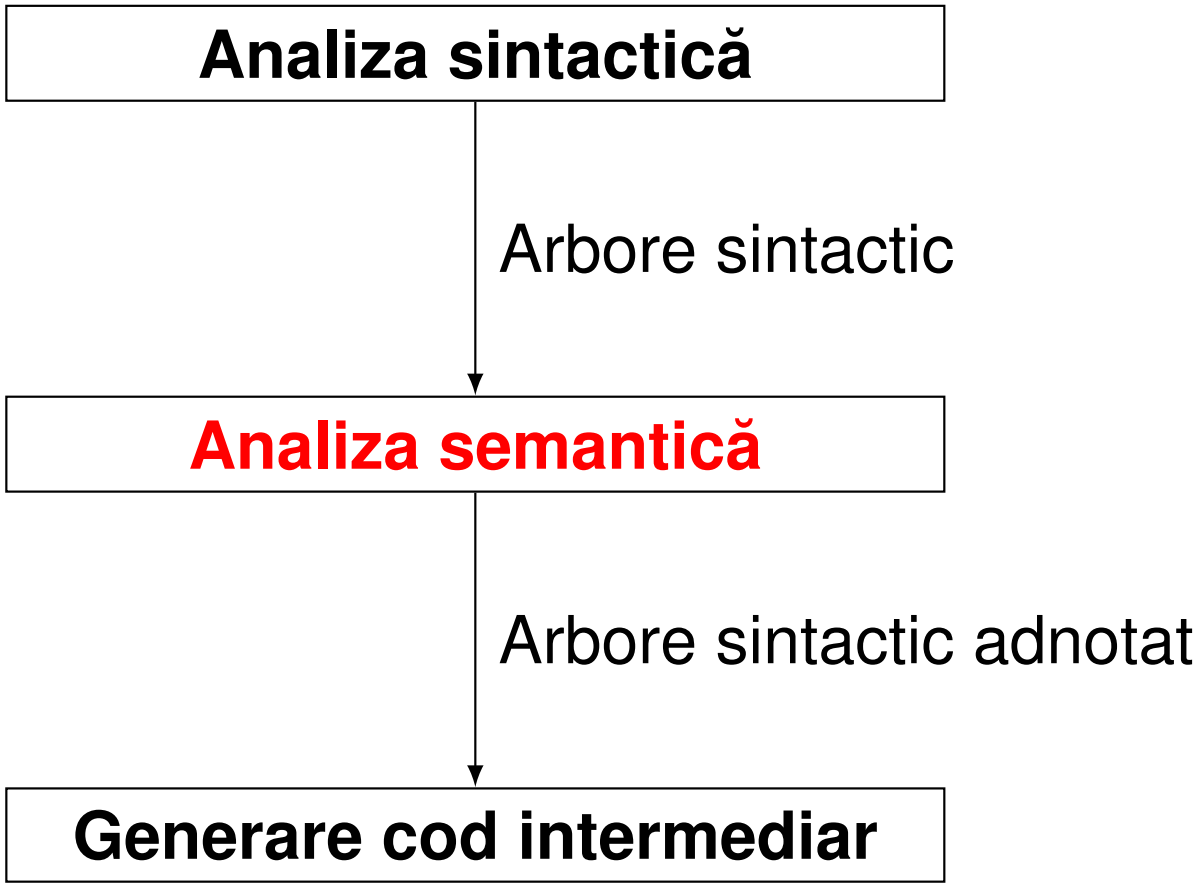
Este acest lucru posibil?

Exemplu în programare:

```
1 Carte c;
2 zboara(c);
```

Parametrul lui `zboară` poate fi de tipul `Carte`?

Context


Analiza sintactică
Arbore sintactic
Analiza semantică
Arbore sintactic adnotat
Generare cod intermediar

Exemple de erori semantice

- Variabilă sau tip, utilizate, dar nedefinite
- Cicluri în ierarhia de clase
- Redefinirea tipurilor sau a metodelor în aceeași clasă
- Incompatibilitatea de tip dintre parametrii formali și cei actuali
- Supradefinirea unei metode cu alt număr și/sau tipuri de parametri formali
- Utilizarea incorectă a identificatorilor rezervați (e.g. atribuire către `self` sau definirea unei clase cu numele `SELF_TYPE`)

Cuprins

Domeniu de vizibilitate (*scope*)

- Porțiunea din program unde o definiție este accesibilă
- Posibilitatea existenței unor definiții multiple ale aceluiași identificator, dar având domenii de vizibilitate disjuncte!
- Posibilitatea restricționării domeniului de vizibilitate (*private*, *protected* etc.)

Determinarea domeniilor de vizibilitate

- Proprietate a limbajului
- **Statică/lexicală**: la compilare, exclusiv pe baza organizării programului, ca în Cool
- **Dinamică**: în funcție de cursul execuției programului

Exemplu de domeniu determinat static

```
1  class A {
2        f(x : Int) : Int {
3             x + (let x : Int <- x + 1
4                  in x)
5               + x
6        };
7  };
```

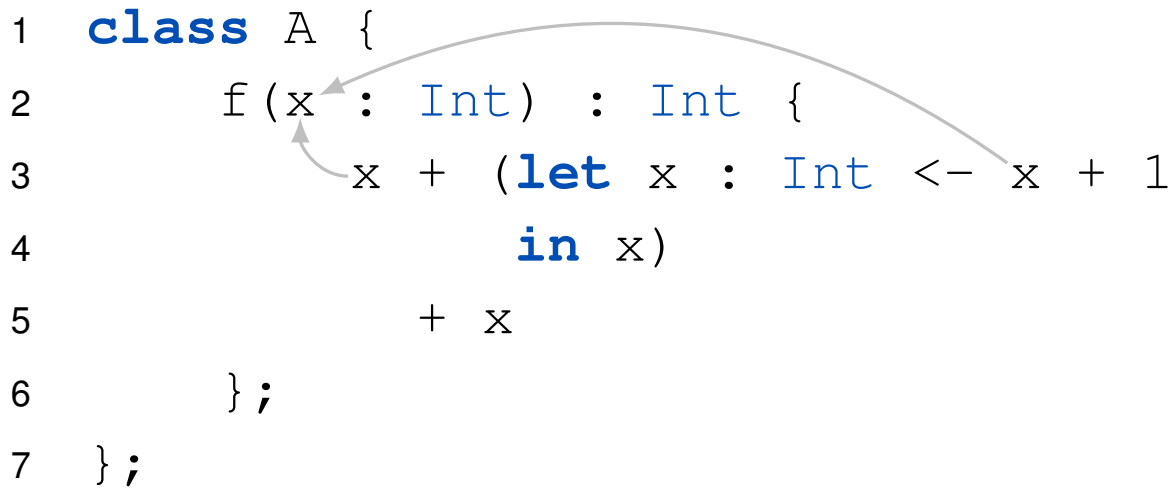
- De obicei, asocierea apariției unui identificator cu cea mai apropiată definiție, conform organizării **textuale** a programului
- Excepție la `let`, unde domeniul unei variabile locale NU include expresia de inițializare

Exemplu de domeniu determinat static

```
1  class A {
2        f(x : Int) : Int {
3            x + (let x : Int <- x + 1
4                    in x)
5                + x
6        };
7  };
```

- De obicei, asocierea apariției unui identificator cu cea mai apropiată definiție, conform organizării **textuale** a programului
- Excepție la `let`, unde domeniul unei variabile locale NU include expresia de inițializare

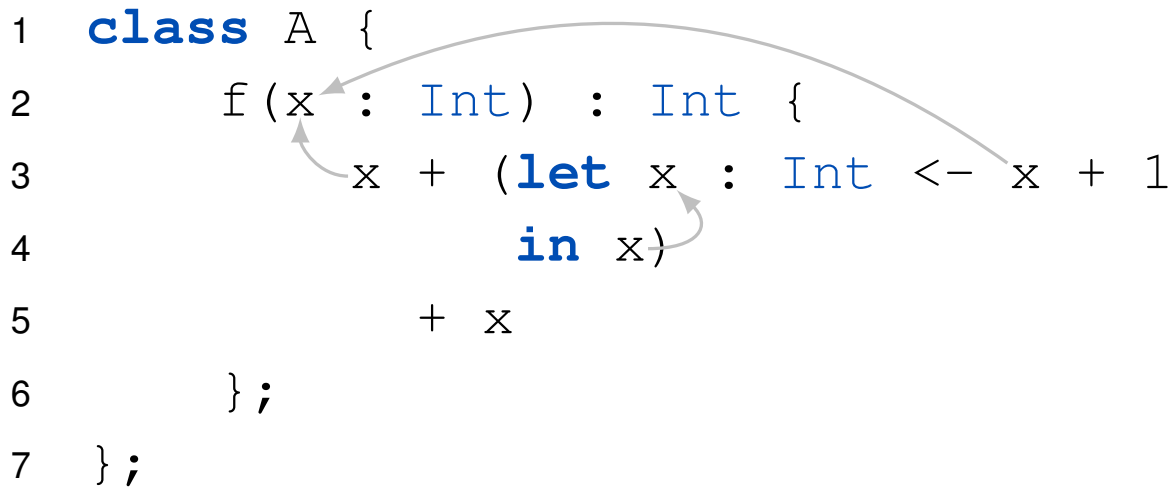
Exemplu de domeniu determinat static



```
1  class A {
2        f(x : Int) : Int {
3            x + (let x : Int <- x + 1
4                 in x)
5                + x
6        };
7  };
```

- De obicei, asocierea apariției unui identificator cu cea mai apropiată definiție, conform organizării **textuale** a programului
- Excepție la `let`, unde domeniul unei variabile locale NU include expresia de inițializare

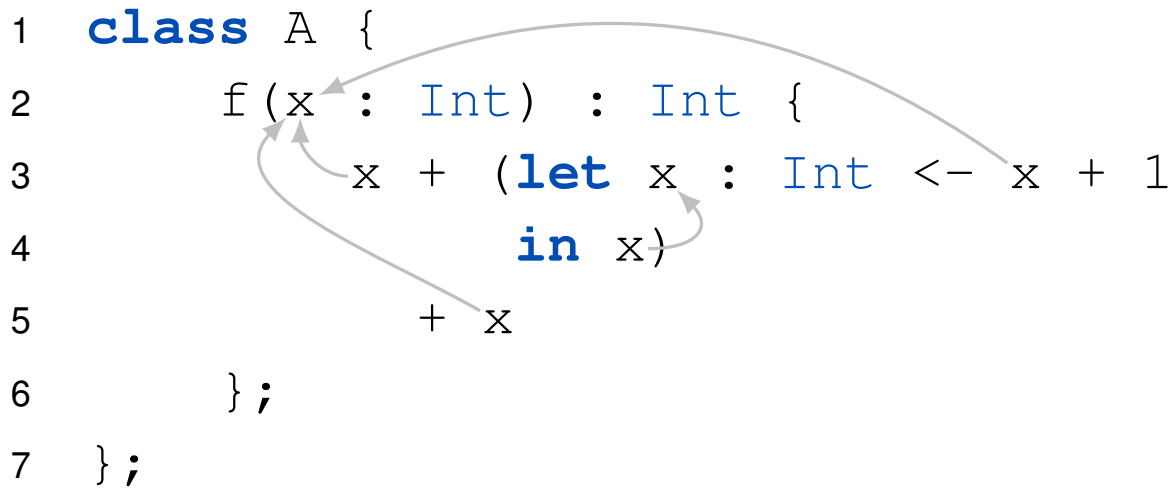
Exemplu de domeniu determinat static



```
1  class A {
2        f(x : Int) : Int {
3            x + (let x : Int <- x + 1
4                 in x)
5                + x
6        };
7  };
```

- De obicei, asocierea apariției unui identificator cu cea mai apropiată definiție, conform organizării **textuale** a programului
- Excepție la `let`, unde domeniul unei variabile locale NU include expresia de inițializare

Exemplu de domeniu determinat static



```
1  class A {
2        f(x : Int) : Int {
3           x + (let x : Int <- x + 1
4                   in x)
5                + x
6        };
7  };
```

- De obicei, asocierea apariției unui identificator cu cea mai apropiată definiție, conform organizării **textuale** a programului
- Excepție la `let`, unde domeniul unei variabile locale NU include expresia de inițializare

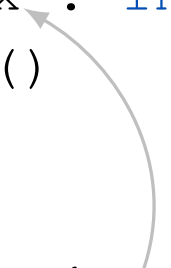
Exemplu de domeniu determinat dinamic

```
1  class A {
2        f() : Int {
3             let x : Int <- 0
4             in g()
5        };
6
7        g() : Int { x };
8  };
```

- Asocierea apariției unui identificator cu cea mai recentă definiție realizată în timpul **execuției**
- Exemplu teoretic, nepermis în Cool

Exemplu de domeniu determinat dinamic

```
1  class A {
2        f() : Int {
3            let x : Int <- 0
4            in g()
5        };
6
7        g() : Int { x };
8  };
```



- Asocierea apariției unui identificator cu cea mai recentă definiție realizată în timpul **execuției**
- Exemplu teoretic, nepermis în Cool

Definirea identificatorilor în Cool

- Clase
- Atribute
- Metode
- Parametri formali
- Variabile de `let`
- Variabile de `case`

Particularități ale domeniilor de vizibilitate în Cool

- Vizibilitatea claselor și a metodelor la nivel global, precum și a atributelor în cadrul unei clase, indiferent de ordinea definirii acestora
- Posibilitatea **utilizării anticipate** a celor de mai sus, înaintea definirii lor (***forward references***)
- Posibilitatea accesării atributelor și metodelor moștenite, în absența definițiilor acestora din clasele în care sunt utilizate
- Posibilitatea supradefinirii metodelor, cu anumite restricții

Exemple de utilizări anticipate

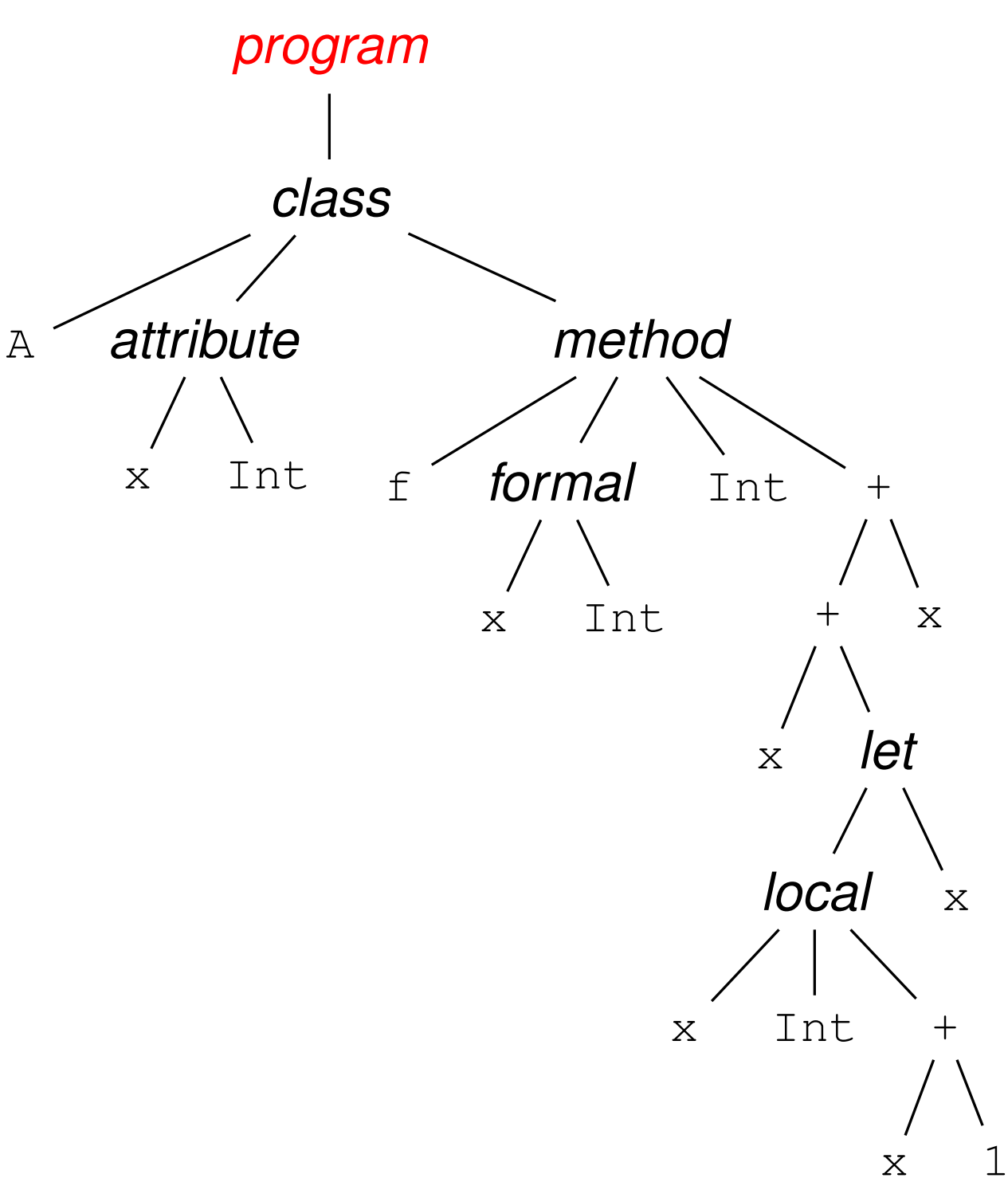
```
1  class B inherits A {                    -- clasa A
2      getSuccNumber() : Int {
3          getNumber() + 1                 -- metoda getNumber
4      };
5  };
6
7  class A {
8      getNumber() : Int {
9          number                          -- atributul number
10     };
11
12     number : Int;
13 };
```

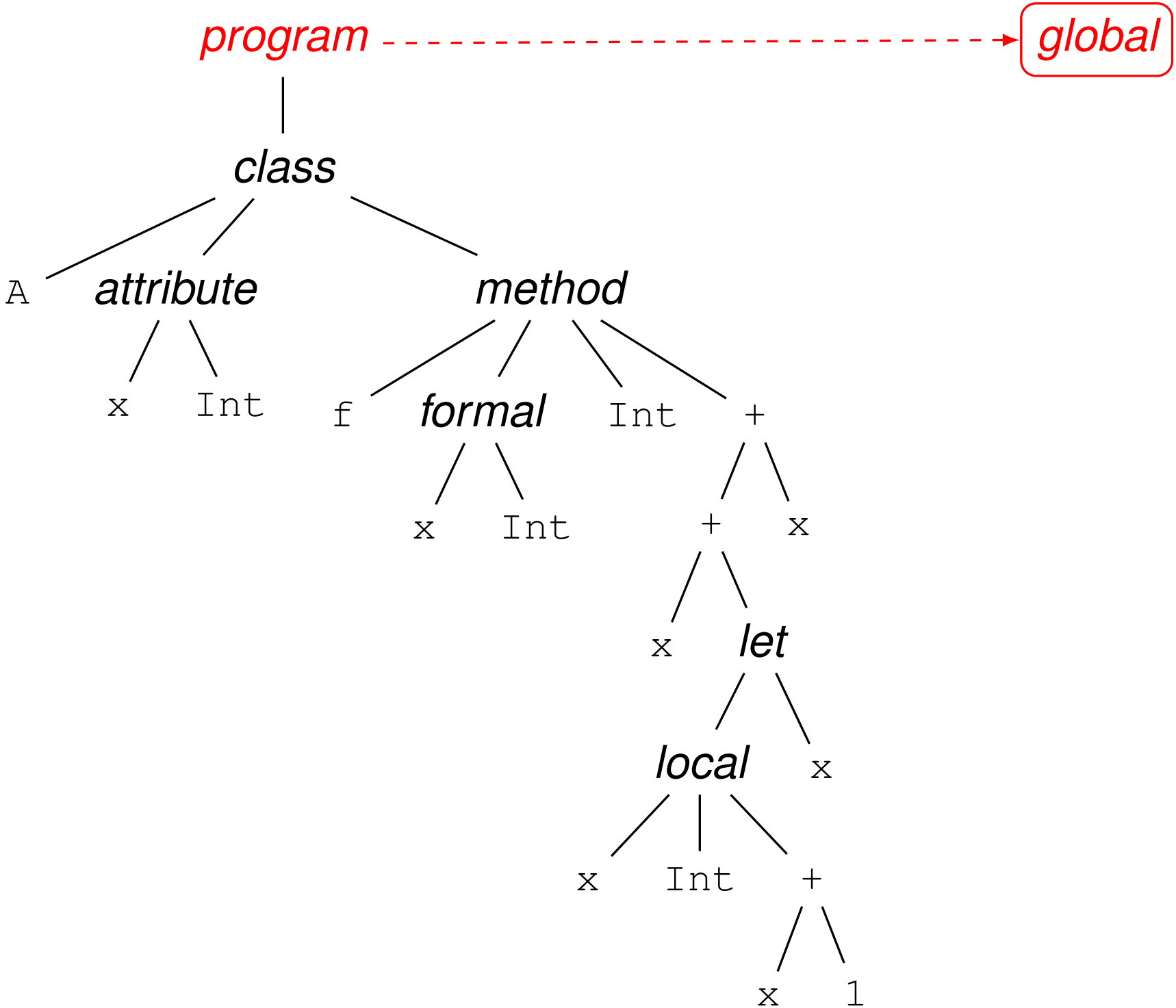
Tabelul de simboluri

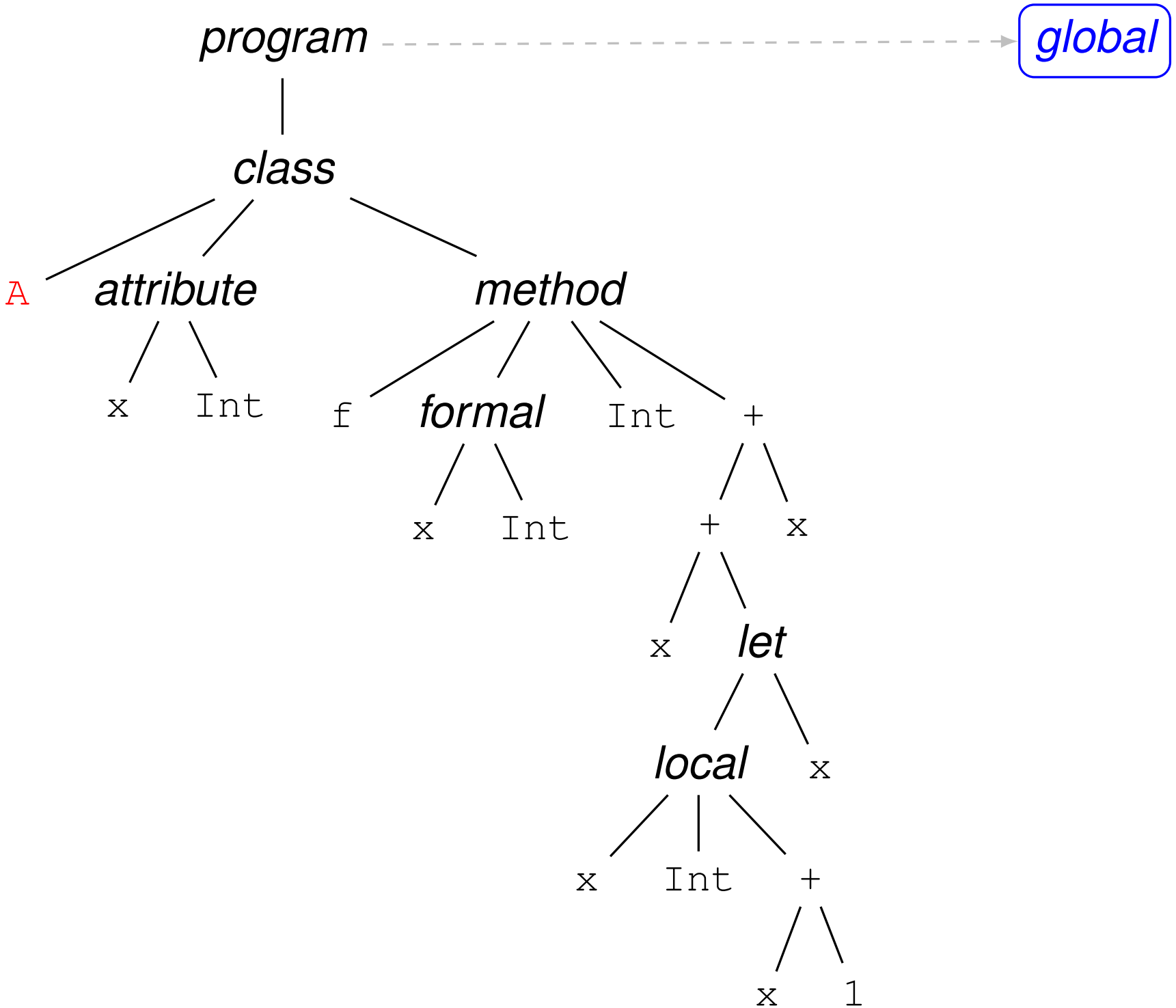
- Arbore de domenii de vizibilitate
- **Adăugarea** unui nou domeniu, e.g. la întâlnirea unei definiții de clasă
- **Părăsirea** domeniului curent, e.g. la încheierea unei definiții de clasă
- **Adăugarea** unui simbol nou, e.g. la definirea unei variabile
- **Verificarea** existenței unui simbol în domeniul curent, pentru evitarea duplicatelor
- **Căutarea** unui simbol, începând din domeniul curent și continuând spre rădăcină până la găsirea acestuia

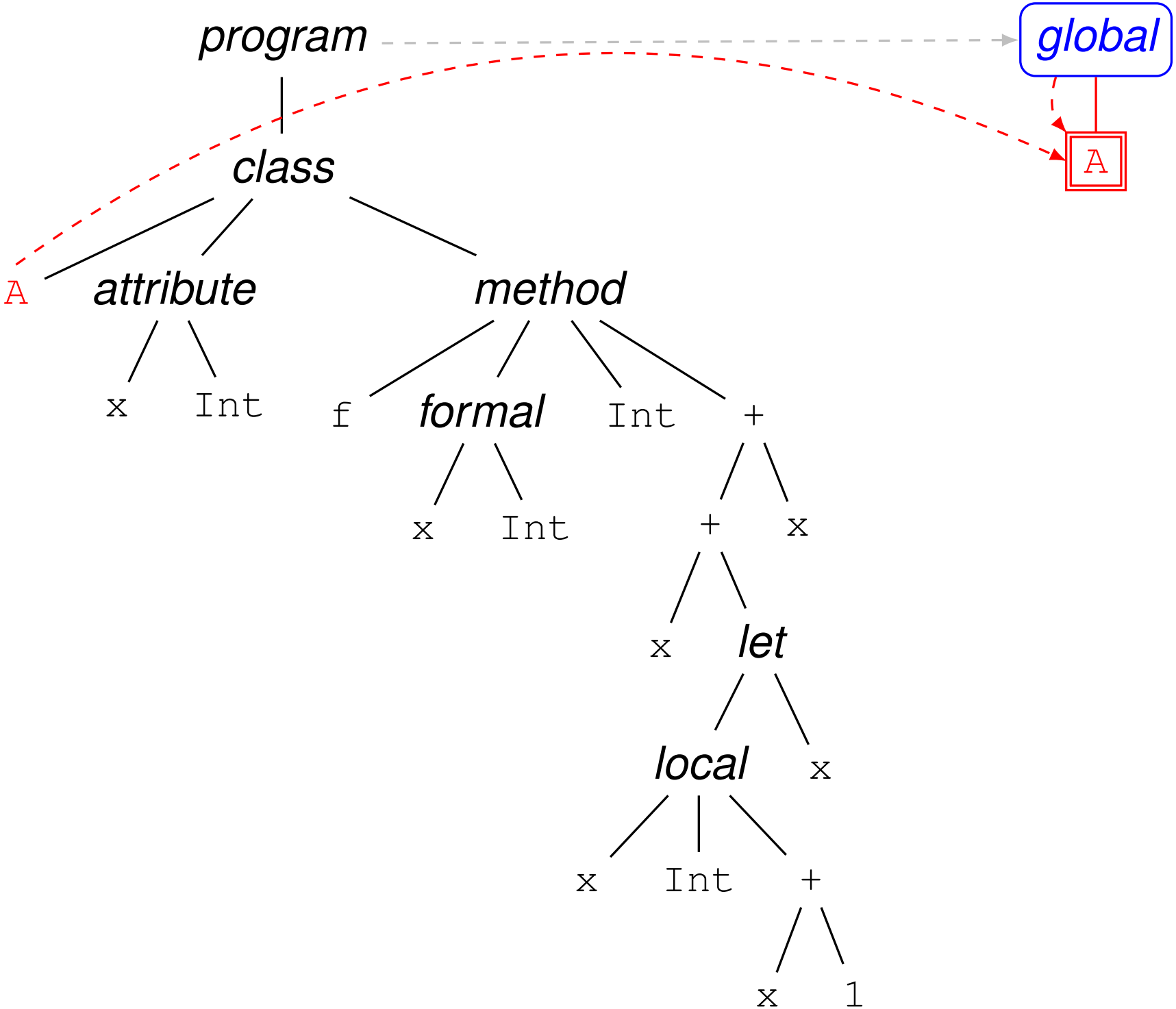
Exemplu

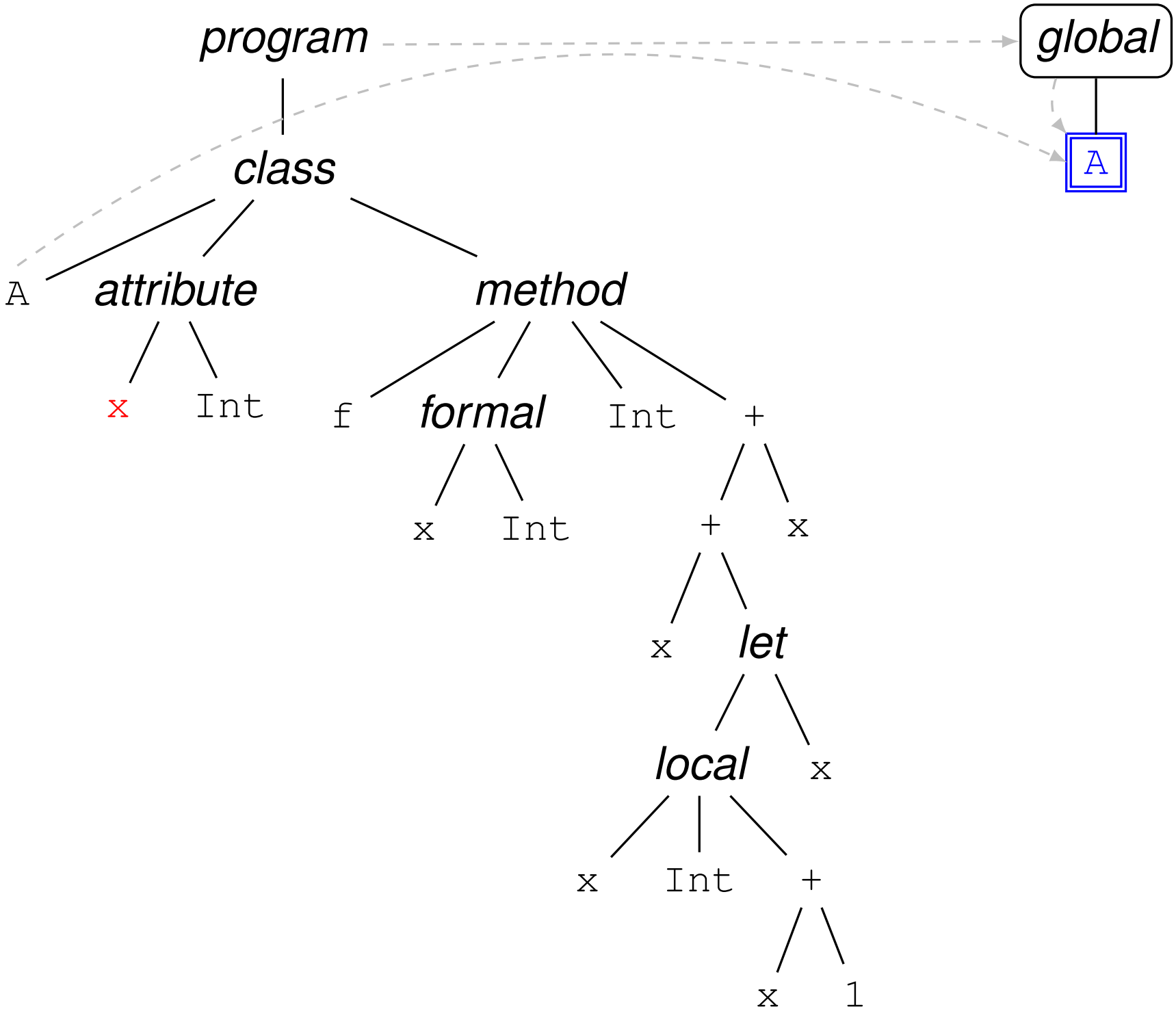
```
1  class A {
2       x : Int;
3
4       f(x : Int) : Int {
5            x + (let x : Int <- x + 1
6                 in x)
7              + x
8       };
9  };
10
11 class B inherits A {
12      y : Int;
13
14      g() : Int { x };
15 };
```

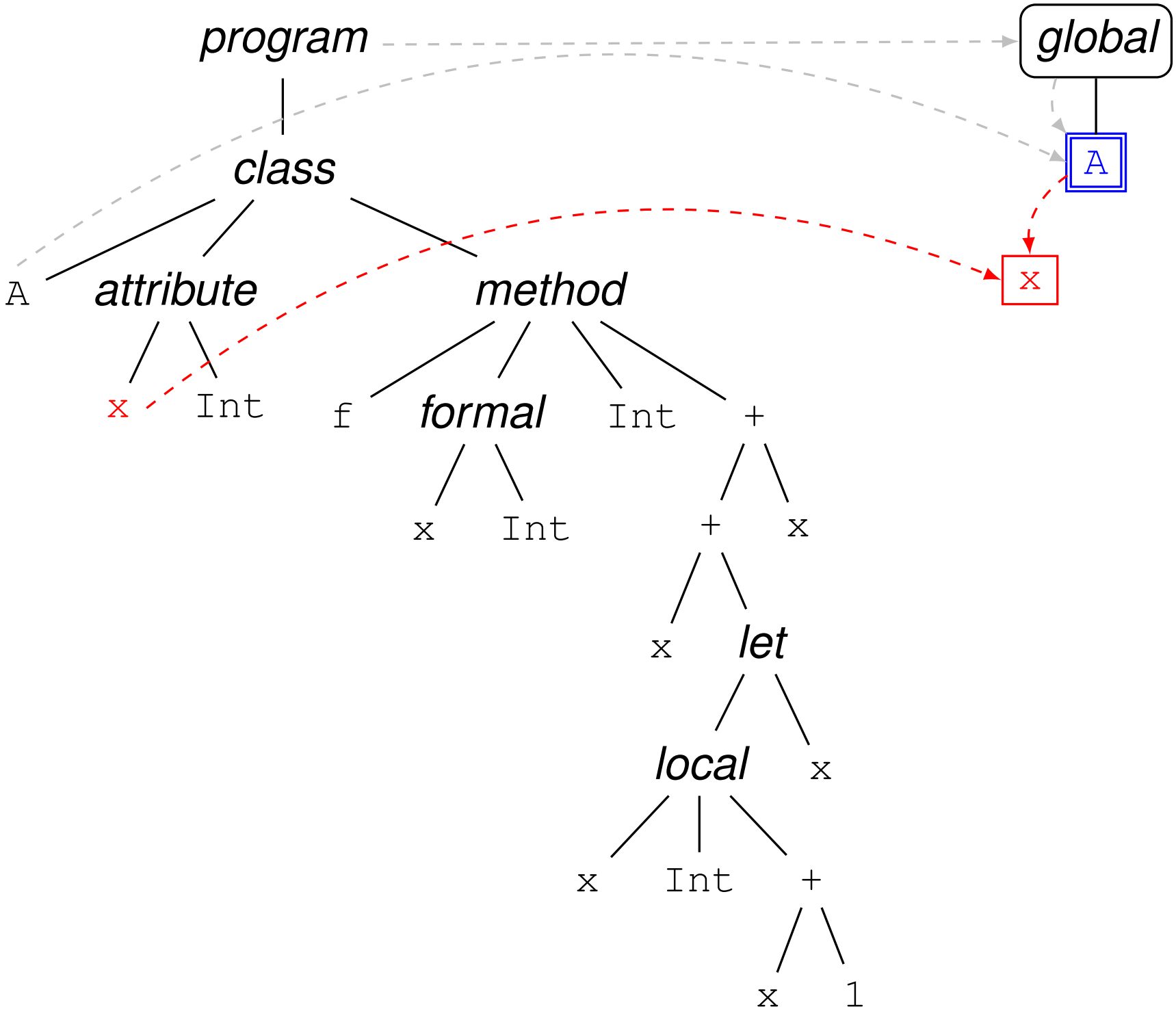

program
class
A
attribute
method
x
Int
f
formal
Int
+
x
Int
+
x
let
x
local
x
x
Int
+
x
1

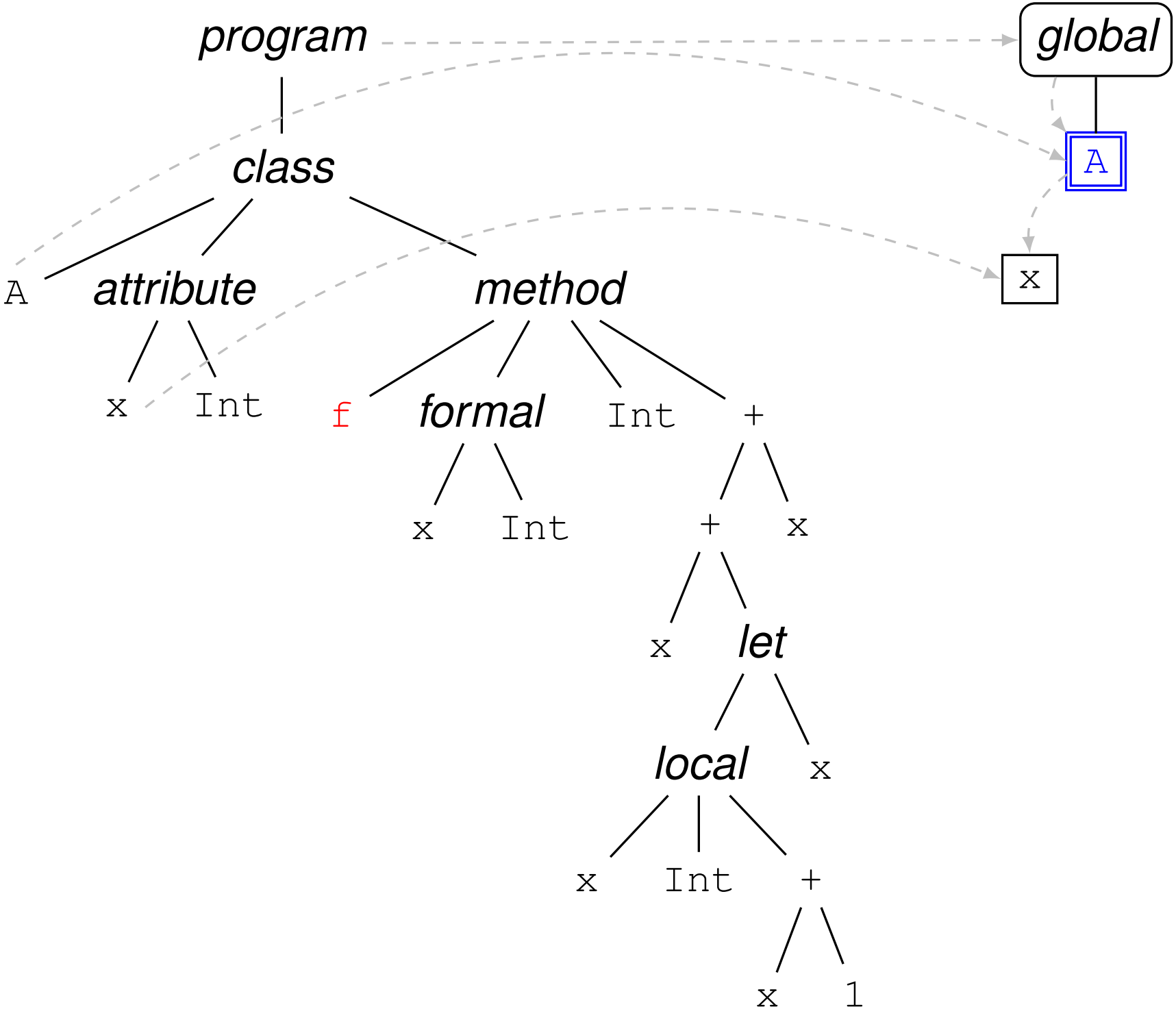

program
global
class
A
attribute
method
x
Int
f
formal
Int
+
x
Int
+
x
let
x
local
x
x
Int
+
x
1

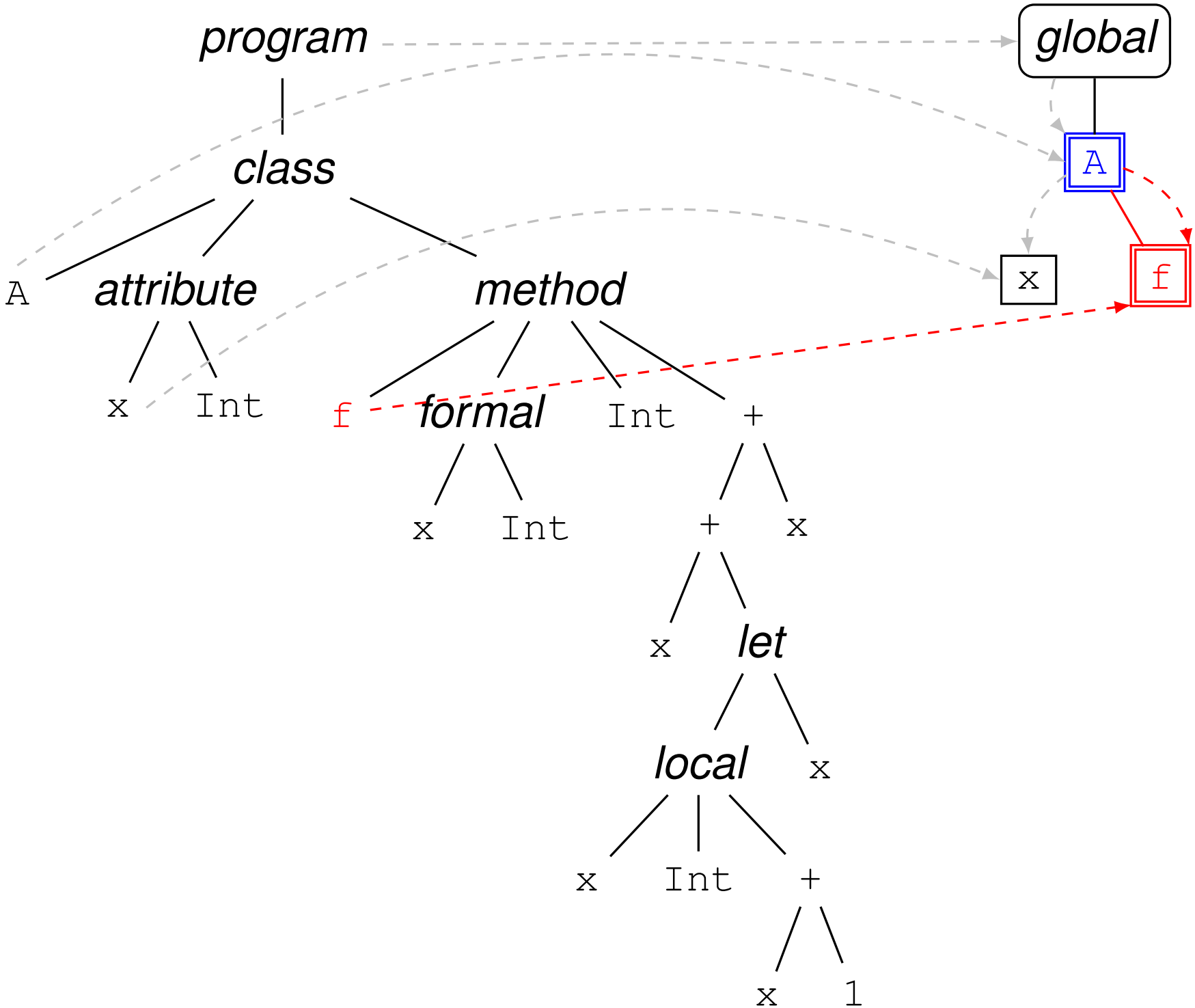

program
global
class
A
attribute
method
x
Int
f
formal
Int
+
x
Int
+
x
let
x
local
x
x
Int
+
x
1

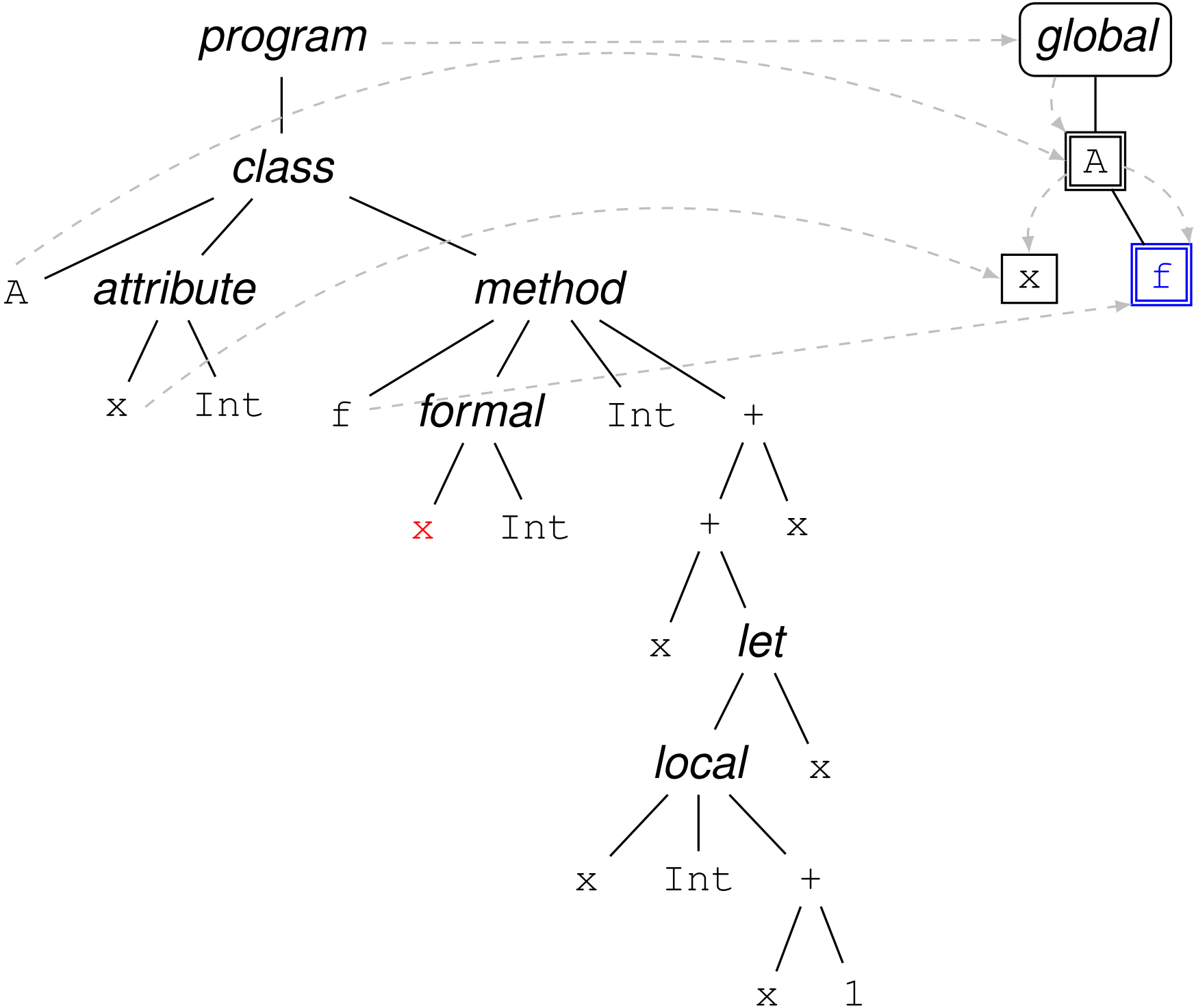

program
global
class
A
A
attribute
method
x
Int
f
formal
Int
+
x
Int
+
x
let
local
x
x
Int
+
x
1

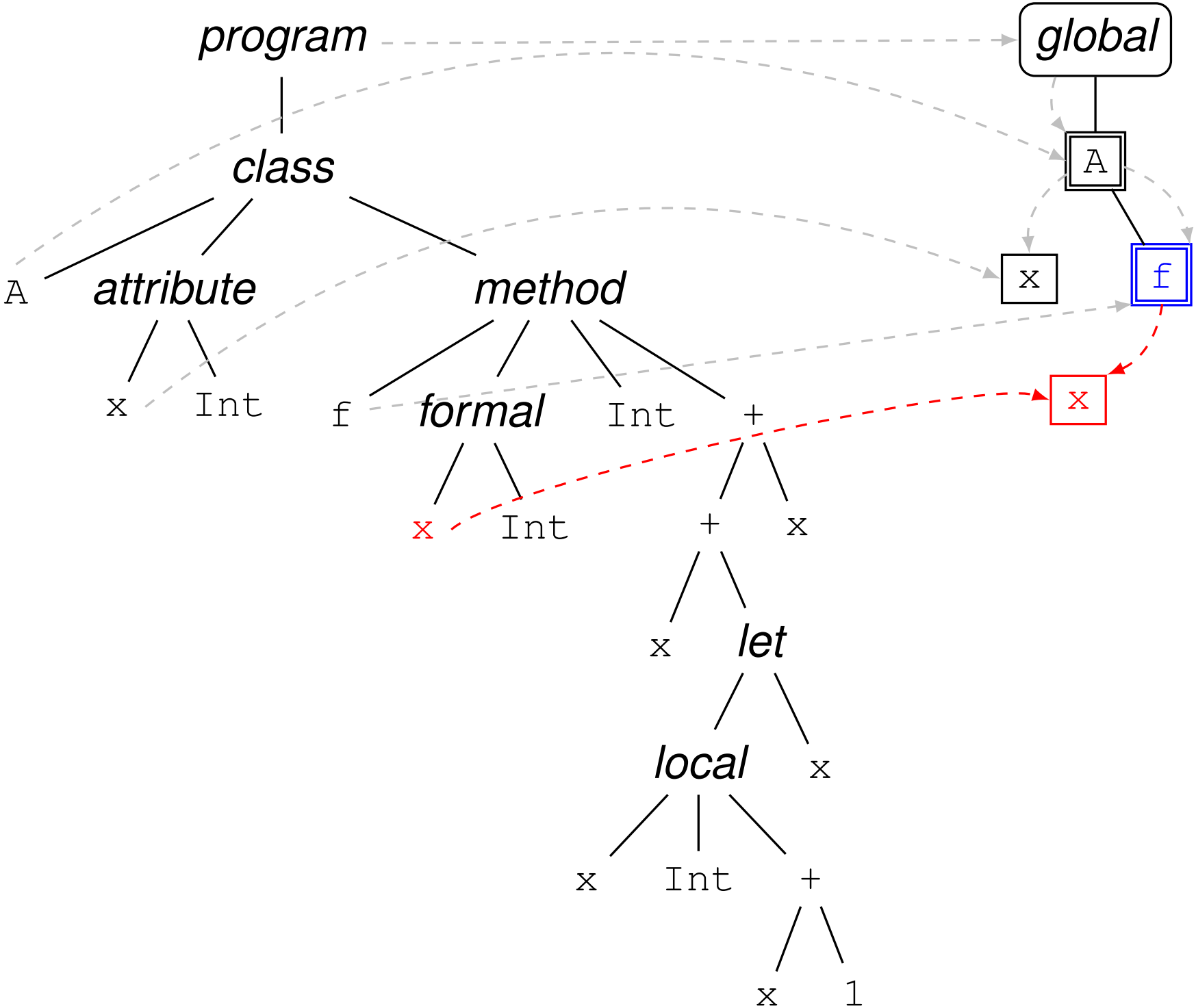

program
global
class
A
A
attribute
method
x
Int
f
formal
Int
+
x
Int
+
x
let
x
local
x
x
Int
+
x
1

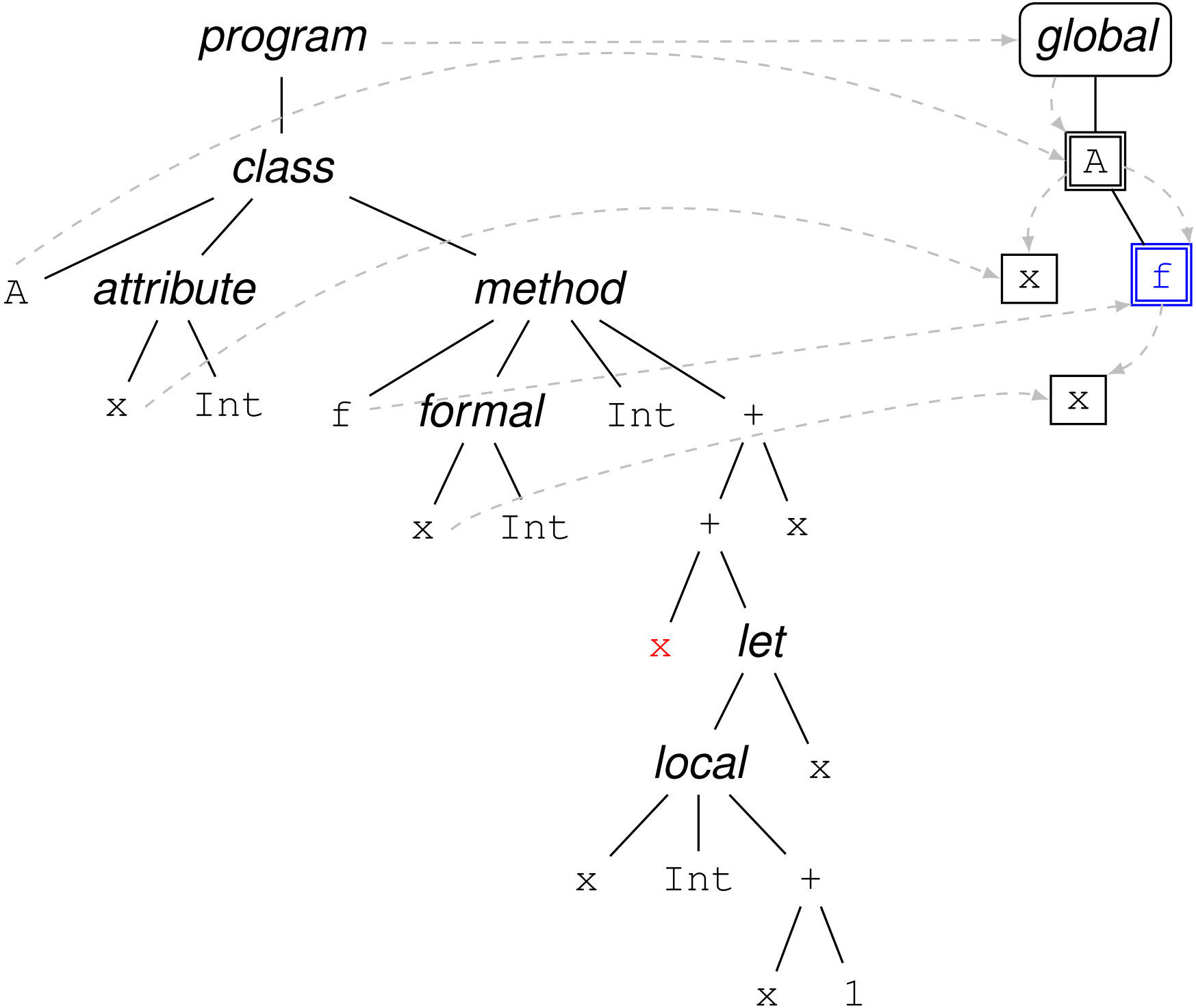

program
global
class
A
A
attribute
method
x
x
Int
f
formal
Int
+
x
Int
+
x
x
let
local
x
x
Int
+
x
1

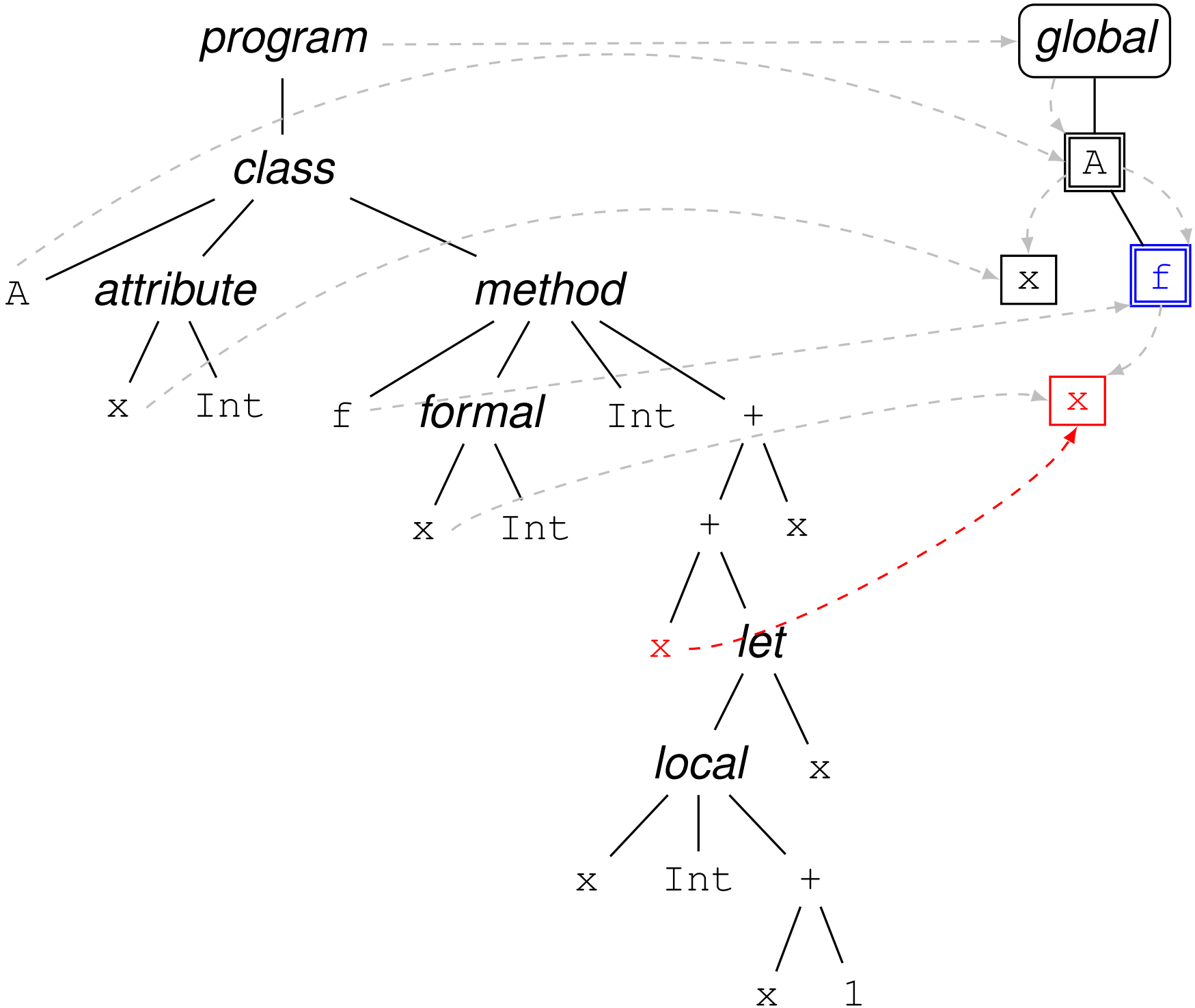

program
global
class
A
A
attribute
method
x
x
Int
f
formal
Int
+
x
Int
+
x
let
x
local
x
x
Int
+
x
1

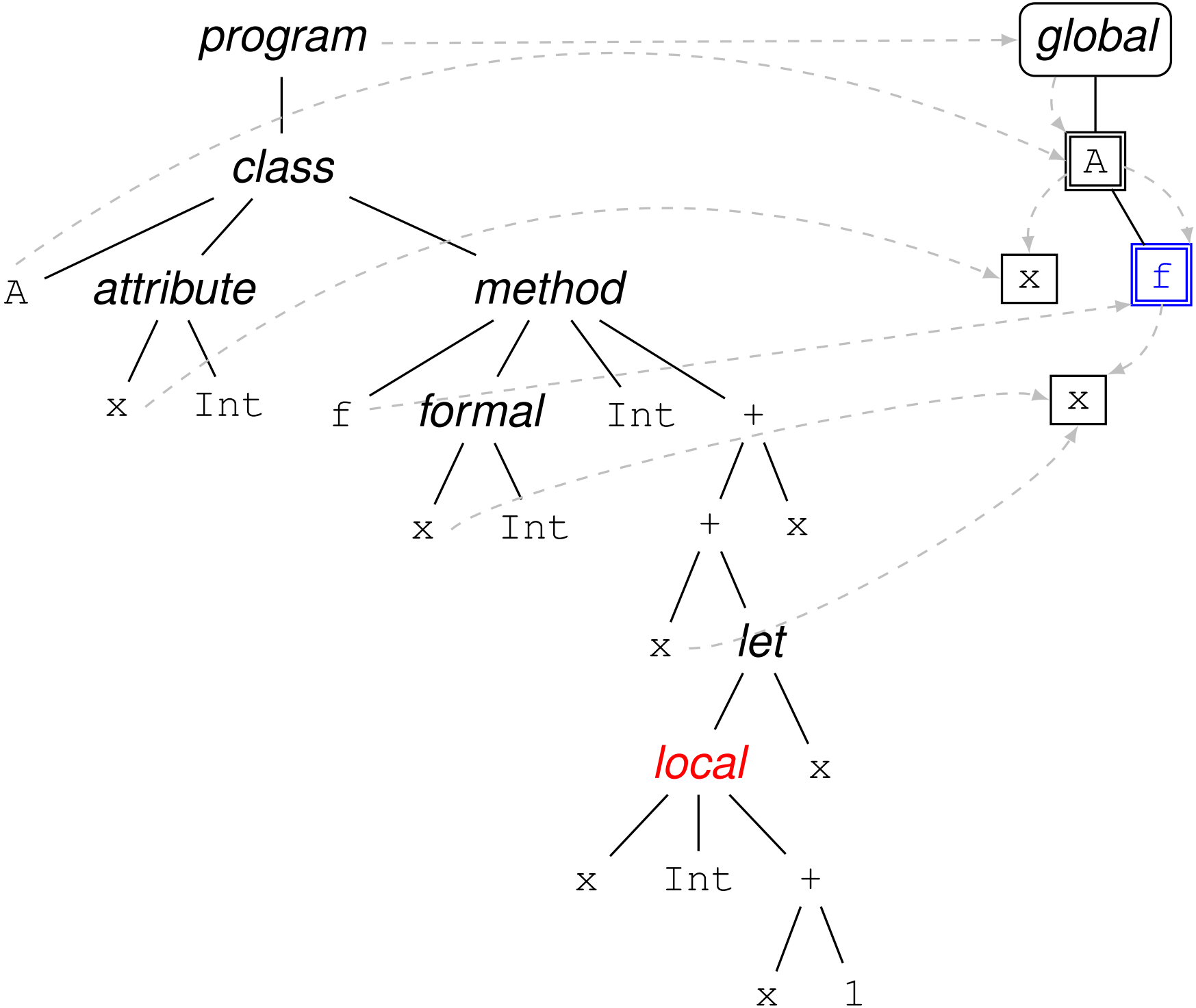

program
global
class
A
A
attribute
method
x
f
x
Int
f
formal
Int
+
x
Int
+
x
let
local
x
x
Int
+
x
1

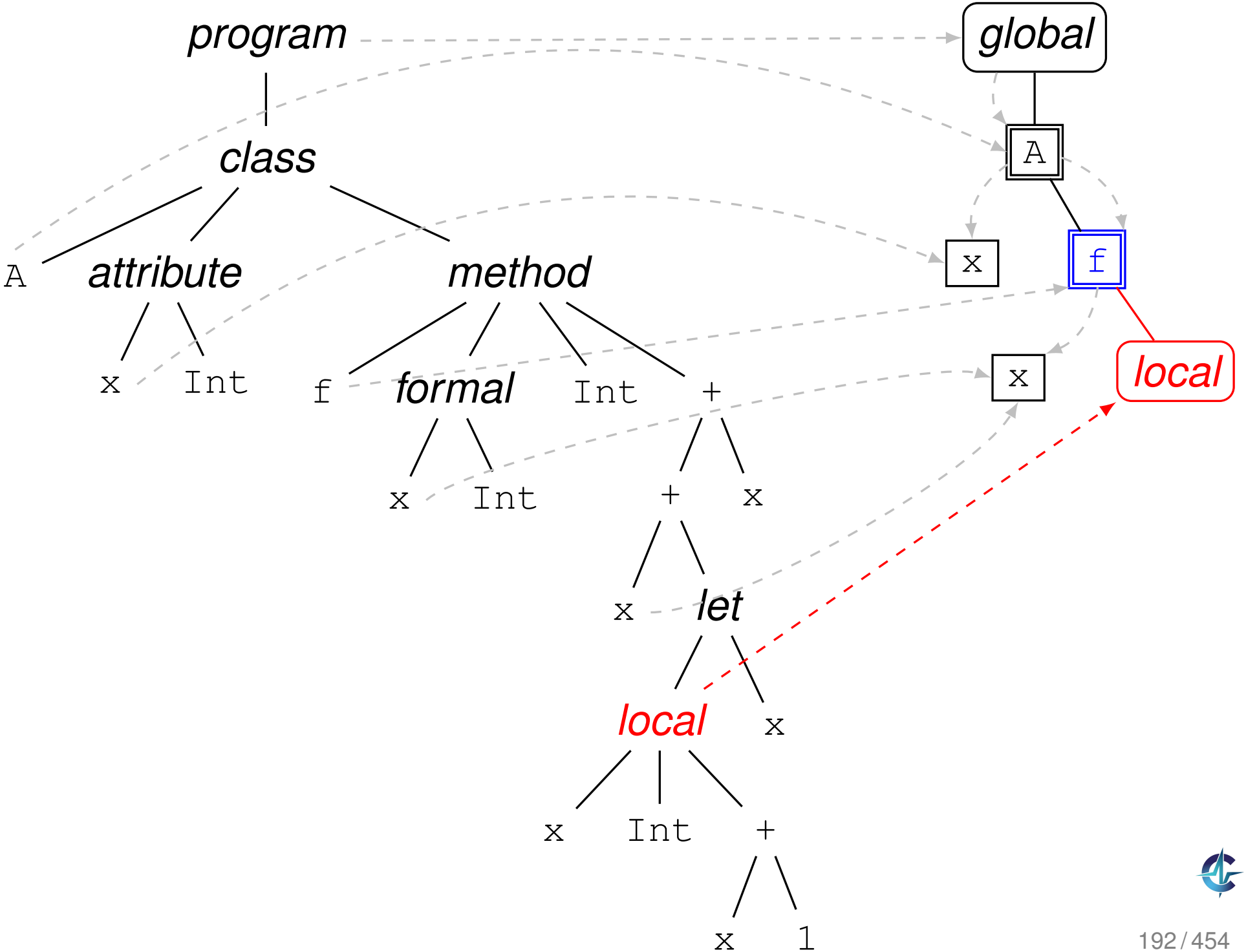

program
class
A
attribute
method
x
Int
f
formal
Int
+
x
Int
+
x
let
x
local
x
x
Int
+
x
1
global
A
x
f

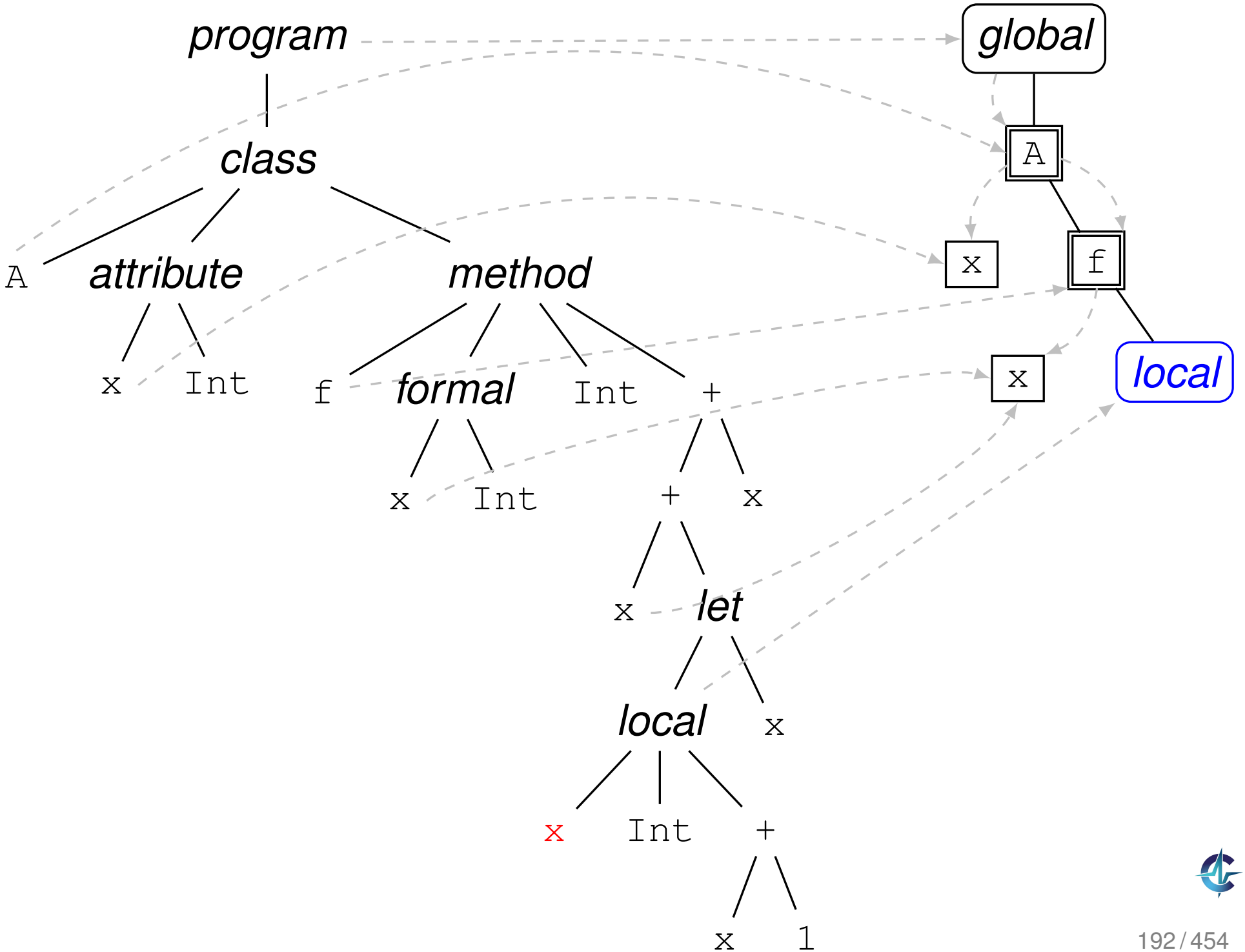

program
class
A
attribute
x
Int
method
f
formal
x
Int
Int
+
+
x
x
let
local
x
x
Int
+
x
1
global
A
x
f
x

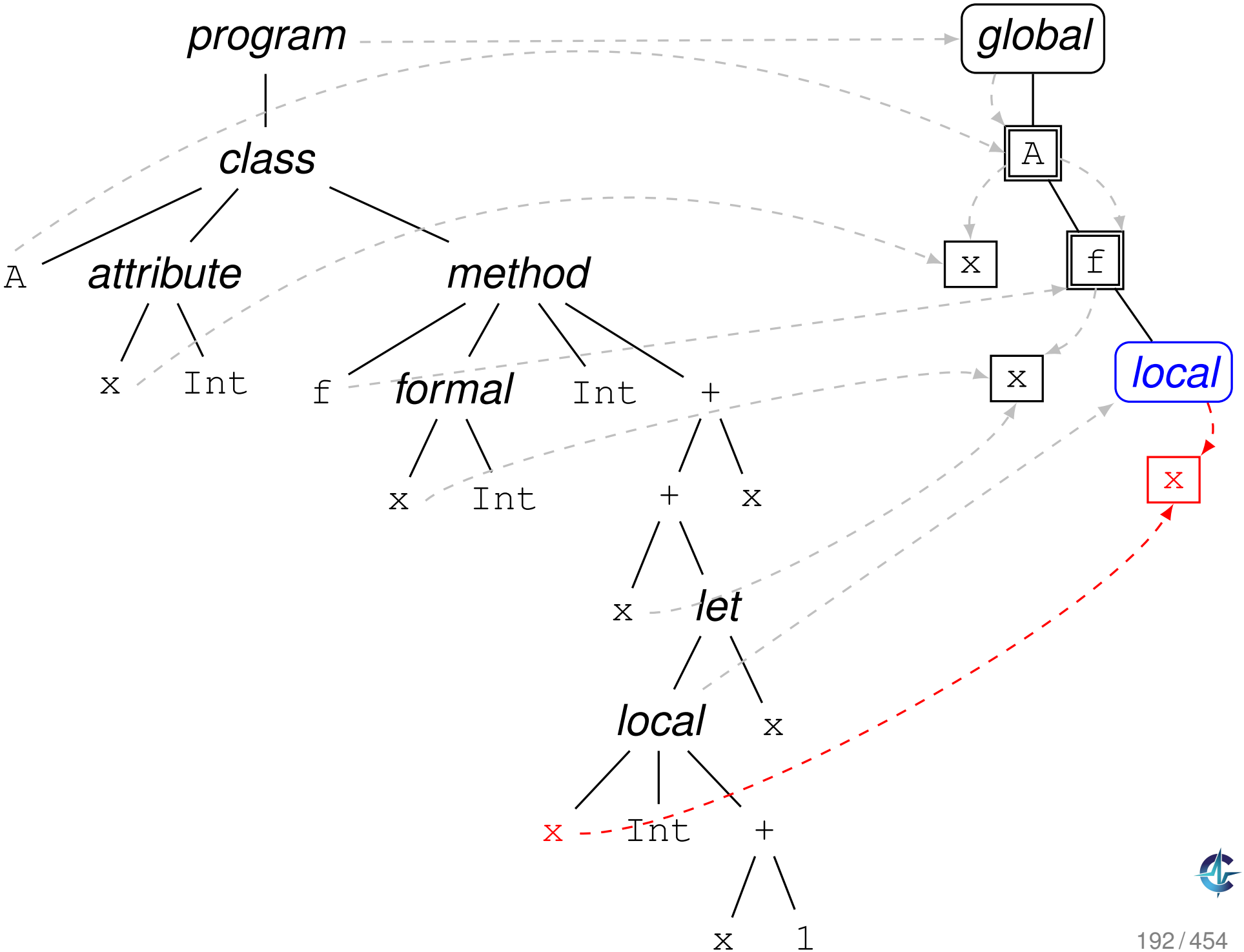

program
class
A
attribute
x
Int
method
f
formal
x
Int
Int
+
+
x
x
let
x
local
x
x
Int
+
x
1
global
A
x
f
x

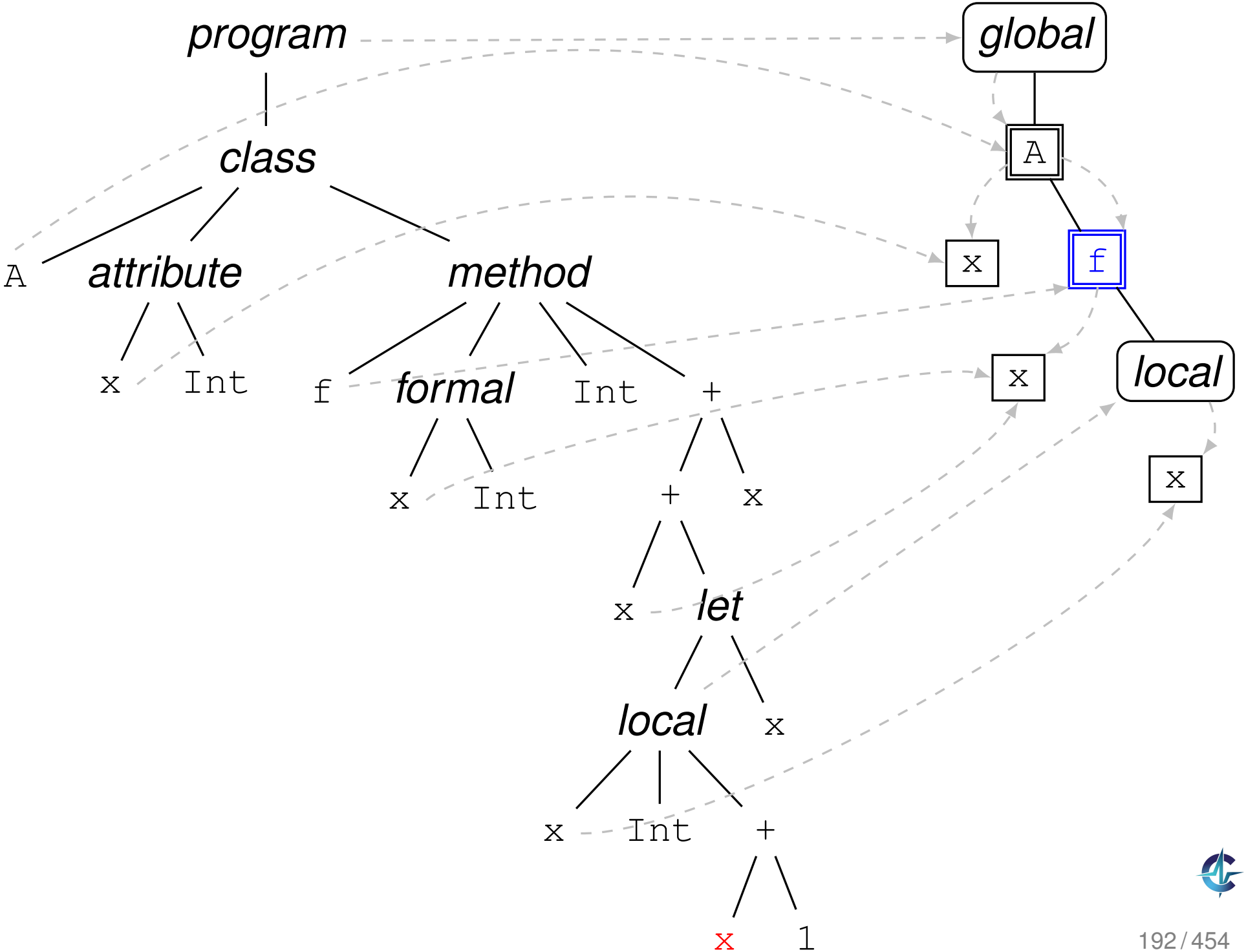

program
class
A
attribute
x
Int
method
f
formal
x
Int
Int
+
x
Int
+
x
let
x
local
x
x
Int
+
x
1
global
A
x
f
x

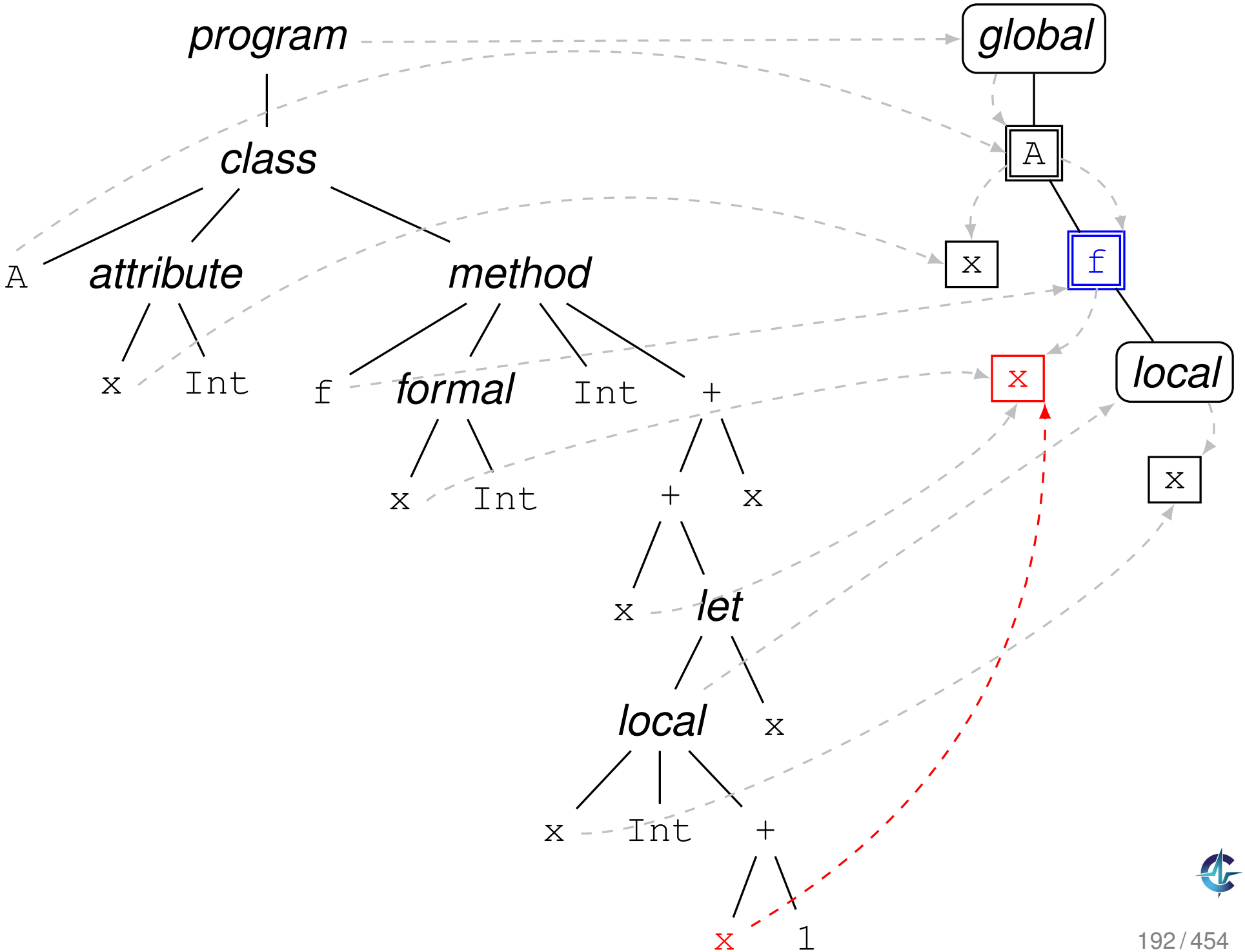

program
class
A
attribute
x
Int
method
f
formal
x
Int
Int
+
+
x
x
let
x
local
x
x
Int
+
x
1
global
A
x
f
x

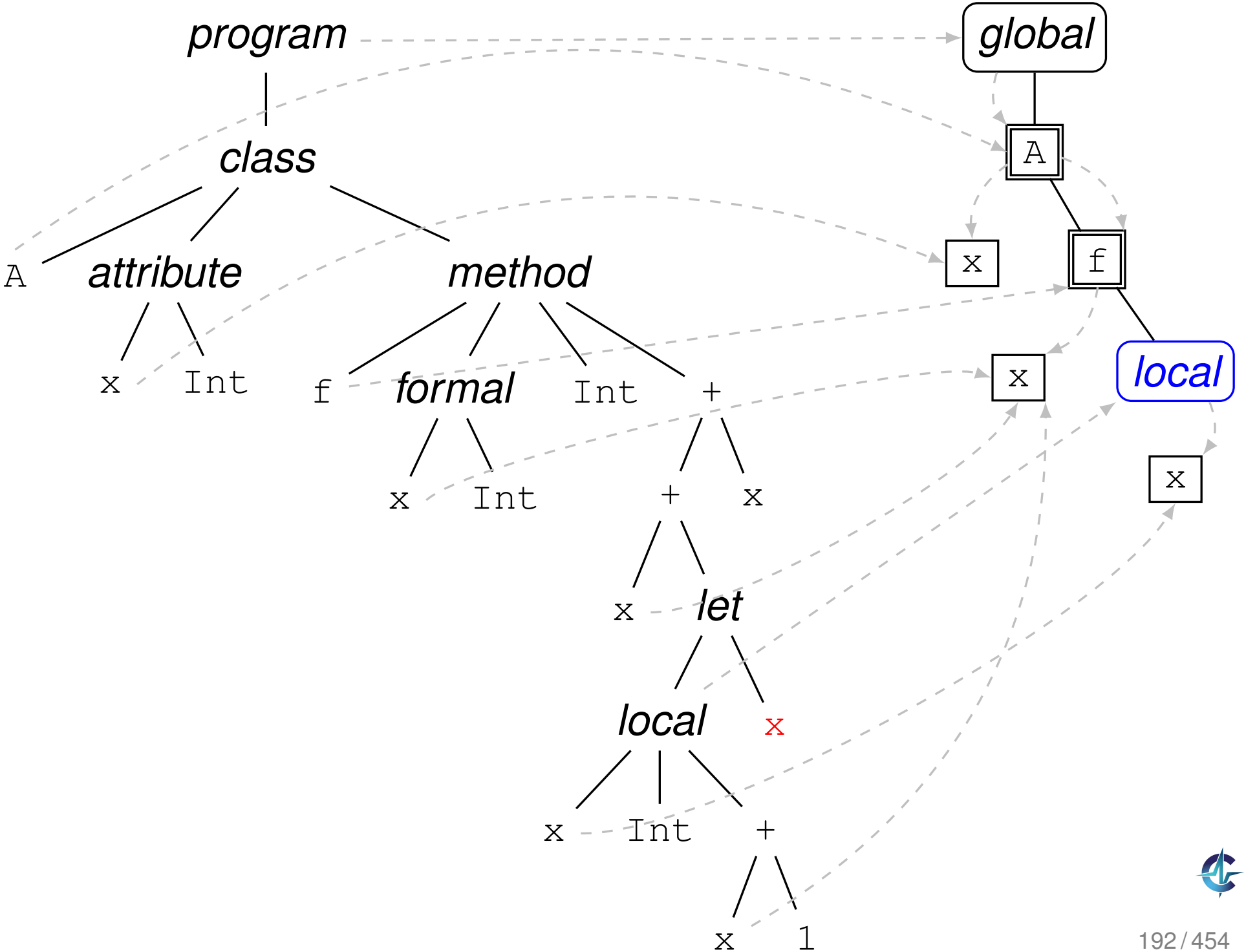

program
class
A
attribute
x
Int
method
f
formal
x
Int
Int
+
x
Int
+
x
let
x
local
x
Int
+
x
1
global
A
x
f
x
local

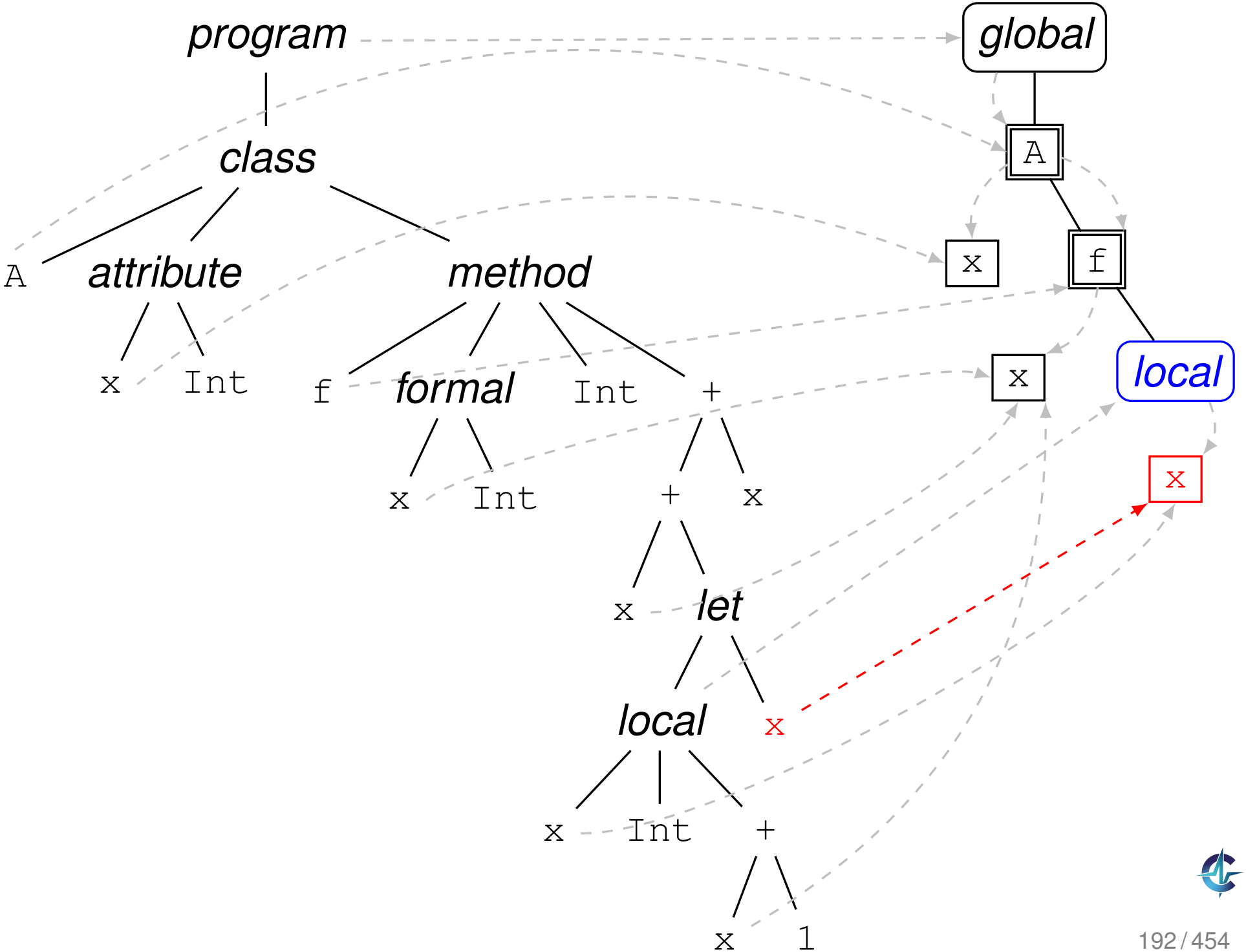

program
class
A
attribute
x
Int
method
f
formal
x
Int
Int
+
+
x
x
let
local
x
x
Int
+
x
1
global
A
x
f
x
local

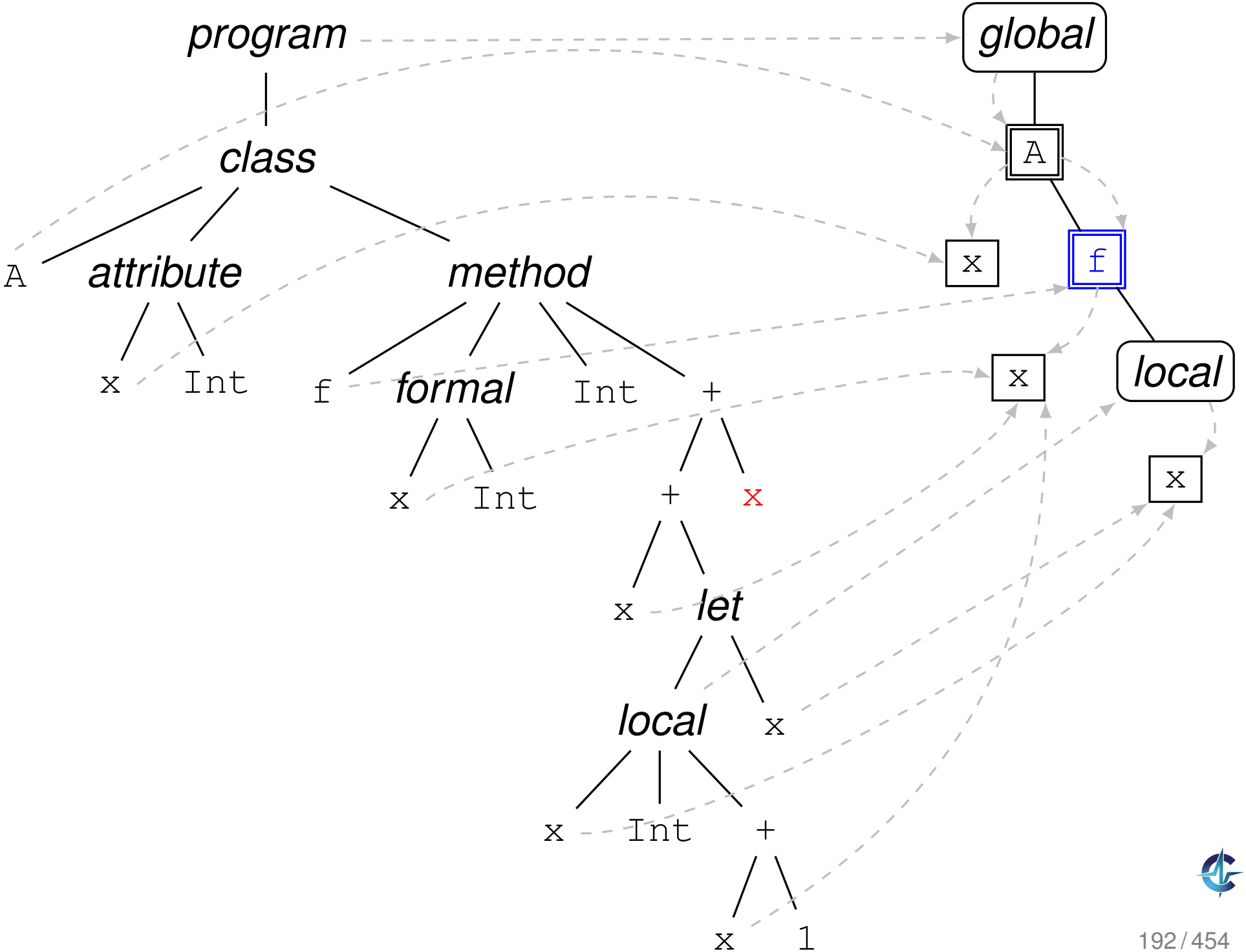

program
class
A
attribute
x
Int
method
f
formal
x
Int
Int
+
+
x
x
let
local
x
x
Int
+
x
1
global
A
x
f
x
local
x

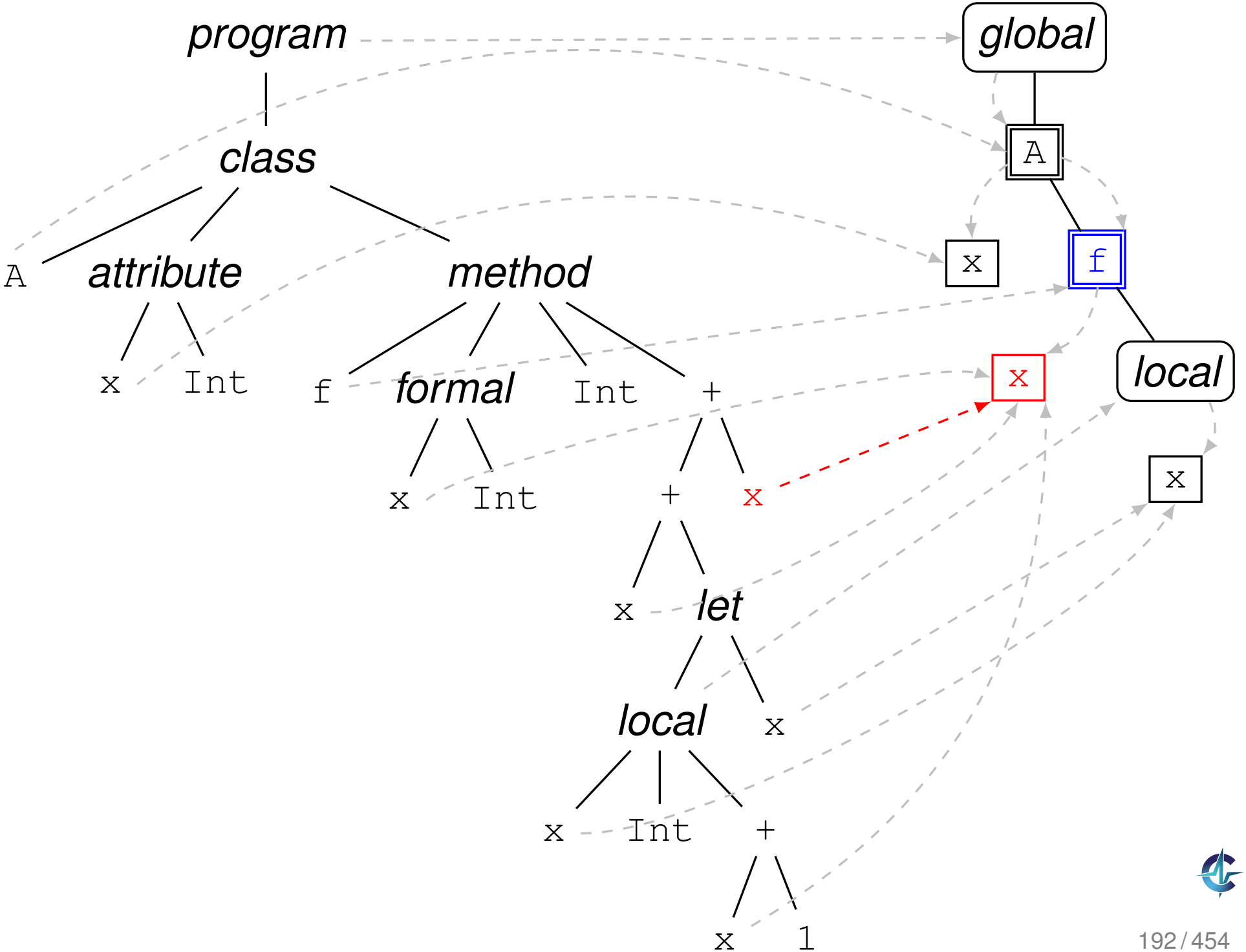

program
class
A
attribute
x
Int
method
f
formal
x
Int
Int
+
x
+
x
let
x
local
x
x
Int
+
x
1
global
A
x
f
x
local
x

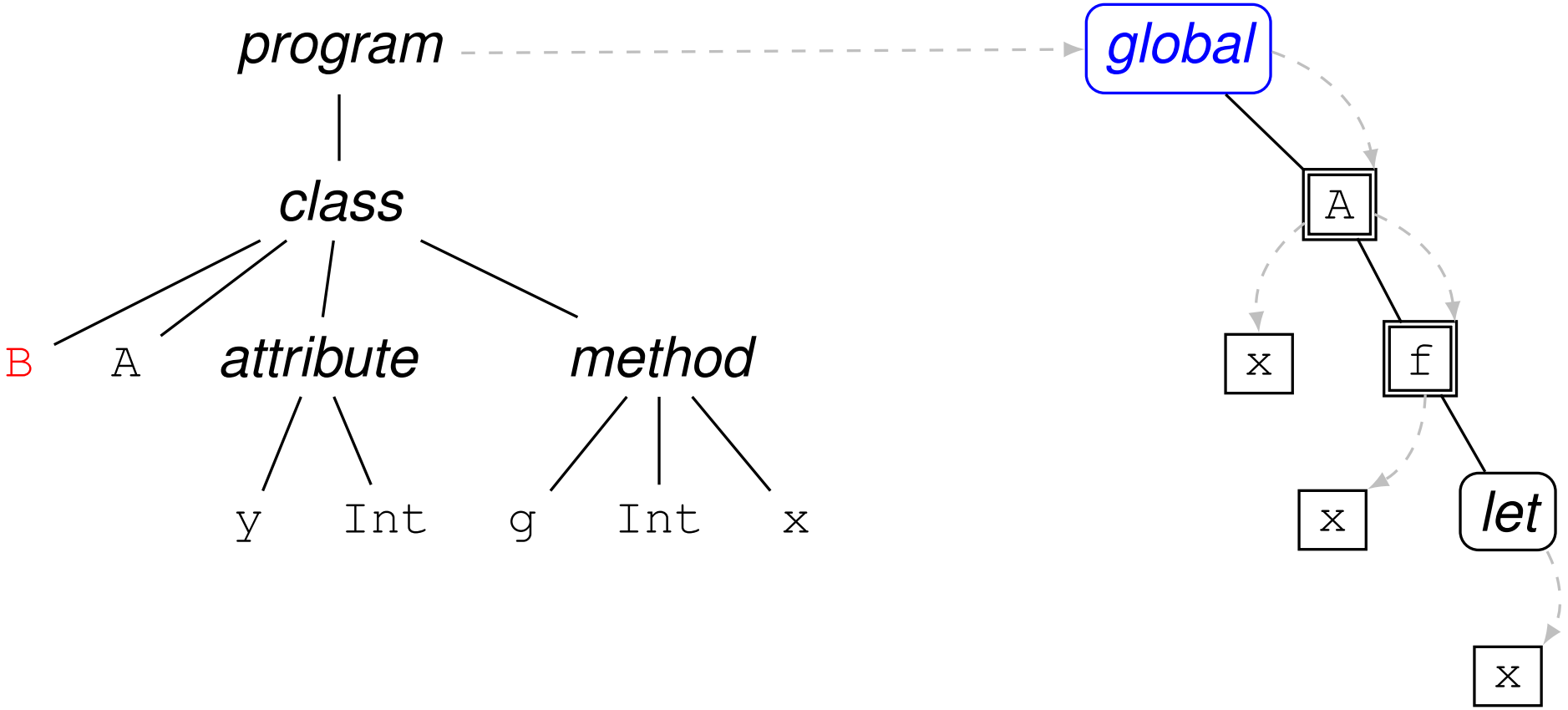

program
class
A
attribute
x
Int
method
f
formal
x
Int
Int
+
+
x
x
let
local
x
x
Int
+
x
1
global
A
x
f
x
local
x

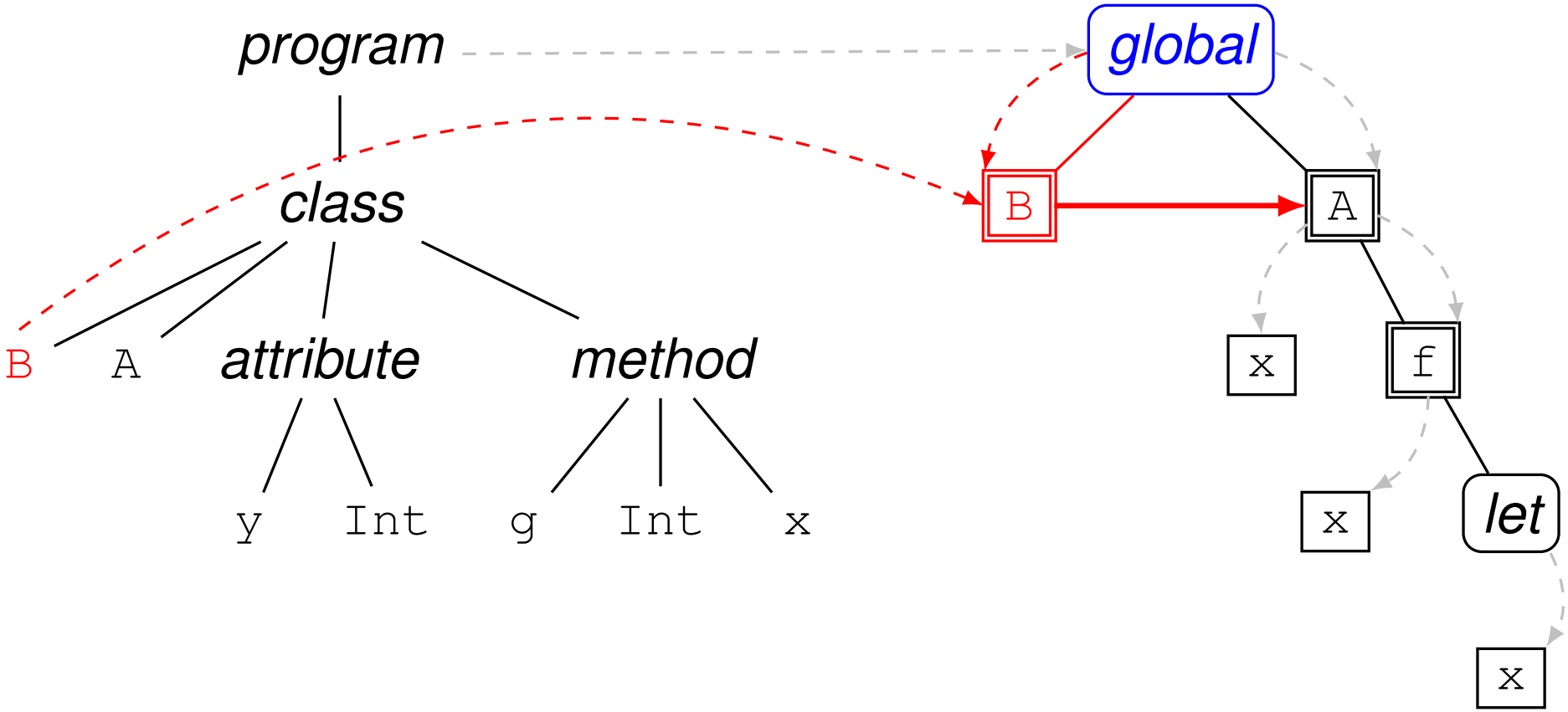

program
class
A
attribute
x
Int
method
f
formal
x
Int
Int
+
+
x
x
let
local
x
x
Int
+
x
1
global
A
x
f
x
local
x


program
class
A
attribute
x
Int
method
f
formal
x
Int
Int
+
+
x
x
let
local
x
x
Int
+
x
1
global
A
x
f
x
local
x

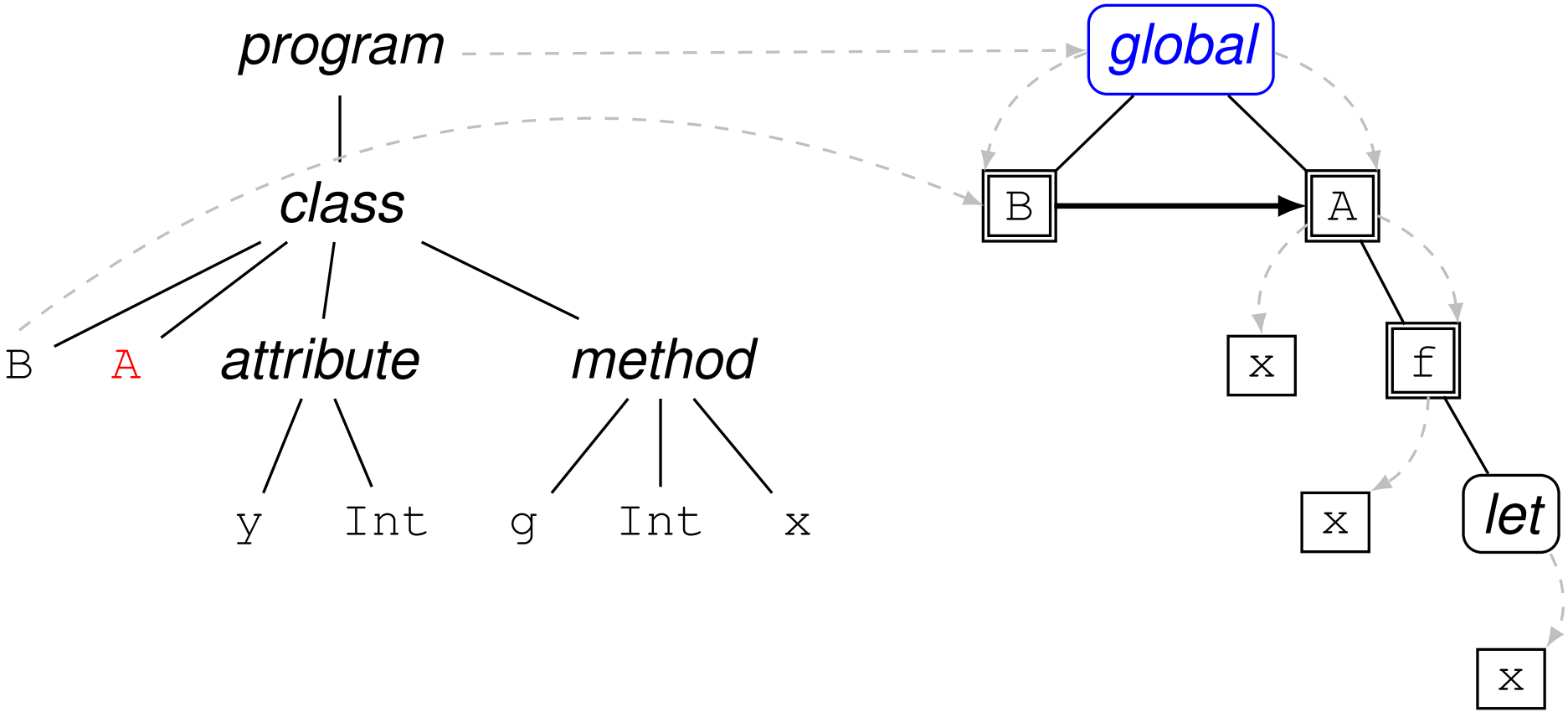

program
class
A
attribute
x
Int
method
f
formal
x
Int
Int
+
x
+
x
let
x
local
x
Int
+
x
1
global
A
x
f
x
local
x


program
class
A
attribute
x
Int
method
f
formal
x
Int
Int
+
x
+
x
let
x
local
x
Int
+
x
1
global
A
x
f
x
local
x

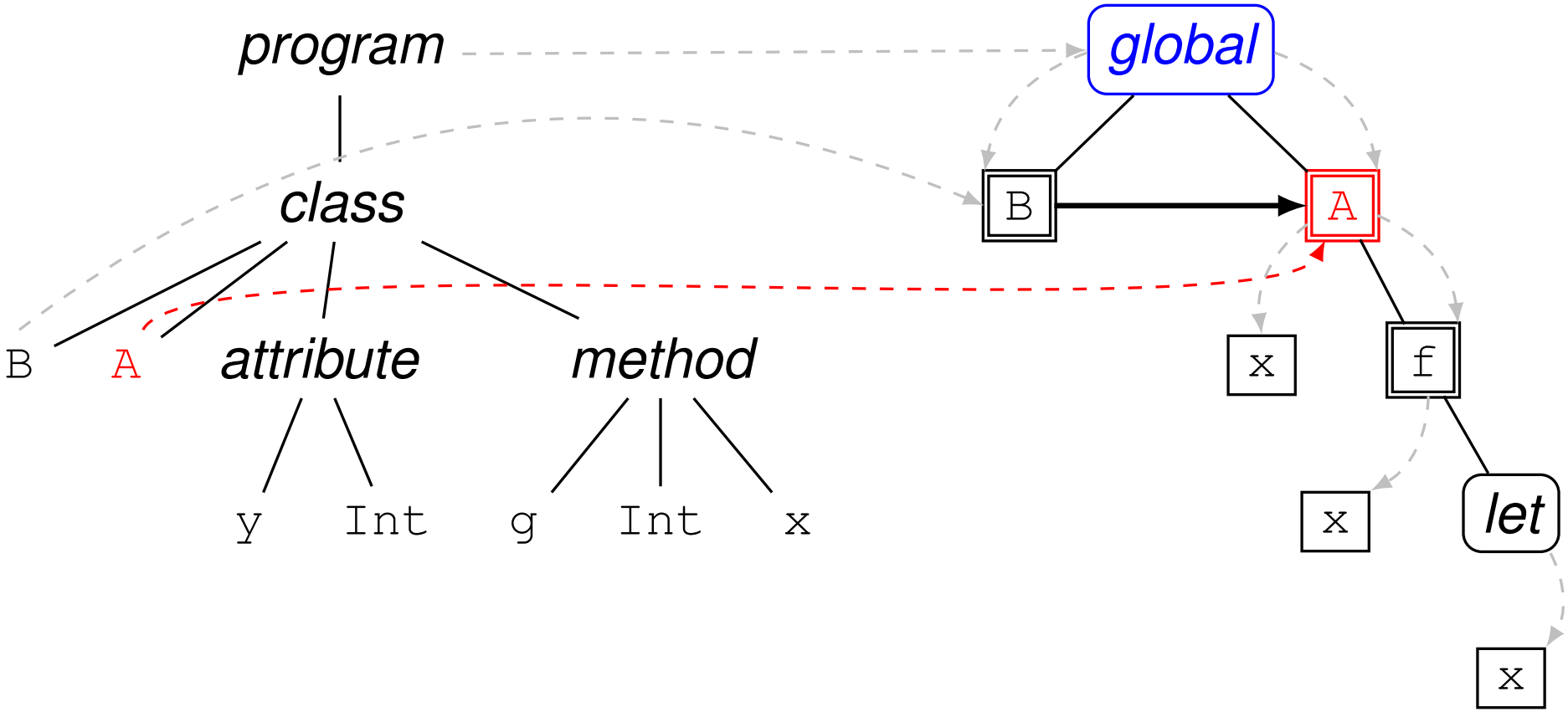

program
class
B
A
attribute
method
y
Int
g
Int
x
global
A
x
f
x
let
x



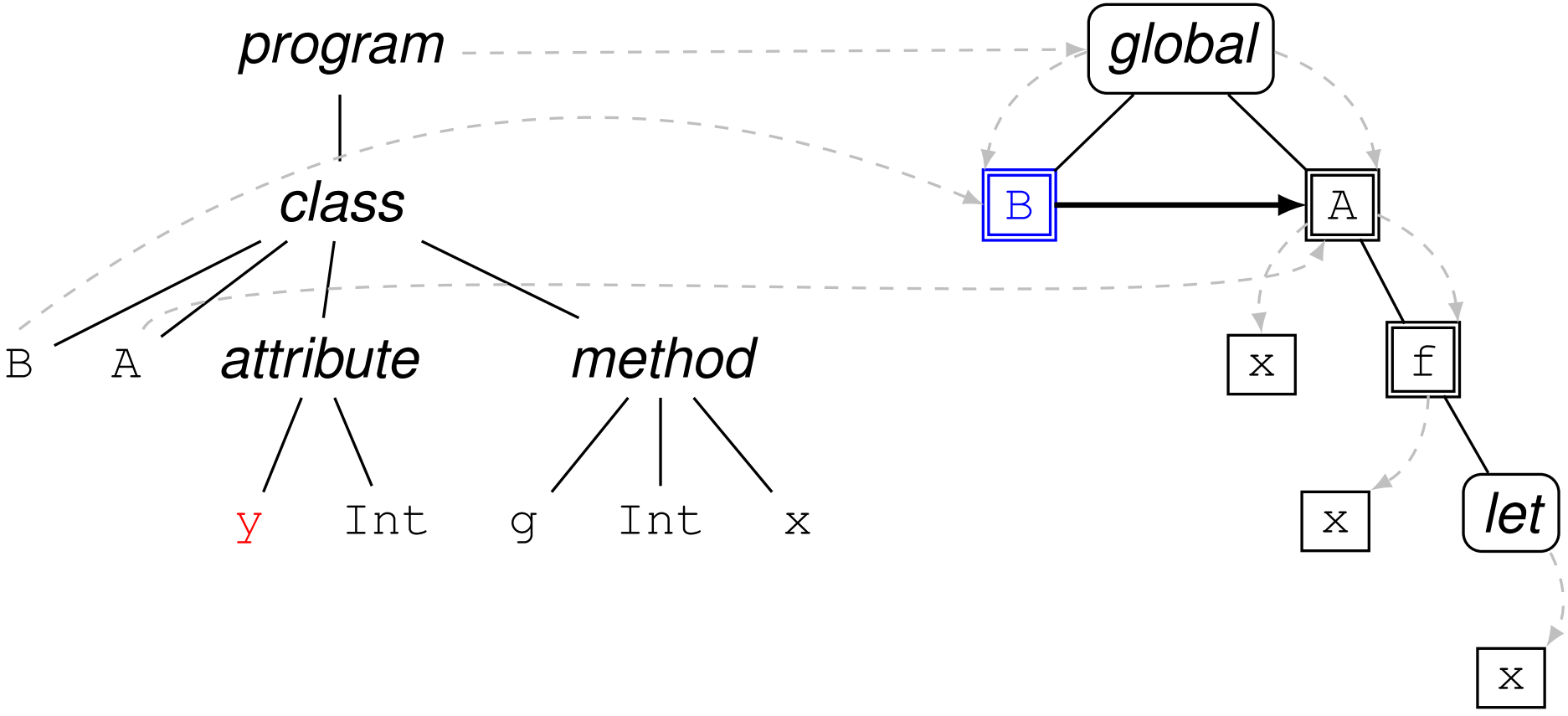
Prezența a doi „părinți” ai domeniului `B`:
`A` (prioritar) și *global*.



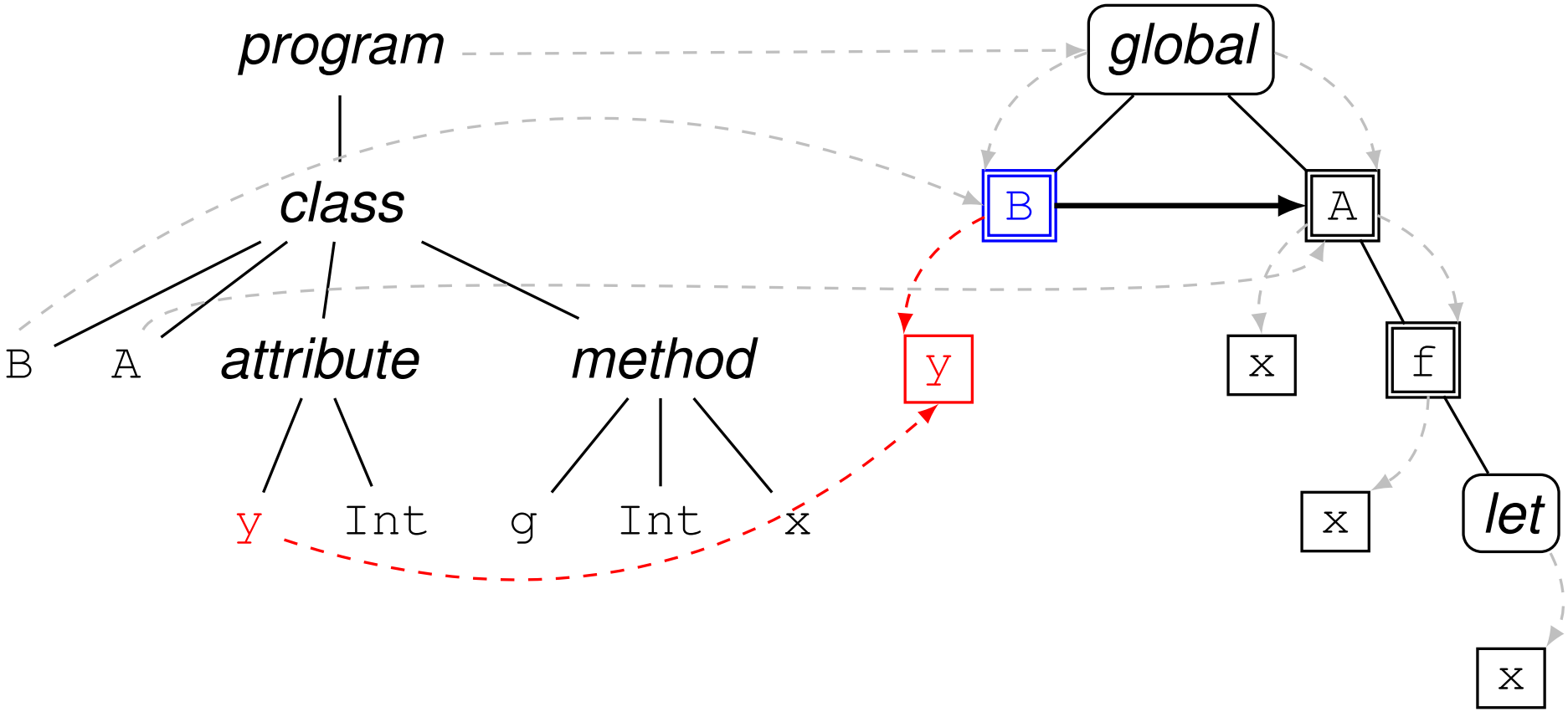
Prezența a doi „părinți” ai domeniului `B`:
`A` (prioritar) și *global*.



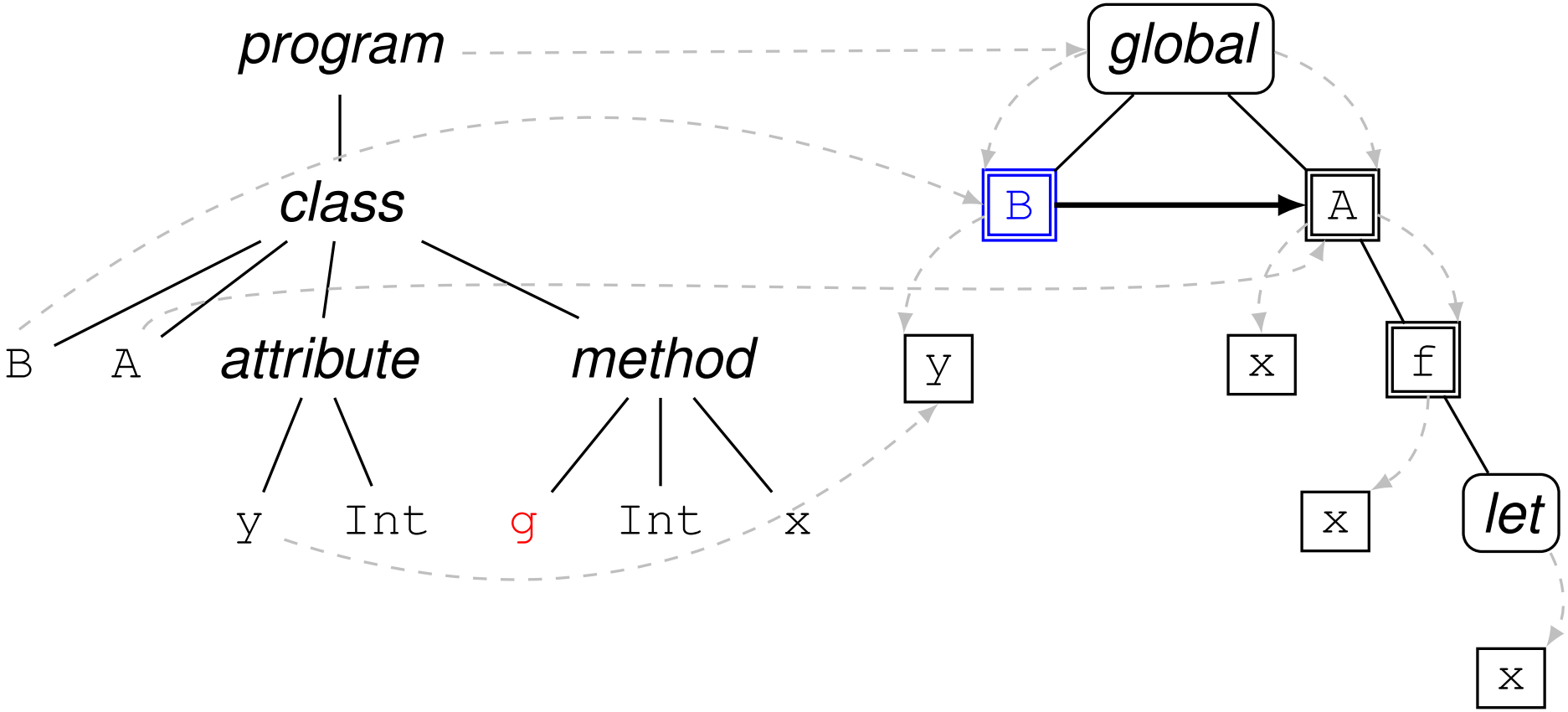
Prezența a doi „părinți” ai domeniului `B`:
`A` (prioritar) și *global*.



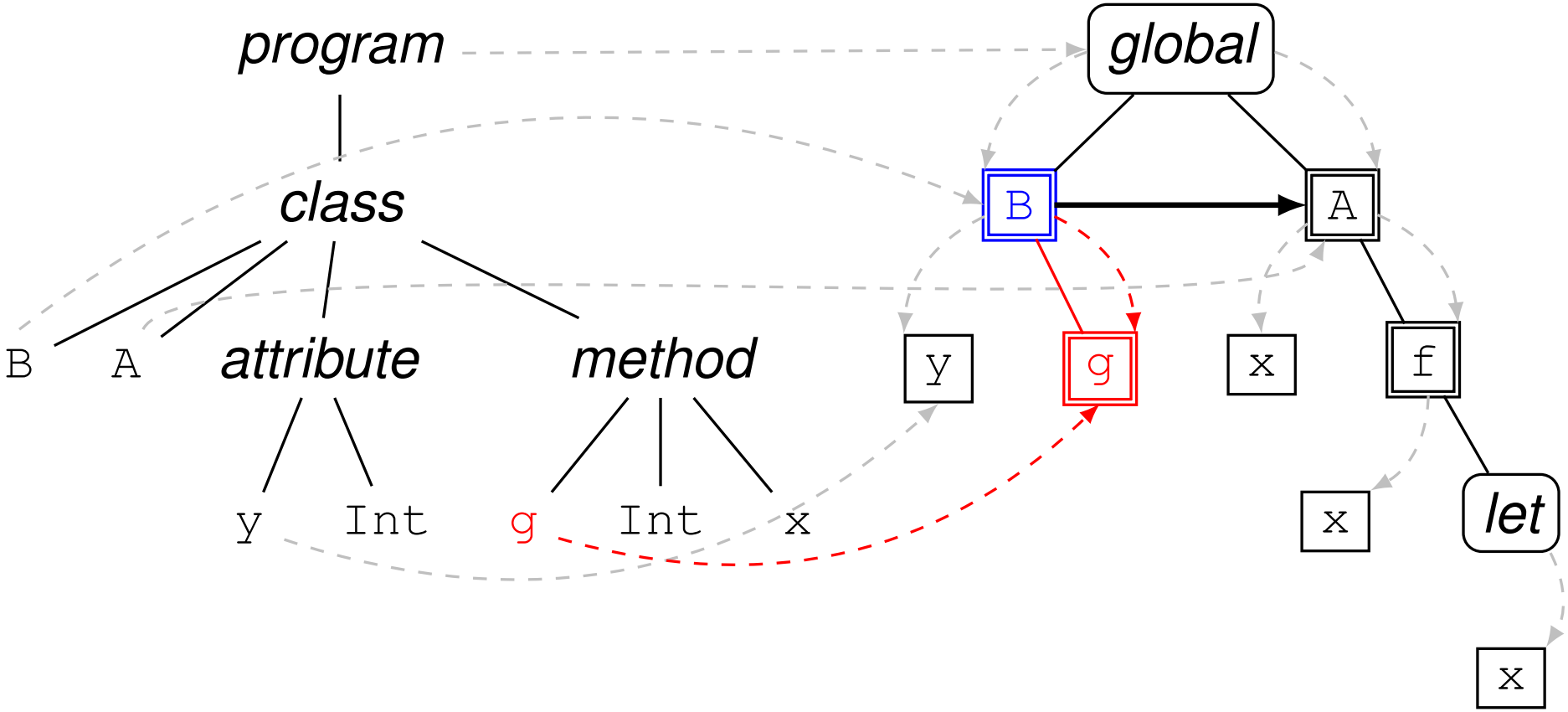
Prezența a doi „părinți” ai domeniului `B`:
`A` (prioritar) și *global*.



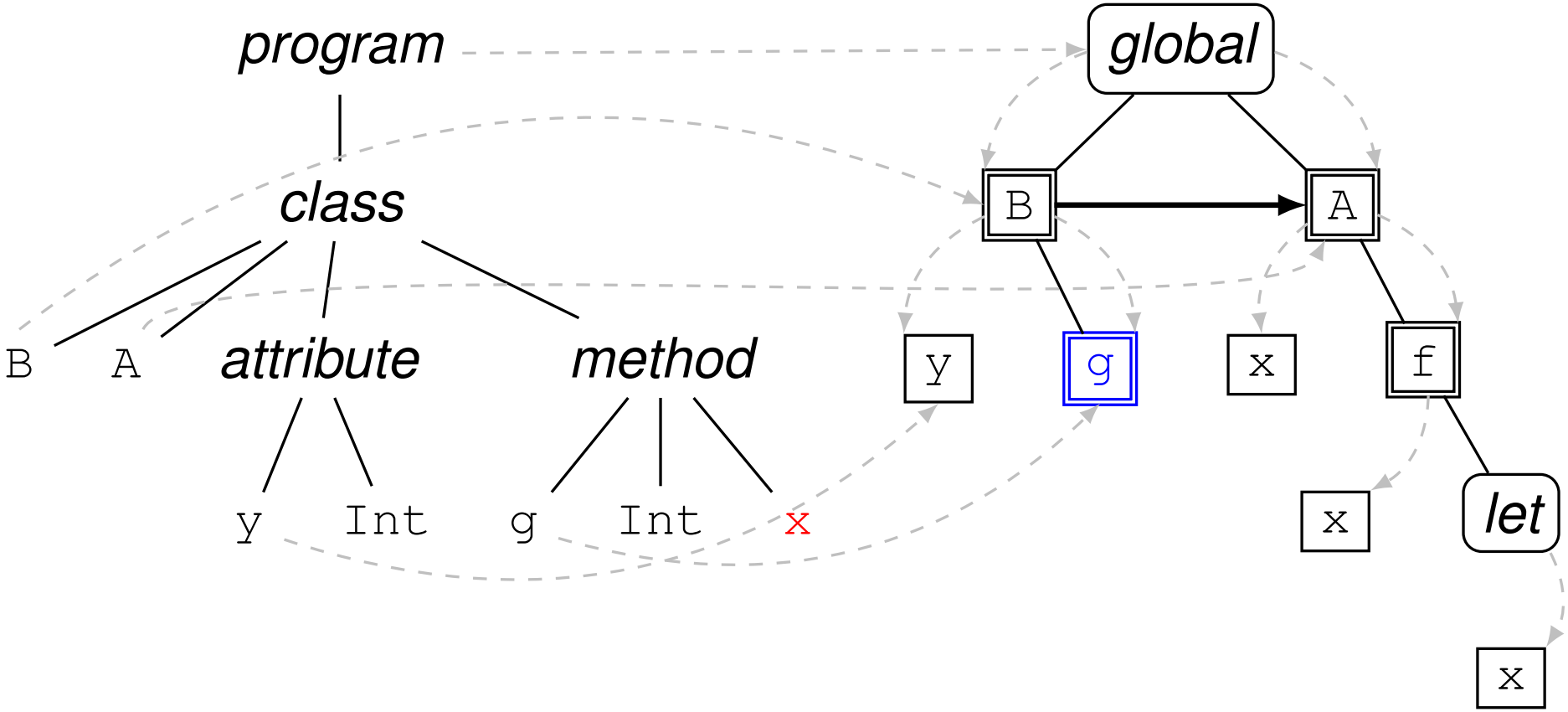
Prezența a doi „părinți” ai domeniului `B`:
`A` (prioritar) și *global*.



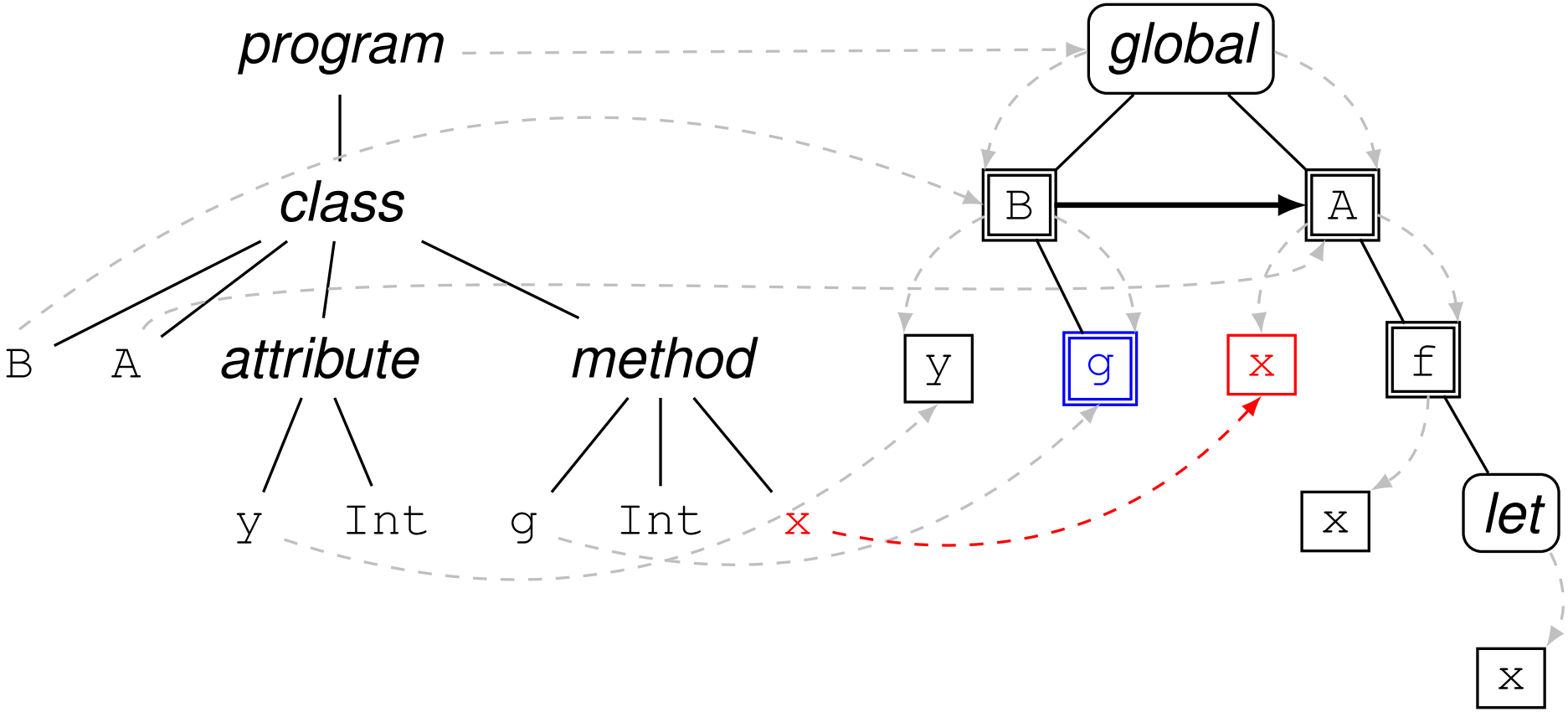
Prezența a doi „părinți” ai domeniului `B`:
`A` (prioritar) și *global*.



Prezența a doi „părinți” ai domeniului `B`:
`A` (prioritar) și *global*.



Prezența a doi „părinți” ai domeniului `B`:
`A` (prioritar) și *global*.



Prezența a doi „părinți” ai domeniului `B`:
`A` (prioritar) și *global*.

Rezolvarea utilizărilor anticipate

- Posibilitatea utilizării claselor, atributelor și metodelor în orice ordine
- Imposibilitatea definirii și căutării simbolurilor într-o singură trecere peste arborele sintactic!
- Necesitatea unor treceri multiple, de obicei, a mai mult de două!

Abordarea în două treceri I

Varianta 1:

- Trecerea de **definire**: simularea intrării și părăsirii domeniilor de vizibilitate, cu adăugarea simbolurilor definite în domeniul curent
- Trecerea de **rezolvare**: reproducerea intrării și părăsirii domeniilor de vizibilitate din etapa 1, cu rezolvarea simbolurilor în domeniul curent

Abordarea în două treceri II

Varianta 2:

- Trecerea de **definire**:
  - Simularea intrării și părăsirii domeniilor de vizibilitate, cu adăugarea simbolurilor definite în domeniul curent
  - Pentru simbolurile ce urmează a fi rezolvate, salvarea domeniului curent, pentru a fi utilizat direct în trecerea următoare
- Trecerea de **rezolvare**: rezolvarea simbolurilor în domeniile salvate în etapa de definire, fără reproducerea intrării și părăsirii domeniilor de vizibilitate

Cuprins

Tipuri

- Mulțime de **valori** și **operații** asociate
- Accepții diverse asupra noțiunii: tipuri de bază (e.g. `int`), structuri, clase
- În Cool, tipuri = **clase**

Rolul tipurilor

- Modalitate de exprimare a intenției programatorului
- Depistarea erorilor
- Documentare: care operații acționează asupra căror obiecte
- Reprezentarea diferențiată a valorilor de tipuri diferite: `1`, `"Hello"`, `true` etc.
- Optimizarea operațiilor specifice
- Verificare formală

Absența tipurilor

- De exemplu, în cazul **codului mașină**
- Toate obiectele, indiferent de natura lor conceptuală, reprezentate prin secvențe de octeți
- Pericolul realizării unor operații fără sens (e.g. suma dintre adresa unui obiect și un boolean)

Sistem de tipuri

- Mulțimea de **reguli și mecanisme** care asigură utilizarea operațiilor în conformitate cu intenția declarată

- Proprietatea de **progres** (*progress*): o expresie fie este o valoare, fie poate fi evaluată în continuare

- Proprietatea de **conservare** (*preservation*): evaluând o expresie, tipul rămâne același

Strategii de verificare a tipurilor

- **Statică**: aproape exclusiv la compilare (C, Java, Cool)
- **Dinamică**: aproape exclusiv la execuție (Python)
- **Absentă**: cod mașină

Comparația strategiilor de verificare a tipurilor

Strategie	Avantaje	Dezavantaje
Statică	▸ Depistarea timpurie a erorilor ▸ Eliminarea efortului la execuție	▸ Uneori, prea restrictivă ▸ Prototipare dificilă
Dinamică	▸ Flexibilitate sporită ▸ Prototipare rapidă	▸ Erori nedepistate ▸ Efort la execuție

Compromisuri între strategii

- Relaxarea verificării **statice** utilizând *cast*-uri
- Dublarea verificării **dinamice** cu anumite adnotări de tip

Verificare vs. sinteză de tip

- **Verificare** de tip: asigurarea corectitudinii în prezența adnotărilor de tip
- Exemplu: știind că `x` are tipul `Int`, este corectă expresia `x + 1`?
- **Sinteză** de tip: determinarea tipului în absența adnotărilor de tip (e.g. Haskell, Java 10)
- Exemplu: ce tip are `x` pentru ca `x + 1` să fie corectă?

Tipuri în Cool

- Tipuri: clase și *SELF_TYPE*
- **Declararea** tipurilor identificatorilor
- **Verificarea** tipurilor expresiilor, utilizând **reguli** de tipare

Reguli de tipare

$$\frac{\vdash Ipoteza_1 \quad \ldots \quad \vdash Ipoteza_n}{\vdash Concluzie} \qquad [Regula]$$

- $\vdash$ = „se poate demonstra că”
- Regula: „Dacă se pot demonstra ipotezele, se poate demonstra și concluzia.”

Reguli de tipare în Cool

$$\frac{i \text{ este un literal întreg}}{\vdash i : Int} \quad [Int]$$

$$\frac{\vdash e_1 : Int \quad \vdash e_2 : Int}{\vdash e_1 + e_2 : Int} \quad [Add]$$

Reguli de tipare în Cool

$$\frac{i \text{ este un literal întreg}}{\vdash i : Int} \quad [Int]$$

$$\frac{\vdash e_1 : Int \quad \vdash e_2 : Int}{\vdash e_1 + e_2 : Int} \quad [Add]$$

Reflectarea structurii arborelui sintactic
în structura demonstrațiilor.

Exemplu

$$\dfrac{\dfrac{0 \text{ este un literal întreg}}{\vdash 0 : \mathit{Int}} \qquad \dfrac{1 \text{ este un literal întreg}}{\vdash 1 : \mathit{Int}}}{\vdash 0 + 1 : \mathit{Int}}$$

Consistență

- Proprietatea de **consistență** (*soundness*):

$$\vdash e : T \quad \Rightarrow \quad e \text{ se evaluează la o valoare de tipul } T$$

- Existența unor multiple reguli consistente, dar nu toate la fel de utile!

$$\frac{i \text{ este un literal întreg}}{\vdash i : \mathit{Object}}$$

Tiparea constantelor

$$\frac{}{\vdash true : Bool}\ [True]$$

$$\frac{s \text{ este un literal șir de caractere}}{\vdash s : String}\ [String]$$

Tiparea lui *new*

$$\frac{}{\vdash new\ T : T}\ [New]$$

Tiparea lui *not* și *while*

$$\frac{\vdash e : Bool}{\vdash not\ e : Bool} \quad [Not]$$

$$\frac{\vdash e_1 : Bool \quad \vdash e_2 : T}{\vdash while\ e_1\ loop\ e_2\ pool : Object} \quad [While]$$

Tiparea variabilelor

$$\frac{x \text{ este o variabilă}}{\vdash x : ?} \quad [Var]$$

Tiparea variabilelor

$$\frac{x \text{ este o variabilă}}{\vdash x : ?} \quad [Var]$$

Necesitatea adăugării unor informații suplimentare despre tipurile variabilelor, utilizând **contexte de tipare**

Contexte de tipare

- **Context de tipare** (*type environment*): dicționar cu declarații de tip pentru variabilele **libere** dintr-o expresie (care nu sunt definite în acea expresie)

- $O \vdash e : T$ = „Se poate demonstra că expresia e are tipul T, pornind de la declarațiile de tip din contextul O.”

Tiparea variabilelor (cont.)

$$\frac{O(x) = T}{O \vdash x : T} \quad [Var]$$

„Variabila x are tipul T,
conform contextului de tipare O.”

Tiparea expresiei *let*

$$\frac{? \vdash e_1 : T_1}{O \vdash \mathit{let}\ x : T_0\ \mathit{in}\ e_1 : T_1} \qquad [\mathit{LetNoInit}]$$

Tiparea expresiei *let*

$$\frac{? \vdash e_1 : T_1}{O \vdash \mathit{let}\ x : T_0\ \mathit{in}\ e_1 : T_1} \qquad [\mathit{LetNoInit}]$$

Necesitatea tipării lui e_1 în contextul O,
îmbogățit cu declarația de tip a lui x.

Îmbogățirea contextelor de tipare

$$O[T/x](x) = T$$
$$O[T/x](y) = O(y), \text{ pentru } y \neq x$$

Îmbogățirea contextelor de tipare

$$O[T/x](x) = T$$
$$O[T/x](y) = O(y), \text{ pentru } y \neq x$$

Comportament de **stivă**:
declarațiile recente le ascund pe cele vechi.

Tiparea expresiei *let* (cont.)

$$\frac{O[T_0/x] \vdash e_1 : T_1}{O \vdash \mathit{let}\ x : T_0\ \mathit{in}\ e_1 : T_1} \qquad [\mathit{LetNoInit}]$$

Tiparea expresiei *let* cu inițializare

$$\frac{O \vdash e_0 : T_0 \qquad O[T_0/x] \vdash e_1 : T_1}{O \vdash \mathit{let}\ x : T_0 \leftarrow e_0\ \mathit{in}\ e_1 : T_1} \qquad [\mathit{LetInit}]$$

Tiparea expresiei *let* cu inițializare

$$\frac{O \vdash e_0 : T_0 \qquad O[T_0/x] \vdash e_1 : T_1}{O \vdash \mathit{let}\ x : T_0 \leftarrow e_0\ \mathit{in}\ e_1 : T_1} \qquad [\mathit{LetInit}]$$

Posibilitatea ca e_0 să aibă drept tip orice subclasă a lui T_0!

Moștenire

Moștenirea = relație de **ordine parțială**:

- $X \leq Y$, dacă X o moștenește pe Y
- **Reflexivitate**: $X \leq X$
- **Tranzitivitate**: dacă $X \leq Y$ și $Y \leq Z$, atunci $X \leq Z$
- **Antisimetrie**: dacă $X \leq Y$ și $Y \leq X$, atunci $X = Y$

Tiparea expresiei *let* cu inițializare (cont.)

$$\frac{O \vdash e_0 : T_0 \quad O[T/x] \vdash e_1 : T_1 \quad T_0 \leq T}{O \vdash \mathit{let}\ x : T \leftarrow e_0\ \mathit{in}\ e_1 : T_1} \qquad [\mathit{LetInit}]$$

Posibilitatea ca e_0 să aibă drept tip orice subclasă a lui T!

Tiparea atribuirilor

$$\frac{\begin{array}{l} O(x) = T_0 \\ O \vdash e_1 : T_1 \\ T_1 \leq T_0 \end{array}}{O \vdash x \leftarrow e_1 : T_1} \qquad [Assign]$$

Posibilitatea ca e_1 să aibă drept tip orice subclasă a lui T_0!

Tiparea atributelor inițializate

$$\frac{O_C(x) = T_0 \quad O_C \vdash e_1 : T_1 \quad T_1 \leq T_0}{O_C \vdash x : T_0 \leftarrow e_1;} \qquad [AttrInit]$$

- O_C = contextul de tipare pentru **atributele** clasei C (x este vizibil în e_1!)
- $O_C(x) = T$, conform declarației $x : T$
- Posibilitatea ca e_1 să aibă drept tip orice subclasă a lui T_0!

Tiparea deciziilor

$$\frac{O \vdash e_0 : Bool \quad O \vdash e_1 : T_1 \quad O \vdash e_2 : T_2}{O \vdash \mathit{if}\ e_0\ \mathit{then}\ e_1\ \mathit{else}\ e_2\ \mathit{fi} : ?} \quad [\mathit{IfThenElse}]$$

Tiparea deciziilor

$$\frac{O \vdash e_0 : Bool \quad O \vdash e_1 : T_1 \quad O \vdash e_2 : T_2}{O \vdash \mathit{if}\ e_0\ \mathit{then}\ e_1\ \mathit{else}\ e_2\ \mathit{fi} : ?} \quad [\mathit{IfThenElse}]$$

Alegerea sigură: cel mai specific tip
care subsumează ambele tipuri, T_1 și T_2.

Marginea superioară minimală

$Z = lub(X, Y)$: **marginea superioară minimală** (*least upper bound*) a două tipuri X și Y:

- Margine **superioară**: $X \leq Z$ și $Y \leq Z$
- Margine superioară **minimală**: dacă $X \leq Z'$ și $Y \leq Z'$, atunci $Z \leq Z'$
- Cel mai adânc strămoș comun al celor două clase în arborele de moștenire

Tiparea deciziilor (cont.)

$$\frac{O \vdash e_0 : Bool \quad O \vdash e_1 : T_1 \quad O \vdash e_2 : T_2}{O \vdash \mathit{if}\ e_0\ \mathit{then}\ e_1\ \mathit{else}\ e_2\ \mathit{fi} : \mathit{lub}(T_1, T_2)} \quad [\mathit{IfThenElse}]$$

Alegerea sigură: cel mai specific tip
care subsumează ambele tipuri, T_1 și T_2.

Tiparea expresiei *case*

$$\frac{\begin{array}{l} O \vdash e_0 : T_0 \\ O[T_1/x_1] \vdash e_1 : T_1' \\ \vdots \\ O[T_n/x_n] \vdash e_n : T_n' \end{array}}{O \vdash \mathit{case}\ e_0\ \mathit{of}\ x_1 : T_1 \Rightarrow e_1;\ \ldots;\ x_n : T_n \Rightarrow e_n;\ \mathit{esac} : \mathit{lub}(T_1', \ldots, T_n')}$$

Alegerea sigură: cel mai specific tip care subsumează tipurile ramurilor.

Tiparea blocurilor

$$\frac{\begin{array}{c} O \vdash e_1 : T_1 \\ \vdots \\ O \vdash e_n : T_n \end{array}}{O \vdash \{\ e_1;\ \ldots;\ e_n;\ \} : T_n} \quad [\mathit{Sequence}]$$

Tipul ultimei expresii din bloc.

Tiparea apelurilor de metodă

$$\frac{\begin{array}{c} O \vdash e_0 : T_0 \\ O \vdash e_1 : T_1 \\ \vdots \\ O \vdash e_n : T_n \end{array}}{O \vdash e_0.f(e_1, \ldots, e_n) : ?} \quad [\mathit{Dispatch}]$$

Tiparea apelurilor de metodă

$$\frac{\begin{array}{c} O \vdash e_0 : T_0 \\ O \vdash e_1 : T_1 \\ \vdots \\ O \vdash e_n : T_n \end{array}}{O \vdash e_0.f(e_1, \ldots, e_n) : ?} \quad [\mathit{Dispatch}]$$

Necesitatea adăugării unor informații suplimentare despre tipurile de retur și ale parametrilor formali ai metodelor.

Contexte de tipare pentru metode

- Dicționar cu declarații pentru tipurile de **retur** și ale parametrilor **formali** ai metodelor din fiecare clasă
- Diferit de contextul de tipare pentru variabile, deoarece metodele și variabilele utilizează spații de nume diferite!
- $M(C, f) = (T_1, \ldots, T_n, T_{n+1})$, conform definiției metodei $f(x_1 : T_1, \ldots, x_n : T_n) : T_{n+1}$ în clasa C

Tiparea apelurilor de metodă (cont.)

$$\frac{\begin{array}{l} O, M \vdash e_0 : T_0 \\ O, M \vdash e_1 : T_1 \\ \vdots \\ O, M \vdash e_n : T_n \\ M(T_0, f) = (T_1', \ldots, T_n', T_{n+1}) \\ T_i \leq T_i', \ 1 \leq i \leq n \end{array}}{O, M \vdash e_0.f(e_1, \ldots, e_n) : T_{n+1}} \quad [\textit{Dispatch}]$$

Posibilitatea parametrilor actuali să aibă drept tipuri subclase ale tipurilor parametrilor formali.

Tiparea apelurilor statice de metodă

$$\frac{\begin{array}{l} O, M \vdash e_0 : T_0 \\ O, M \vdash e_1 : T_1 \\ \vdots \\ O, M \vdash e_n : T_n \\ T_0 \leq T \\ M(T, f) = (T'_1, \ldots, T'_n, T_{n+1}) \\ T_i \leq T'_i,\ 1 \leq i \leq n \end{array}}{O, M \vdash e_0@T.f(e_1, \ldots, e_n) : T_{n+1}} \quad [StaticDispatch]$$

Posibilitatea parametrilor actuali să aibă drept tipuri subclase ale tipurilor parametrilor formali.

Tip static vs. tip dinamic

- Tip **static**: precizat la declararea unui obiect, e.g. $x : A$
- Tip **dinamic**: precizat la instanțierea unui obiect, e.g. $x \leftarrow new\ A$ sau $x \leftarrow new\ B$, unde $B \leq A$
- Proprietatea de **consistență** în absența moștenirii:

$$\forall E.\ dynamic\_type(E) = static\_type(E)$$

- Proprietatea de **consistență** în prezența moștenirii:

$$\forall E.\ dynamic\_type(E) \leq static\_type(E)$$

Restricții impuse de tiparea statică

```
1  class Accumulator {
2      value : Int <- 0;
3
4      accumulate(x : Int) : Accumulator {{
5          value <- value + x;
6          self;
7      }};
8  };
9  class NamedAccumulator inherits Accumulator {
10     name : String;
11 };
12 class Main {
13     -- Eroare: accumulate intoarce Accumulator,
14     -- nu NamedAccumulator!
15     namedAcc : NamedAccumulator
16         <- new NamedAccumulator.accumulate(3);
17 };
```

Relaxarea tipării statice cu *SELF_TYPE*

```
1  class Accumulator {
2      value : Int <- 0;
3
4      accumulate(x : Int) : SELF_TYPE {{
5          value <- value + x;
6          self;
7      }};
8  };
9  class NamedAccumulator inherits Accumulator {
10     name : String;
11 };
12 class Main {
13     -- Corect: accumulate intoarce acum
14     -- NamedAccumulator!
15     namedAcc : NamedAccumulator
16         <- new NamedAccumulator.accumulate(3);
17 };
```

SELF_TYPE I

- Tipul lui *self*, i.e. *Accumulator* sau orice subclasă a sa
- Tip **static**, nu dinamic!
- Sporirea expresivității sistemului de tipuri, prin surprinderea mai fidelă a varietății tipurilor dinamice:

$$\ldots \vdash \mathit{new}\ \mathit{NamedAccumulator}.\mathit{accumulate}(3) : \mathit{NamedAccumulator}$$

- În absența lui:

$$\ldots \vdash \mathit{new}\ \mathit{NamedAccumulator}.\mathit{accumulate}(3) : \mathit{Accumulator}$$

SELF_TYPE II

- În cazul declarației $E : SELF\_TYPE$ din clasa C:

$$dynamic\_type(E) \leq C$$

- Dependența semnificației lui *SELF_TYPE* de clasa C în care apare: $SELF\_TYPE_C$

- Regulă de tipare:

$$SELF\_TYPE_C \leq C$$

- Necesitatea extinderii relațiilor $\leq$ și *lub* pentru a încorpora *SELF_TYPE*

Extinderea $\leq$

- $SELF\_TYPE_C \leq SELF\_TYPE_C$: doar în cadrul aceleiași clase, C
- $SELF\_TYPE_C \quad T$
- $T \quad SELF\_TYPE_C$:

Extinderea $\leq$

- $SELF\_TYPE_C \leq SELF\_TYPE_C$: doar în cadrul aceleiași clase, C
- $SELF\_TYPE_C \leq T$, dacă $C \leq T$
- T $SELF\_TYPE_C$:

Extinderea $\leq$

- $SELF\_TYPE_C \leq SELF\_TYPE_C$: doar în cadrul aceleiași clase, C
- $SELF\_TYPE_C \leq T$, dacă $C \leq T$
- $T \not\leq SELF\_TYPE_C$: posibilitatea adăugării unei subclase A, pe lanțul de moștenire $A \leq T \leq C$, care ar invalida relația

Extinderea *lub*

- $lub(SELF\_TYPE_C, SELF\_TYPE_C) = SELF\_TYPE_C$: doar în cadrul aceleiași clase, C
- $lub(SELF\_TYPE_C, T) = lub(C, T)$
- $lub(T, SELF\_TYPE_C) = lub(T, C)$

Utilizarea lui *SELF_TYPE* I

- *class* T *inherits* T' $\{\ldots\}$: interzisă pentru T și T'
- $x : T$: permisă pentru T
- *let* $x : T$ *in* e: permisă pentru T
- *new* T: permisă pentru T, având ca efect instanțierea unui obiect cu tipul dinamic al lui *self*
- $e_0@T.f(e_1, \ldots, e_n)$: interzisă pentru T
- $f(x : T) : T' \ \{\ldots\}$: interzisă pentru T (de ce?), permisă pentru T' (utilitatea **primară**)

Utilizarea lui *SELF_TYPE* II

De ce în $f(x : T) : T' \{\ldots\}$, T nu poate fi *SELF_TYPE*?

- Posibil apel al lui f: $e_0.f(e_1)$
- Dacă T este *SELF_TYPE*, atunci, conform regulii [*Dispatch*], $e_1 : T_1 \leq T = SELF\_TYPE_C$, întotdeauna falsă

Utilizarea lui *SELF_TYPE* III

De ce în $f(x : T) : T' \{\ldots\}$, T nu poate fi *SELF_TYPE*?

```
1  class A {
2      f(x : SELF_TYPE) : Bool {...};
3  };
4
5  class B inherits A {
6      b : Int;
7      f(x : SELF_TYPE) : Bool {... x.b ...};
8  };
9
10 ...
11     -- Tipare corecta, dar eroare la executie,
12     -- deoarece new A nu contine un b!
13     let x : A <- new B in ... x.f(new A); ...
14 ...
```

Tiparea lui *self*

$$\frac{}{O, M \vdash self : ?} \quad [Self]$$

$$\frac{}{O, M \vdash new\ SELF\_TYPE : ?} \quad [NewSelfType]$$

Tiparea lui *self*

$$\frac{}{O, M \vdash self : ?} \quad [Self]$$

$$\frac{}{O, M \vdash new\ SELF\_TYPE : ?} \quad [NewSelfType]$$

Necesitatea adăugării de informații suplimentare despre clasa curentă în care se află expresia!

Tiparea lui *self* (cont.)

$$\frac{}{O, M, C \vdash self : SELF\_TYPE_C} \quad [Self]$$

$$\frac{}{O, M, C \vdash new\ SELF\_TYPE : SELF\_TYPE_C} \quad [NewSelfType]$$

C = clasa curentă în care se află expresia.

Extinderea regulilor de tipare

- Majoritatea regulilor existente, nemodificate, dar înzestrate cu noile definiții ale relațiilor $\leq$ și *lub*

- Modificări în special în regulile de tipare a apelurilor de metodă

Tiparea apelurilor de metodă I

$$\frac{\begin{array}{l} O, M, C \vdash e_0 : T_0 \\ O, M, C \vdash e_1 : T_1 \\ \vdots \\ O, M, C \vdash e_n : T_n \\ M(T_0, f) = (T'_1, \ldots, T'_n, T_{n+1}) \\ T_{n+1} \neq \mathit{SELF\_TYPE} \\ T_i \leq T'_i,\ 1 \leq i \leq n \end{array}}{O, M, C \vdash e_0.f(e_1, \ldots, e_n) : T_{n+1}}$$

$$\frac{\begin{array}{l} O, M, C \vdash e_0 : T_0 \\ O, M, C \vdash e_1 : T_1 \\ \vdots \\ O, M, C \vdash e_n : T_n \\ M(T_0, f) = (T'_1, \ldots, T'_n, \mathit{SELF\_TYPE}) \\ T_i \leq T'_i,\ 1 \leq i \leq n \end{array}}{O, M, C \vdash e_0.f(e_1, \ldots, e_n) : T_0}$$

Tiparea apelurilor de metodă II

- Dar dacă $O, M, C \vdash e_0 : SELF\_TYPE_C$?
- Atunci, $M(C, f) = \ldots$

Tiparea apelurilor de metodă III

$$\frac{\begin{array}{l} O, M, C \vdash e_0 : T_0 \\ O, M, C \vdash e_1 : T_1 \\ \vdots \\ O, M, C \vdash e_n : T_n \\ T_0' = \begin{cases} C, & \text{dacă } T_0 = SELF\_TYPE_C \\ T_0, & \text{altfel} \end{cases} \\ M(T_0', f) = (T_1', \ldots, T_n', T_{n+1}') \\ T_i \leq T_i', \ 1 \leq i \leq n \\ T_{n+1} = \begin{cases} T_0, & \text{dacă } T_{n+1}' = SELF\_TYPE \\ T_{n+1}', & \text{altfel} \end{cases} \end{array}}{O, M, C \vdash e_0.f(e_1, \ldots, e_n) : T_{n+1}}$$

Partea VII

Organizarea execuției și generarea de cod

Cuprins

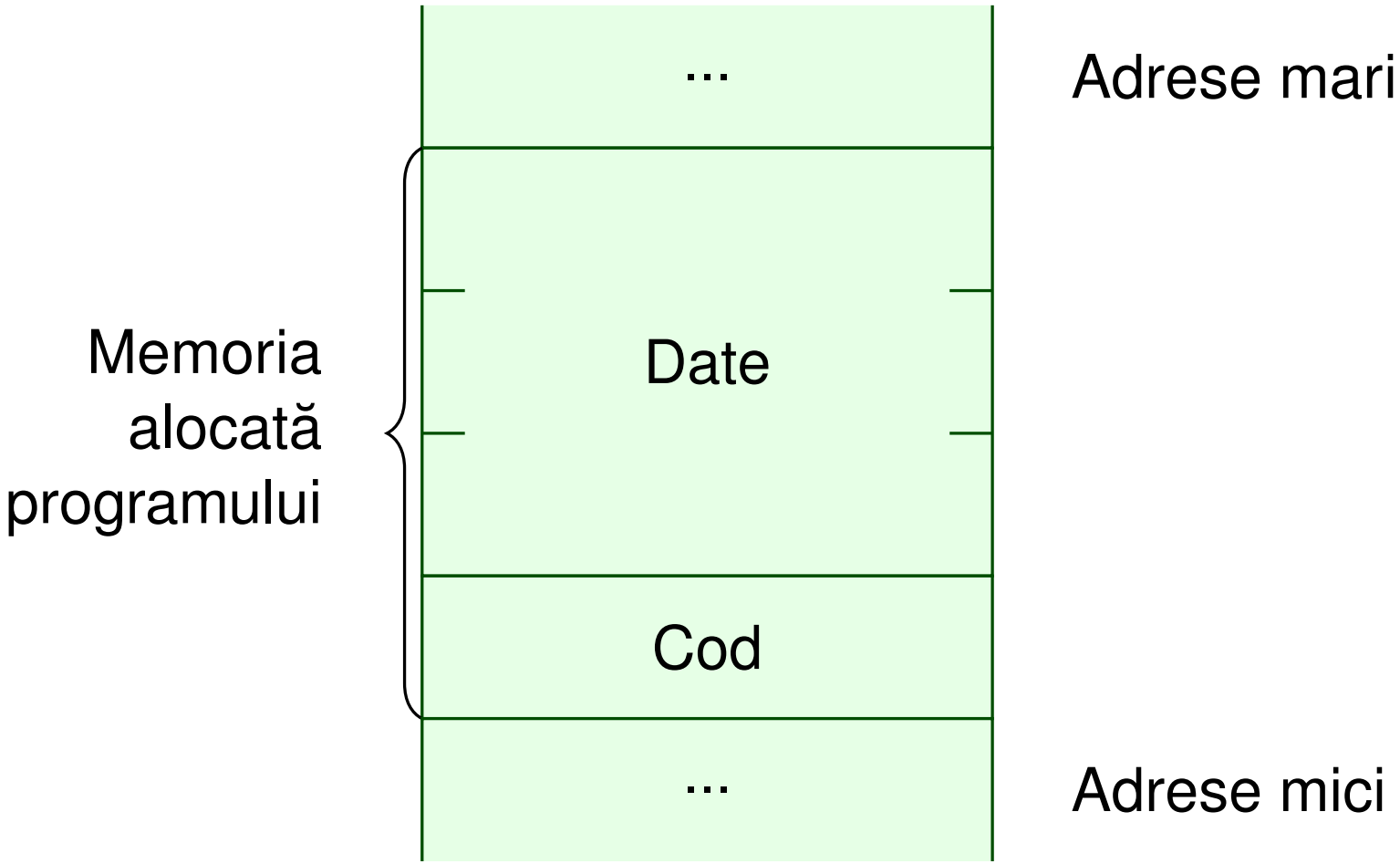
Cuprins

Etapa curentă

- ***Front-end***-ul compilatorului: analiză lexicală, sintactică și semantică
- ***Middle-end***-ul compilatorului: optimizare
- ***Back-end***-ul compilatorului: generare de cod, constând în selecția și planificarea instrucțiunilor, și alocarea registrelor
- Proiectare concertată a modalităților de generare de cod și de gestiune a memoriei

Cuprins

Organizarea memoriei



Presupuneri despre execuție

- Flux secvențial de instrucțiuni, executate într-o ordine prestabilită (fără paralelism/concurență)
- După apelul unei proceduri, revenirea în punctul consecutiv apelului (fără excepții sau continuări)

Proceduri

- **Activare** a unei proceduri = apel al acelei proceduri
- **Durata de viață** a unei activări = totalitatea instrucțiunilor necesare execuției procedurii, inclusiv a apelurilor imbricate
- Imbricarea activărilor, cu posibilitatea reprezentării lor utilizând **arbori de activare** sau o **stivă**

Variabile

- **Durata de viață** a unei variabile = intervalul în care variabila este definită, între momentul alocării și cel al eliberării memoriei aferente

- Durată de viață (concept dinamic) vs. **domeniu de vizibilitate** (concept static)

Arbori de activare și stiva

```
1  class Main {
2        f(x : Int) : Int { x };
3        g(x : Int) : Int { f(x) };
4        main() : Int {{ f(0); g(1); }};
5  };
```

Arbore de activare

Stivă

Arbori de activare și stiva

```
1  class Main {
2        f(x : Int) : Int { x };
3        g(x : Int) : Int { f(x) };
4        main() : Int {{ f(0); g(1); }};
5  };
```

main

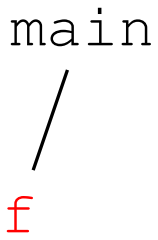
main

Arbore de activare

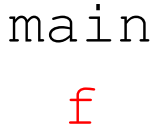
Stivă

Arbori de activare și stiva

```
1  class Main {
2        f(x : Int) : Int { x };
3        g(x : Int) : Int { f(x) };
4        main() : Int {{ f(0); g(1); }};
5  };
```



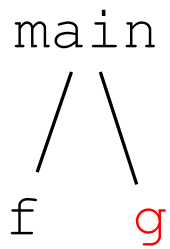
Arbore de activare



Stivă

Arbori de activare și stiva

```
1  class Main {
2        f(x : Int) : Int { x };
3        g(x : Int) : Int { f(x) };
4        main() : Int {{ f(0); g(1); }};
5  };
```

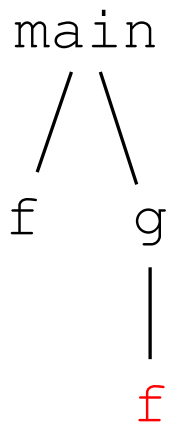


Arbore de activare

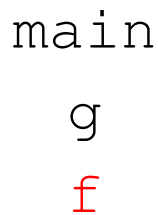
Stivă

Arbori de activare și stiva

```
1  class Main {
2        f(x : Int) : Int { x };
3        g(x : Int) : Int { f(x) };
4        main() : Int {{ f(0); g(1); }};
5  };
```

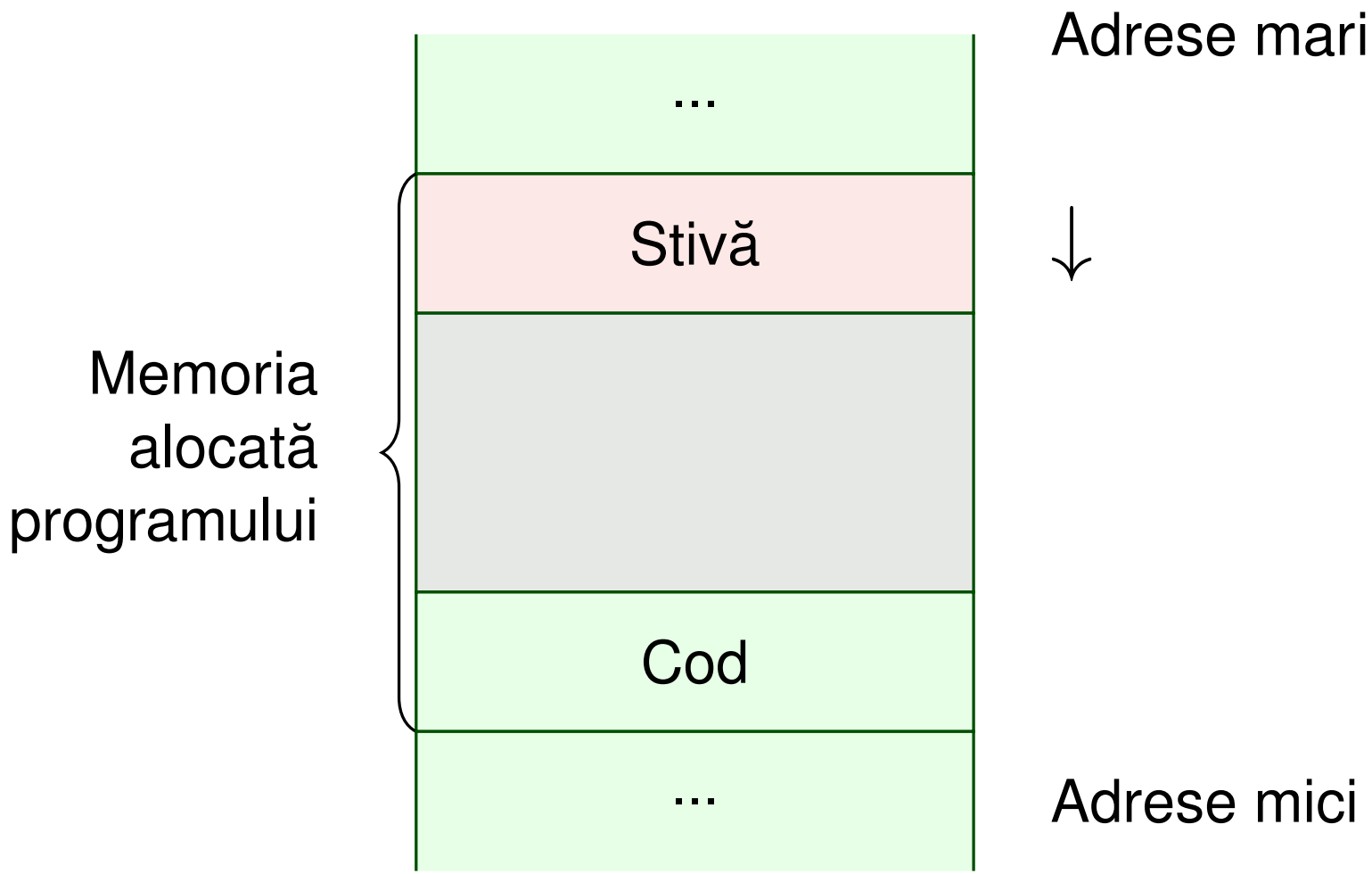


Arbore de activare



Stivă

Organizarea memoriei, nuanțată



Înregistrări de activare

- **Înregistrare de activare** (*activation record*, AR) = totalitatea informațiilor necesare unei activări de procedură

- Informații mixte, legate atât de apelat, cât și de apelant și de starea mașinii

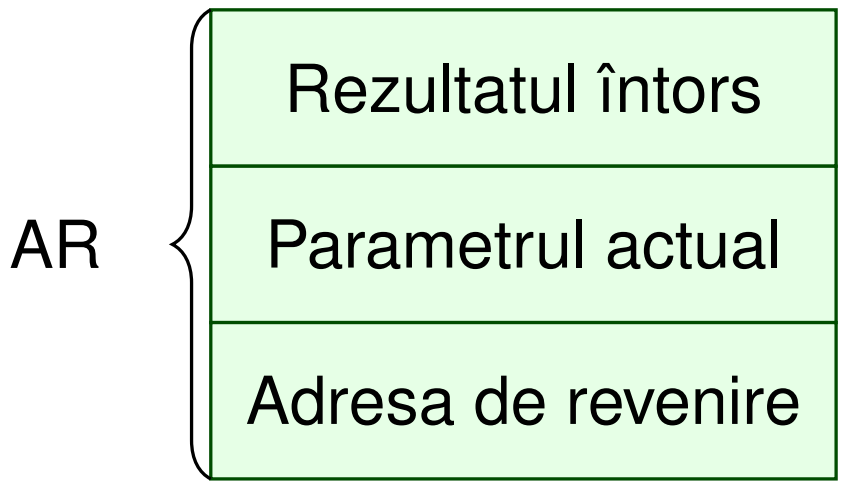
Conținutul înregistrărilor de activare

Conținutul AR aferent procedurii `f`, apelate de `g`:

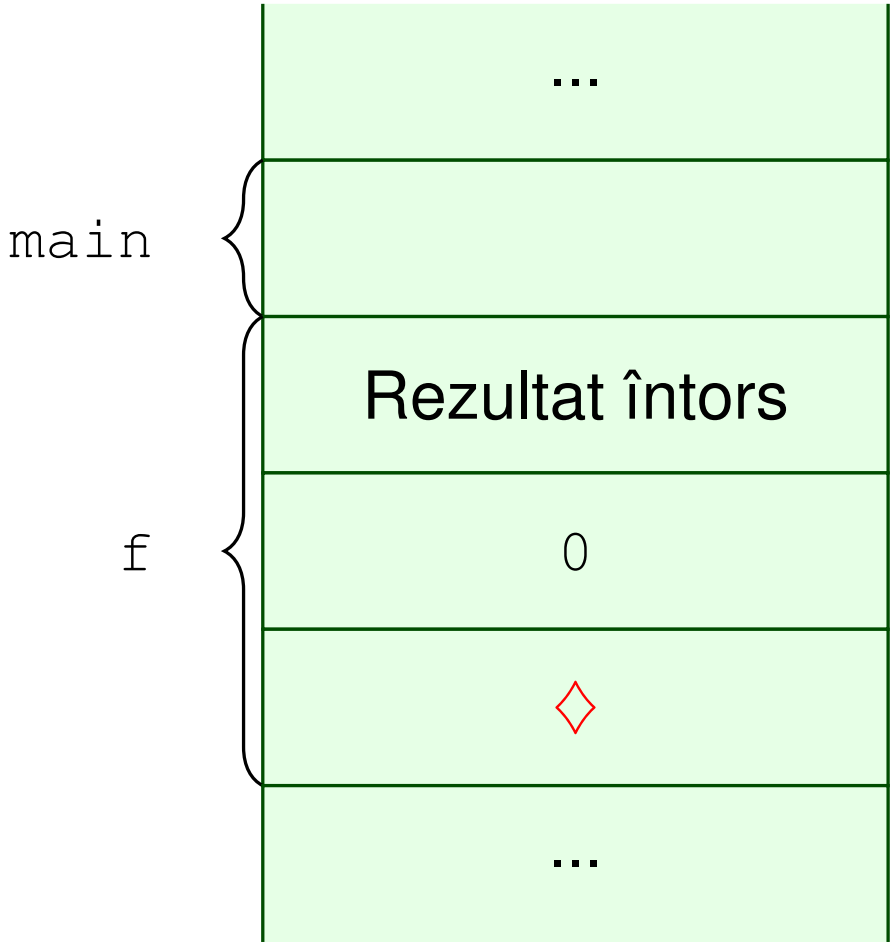
- Valoarea **întoarsă** de `f`
- **Parametrii** formali ai lui `f`
- Variabilele **locale** ale lui `f`
- Adresa de **revenire** în cadrul lui `g`
- Eventuale **registre** salvate, care vor fi modificați de `f` în timpul execuției sale

Exemplu reluat

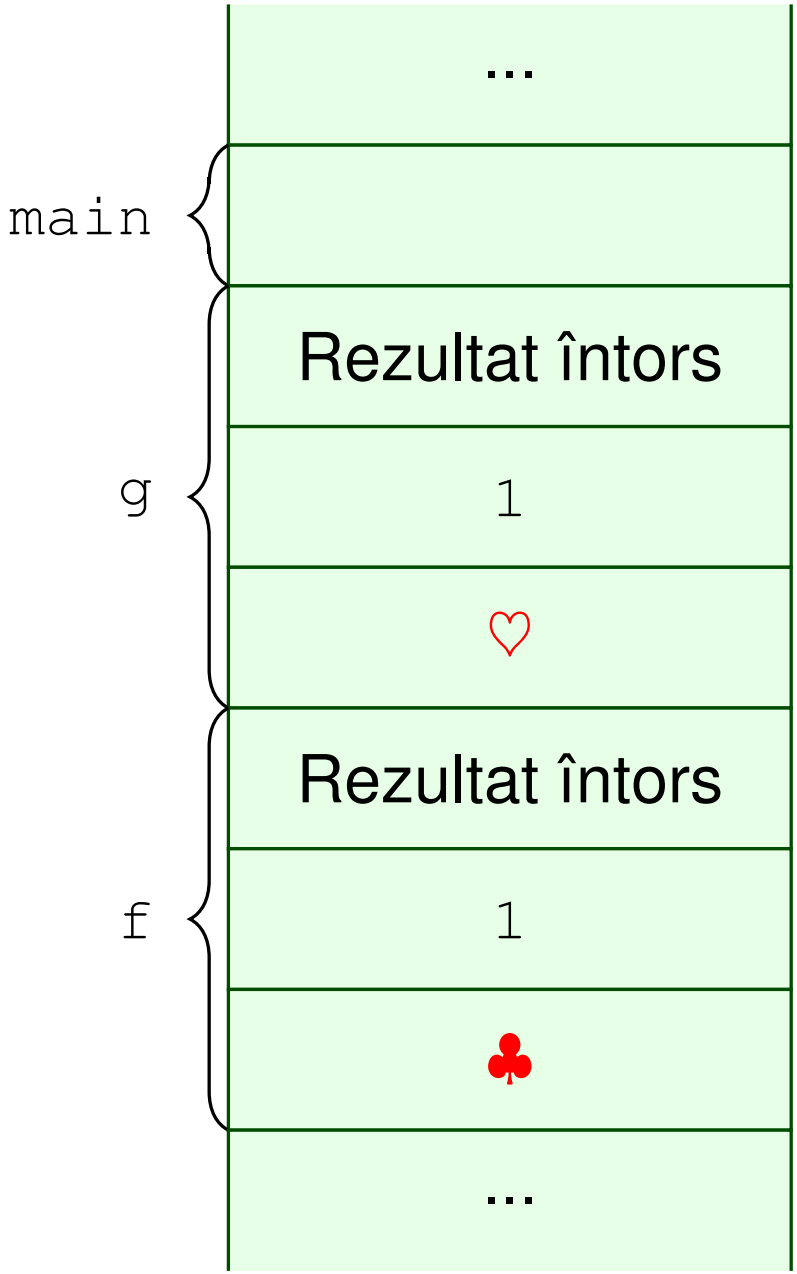
```
1  class Main {
2        f(x : Int) : Int { x };
3        g(x : Int) : Int { f(x) ♣ };
4        main() : Int {{ f(0); ♢ g(1); ♡ }};
5  };
```



Exemplu reluat (cont.)



Exemplu reluat (cont.)



Observații

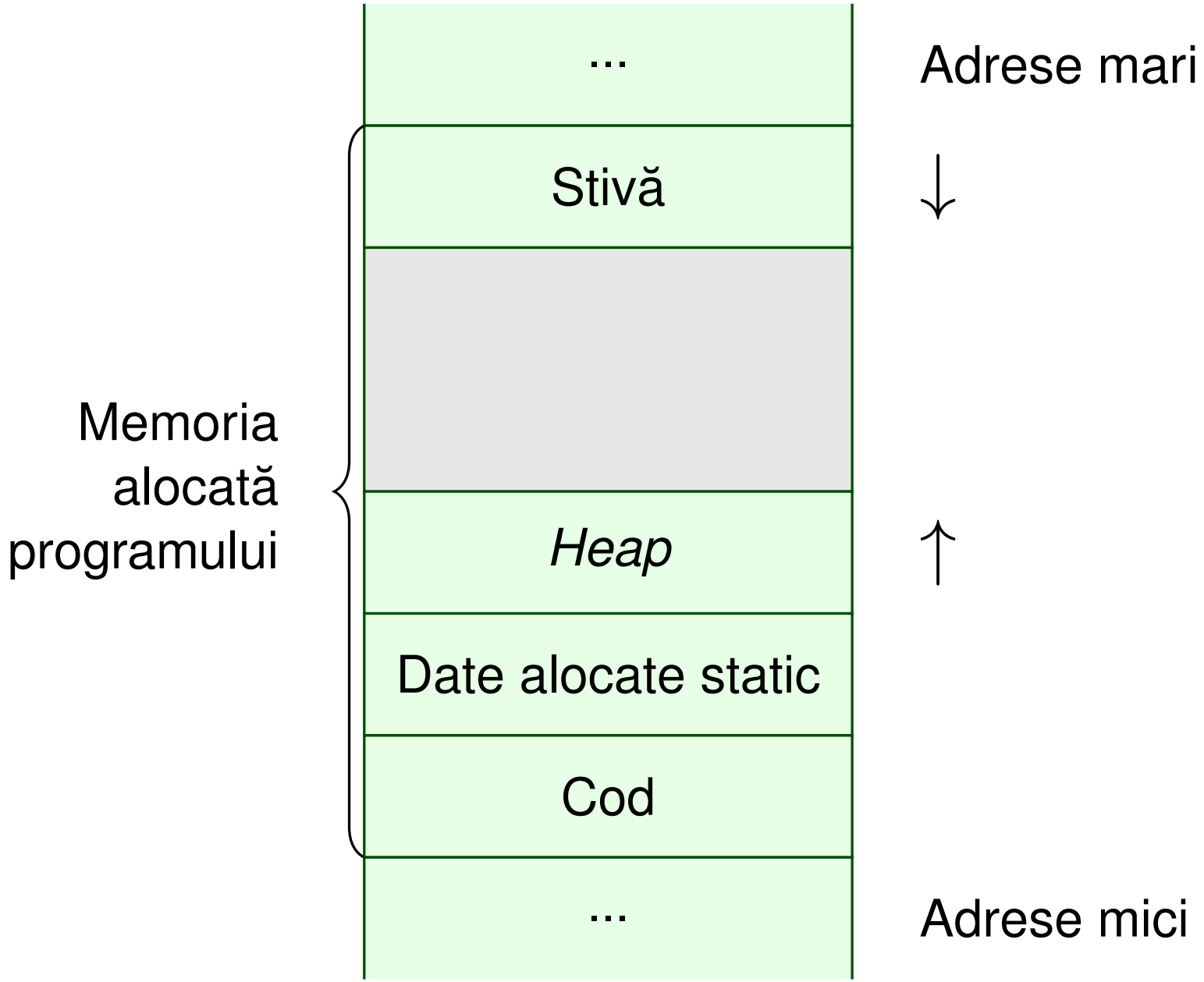
- Situarea rezultatului întors pe prima poziție din AR-ul apelatului $\Rightarrow$ accesarea lui la o distanță fixă față de AR-ul apelantului
- Posibilitatea varierii structurii AR în funcție de nevoi
- Pe cât posibil, păstrarea cât mai multor informații direct în registre, în special a rezultatului întors și a parametrilor, în vederea accesului rapid

Zonele globală și de heap

- Zona **globală**: entități accesibile permanent, în întregul program, alocate static

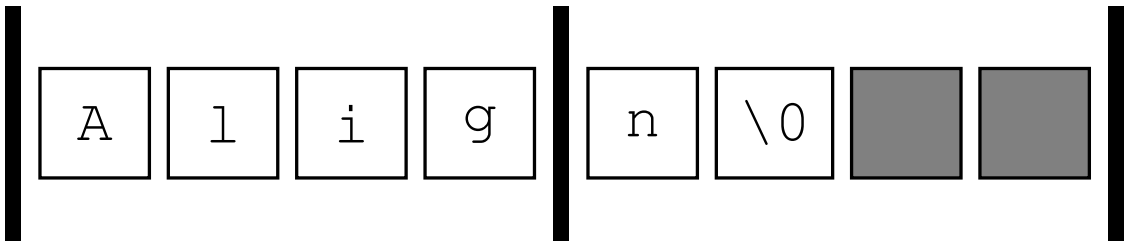
- Zona de ***heap***: entități alocate dinamic, în timpul execuției

Organizarea memoriei, varianta finală



Aliniere

- Memoria, **adresabilă** la nivel octet și/sau de cuvânt (4 sau 8 octeți)
- **Aliniere** = situarea datelor la un început de cuvânt (adresă multiplu de 4 sau 8 octeți)
- În caz de nealiniere, eroare sau afectare serioasă a vitezei
- Exemplu de aliniere pe 32 de biți, cu *padding*, pentru un șir de caractere:



Cuprins

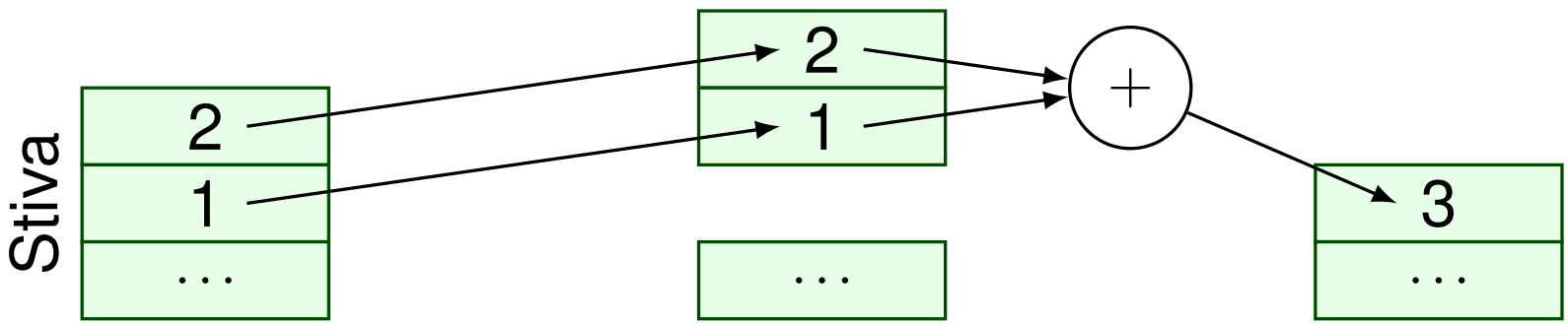
Mașini cu stivă

- Model simplu de evaluare
- Absența variabilelor și a registrelor
- Existența exclusiv a unei **stive** pentru stocarea rezultatelor intermediare
- Pentru **execuția** unei instrucțiuni: înlăturarea operanzilor din vârful stivei și adăugarea rezultatului în loc

Exemplu

Calculul $1 + 2$

```
1  push 1
2  push 2
3  add
```



Mașini cu stivă vs. mașini cu registre

Mașini cu stivă:

- Absența necesității de a preciza locația parametrilor sau a rezultatului
- Programe mai compacte
- Codificare mai compactă a instrucțiunilor

Mașini cu stivă și acumulator

- Soluția de mijloc: stivă și un singur registru (**acumulator**)
- Vârful stivei, accesat frecvent, reținut în acumulator
- Înainte, pentru `add`, două citiri și o scriere în stivă
- Acum, pentru `add`, o singură operație cu memoria!

$$acc \leftarrow acc + varf(stiva)$$

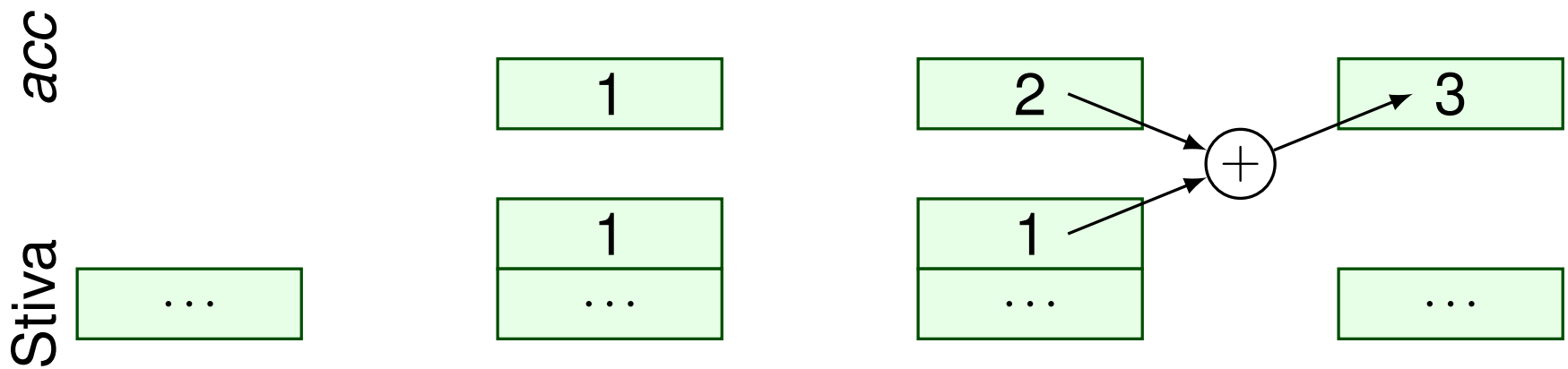
Principiile mașinii cu stivă și acumulator

- Rezultatul evaluării un expresii, depus în acumulator
- Pentru evaluarea $op(e_1, \ldots, e_n)$:
  - Adăugarea acumulatorului în stivă după evaluarea $e_1, \ldots, e_{n-1}$
  - La final, extragerea din stivă a celor $n-1$ operanzi și aplicarea operației op pe aceștia și pe acumulator
- La finalul evaluării unei expresii, revenirea la aceeași stare a stivei ca la început!

Exemplu reluat

Calcul $1 + 2$

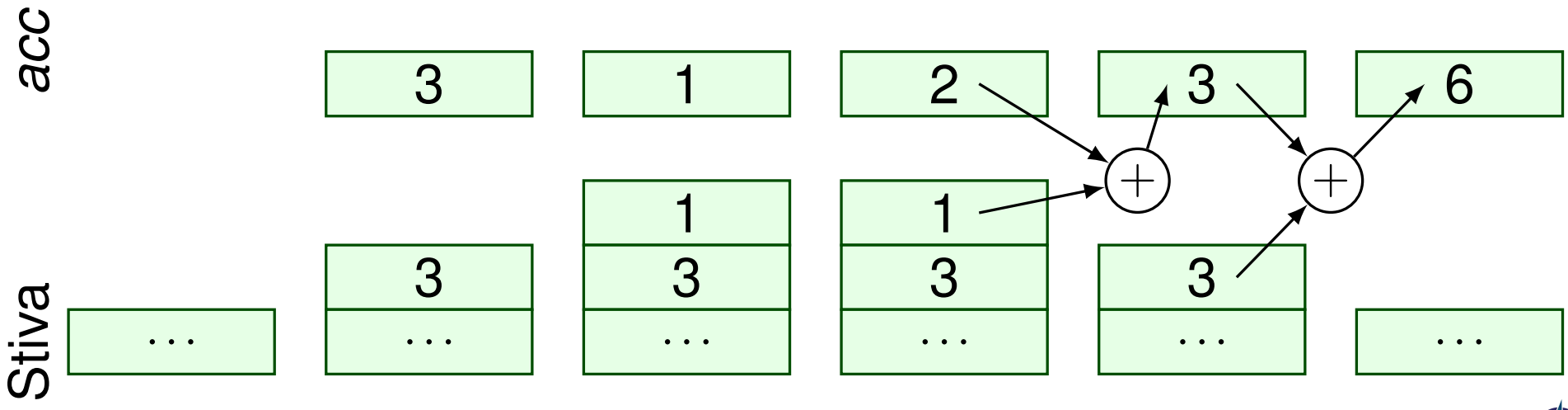
```
1  acc <- 1
2  push acc
3
4  acc <- 2
5
6  acc <- acc + varf(stiva)
7  pop
```



Exemplu mai complex

Calcul
$3+(1+2)$

```
1  acc <- 3
2  push acc
3
4  acc <- 1
5  push acc
6
7  acc <- 2
9  acc <- acc + varf(stiva)
10 pop
11
12 acc <- acc + varf(stiva)
13 pop
```



Cuprins

Obiective

- Generarea de cod pentru o **mașină cu stivă și acumulator**

- Utilizarea arhitecturii **MIPS** pentru simularea acestei mașini

Arhitectura MIPS

- RISC (*Reduced Instruction Set Computer*)
- Utilizarea **registrelor** pentru parametrii și rezultatele majorității operațiilor
- Operații *load* și *store* pentru citirea și scrierea în **memorie**
- 32 de registre generale pe 32 de biți

Simularea mașinii cu stivă și acumulator pe MIPS

- **Acumulatorul**, stocat în registrul `$a0`
- **Stiva**, reținută în memorie, crescând spre adrese mici
- Adresa următoarei poziții libere din stivă, stocată în registrul `$sp` (*stack pointer*), vârful aflându-se la `$sp + 4`

Câteva instrucțiuni MIPS

- `lw reg1 offset(reg2)`
  Load word: $reg_1 \leftarrow memory(reg_2 + offset)$
- `sw reg1 offset(reg2)`
  Store word: $memory(reg_2 + offset) \leftarrow reg_1$
- `li reg imm`
  Load immediate: $reg \leftarrow imm$
- `add reg1 reg2 reg3`
  $reg_1 \leftarrow reg_2 + reg_3$
- `addiu reg1 reg2 imm`
  Add immediate, unchecked overflow:
  $reg_1 \leftarrow reg_2 + imm$

Exemplu reluat

Calcul $1 + 2$

Mașina cu stivă și acumulator:

```
1  acc <- 1
2  push acc
3
4  acc <- 2
5  acc <- acc + varf(stiva)
6
7  pop
```

MIPS:

```
1  li $a0 1
2  sw $a0 0($sp)
3  addiu $sp $sp -4
4  li $a0 2
5  lw $t1 4($sp)
6  add $a0 $a0 $t1
7  addiu $sp $sp 4
```

Principiile generării de cod

- Funcția **`cgen(e)`**: codul generat pentru expresia `e`
- Depunerea **valorii** lui `e` în registrul `$a0`
- Asigurarea proprietății de **conservare** a stivei, i.e. revenirea la starea anterioară a stivei după evaluare

Generarea de cod pentru constante

```
1  cgen(i) =
2        li $a0 i
```

Depunerea constantei în acumulator.

Generarea de cod pentru adunare

```
1  cgen(e1 + e2) =
2       cgen(e1)
3       sw $a0 0($sp)
4       addiu $sp $sp -4
5       cgen(e2)
6       lw $t1 4($sp)
7       add $a0 $t1 $a0
8       addiu $sp $sp 4
```

Generarea de cod pentru operanzi, într-o manieră recursivă, exploatând proprietatea de **conservare** a stivei.

Generarea de cod pentru decizii I

- Utilizarea instrucțiunilor de **control** al fluxului de execuție

- `beq reg1 reg2 label`
  Salt la etichetă în caz de **egalitate** a celor două registre

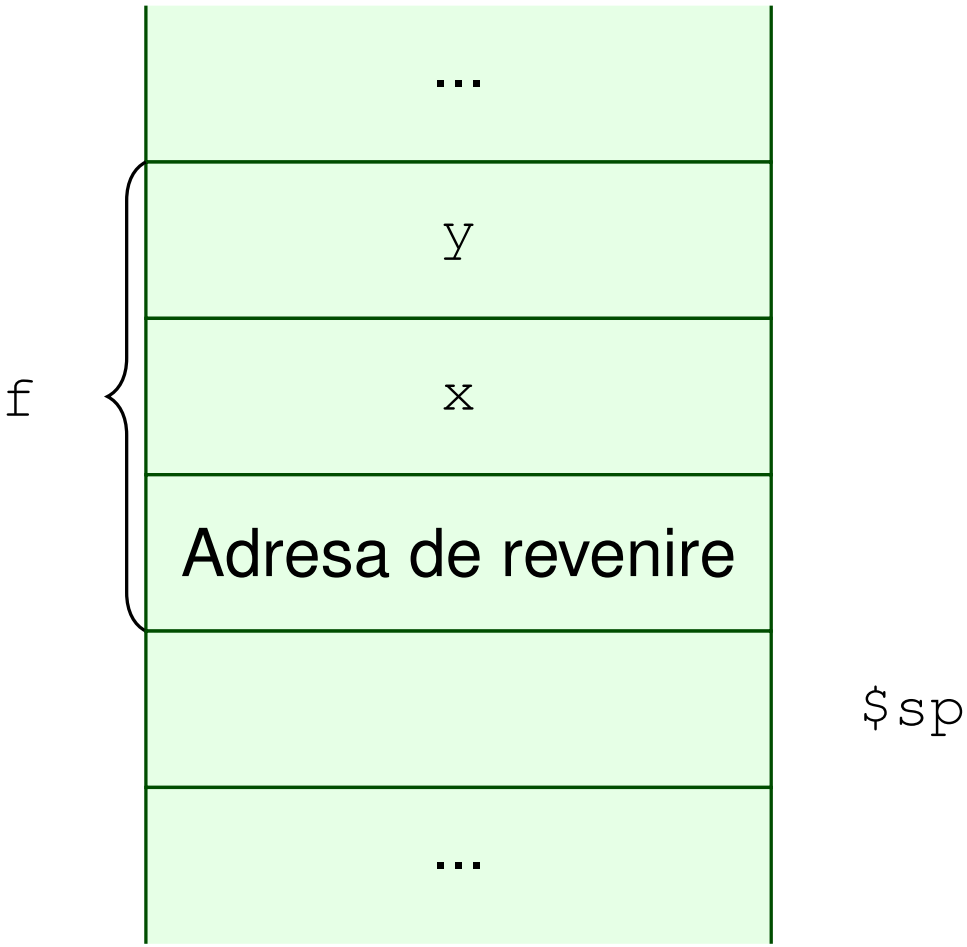
- `b label`
  Salt **necondiționat**

Generarea de cod pentru decizii II

```
1  cgen(if e1 = e2 then e3 else e4) =
2       cgen(e1)
3       sw $a0 0($sp)
4       addiu $sp $sp -4
5       cgen(e2)
6       lw $t1 4($sp)
7       addiu $sp $sp 4
8       beq $t1 $a0 true_branch
9     false_branch:
10      cgen(e4)
11      b end_if
12    true_branch:
13      cgen(e3)
14    end_if:
```

Înregistrarea de activare I

Funcția: `f(x, y)`

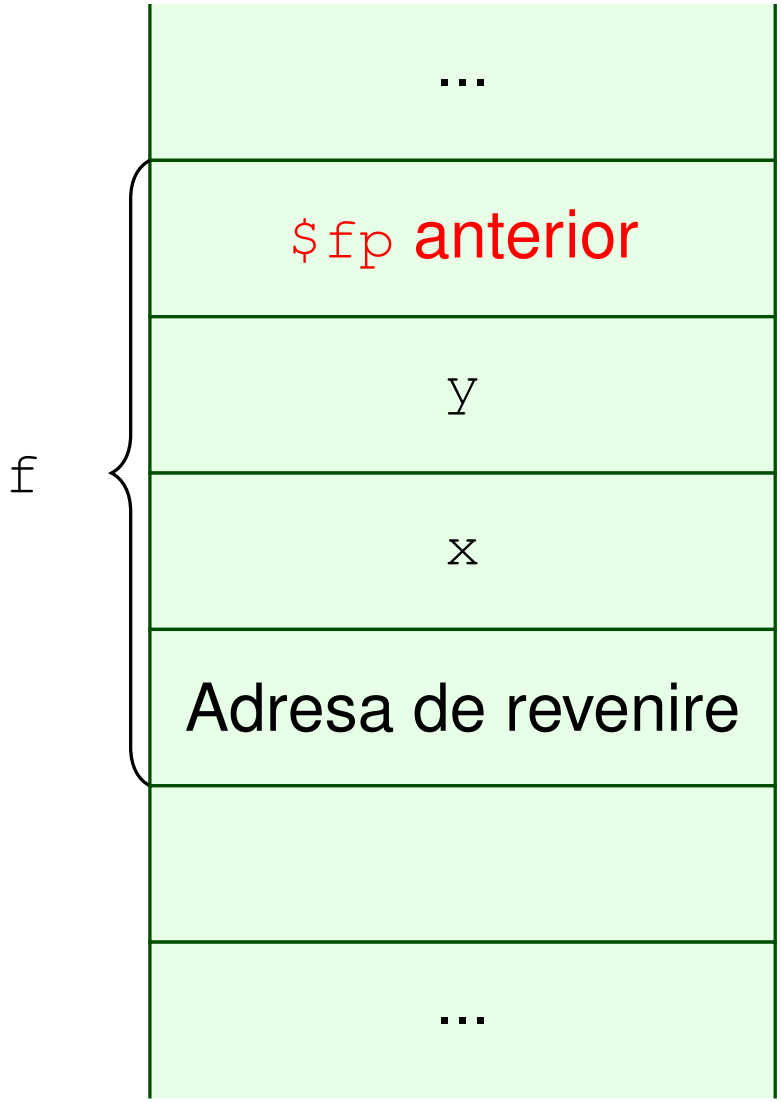


Rezultat, în `$a0`

Înregistrarea de activare II

- Problemă: posibilitatea `$sp` de a se modifica pe parcursul execuției corpului lui `f` $\Rightarrow$ variabilitatea adreselor parametrilor relativ la `$sp`
- Soluție: utilizarea unui reper fix, stocat în ***frame pointer*** (`$fp`)
- Necesitatea salvării pe stivă a valorii anterioare a `$fp`

Înregistrarea de activare III

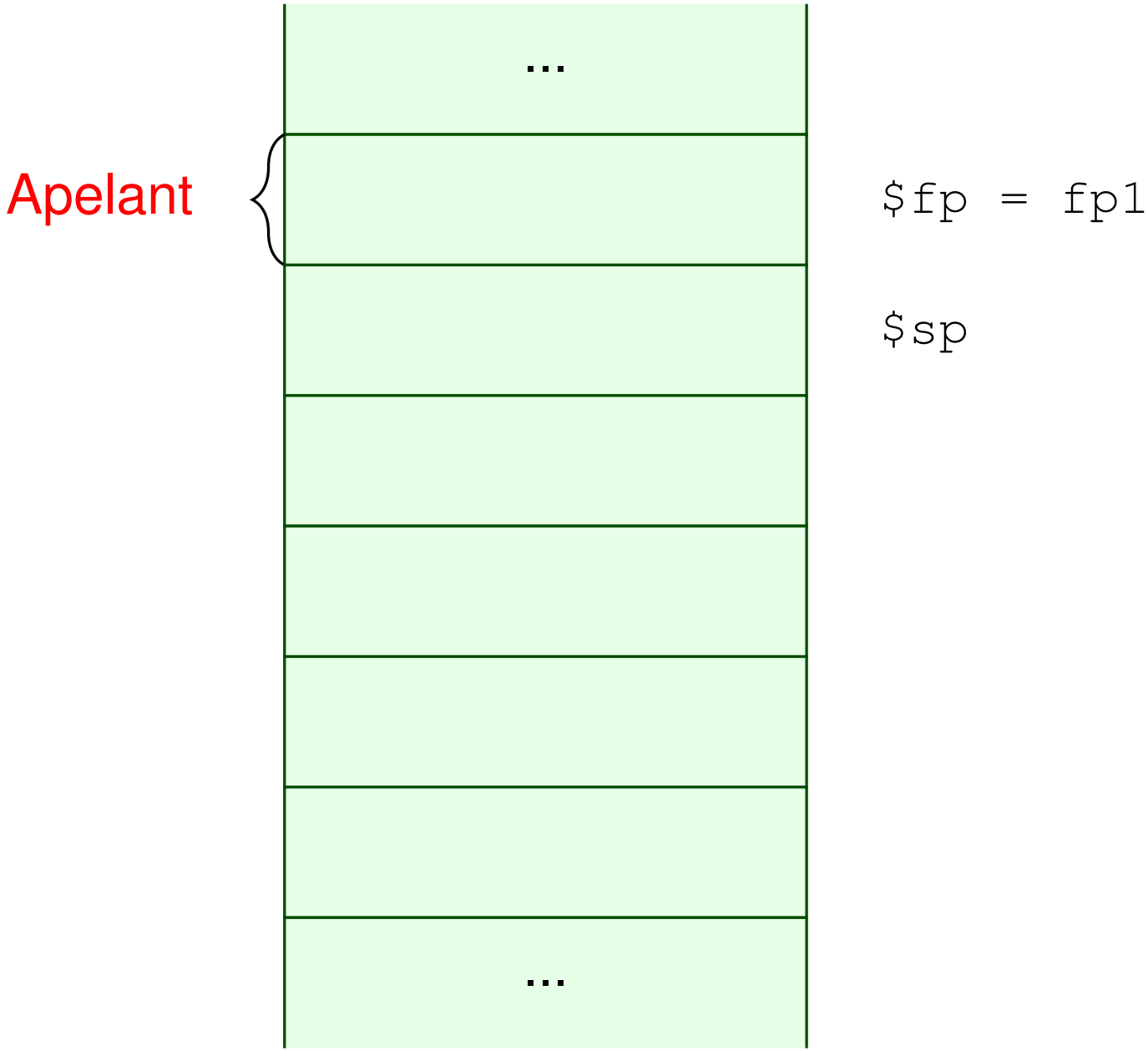


Rezultat, în `$a0`

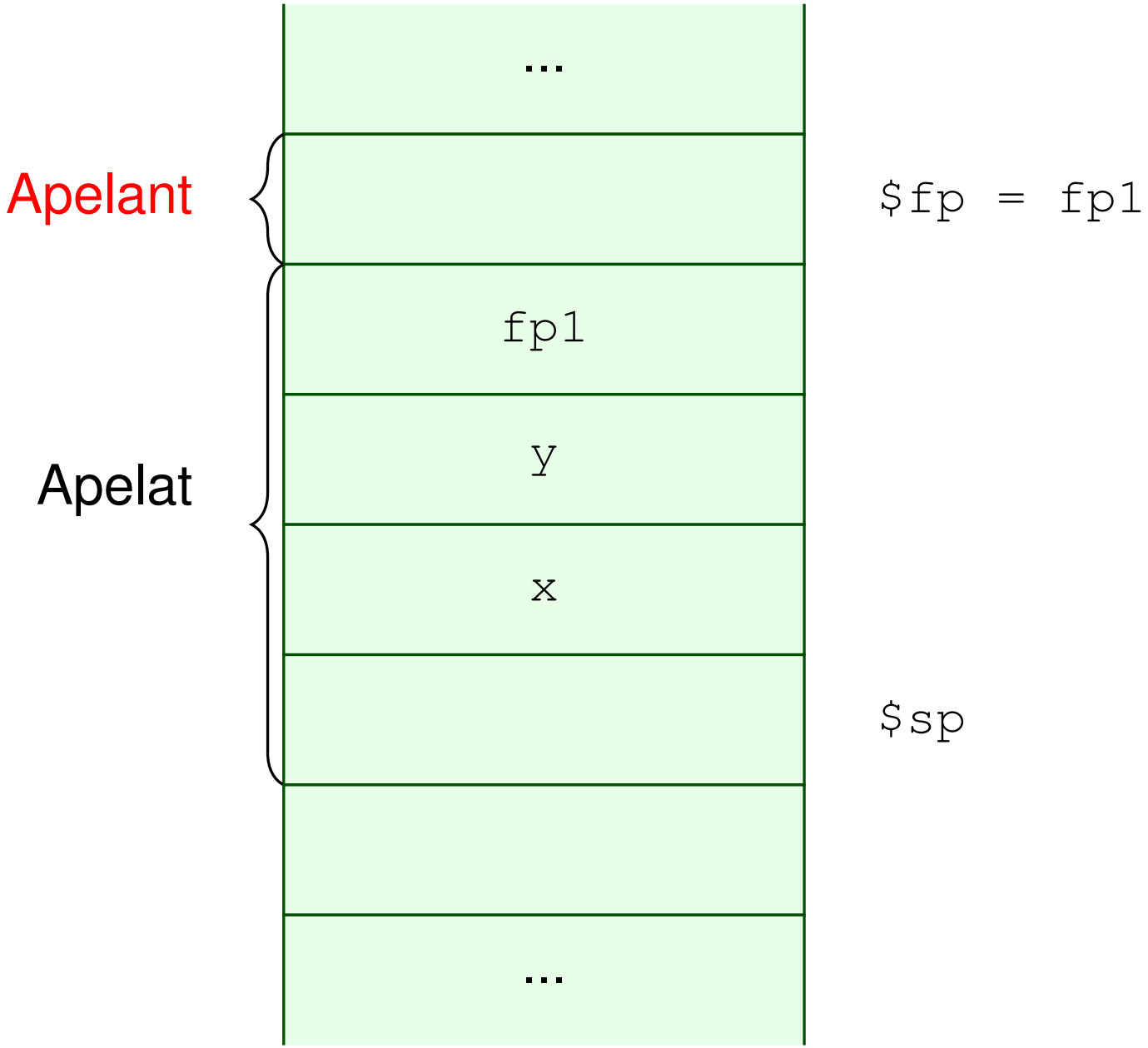
Secvența de apel I

- **Secvența de apel** = totalitatea instrucțiunilor executate atât de apelant, cât și de apelat, pentru invocarea unei funcții
- `jal label`
  Jump and link: Salt la etichetă, cu salvarea **adresei de revenire** în registrul `$ra` (*return address*)
- **Apelant**: salvarea `$fp` și a parametrilor în ordine inversă, apoi salt la apelat
- **Apelat**: salvarea `$ra` (doar el o știe, urmare lui `jal`!), modificarea `$fp`, și, în final, eliminarea AR de pe stivă, cu restaurarea `$fp` anterior și salt la adresa de revenire

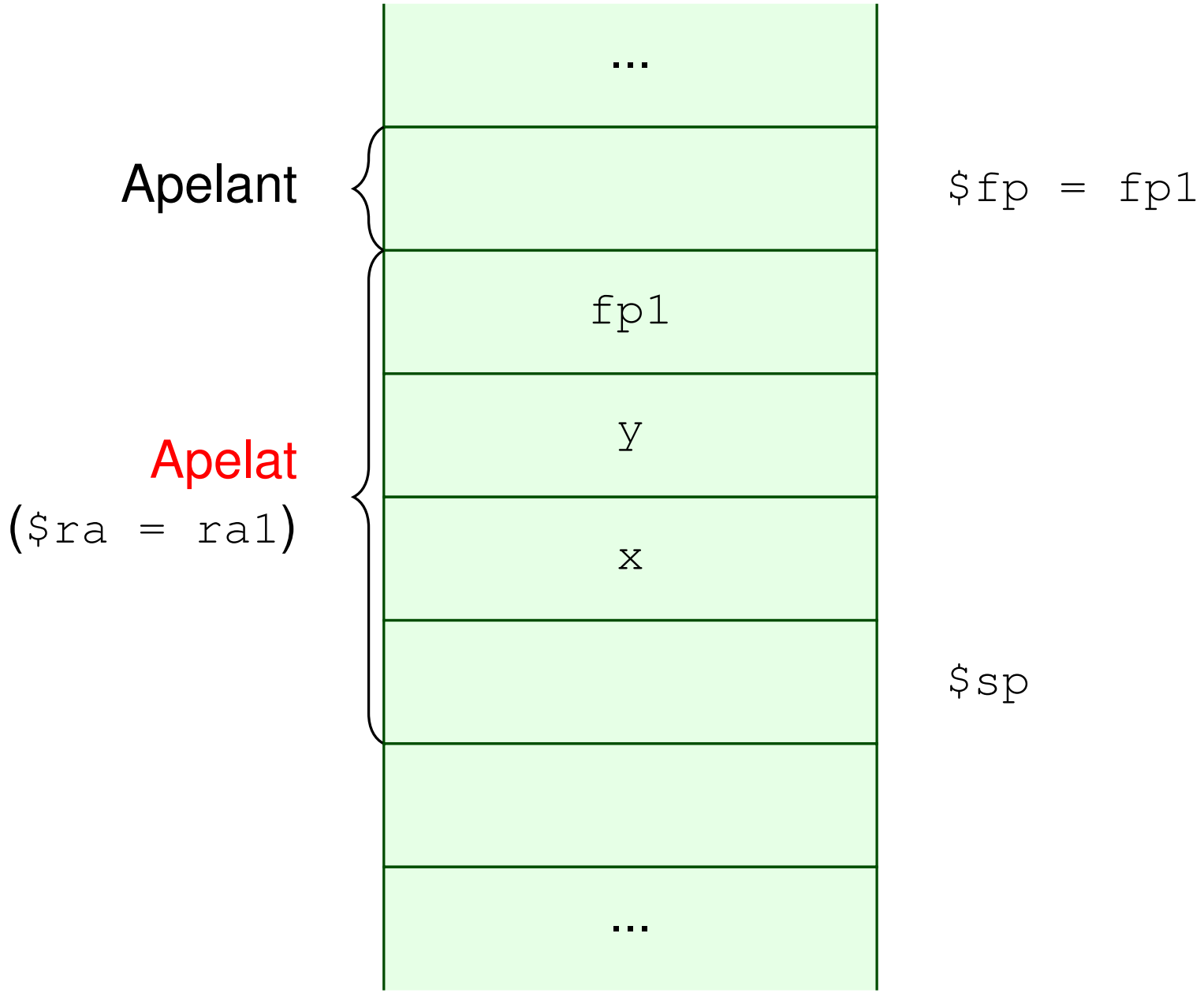
Secvența de apel II



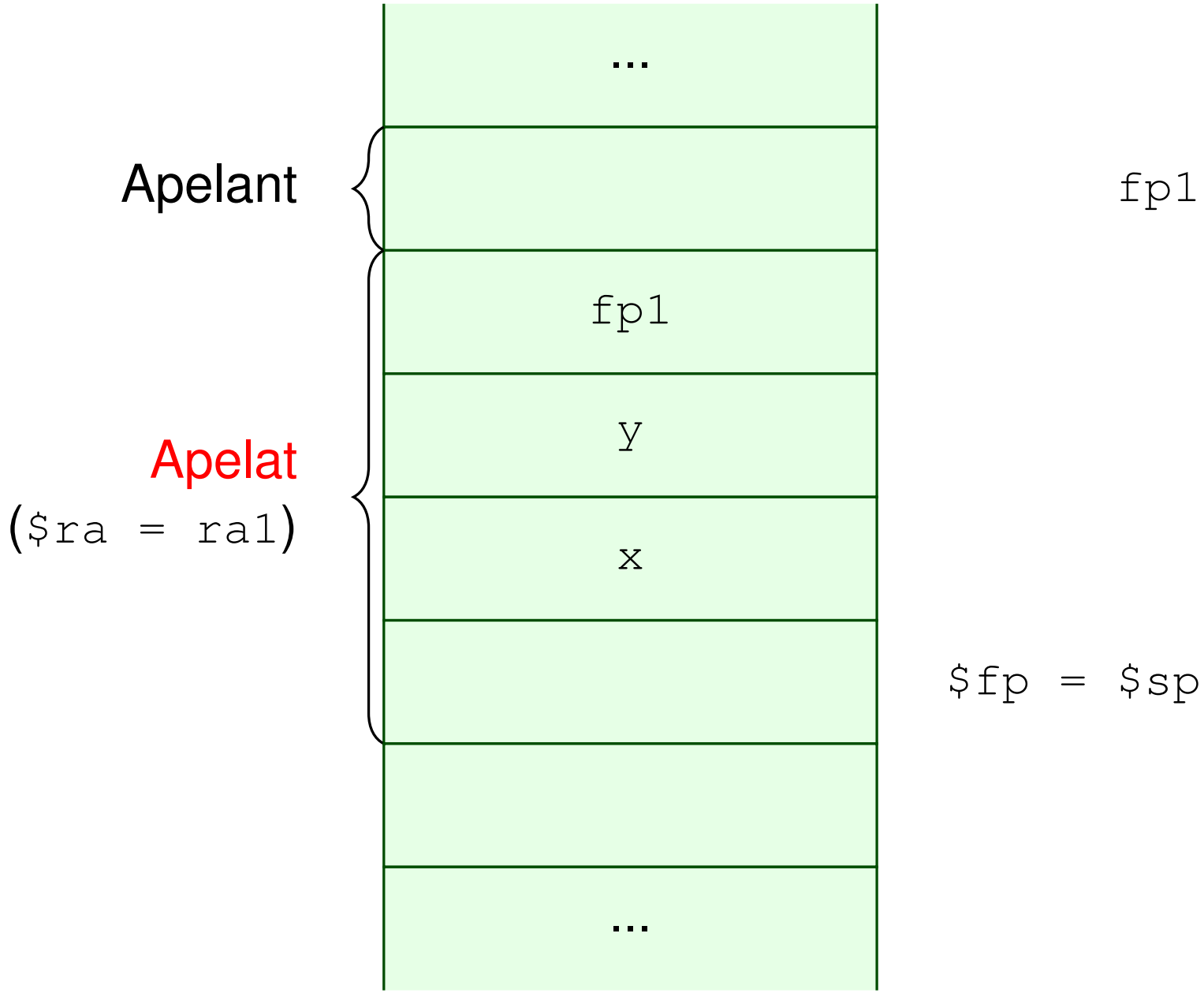
Secvența de apel II



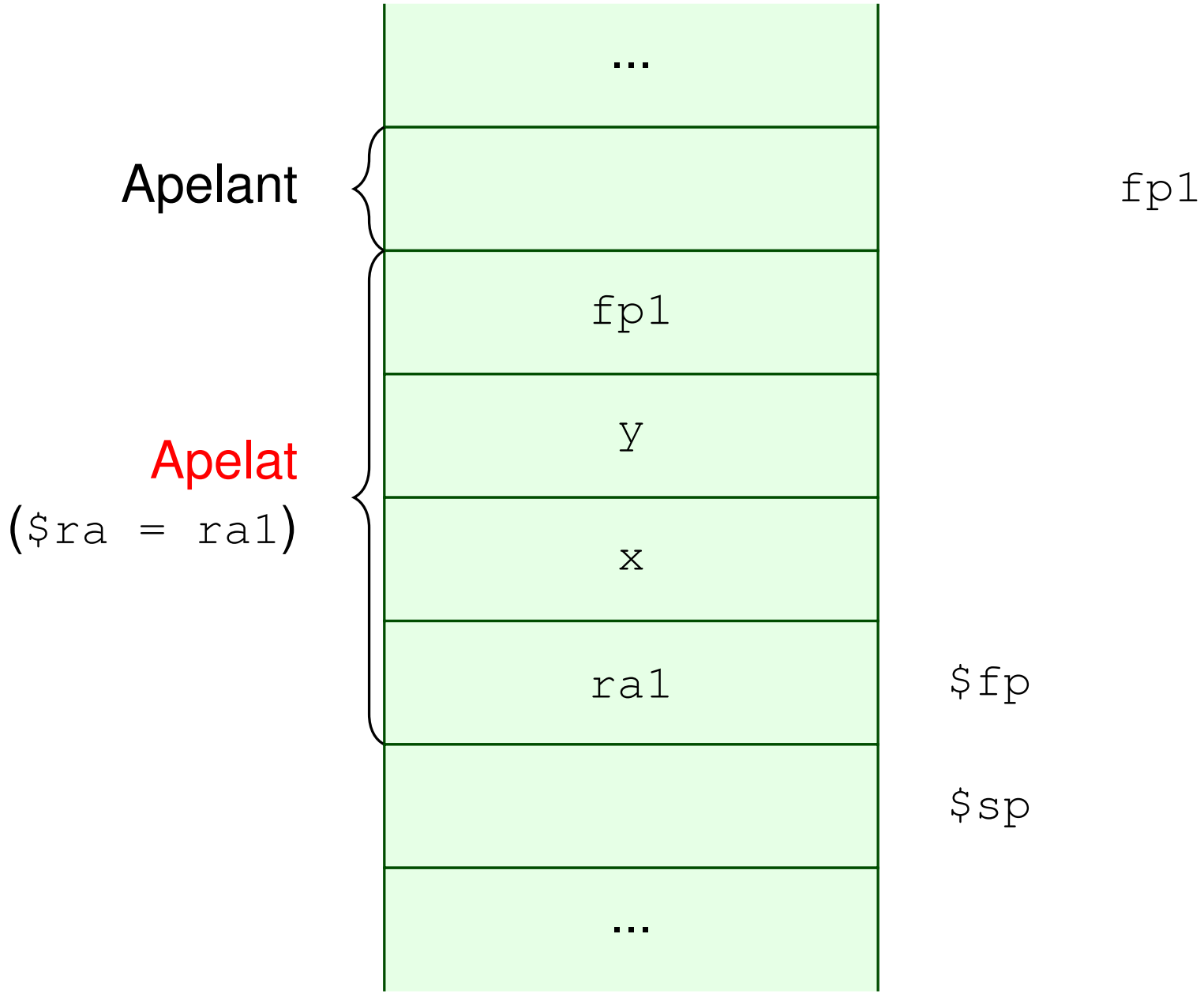
Secvența de apel II



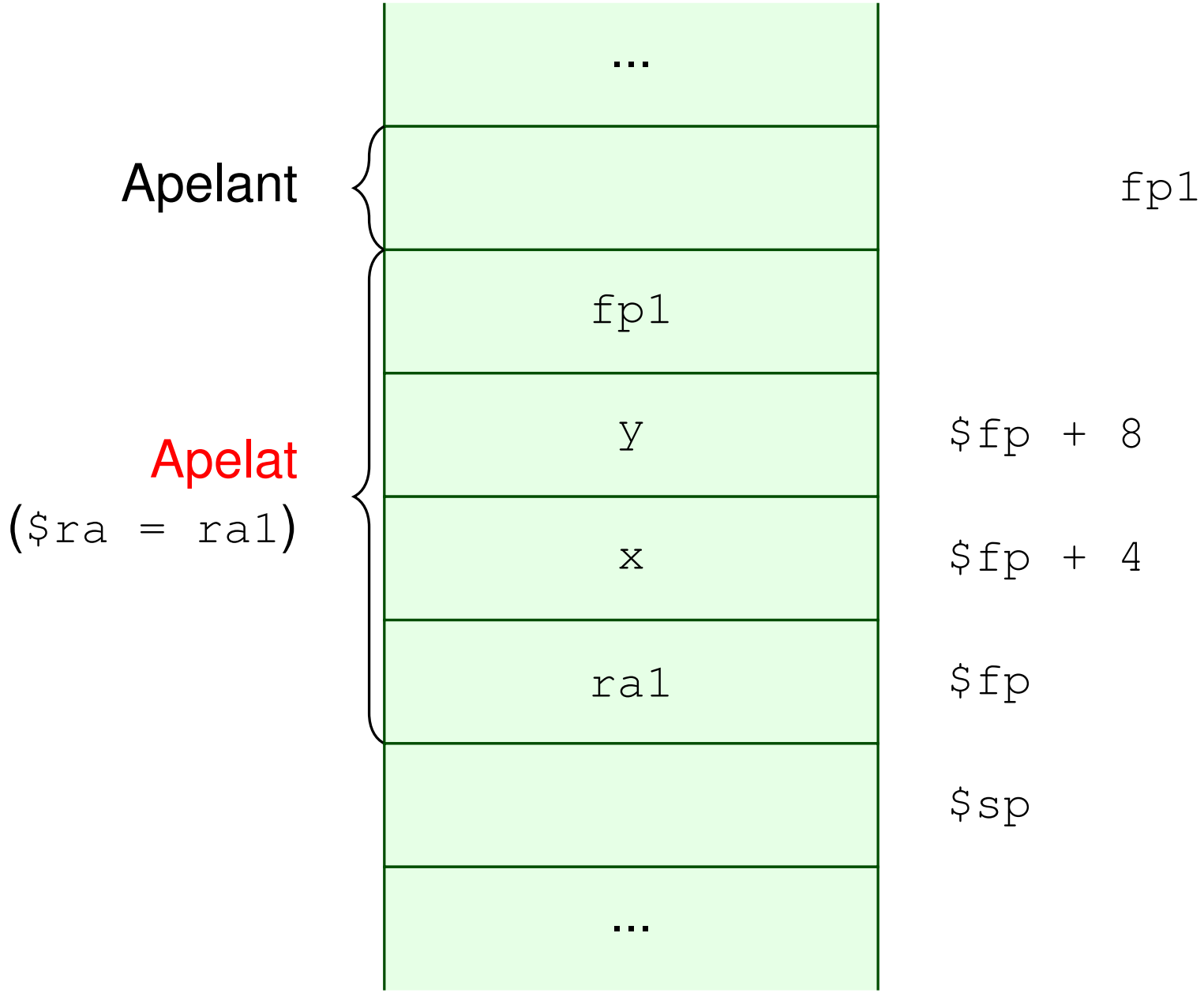
Secvența de apel II



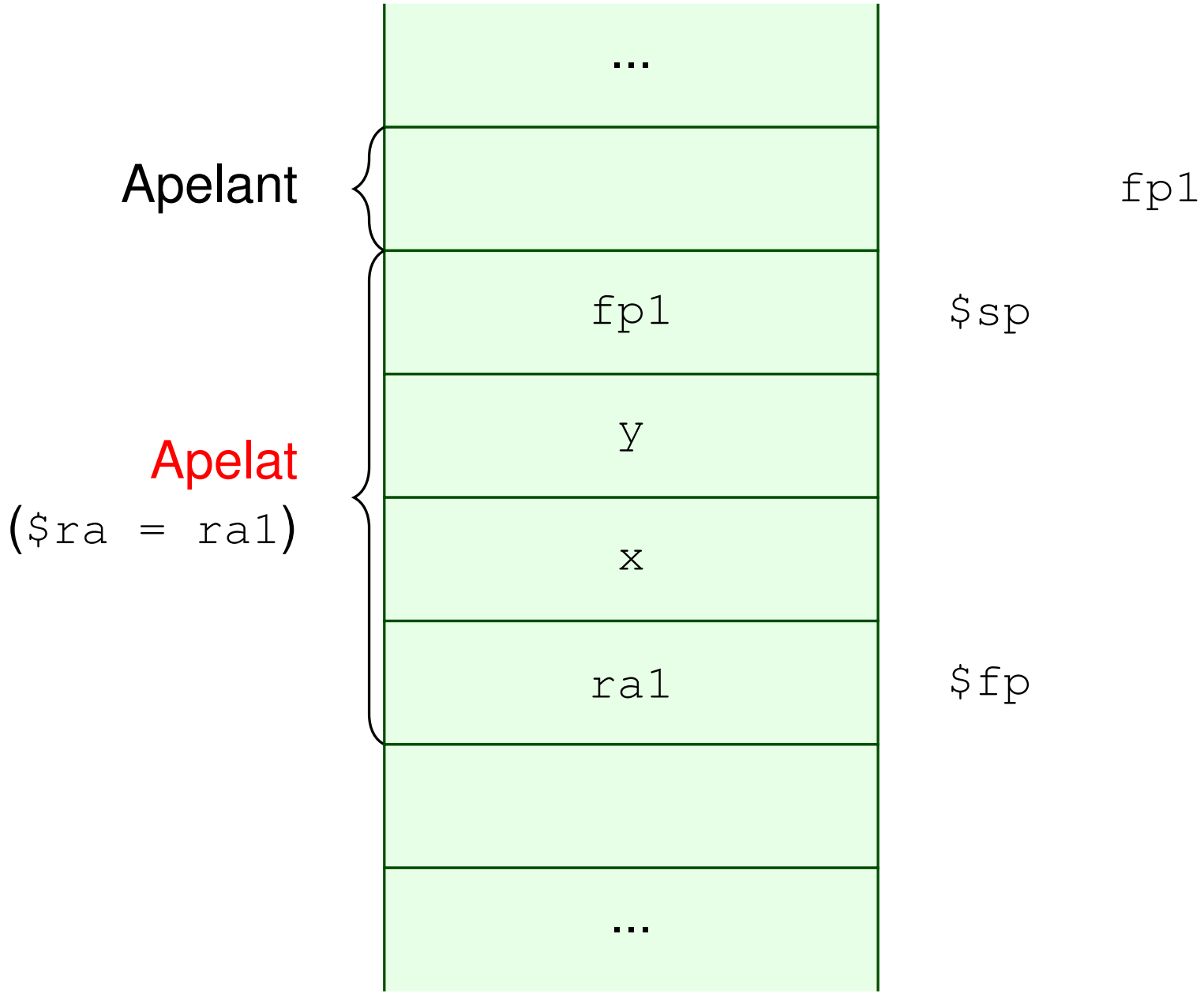
Secvența de apel II



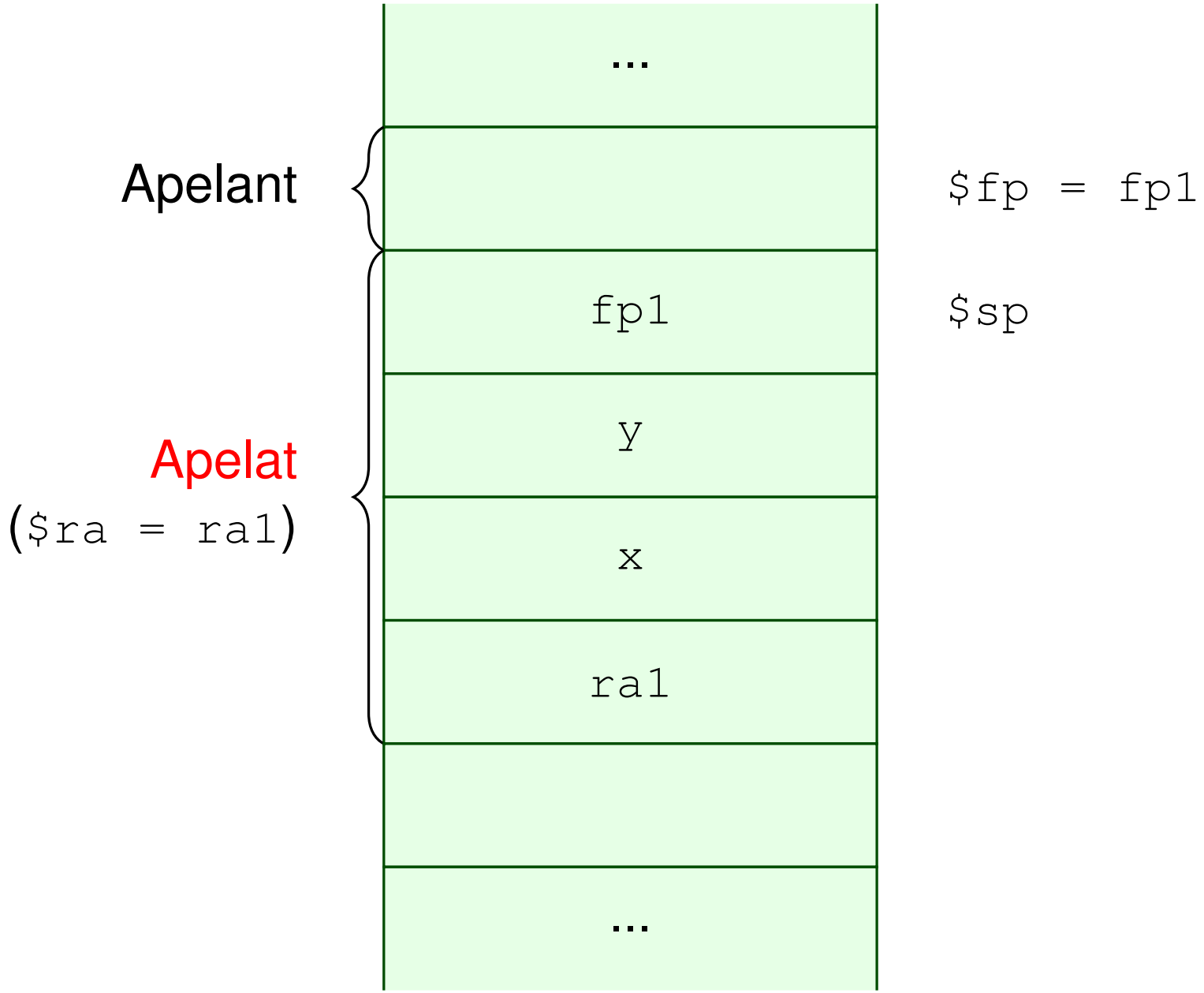
Secvența de apel II



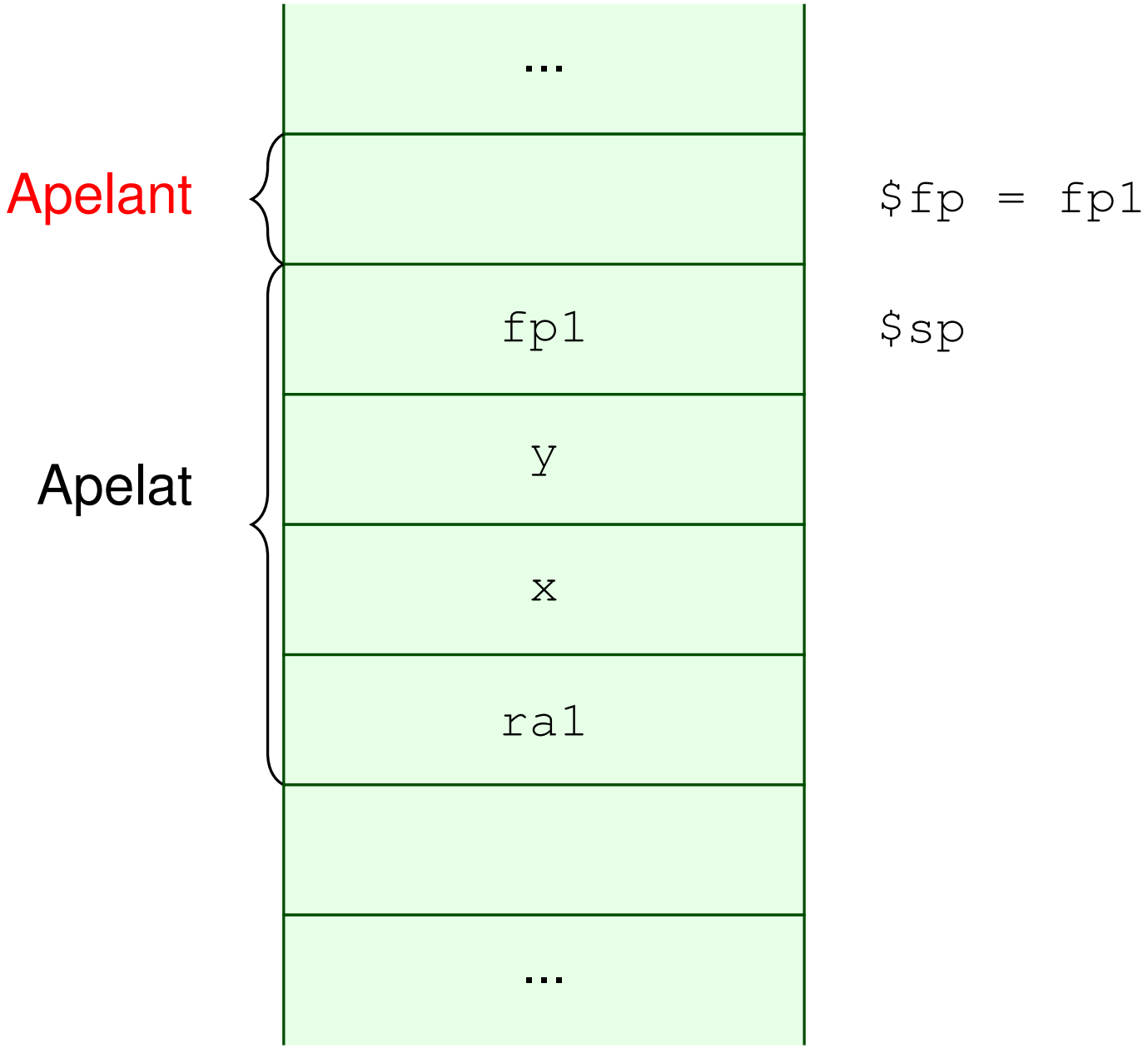
Secvența de apel II



Secvența de apel II



Secvența de apel II



Generarea de cod pentru apelurile de funcție

```
1  cgen(f(e1, ..., en)) =
2      sw $fp 0($sp)
3      addiu $sp $sp -4
4      cgen(en)
5      sw $a0 0($sp)
6      addiu $sp $sp -4
7      ...
8      cgen(e1)
9      sw $a0 0($sp)
10     addiu $sp $sp -4
11     jal f
```

Depunerea parametrilor în ordine inversă.

Generarea de cod pentru definițiile de funcție

```
1  cgen(f(x1, ..., xn) { e }) =
2       move $fp $sp
3       sw $ra 0($sp)
4       addiu $sp $sp -4
5       cgen(e)
6       lw $ra 4($sp)
7       addiu $sp $sp offset
8       lw $fp 0($sp)
9       jr $ra
```

offset $= 4n + 8$: n parametri, `$ra`, `$fp`

Generarea de cod pentru parametri

```
1  cgen(xi) =
2       lw $a0 offset($fp)
```

Pentru parametrul x_i, $offset = 4i$.

Îmbunătățiri

- Păstrarea anumitor componente ale AR direct în registre

- Stabilirea anticipată în cadrul AR a adreselor eventualelor locații temporare, pentru a evita deplasarea frecventă a *stack pointer*

Stabilirea anticipată a locațiilor temporare I

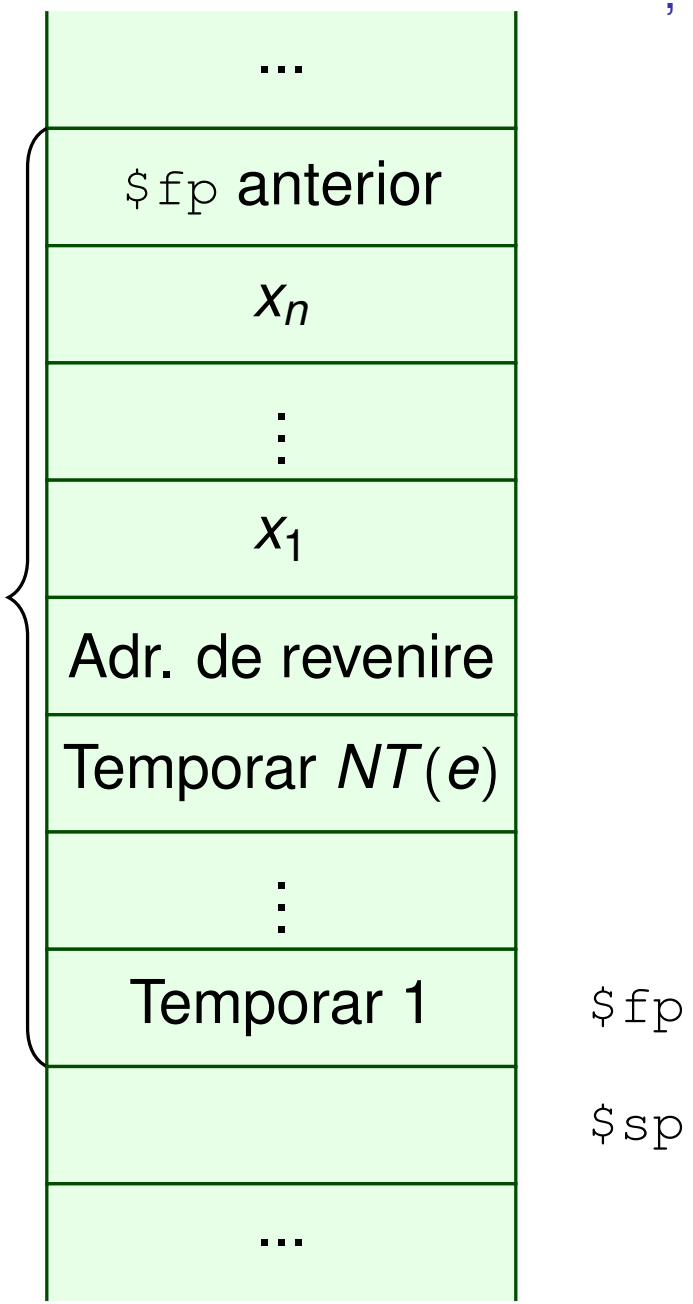
- $NT(e)$ = **numărul** de locații temporare necesare evaluării expresiei e
- $NT(e_1 + e_2)$: cel puțin $NT(e_1)$, și cel puțin $1 + NT(e_2)$, care include locația temporară pentru rezultatul lui e_1
- Posibilitatea reutilizării locațiilor temporare necesare evaluării lui e_1 în cursul evaluării lui e_2

Stabilirea anticipată a locațiilor temporare II

$$
\begin{aligned}
NT(1) &= 0 \\
NT(x) &= 0 \\
NT(e_1 + e_2) &= \max(NT(e_1), 1 + NT(e_2)) \\
NT(\mathit{if}\ e_1 = e_2\ \mathit{then}\ e_3\ \mathit{else}\ e_4) &= \max(NT(e_1), 1 + NT(e_2), \\
&\qquad NT(e_3), NT(e_4)) \\
NT(f(e_1, \ldots, e_n)) &= \max(NT(e_1), \ldots, NT(e_n))
\end{aligned}
$$

Rezultatele evaluării parametrilor funcțiilor apelate, depuse în AR-ul apelatului, nu al apelantului!

Înregistrarea de activare cu locații temporare



Generarea de cod cu locații temporare

```
1  cgen(e1 + e2) =
2       cgen(e1)
3       sw $a0 0($sp)
4       addiu $sp $sp -4
5       cgen(e2)
6       lw $t1 4($sp)
7       add $a0 $t1 $a0
8       addiu $sp $sp 4
```

```
1  cgen(e1 + e2, nt) =
2       cgen(e1, nt)
3       sw $a0 nt($fp)
4       cgen(e2, nt + 4)
5       lw $t1 nt($fp)
6       add $a0 $t1 $a0
```

În noua variantă, utilizarea `$fp` în locul `$sp`!

Cuprins

Principii

- Principiul **substituibilității**: dacă $B \leq A$, atunci un obiect de tipul B poate fi utilizat în locul unuia de tipul A

- Consecință: codul generat pentru clasa A, utilizabil ca atare asupra oricărui obiect de tipul B

Aspecte fundamentale

- Reprezentarea **obiectelor** în memorie
- Implementarea **apelurilor dinamice** de metodă (*dynamic dispatch*)

Constrângeri de reprezentare

```
1  class A {
2        a : Int;
3        f() : Int { 0 };
4  };
```

```
6  class B inherits A {
7        b : Int;
8        g() : Int { 0 };
9  };
10
11 class C inherits A {
12       c : Int;
13       f() : Int { 0 };
14       h() : Int { 0 };
15 };
```

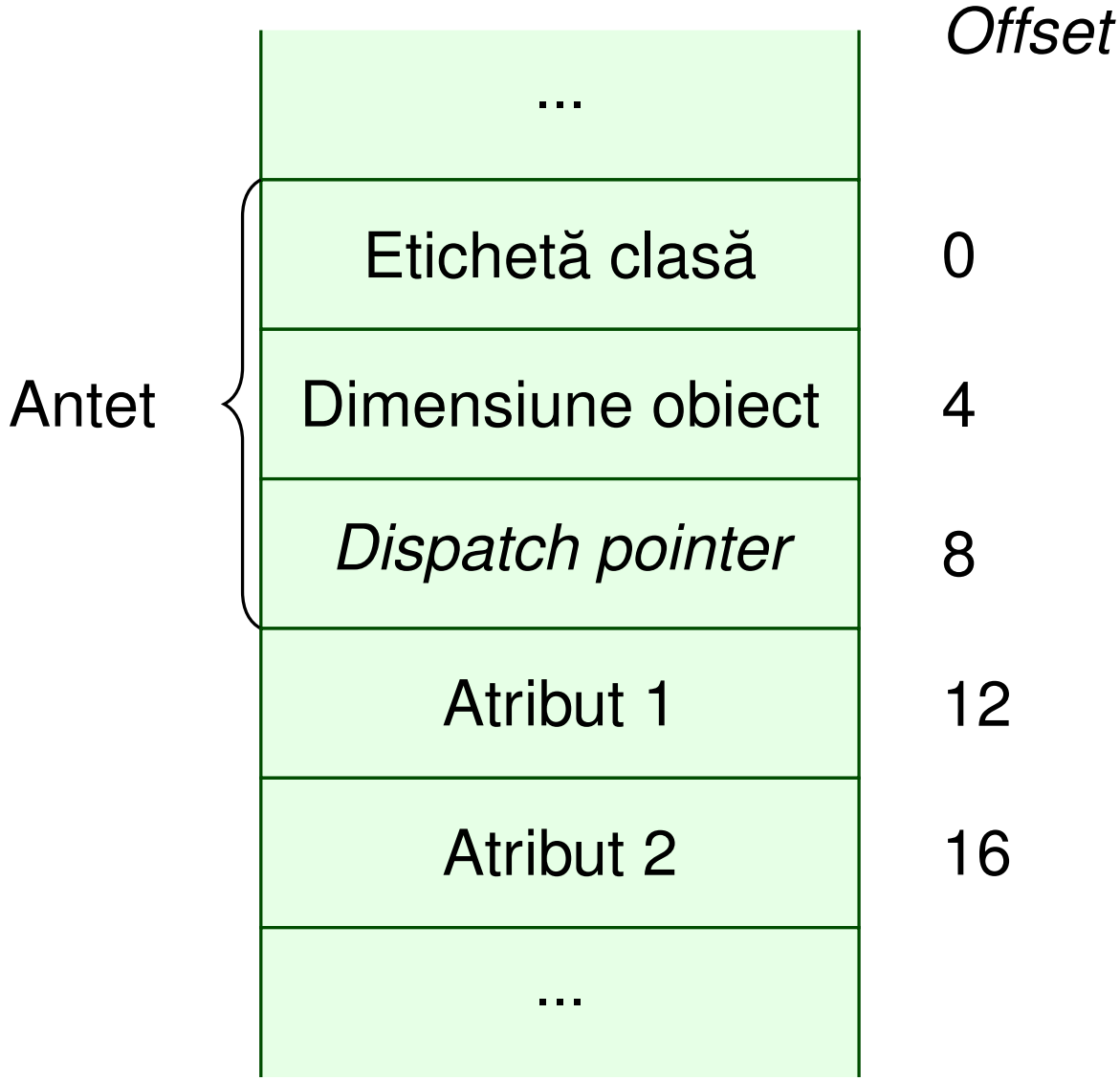
Constrângeri de reprezentare

```
1  class A {
2        a : Int;
3        f() : Int { 0 };
4  };
```

```
6  class B inherits A {
7        b : Int;
8        g() : Int { 0 };
9  };
10
11 class C inherits A {
12       c : Int;
13       f() : Int { 0 };
14       h() : Int { 0 };
15 };
```

Necesitatea utilizării ca atare a codului generat pentru superclasă asupra obiectelor subclaselor $\Rightarrow$ necesitatea situării **atributelor** moștenite în același loc în cadrul reprezentării obiectelor din ierarhie

Reprezentarea obiectelor în memorie I



Reprezentarea obiectelor în memorie II

- **Eticheta** clasei: număr întreg
- **Dimensiunea**, măsurată în cuvinte
- ***Dispatch pointer***: către tabelul de metode
- **Atributele**, adăugate în continuarea celor moștenite

Exemplu reluat

Clasa `A`:

...
1
4
Dispatch p. `A`
`a`
...

Clasa `B`:

...
2
5
Dispatch p. `B`
`a`
`b`
...

Clasa `C`:

...	
3	0
5	4
Dispatch p. `C`	8
`a`	12
`c`	16
...	

Implementarea apelurilor dinamice de metodă

- **Tabel de metode** (*dispatch table*): lista adreselor metodelor unei clase

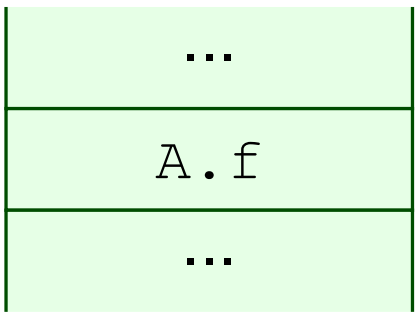
- Întrebare: De ce utilizăm un **pointer** către acest tabel, în loc să dispunem adresele metodelor direct în reprezentarea obiectului, ca în cazul atributelor?

Implementarea apelurilor dinamice de metodă

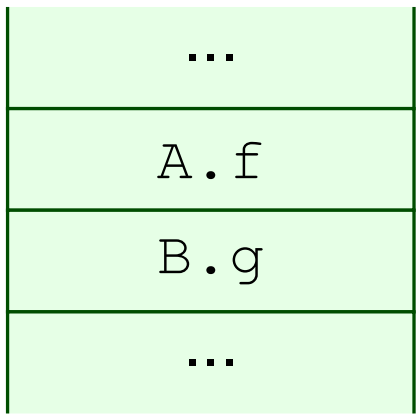
- **Tabel de metode** (*dispatch table*): lista adreselor metodelor unei clase
- Întrebare: De ce utilizăm un **pointer** către acest tabel, în loc să dispunem adresele metodelor direct în reprezentarea obiectului, ca în cazul atributelor?
- Răspuns: Deoarece atributele se pot modifica, în timp ce metodele nu, și pot fi partajate de toate instanțele clasei

Exemplu reluat

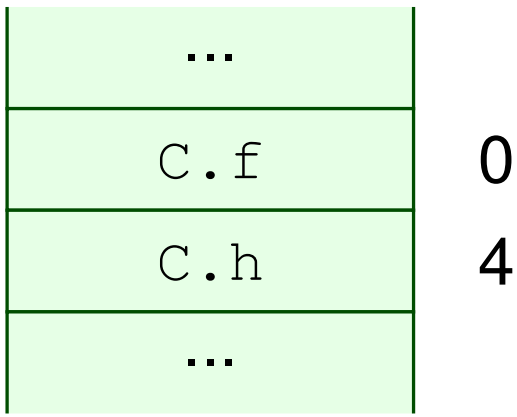
Clasa `A`:



Clasa `B`:



Clasa `C`:



Cuprins

Motivație

- Necesitatea descrierii modalității de evaluare a programelor, independent de mașina pentru care se generează cod
- Limbajul de asamblare, bazat pe prea multe detalii specifice arhitecturii alese: zonele de memorie, registrele, absența sau prezența stivei, sensul în care aceasta crește etc.
- Soluție: **notație formală**, pentru descrierea semnificației programelor

Semantica operațională

- „Cum decurge **evaluarea**?”
- Descrierea semnificației programului prin secvența de **operații** desfășurate în timpul evaluării pe o mașină abstractă, cu stare modificabilă
- Varianta cea mai adecvată pentru specificarea implementărilor
- Exemplu pentru **atribuire**:

$$\frac{S \vdash e \mapsto v, S'}{S \vdash x \leftarrow e \mapsto v, S'[v/x]}$$

„Dacă, în starea S, e se evaluează la valoarea v, mașina trecând în starea S', atunci, tot în starea S, atribuirea $x \leftarrow e$ se evaluează la v, mașina trecând în starea $S'[v/x]$, conținând noua valoare a lui x.”

Semantica denotațională

- „Ce **obiect matematic** denotă programul?”
- Descrierea semnificației programului printr-o **funcție matematică** ce transformă starea acestuia
- Exemplu pentru **atribuire**:

$$meaning(x \leftarrow e)(S) = (v, S'[v/x]), \text{ unde}$$
$$meaning(e)(S) = (v, S')$$

Pentru o expresie, calculul de către funcția *meaning* a unei **funcții** aferente expresiei, care transformă starea.

Semantica axiomatică

- „Ce **proprietăți** pot fi demonstrate despre program?”
- Descrierea semnificației programului prin **formule logice** care ar trebui satisfăcute înainte și după încheierea execuției
- Baza sistemelor de **verificare** formală a corectitudinii
- Exemplu pentru **atribuire**:

$$\{e = v\}\ x \leftarrow e\ \{x = v\}$$

„Dacă, înainte de atribuire, $e = v$, atunci, după evaluarea atribuirii $x \leftarrow e$, $x = v$.”

Notația pentru semantică operațională

- Analogie cu notația regulilor de **tipare**:

$$\vdash e : T$$

„Expresia e are tipul T.”

- Notația regulilor de **evaluare**, în semantică **operațională**:

$$\vdash e \mapsto v$$

„Expresia e se evaluează la valoarea v.”

Contexte de evaluare

- La fel ca în cazul regulilor de tipare, de unde obținem valorile **variabilelor**?
- În plus, unde scriem valorile variabilelor în urma atribuirilor (efecte laterale)?
- **Mediu de evaluare** (*environment*, E): dicționar de asocieri între variabile și locații de memorie:

$$E = [x_1 : l_1, \ldots, x_n : l_n]$$

- **Memorie** (*store*, S): dicționar de asocieri între locații de memorie și valori:

$$S = [l_1 \to v_1, \ldots, l_n \to v_n]$$

Exemplu

- Contextul de evaluare:

$$E = [x : l_x, y : l_y]$$
$$S = [l_x \to 1, l_y \to 2]$$

- Accesarea valorii lui x, prin dublă indirectare:

$$E(x) = l_x$$
$$S(l_x) = 1$$

Afirmațiile despre evaluare

$$E, S \vdash e \mapsto v, S'$$

„Dacă evaluarea expresiei e, în raport cu mediul de evaluare E și memoria S, **se termină**, atunci se obțin valoarea v, și memoria potențial modificată S'.”

Valorile din Cool

- Toate valorile = **instanțe** ale claselor!

- Notația pentru instanțe:

$$x = X(a_1 = l_1, \ldots, a_n = l_n), \text{ unde}$$
$$X = \textbf{clasa} \text{ lui } x$$
$$a_i = \textbf{atributele}, \text{ inclusiv cele moștenite!}$$
$$l_i = \textbf{locația} \text{ atributului } a_i$$

Valorile de bază

- Întregi: *Int*(1)
  (valoarea)
- Șiruri de caractere: *String*(6,"*string*")
  (lungimea și șirul)
- Booleeni: *Bool*(*false*)
  (valoarea)
- *void*

Evaluarea valorilor de bază

$$\frac{}{E, S \vdash true \mapsto Bool(true), S}$$

$$\frac{i \text{ este un literal întreg}}{E, S \vdash i \mapsto Int(i), S}$$

$$\frac{s \text{ este un literal șir de caractere de lungime } n}{E, S \vdash s \mapsto String(n, s), S}$$

Evaluarea variabilelor

$$\frac{E(x) = l_x \qquad S(l_x) = v_x}{E, S \vdash x \mapsto v_x, S}$$

Dublă indirectare.

Evaluarea lui *self*

$$\frac{}{E, S \vdash \mathit{self} \mapsto ?, S}$$

Evaluarea lui *self*

$$\frac{}{E, S \vdash \mathit{self} \mapsto ?, S}$$

Necesitatea adăugării **obiectului curent** (*self object*) la contextul de evaluare.

Evaluarea lui *self* (cont.)

$$\frac{}{so, E, S \vdash self \mapsto so, S}$$

Evaluarea atribuirilor

$$\frac{so, E, S \vdash e \mapsto v, S_1 \quad E(x) = l_x \quad S_2 = S_1[v/l_x]}{so, E, S \vdash x \leftarrow e \mapsto v, S_2}$$

Surprinderea **efectelor laterale**
prin modificarea memoriei.

Evaluarea deciziilor

$$\frac{so, E, S \vdash e_1 \mapsto Bool(true), S_1 \qquad so, E, S_1 \vdash e_2 \mapsto v, S_2}{so, E, S \vdash \mathit{if}\ e_1\ \mathit{then}\ e_2\ \mathit{else}\ e_3\ \mathit{fi} \mapsto v, S_2}$$

În cazul satisfacerii condiției,
evaluarea exclusivă a ramurii *then*.

Evaluarea blocurilor

$$\frac{\begin{array}{c} so, E, S \vdash e_1 \mapsto v_1, S_1 \\ so, E, S_1 \vdash e_2 \mapsto v_2, S_2 \\ \vdots \\ so, E, S_{n-1} \vdash e_n \mapsto v_n, S_n \end{array}}{so, E, S \vdash \{\ e_1;\ \ldots;\ e_n;\ \} \mapsto v_n, S_n}$$

Valoarea ultimei expresii din bloc.

Evaluarea buclelor

$$\frac{so, E, S \vdash e_1 \mapsto Bool(false), S_1}{so, E, S \vdash while\ e_1\ loop\ e_2\ pool \mapsto void, S_1}$$

$$\frac{\begin{array}{l} so, E, S \vdash e_1 \mapsto Bool(true), S_1 \\ so, E, S_1 \vdash e_2 \mapsto v, S_2 \\ so, E, S_2 \vdash while\ e_1\ loop\ e_2\ pool \mapsto void, S_3 \end{array}}{so, E, S \vdash while\ e_1\ loop\ e_2\ pool \mapsto void, S_3}$$

Aplicarea recursivă a regulii de evaluare.

Evaluarea expresiei *let*

$$\frac{so, E, S \vdash e_1 \mapsto v_1, S_1 \qquad so, ?, ? \vdash e_2 \mapsto v_2, S_2}{so, E, S \vdash let\ x : T \leftarrow e_1\ in\ e_2 \mapsto v_2, S_2}$$

Evaluarea expresiei *let*

$$\frac{so, E, S \vdash e_1 \mapsto v_1, S_1 \qquad so, ?, ? \vdash e_2 \mapsto v_2, S_2}{so, E, S \vdash let\ x : T \leftarrow e_1\ in\ e_2 \mapsto v_2, S_2}$$

Necesitatea rezervării unei noi locații pentru x.

Evaluarea expresiei *let* (cont.)

$$\frac{so, E, S \vdash e_1 \mapsto v_1, S_1 \quad l_x = newloc(S_1) \quad so, E[l_x/x], S_1[v_1/l_x] \vdash e_2 \mapsto v_2, S_2}{so, E, S \vdash let\ x : T \leftarrow e_1\ in\ e_2 \mapsto v_2, S_2}$$

$newloc(S)$ = locație neutilizată din S.

Evaluarea lui *new*

- Rezervarea locațiilor pentru **atribute**
- Constituirea noului **obiect**
- Inițializarea atributelor cu valorile **implicite** (necesară din cauza vizibilității atributelor în întreaga clasă, inclusiv în propriile expresii de inițializare și ale atributelor definite mai devreme)
- Evaluarea expresiilor de **inițializare** a atributelor și actualizarea ultimelor

Notația pentru clase

$$class(A) = (a_1 : T_1 \leftarrow e_1, \ldots, a_n : T_n \leftarrow e_n), \text{ unde}$$

$a_i =$ **atributele**, inclusiv cele moștenite!

$T_i =$ **tipul** lui a_i

$e_i =$ expresia de **inițializare** a lui a_i

Valorile implicite

$$D_{Int} = Int(0)$$
$$D_{String} = String(0,"")$$
$$D_{Bool} = Bool(false)$$
$$D_A = void$$

D_T = valoarea **implicită** pentru tipul T.

Evaluarea lui *new* (cont.)

$$T_0 = \begin{cases} X, & \text{dacă } T = SELF\_TYPE \text{ și } so = X(\ldots) \\ T, & \text{altfel} \end{cases}$$

$$class(T_0) = (a_1 : T_1 \leftarrow e_1, \ldots, a_n : T_n \leftarrow e_n)$$

$$l_i = newloc(S),\ 1 \leq i \leq n$$

$$v = T_0(a_1 = l_1, \ldots, a_n = l_n)$$

$$E_1 = [a_1 : l_1, \ldots, a_n : l_n]$$

$$S_1 = S[D_{T_1}/l_1, \ldots, D_{T_n}/l_n]$$

$$\frac{v, E_1, S_1 \vdash \{\ a_1 \leftarrow e_1;\ \ldots;\ a_n \leftarrow e_n;\ \} \mapsto v_n, S_2}{so, E, S \vdash new\ T \mapsto v, S_2}$$

Utilizarea noului obiect, v, în calitate de *self object* pentru evaluarea inițializărilor.

Evaluarea apelurilor de metodă

- Evaluarea **parametrilor**
- Evaluarea **expresiei** pentru care realizăm apelul
- Determinarea **implementării** corecte a metodei, în funcție de tipul dinamic al obiectului
- Rezervarea **locațiilor** pentru parametrii formali și inițializarea acestora cu valorile parametrilor actuali
- Evaluarea **corpului** metodei, utilizând obiectul pentru care realizăm apelul în rol de *self object*

Notația pentru metode

$$impl(A, f) = (x_1, \ldots, x_n, e), \text{ unde}$$
$$x_i = \textbf{parametrii} \text{ formali}$$
$$e = \textbf{corpul}$$

Evaluarea apelurilor de metodă (cont.)

$$\frac{\begin{array}{l} so, E, S \vdash e_1 \mapsto v_1, S_1 \\ so, E, S_1 \vdash e_2 \mapsto v_2, S_2 \\ \vdots \\ so, E, S_{n-1} \vdash e_n \mapsto v_n, S_n \\ so, E, S_n \vdash e_0 \mapsto v_0, S_{n+1} \\ v_0 = X(a_1 = l_1, \ldots, a_m = l_m) \\ impl(X, f) = (x_1, \ldots, x_n, e) \\ l_{x_i} = newloc(S_{n+1}),\ 1 \leq i \leq n \\ E_1 = [a_1 : l_1, \ldots, a_m : l_m][l_{x_1}/x_1, \ldots, l_{x_n}/x_n] \\ S_{n+2} = S_{n+1}[v_1/l_{x_1}, \ldots, v_n/l_{x_n}] \\ v_0, E_1, S_{n+2} \vdash e \mapsto v, S_{n+3} \end{array}}{so, E, S \vdash e_0.f(e_1, \ldots, e_n) \mapsto v, S_{n+3}}$$

Erori la execuție

- **Apel** de metodă pe un obiect *void*
- ***case*** pe un obiect *void*
- ***case*** cu toate ramurile nepotrivite
- Index sau lungime în afara intervalului la ***String*.*substr***
- **Împărțire** la zero
- Depășire ***heap***

Partea VIII

Optimizări și cod intermediar

Cuprins

Cuprins

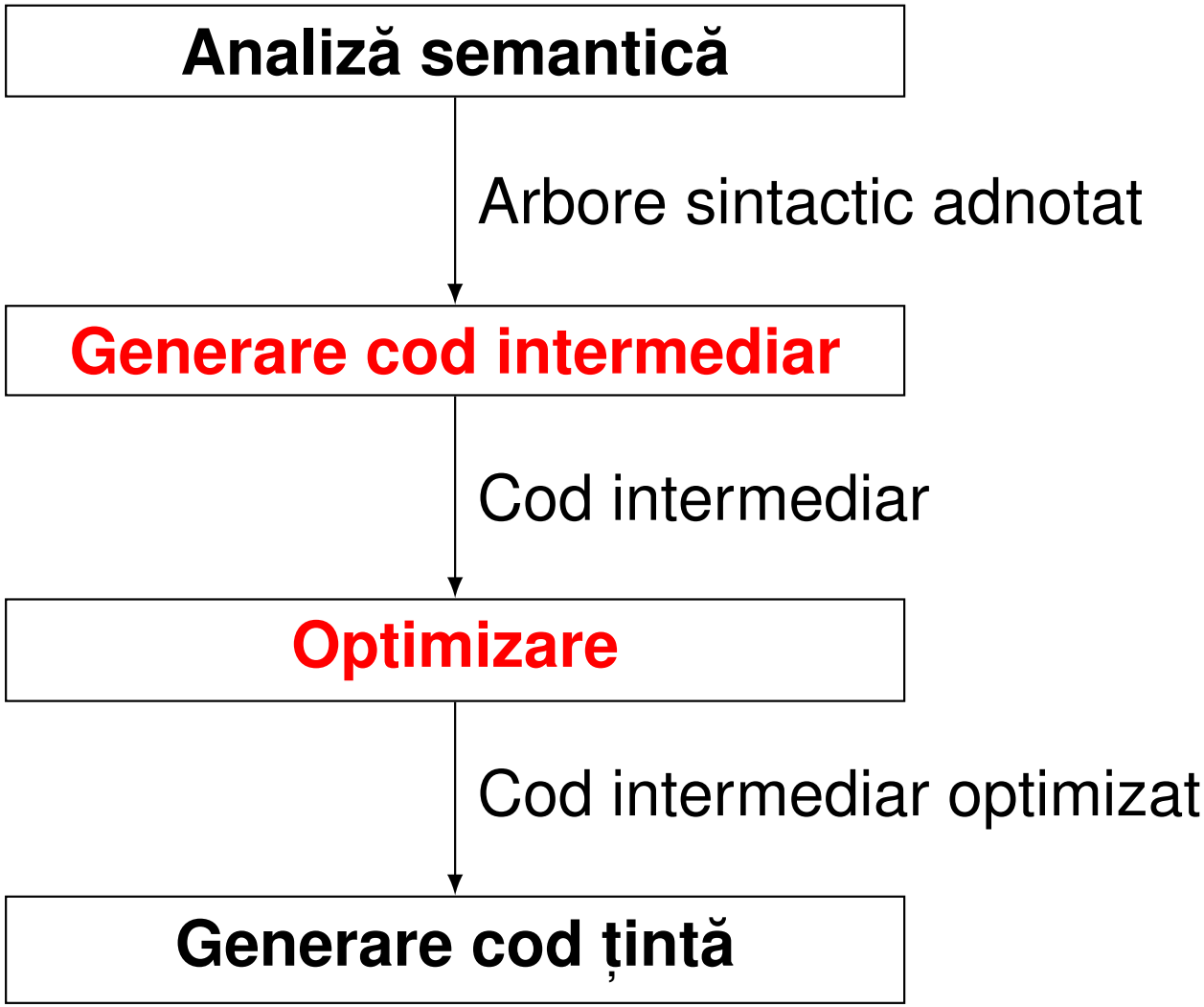
Optimizare

- Prelucrarea automată a programului, în scopul reducerii resurselor utilizate: timp de execuție, memorie, dimensiunea codului generat
- Exemplu: `x + 0` $\rightarrow$ `x` (simplificare algebrică)
- Termen ușor impropriu; alternativ, *îmbunătățire*
- În prezent, cea mai elaborată etapă a compilării!

Niveluri de optimizare

- **Arbore sintactic**: nivel prea înalt, neexpunând detaliile mașinii
- **Cod țintă**: total dependent de mașină, nepermițând reutilizarea optimizărilor
- Obiectiv: expunerea unor detalii de implementare, permițând totodată reutilizarea optimizărilor
- Soluție: **cod intermediar**, de nivel mai scăzut decât cel sursă, dar mai înalt decât cel țintă

Context


Analiză semantică
Arbore sintactic adnotat
Generare cod intermediar
Cod intermediar
Optimizare
Cod intermediar optimizat
Generare cod țintă

Cuprins

Cod intermediar

- Similar unui cod de asamblare de nivel mai înalt
- Utilizarea **registrelor**, dar în număr nelimitat
- Prezența **structurilor de control**
- Folosirea **instrucțiunilor**, în corespondență 1 : 1 sau 1 : *n* cu cele din limbajul de asamblare

Three-address code

- Prezența exclusivă a operațiilor **unare și binare**, cu „numirea” rezultatului:

$$x := y \; op \; z$$
$$x := op \; y,$$

  unde y și z sunt **registre sau constante**

- Exemplu pentru $x + y \cdot z$:

$$t_1 := y \cdot z$$
$$t_2 := x + t_1$$

Generarea codului intermediar

- Similară generării codului țintă, dar exploatând un număr nelimitat de registre

- Exemplu pentru **adunare**, unde al doilea parametru indică registrul în care este depus rezultatul:

```
1  igen(e1 + e2, t) =
2       igen(e1, t1)
3       igen(e2, t2)
4       t := t1 + t2
```

Un posibil limbaj intermediar

- $x := y \ op \ z$
- $x := op \ y$
- $x := y$
- *push x*
- $x := pop$
- $if \ x \leq y \ goto \ L$
- *jump L*
- L : (etichetă)

Blocuri de bază

- **Bloc de bază** (*basic block*, BB) = secvență maximală de instrucțiuni executate întotdeauna în ordine, cu un singur punct de intrare (la început) și un singur punct de ieșire (la sfârșit)
- **Etichete**, posibile exclusiv la început
- **Salturi**, posibile exclusiv la sfârșit

Exemplu

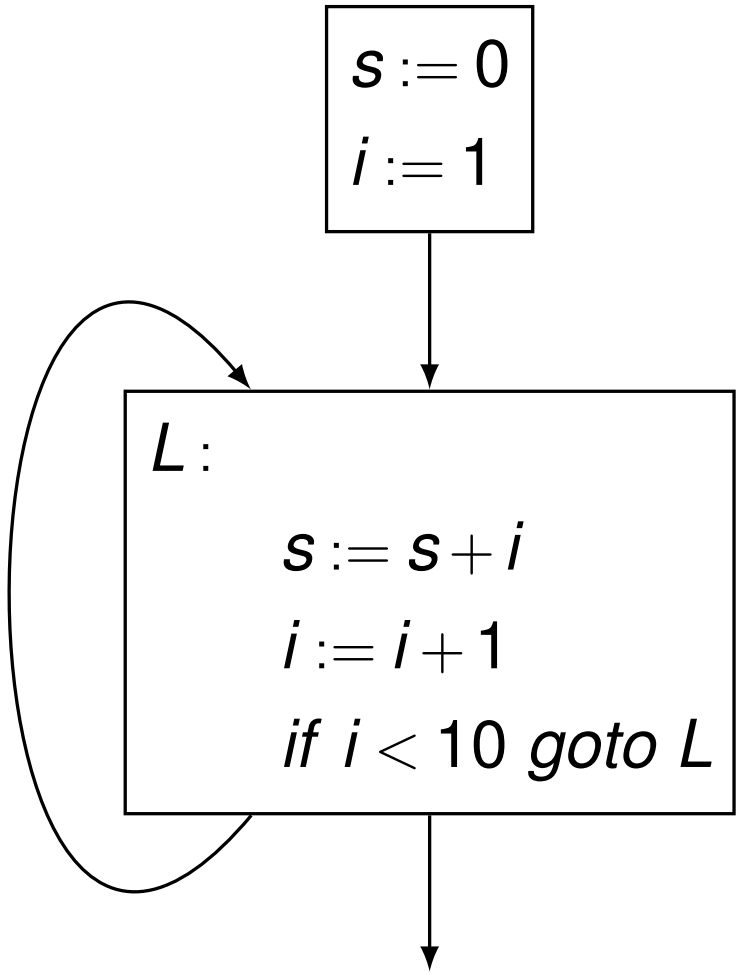
$$
\begin{aligned}
L: \quad & \\
& t := x \\
& y := t + x \\
& \textit{if } y > 2 \textit{ goto } L'
\end{aligned}
$$

- Posibilitatea înlocuirii $y := t + x$ cu $y := x + x$, pe baza garanției oferite de BB, cum că prima atribuire se execută întotdeauna înaintea celei de-a doua
- Imposibilitatea eliminării atribuirii $t := x$, în absența informației despre alte utilizări ale lui t, în afara BB

Grafuri ale fluxului de control

- **Graf al fluxului de control** (*control-flow graph*, CFG) = graf orientat, format din BB-uri, care surprinde punctele de decizie
- **Noduri**: BB-uri
- **Arc** între nodurile *A* și *B*, dacă execuția poate trece de la ultima instrucțiune a lui *A* la prima instrucțiune a lui *B* (natural sau prin salt)
- **Corpurile** metodelor, reprezentabile prin CFG-uri

Exemplu



Clasificarea optimizărilor

- **Locale**: la nivelul unui singur BB
- **Globale**: la nivelul CFG (i.e. corp de metodă)
- **Inter-procedurale**: dincolo de granițele metodelor (rar implementate)

Cuprins

Optimizări locale

- Cele mai simple
- Aplicate la nivelul unui singur BB
- Exemplu: simplificarea algebrică $x \cdot 1 = x$

Simplificări algebrice

$x := x + 0$	$\rightarrow$	eliminată
$x := x \cdot 1$	$\rightarrow$	eliminată
$x := x \cdot 0$	$\rightarrow$	$x := 0$
$x := x^2$	$\rightarrow$	$x := x \cdot x$
$x := x \cdot 4$	$\rightarrow$	$x := x \ll 2$

Evaluarea operațiilor pe constante

- ***Constant folding***: evaluarea operațiilor pe constante la compilare, în loc de execuție
- Exemple:

$x := 1 + 2$	$\rightarrow$	$x := 3$
if $1 < 0$ *goto L*	$\rightarrow$	eliminată
if $0 < 1$ *goto L*	$\rightarrow$	*jump L*

- Riscantă în situații de cross-compilare

Eliminarea BB-urilor moarte

- BB ***unreacheable***: nu îi poate urma natural altui BB, și nici nu constituie ținta unui salt
- Avantaj: reducerea dimensiunii codului, cu eventuala sporire a vitezei, datorită efectelor de *cache*
- Întâlnite datorită:
  - Instrucțiunilor de depanare: *if* (*DEBUG*) *then* ...
  - Părților neutilizate ale bibliotecilor
  - Optimizărilor anterioare

Forma cu atribuiri unice

- ***Static single-assignment form*** (SSA): fiecărui registru îi este atribuită o valoare o singură dată, eventual în urma rescrierii codului intermediar, în vederea facilitării anumitor optimizări
- Exemplu:

$$\begin{array}{lcl} x := y + z & & a := y + z \\ u := x & \rightarrow & u := a \\ x := x \cdot 2 & & x := a \cdot 2 \end{array}$$

- În general, rescriere nebanală, din cauza deciziilor și a buclelor

Eliminarea subexpresiilor comune

- ***Common subexpression elimination***: în forma SSA, identitatea expresiilor din partea dreaptă a atribuirilor $\Rightarrow$ identitatea de rezultat

- Exemplu:

$$\begin{array}{lcl} x := y + z & & x := y + z \\ \ldots & \rightarrow & \ldots \\ u := y + z & & u := x \end{array}$$

(garanția constanței lui x, y, z între cele două atribuiri)

Propagarea copiilor și a constantelor

- ***Copy/constant propagation***: în forma SSA, urmare definiției $u := x$, înlocuirea aparițiilor ulterioare ale lui u cu x

- Exemplu:

$$\begin{array}{l} a := y + z \\ u := a \\ x := u \cdot 2 \end{array} \quad \rightarrow \quad \begin{array}{l} a := y + z \\ u := a \\ x := a \cdot 2 \end{array}$$

- Utilă nu *per se*, ci prin facilitarea altor optimizări (e.g. *constant folding* sau eliminarea codului mort)

Exemplu

$$x := 5$$
$$y := x \cdot 2$$
$$z := y + 1$$
$$u := y \cdot z$$

Exemplu

$$x := 5$$
$$y := x \cdot 2$$
$$z := y + 1$$
$$u := y \cdot z$$

Constant propagation

Exemplu

$$x := 5$$
$$y := 5 \cdot 2$$
$$z := y + 1$$
$$u := y \cdot z$$

Constant propagation

Exemplu

$$x := 5$$
$$y := 5 \cdot 2$$
$$z := y + 1$$
$$u := y \cdot z$$

Constant folding

Exemplu

$$x := 5$$
$$y := 10$$
$$z := y + 1$$
$$u := y \cdot z$$

Constant folding

Exemplu

$$x := 5$$
$$y := 10$$
$$z := y + 1$$
$$u := y \cdot z$$

Constant propagation

Exemplu

$$x := 5$$
$$y := 10$$
$$z := 10 + 1$$
$$u := 10 \cdot z$$

Constant propagation

Exemplu

$$x := 5$$
$$y := 10$$
$$z := 10 + 1$$
$$u := 10 \cdot z$$

Constant folding

Exemplu

$$x := 5$$
$$y := 10$$
$$z := 11$$
$$u := 10 \cdot z$$

Constant folding

Exemplu

$$x := 5$$
$$y := 10$$
$$z := 11$$
$$u := 10 \cdot z$$

Constant propagation

Exemplu

$$x := 5$$
$$y := 10$$
$$z := 11$$
$$u := 10 \cdot 11$$

Constant propagation

Exemplu

$$x := 5$$
$$y := 10$$
$$z := 11$$
$$u := 10 \cdot 11$$

Constant folding

Exemplu

$$x := 5$$
$$y := 10$$
$$z := 11$$
$$u := 110$$

Constant folding

Eliminarea codului mort

- **Cod mort**: instrucțiune de forma $u := x$, unde u nu mai este folosit nicăieri

- Exemplu de *copy propagation*:

$$\begin{array}{lcl} a := y + z & & a := y + z \\ u := a & \rightarrow & u := a \\ x := u \cdot 2 & & x := a \cdot 2 \end{array}$$

- Posibilitatea schimbării stării unei instrucțiuni în urma aplicării unor optimizări

Eliminarea codului mort

- **Cod mort**: instrucțiune de forma $u := x$, unde u nu mai este folosit nicăieri

- Exemplu de *copy propagation*:

$$\begin{array}{lcll} a := y + z & & a := y + z & \\ u := a & \rightarrow & u := a & \text{(cod mort)} \\ x := u \cdot 2 & & x := a \cdot 2 & \end{array}$$

- Posibilitatea schimbării stării unei instrucțiuni în urma aplicării unor optimizări

Interacțiunea optimizărilor

- Contribuții reduse ale optimizărilor individuale
- Facilitarea anumitor optimizări, neaplicabile inițial, în urma realizării altora
- Aplicarea repetată a optimizărilor până la absența modificărilor

Exemplu complex

$$a := x^2$$
$$b := 3$$
$$c := x$$
$$d := c \cdot c$$
$$e := b \cdot 2$$
$$f := a + d$$
$$g := e \cdot f$$

Exemplu complex

$$a := x^2$$
$$b := 3$$
$$c := x$$
$$d := c \cdot c$$
$$e := b \cdot 2$$
$$f := a + d$$
$$g := e \cdot f$$

Simplificare algebrică

Exemplu complex

$$a := x \cdot x$$
$$b := 3$$
$$c := x$$
$$d := c \cdot c$$
$$e := b \ll 1$$
$$f := a + d$$
$$g := e \cdot f$$

Simplificare algebrică

Exemplu complex

$$a := x \cdot x$$
$$b := 3$$
$$c := x$$
$$d := c \cdot c$$
$$e := b \ll 1$$
$$f := a + d$$
$$g := e \cdot f$$

Copy/ constant propagation

Exemplu complex

$$a := x \cdot x$$
$$b := 3$$
$$c := x$$
$$d := x \cdot x$$
$$e := 3 \ll 1$$
$$f := a + d$$
$$g := e \cdot f$$

Copy/ constant propagation

Exemplu complex

$$a := x \cdot x$$
$$b := 3$$
$$c := x$$
$$d := x \cdot x$$
$$e := 3 \ll 1$$
$$f := a + d$$
$$g := e \cdot f$$

Constant folding

Exemplu complex

$$a := x \cdot x$$
$$b := 3$$
$$c := x$$
$$d := x \cdot x$$
$$e := 6$$
$$f := a + d$$
$$g := e \cdot f$$

Constant folding

Exemplu complex

$$a := x \cdot x$$
$$b := 3$$
$$c := x$$
$$d := x \cdot x$$
$$e := 6$$
$$f := a + d$$
$$g := e \cdot f$$

Eliminarea subexpresiilor comune

Exemplu complex

$$a := x \cdot x$$
$$b := 3$$
$$c := x$$
$$d := a$$
$$e := 6$$
$$f := a + d$$
$$g := e \cdot f$$

Eliminarea subexpresiilor comune

Exemplu complex

$$a := x \cdot x$$
$$b := 3$$
$$c := x$$
$$d := a$$
$$e := 6$$
$$f := a + d$$
$$g := e \cdot f$$

Copy/ constant propagation

Exemplu complex

$$a := x \cdot x$$
$$b := 3$$
$$c := x$$
$$d := a$$
$$e := 6$$
$$f := a + a$$
$$g := 6 \cdot f$$

Copy/ constant propagation

Exemplu complex

$$a := x \cdot x$$
$$b := 3$$
$$c := x$$
$$d := a$$
$$e := 6$$
$$f := a + a$$
$$g := 6 \cdot f$$

Eliminarea codului mort
(presupunând că b, c, d, e nu mai sunt folosite)

Optimizare în fereastră pe codul țintă

- ***Peephole optimization***: înlocuirea unei secvențe (de obicei, contigue) de instrucțiuni cu alta mai eficientă

- Reprezentare sub forma unor **reguli de substituție**:

$$i_1, \ldots, i_n \rightarrow j_1, \ldots, j_m$$

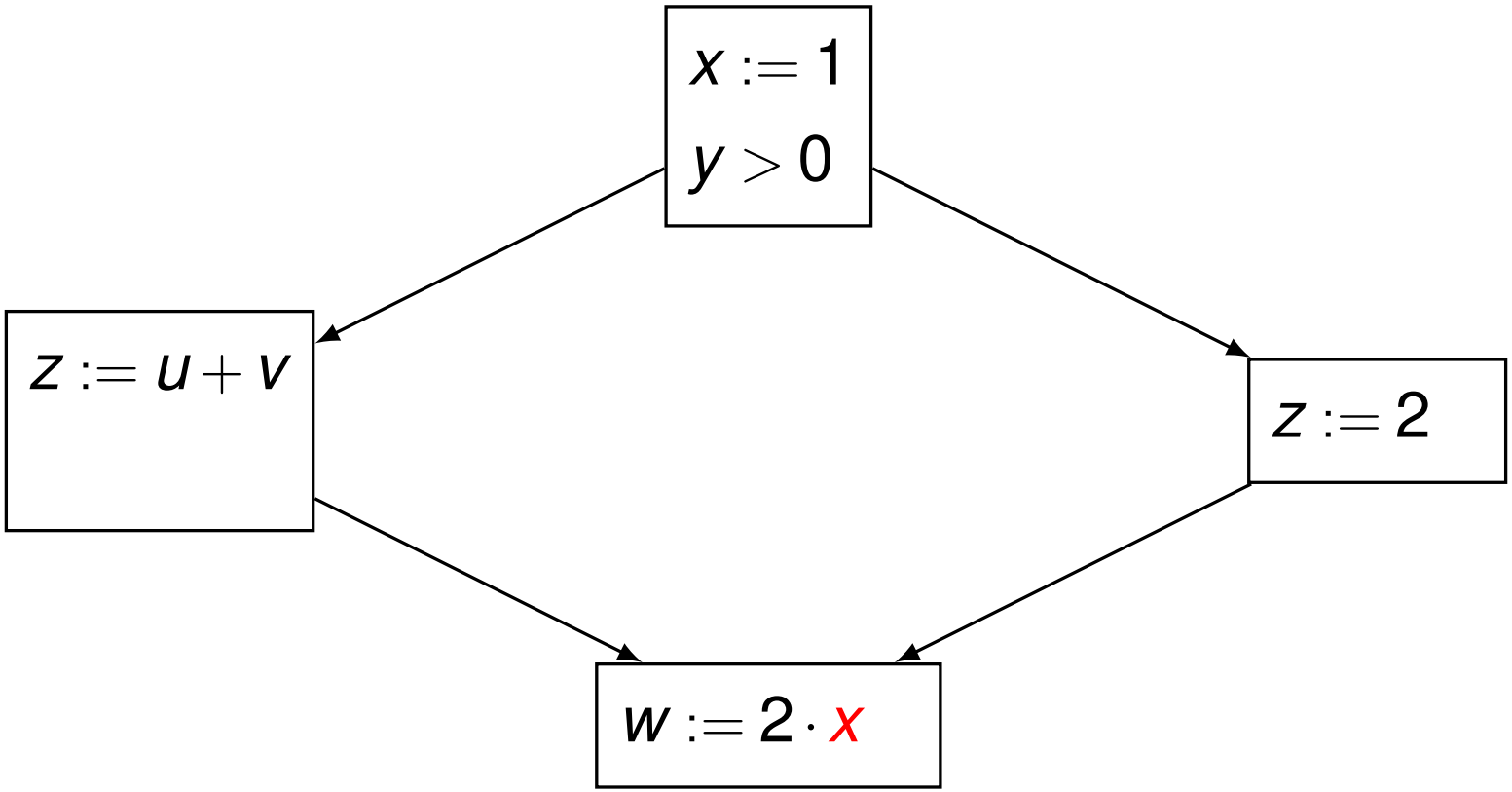
- Necesitatea aplicării repetate, conform principiilor anterioare

Exemplu

```
move $a $b, move $b $a          →  move $a $b
addiu $a $a i, addiu $a $a j    →  addiu $a $a i+j
addiu $a $b 0                   →  move $a $b
move $a $a                      →
```

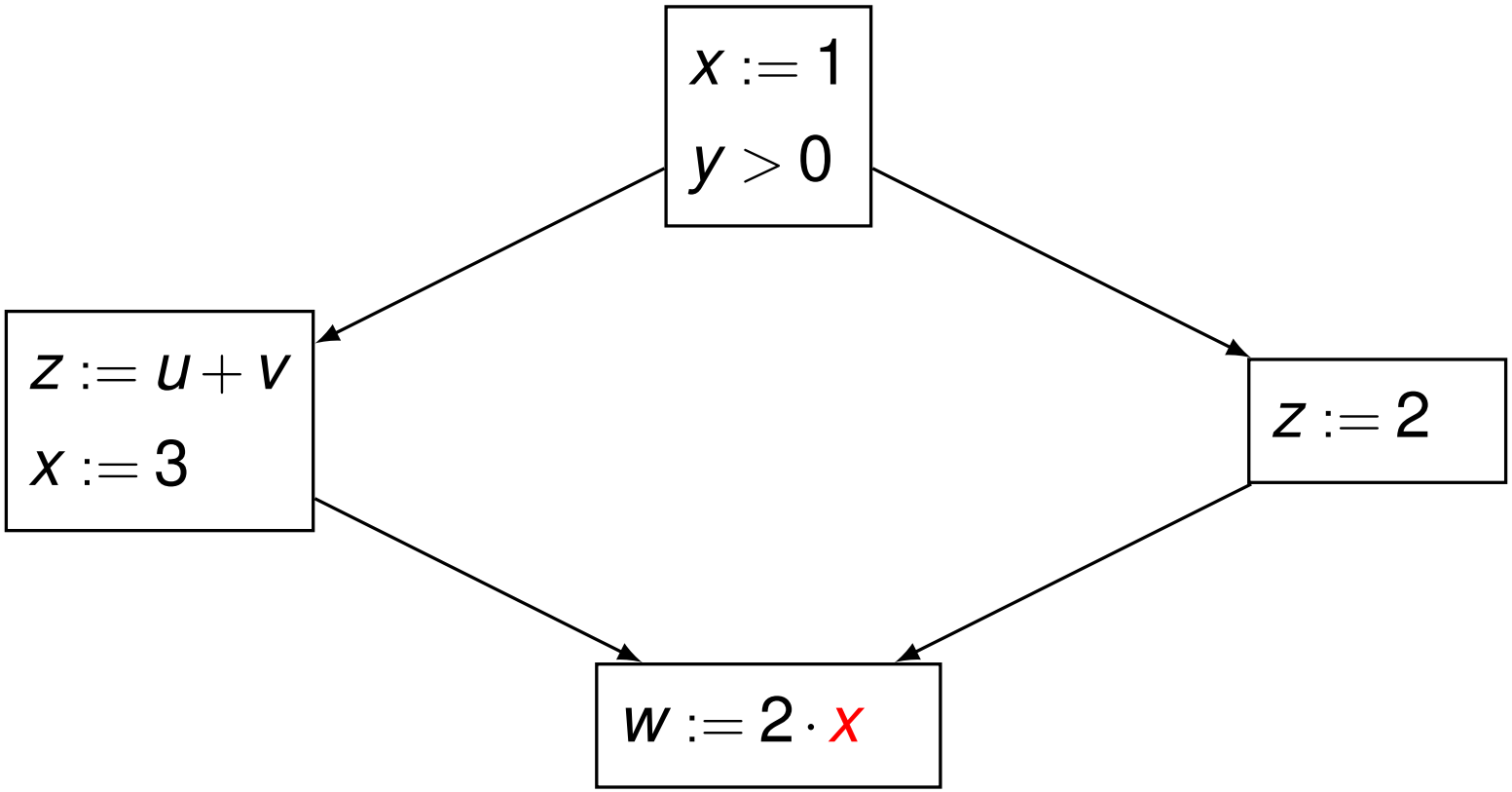
Cuprins

Motivație



Putem aplica propagarea constantelor pentru x,
în cadrul întregului CFG?

Motivație



Dar acum?

Răspuns intuitiv

- Posibilitatea înlocuirii unei utilizări a lui x cu constanta c dacă, pe orice cale către acea utilizare, ultima atribuire a lui x este $x := c$
- Posibilitatea întâlnirii de decizii și bucle pe aceste căi
- Necesitatea unei analize **globale**

Optimizări globale

- Necesitatea demonstrării unor **proprietăți** în diferite puncte din program
- Dependența validității acestor proprietăți de structura întregului CFG
- Strategia utilizată: **analiza globală a fluxului de date**
- Răspunsuri furnizate despre validitate proprietăților: „da” sau „nu știu” (soluția exactă, prea costisitoare)

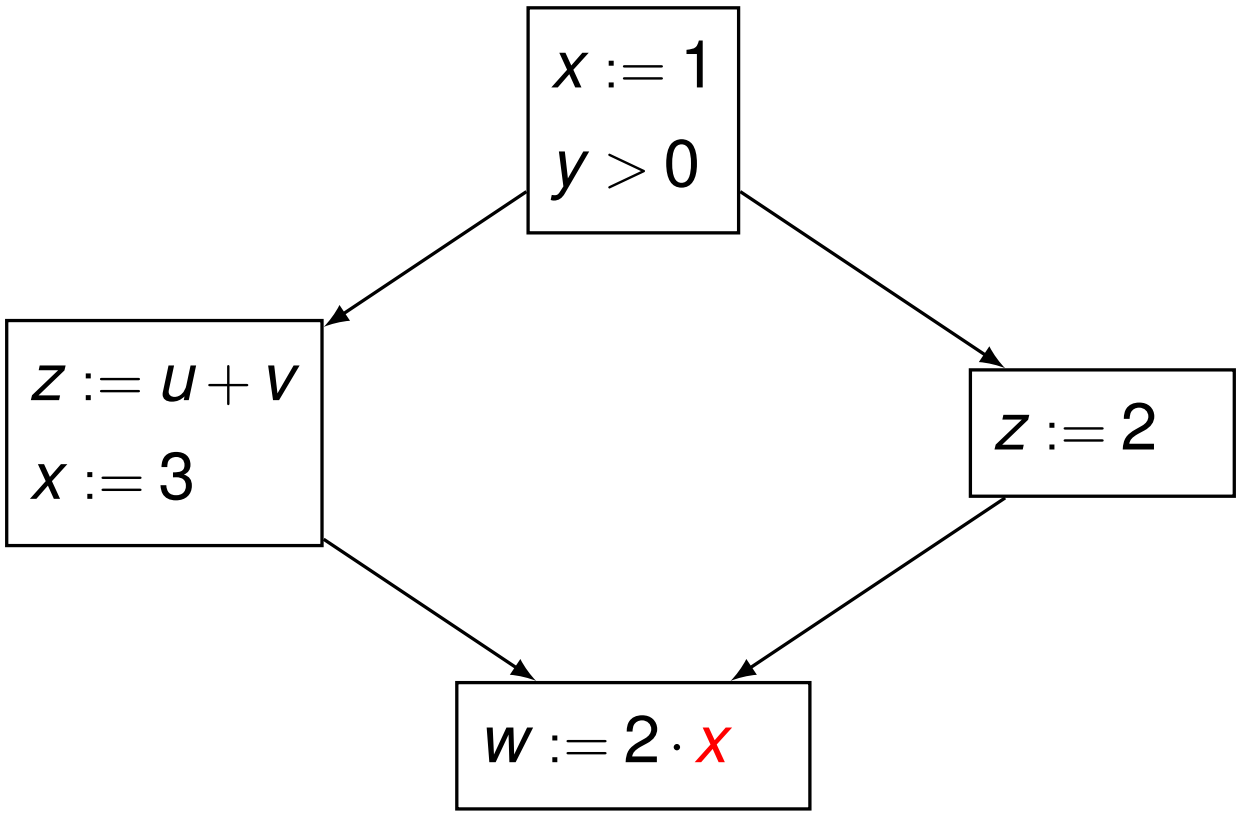
Propagarea globală a constantelor

- Aplicabilă în punctele din program unde ține proprietatea de mai devreme, i.e. „pe orice cale ...”

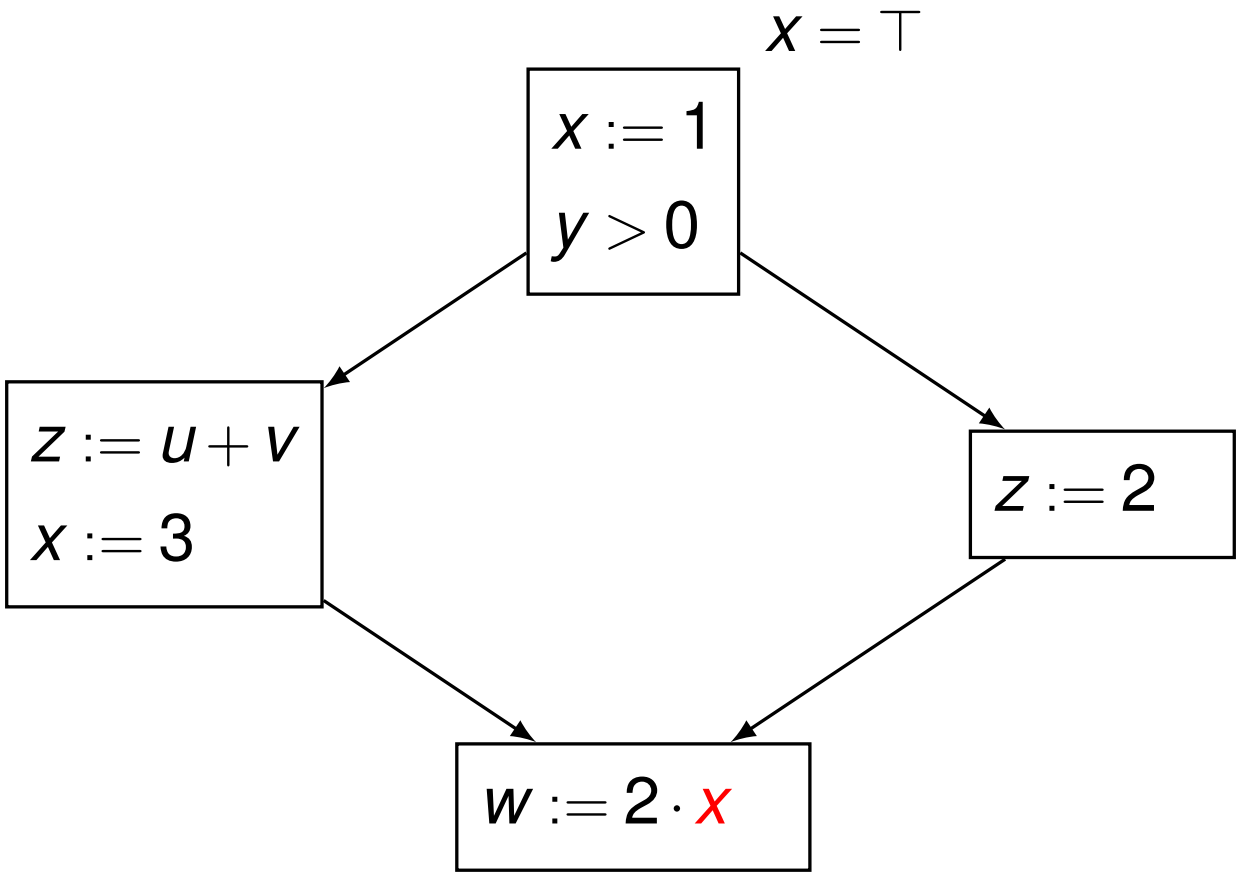
- Asocierea unei valori pentru variabila țintă x, în fiecare punct din program:

Valoare	Semnificație
$\bot$	Instrucțiune neexecutabilă
c	x are valoarea c
$\top$	x nu este constantă

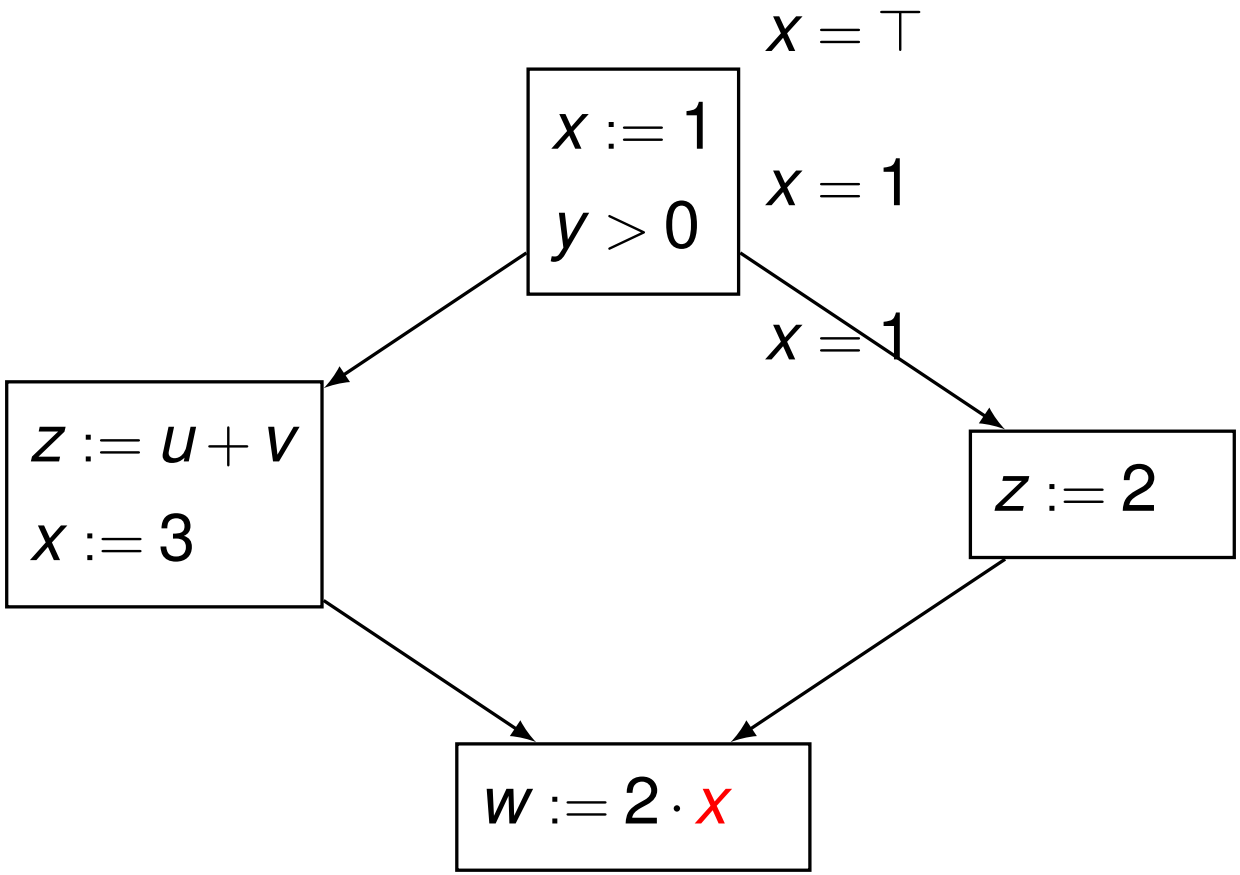
Exemplu reluat



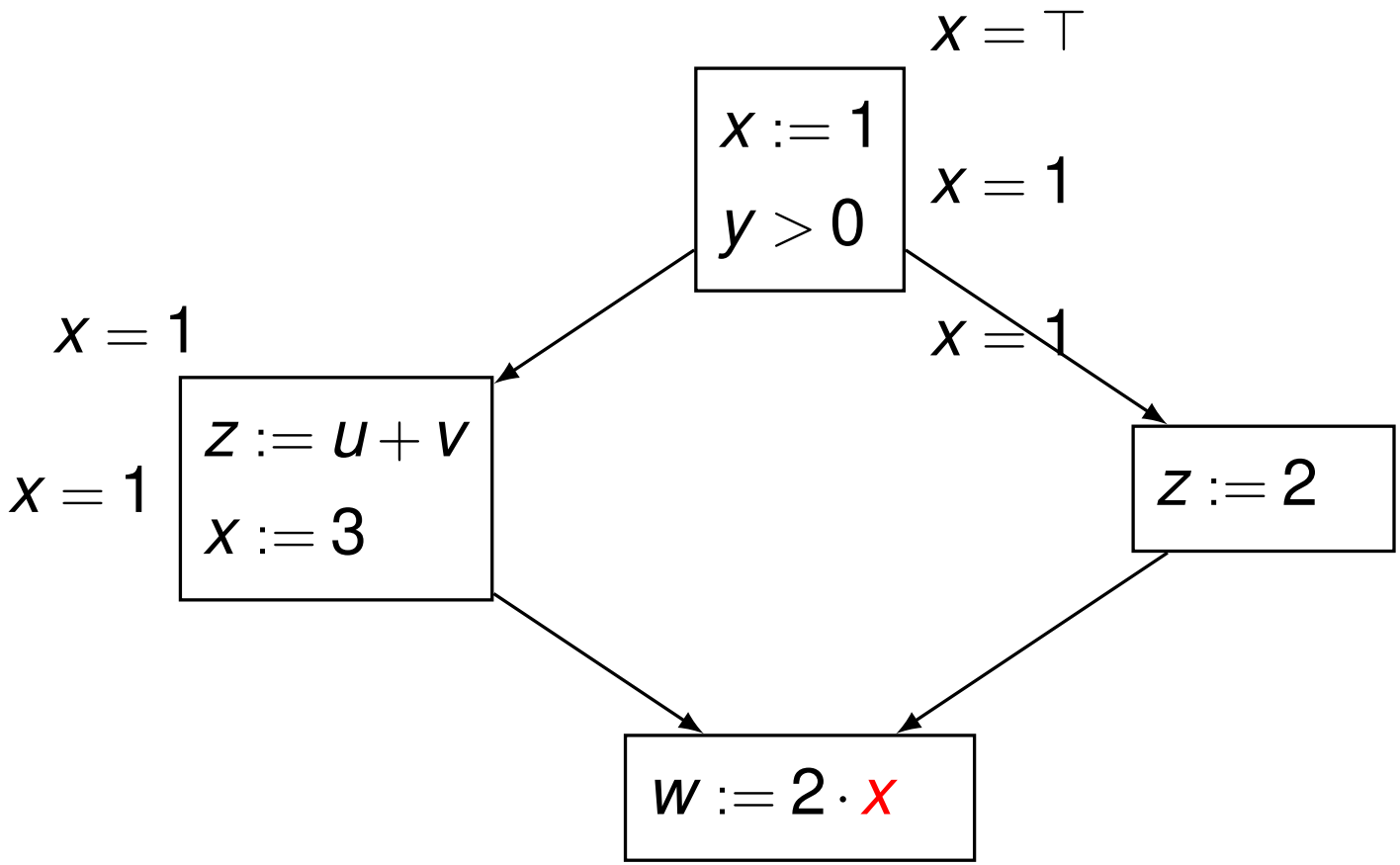
Exemplu reluat



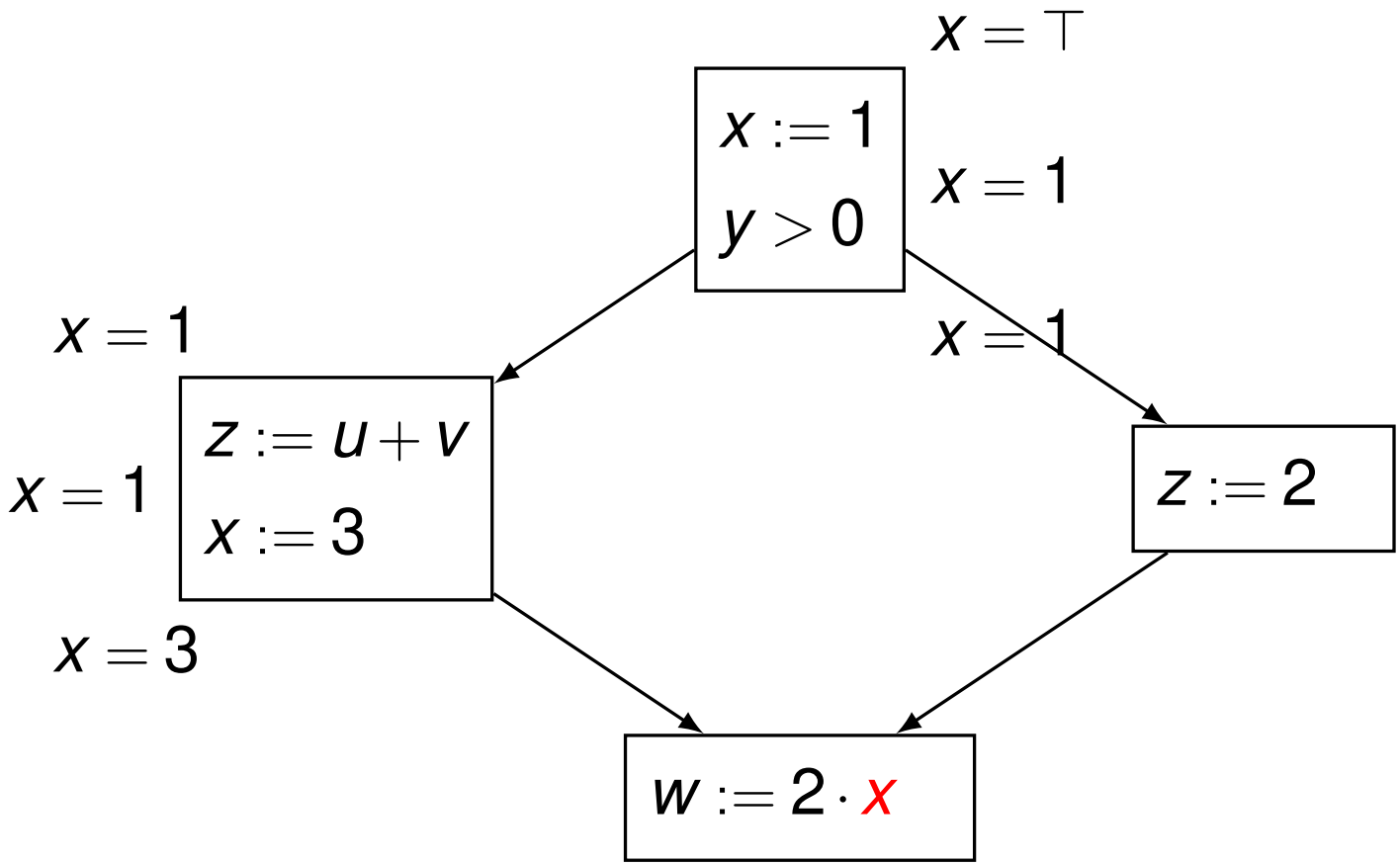
Exemplu reluat



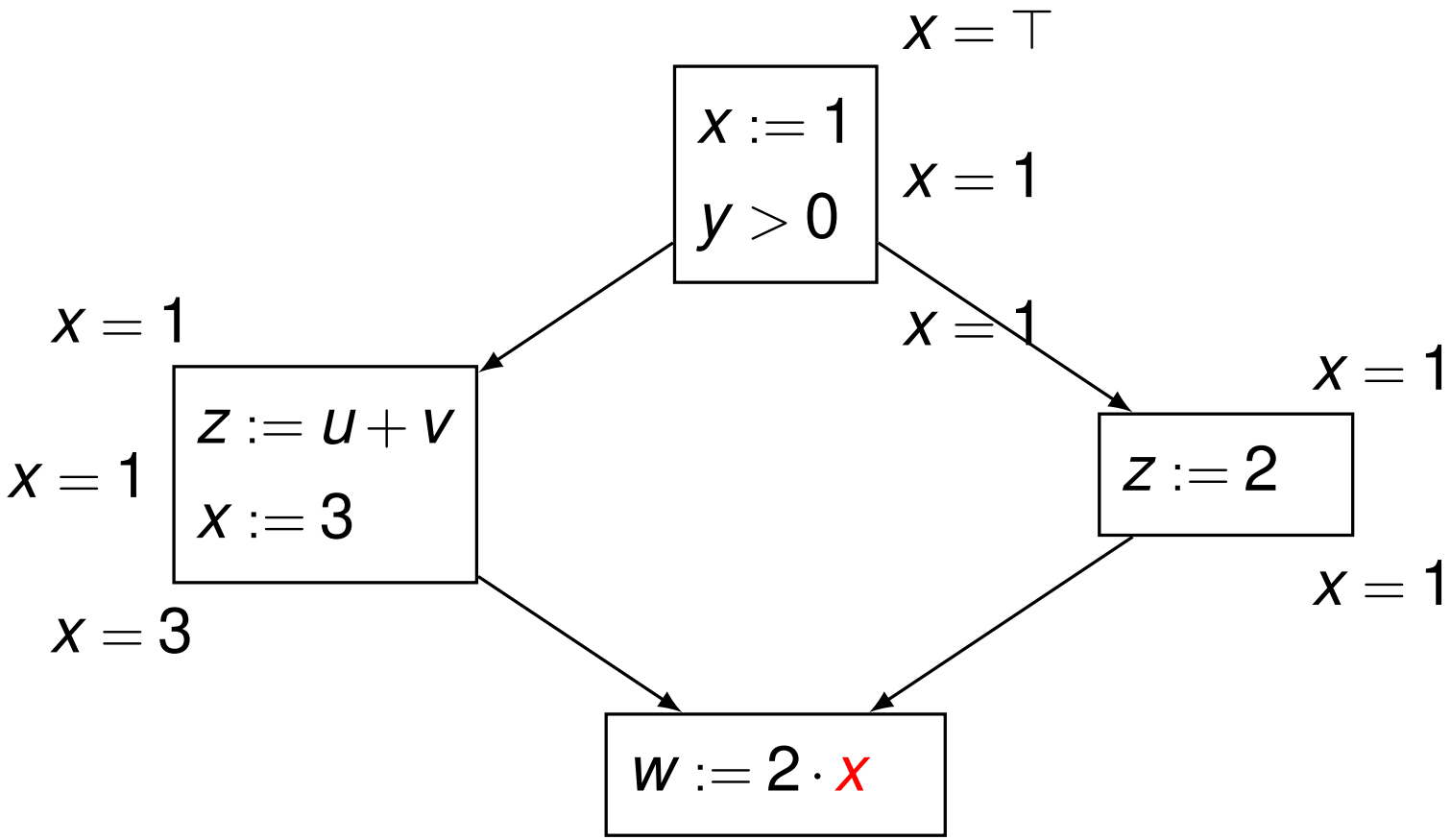
Exemplu reluat



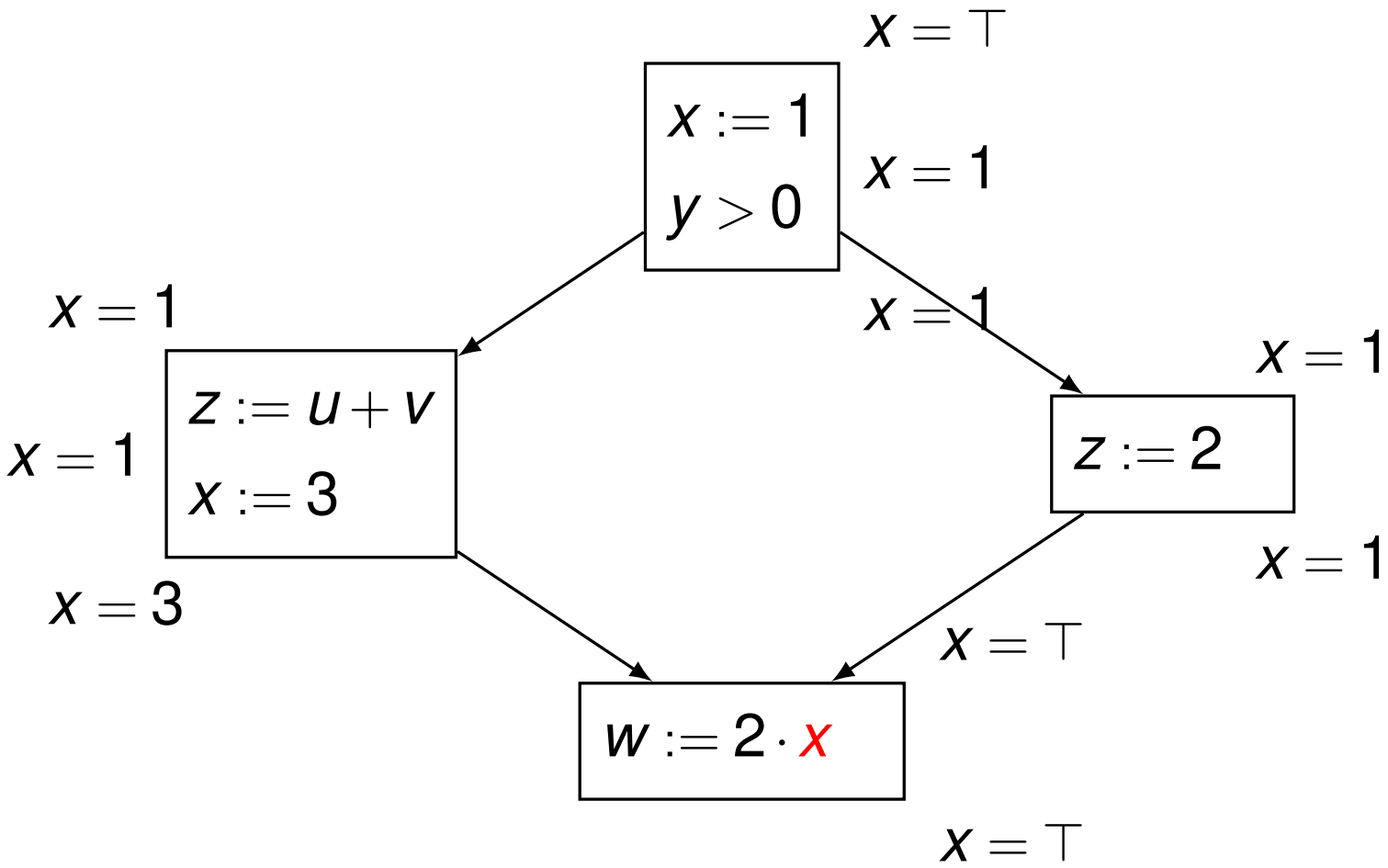
Exemplu reluat



Exemplu reluat



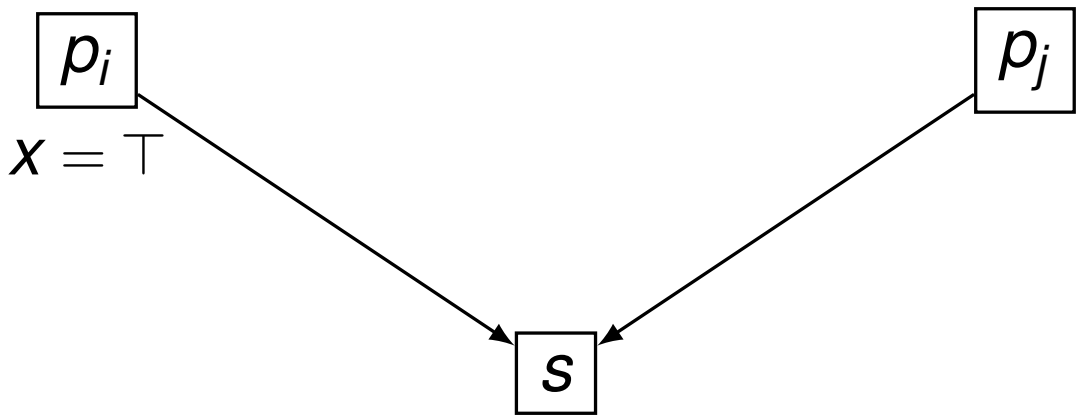
Exemplu reluat



Aplicabilitatea propagării globale a constantelor

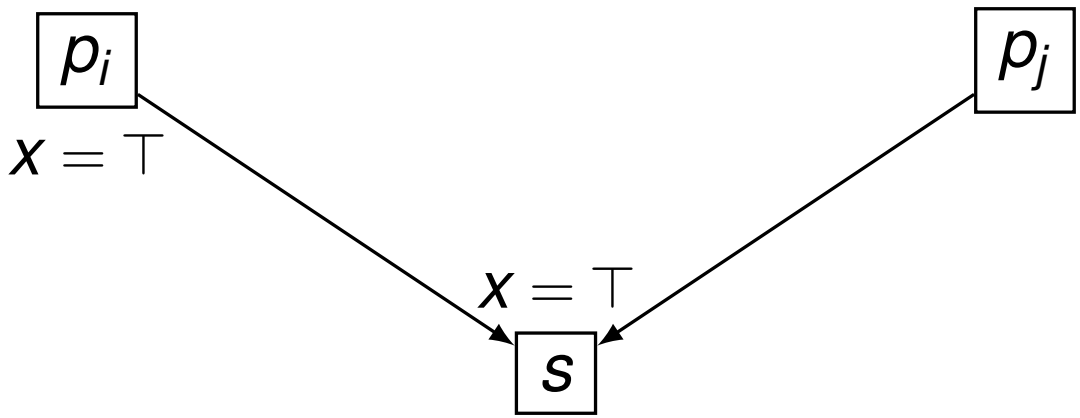
- În orice punct în care $x = c$
- Cum calculăm valorile asociate în fiecare punct?
- Numirea valorii lui x înainte și după instrucțiunea s: $C(s, x, in)$, $C(s, x, out)$
- Specificarea modului în care valorile sunt transferate inter- și intrainstrucțiune

Regula 1



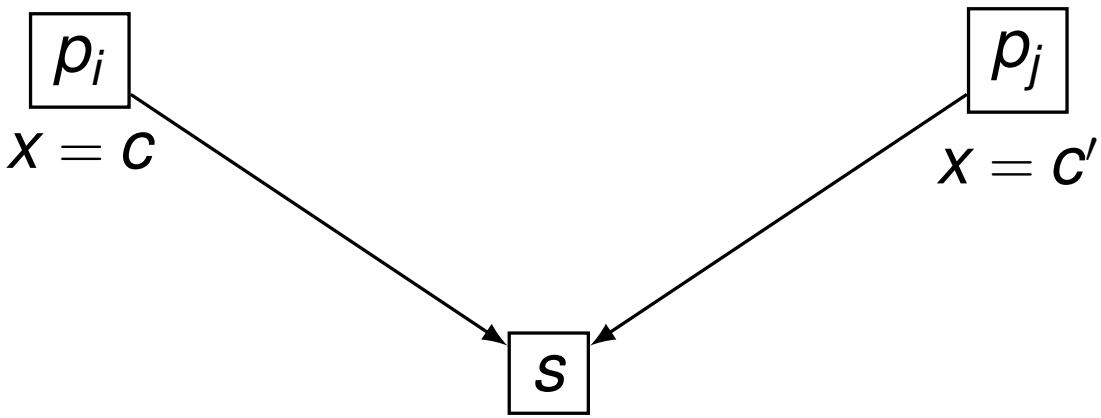
Dacă $C(p_i, x, out) = \top$,
atunci

Regula 1



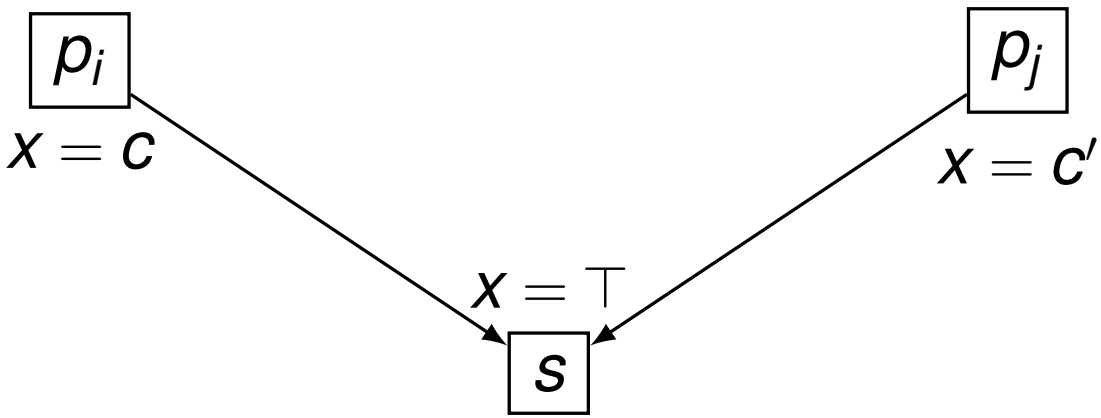
Dacă $C(p_i, x, out) = \top$,
atunci $C(s, x, in) = \top$

Regula 2



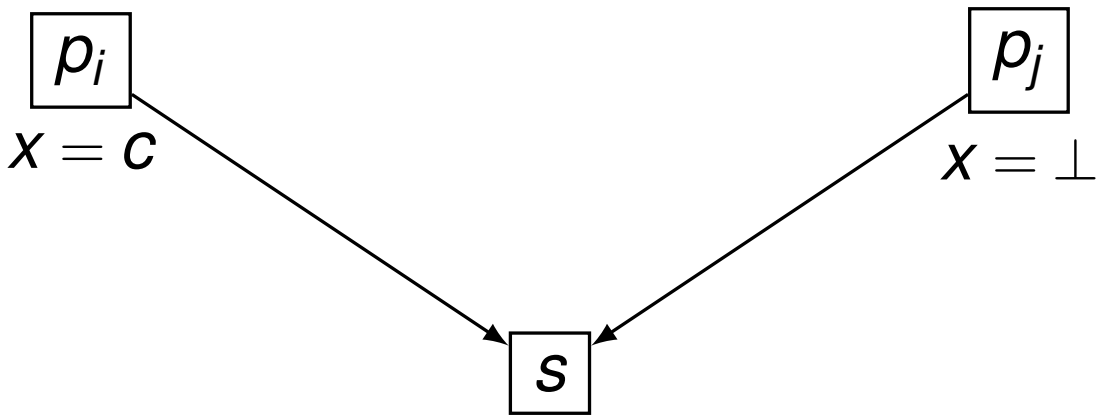
Dacă $C(p_i, x, out) = c$ și $C(p_j, x, out) = c' \neq c$, atunci

Regula 2



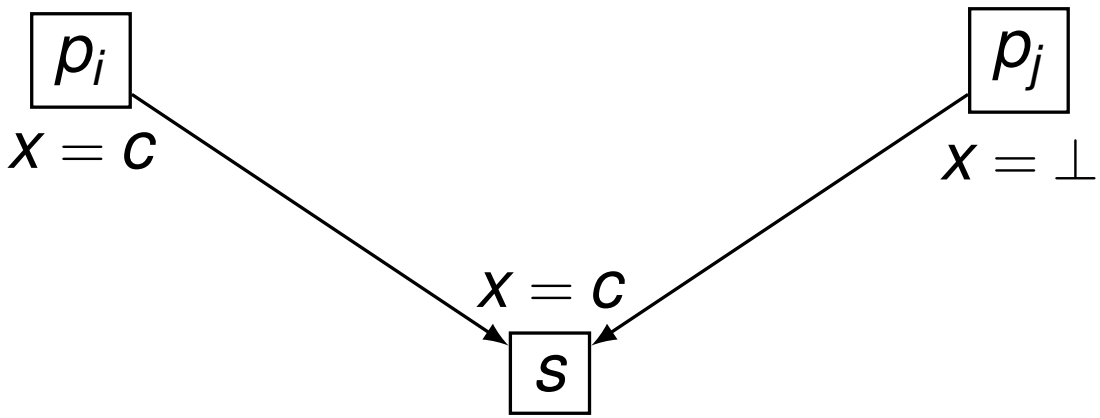
Dacă $C(p_i, x, out) = c$ și $C(p_j, x, out) = c' \neq c$,
atunci $C(s, x, in) = \top$

Regula 3



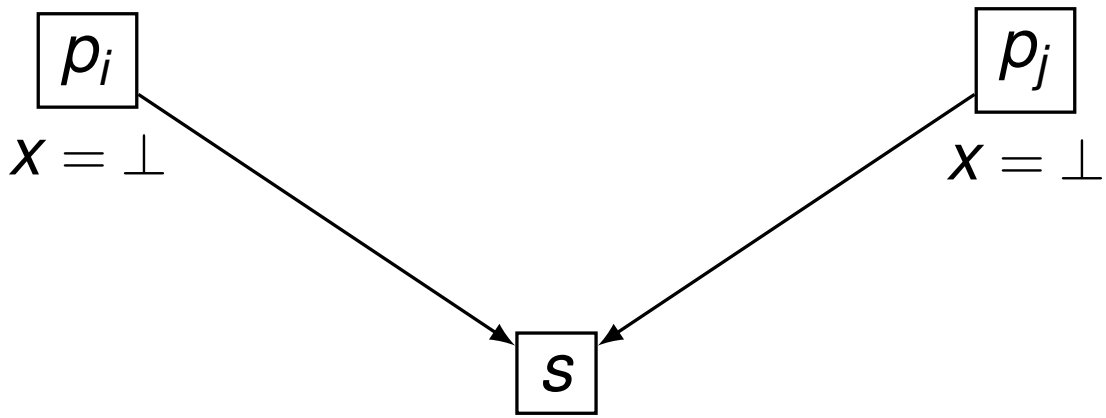
Dacă $C(p_k, x, out) = c$ sau $\perp$, pentru orice k, atunci

Regula 3



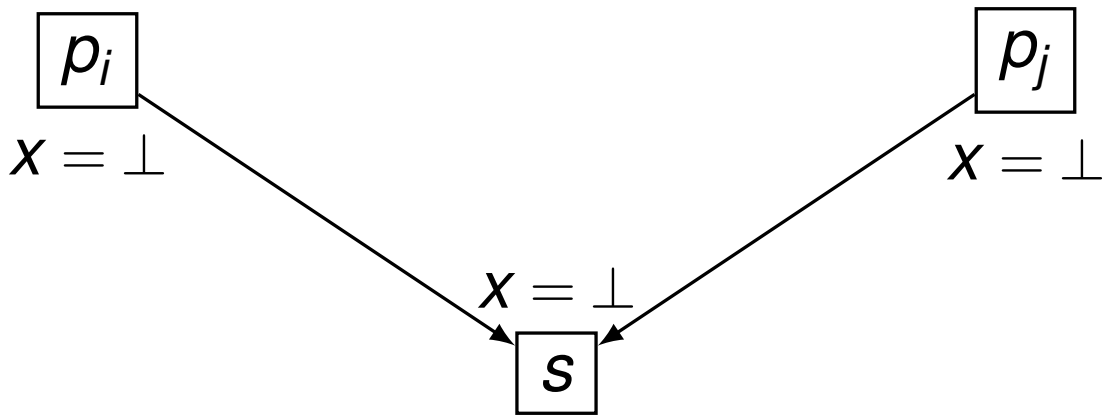
Dacă $C(p_k, x, out) = c$ sau $\perp$, pentru orice k,
atunci $C(s, x, in) = c$

Regula 4



Dacă $C(p_k, x, out) = \perp$, pentru orice k, atunci

Regula 4



Dacă $C(p_k, x, out) = \perp$, pentru orice k,
atunci $C(s, x, in) = \perp$

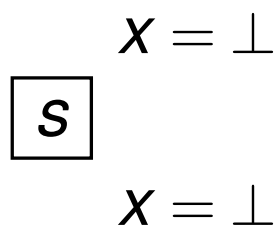
Regula 5

$x = \perp$

$\boxed{s}$

Dacă $C(s, x, in) = \perp$,
atunci

Regula 5



Dacă $C(s, x, in) = \perp$,
atunci $C(s, x, out) = \perp$

Regula 6

$$\frac{x \neq \bot}{\boxed{x := c}}$$

Dacă $C(x := c, x, in) \neq \bot$,
atunci

Regula 6

$$\frac{x \neq \bot}{\boxed{x := c}\quad x = c}$$

Dacă $C(x := c, x, in) \neq \bot$,
atunci $C(x := c, x, out) = c$

Regula 7

$$\frac{x \neq \bot}{\boxed{x := f(\ldots)}}$$

Dacă $C(x := f(\ldots), x, in) \neq \bot$,
atunci

Regula 7

$$\frac{x \neq \bot}{\boxed{x := f(\ldots)}}$$
$$x = \top$$

Dacă $C(x := f(\ldots), x, in) \neq \bot$,
atunci $C(x := f(\ldots), x, out) = \top$

Regula 8

$$\frac{}{\boxed{y := \ldots}}\ x = a$$

Dacă $C(y := \ldots, x, in) = a$ și $y \neq x$,
atunci

Regula 8

$$\begin{array}{r} x = a \\ \boxed{y := \ldots} \\ x = a \end{array}$$

Dacă $C(y := \ldots, x, in) = a$ și $y \neq x$,
atunci $C(y := \ldots, x, out) = a$

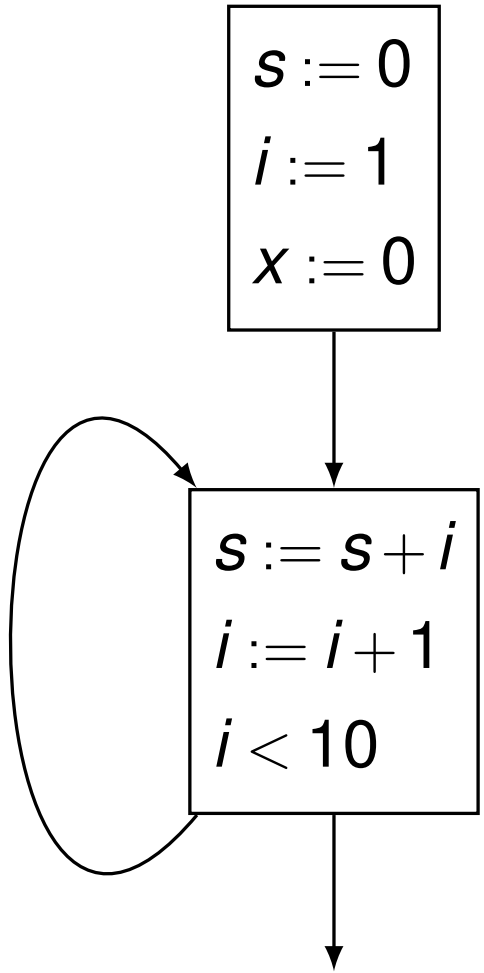
Algoritmul

1. Pentru fiecare instrucțiune **inițială** s, $C(s, x, in) = \top$

2. Pentru fiecare dintre **celelalte** instrucțiuni s, $C(s, x, in) = C(s, x, out) = \bot$

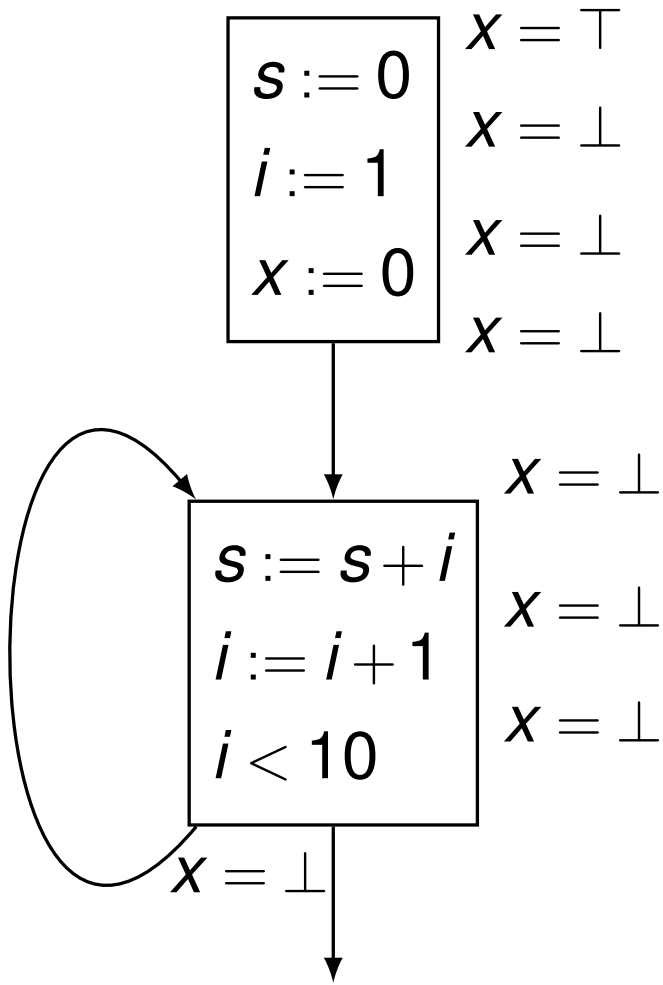
3. Până la absența modificărilor, repetă alegerea unei instrucțiuni care încalcă una dintre regulile 1–8 și aplică acea regulă

Analiza buclelor



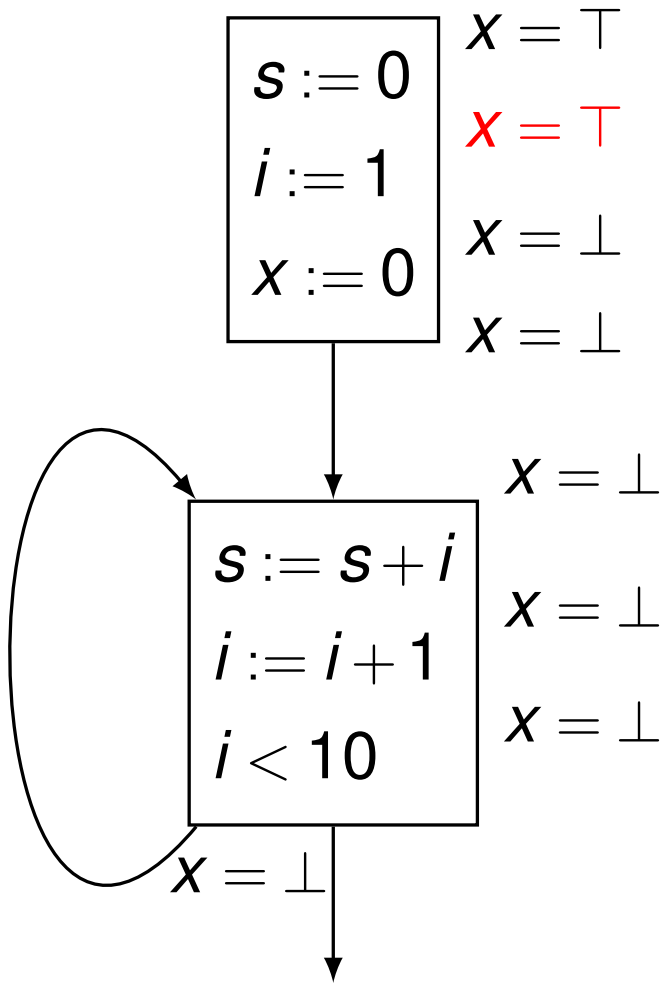
Dependența $C(s := s + i, x, in)$ de $C(i < 10, x, out)$,
și a $C(i < 10, x, in)$ de $C(s := s + i, x, out)$,
într-o manieră ciclică!

Analiza buclelor



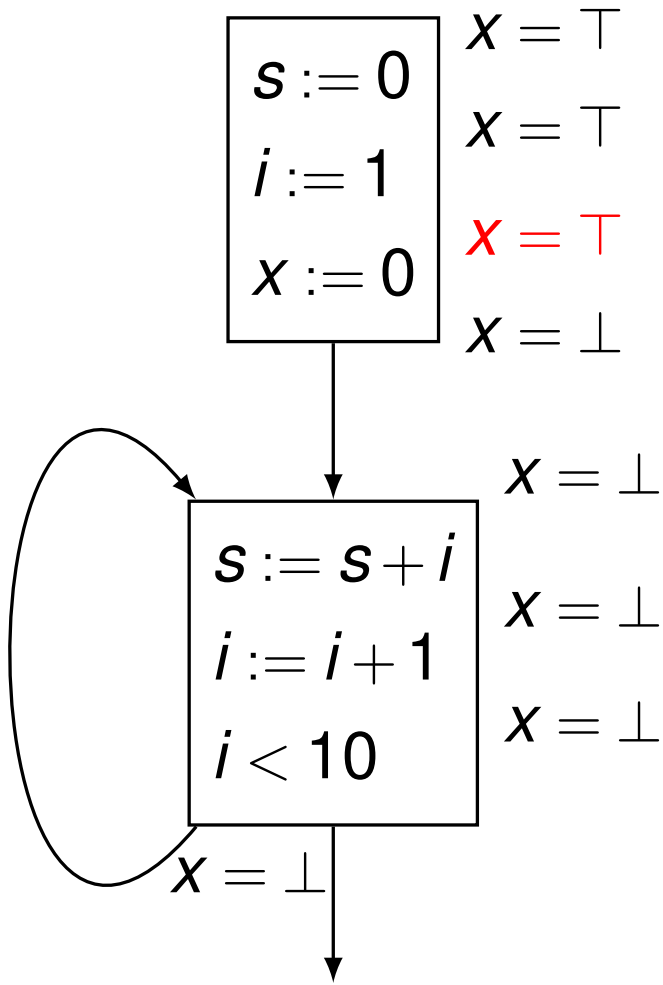
Utilitatea lui $\bot$ pentru ruperea ciclicității!

Analiza buclelor



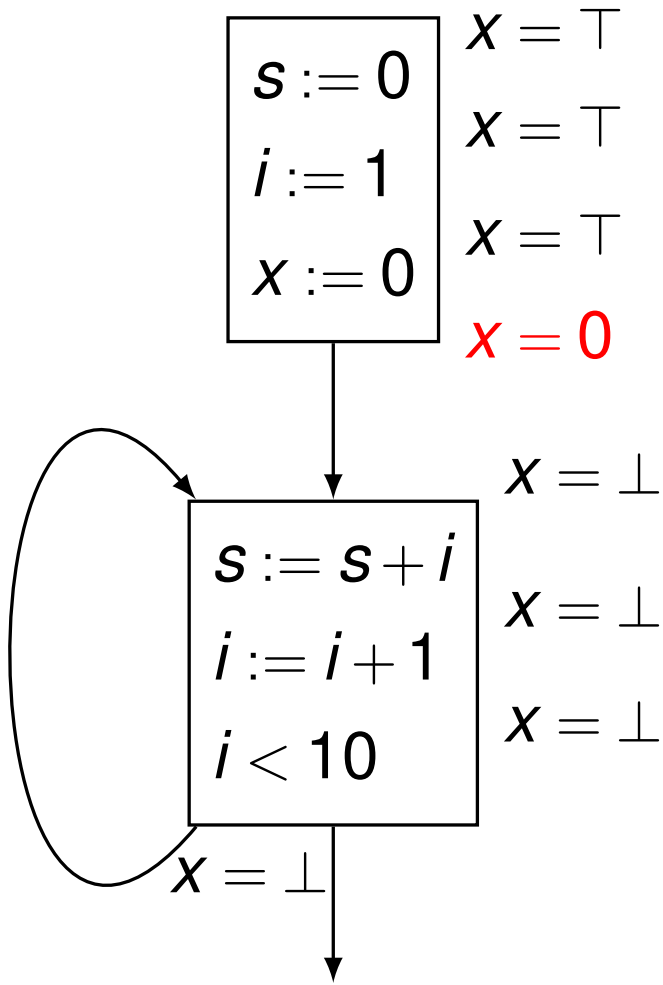
Utilitatea lui $\bot$ pentru ruperea ciclicității!

Analiza buclelor



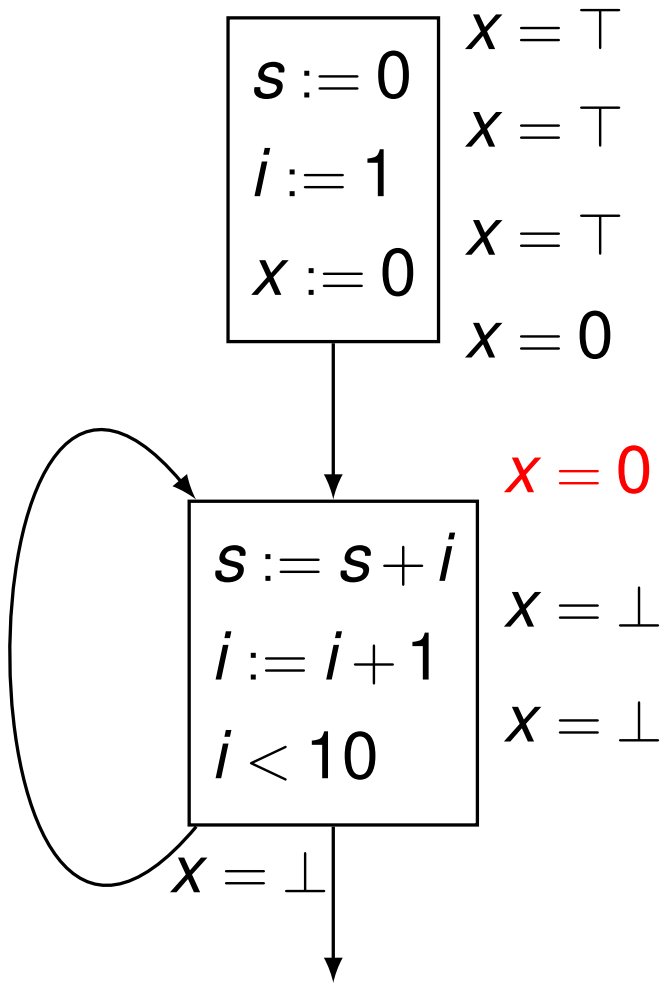
Utilitatea lui $\bot$ pentru ruperea ciclicității!

Analiza buclelor



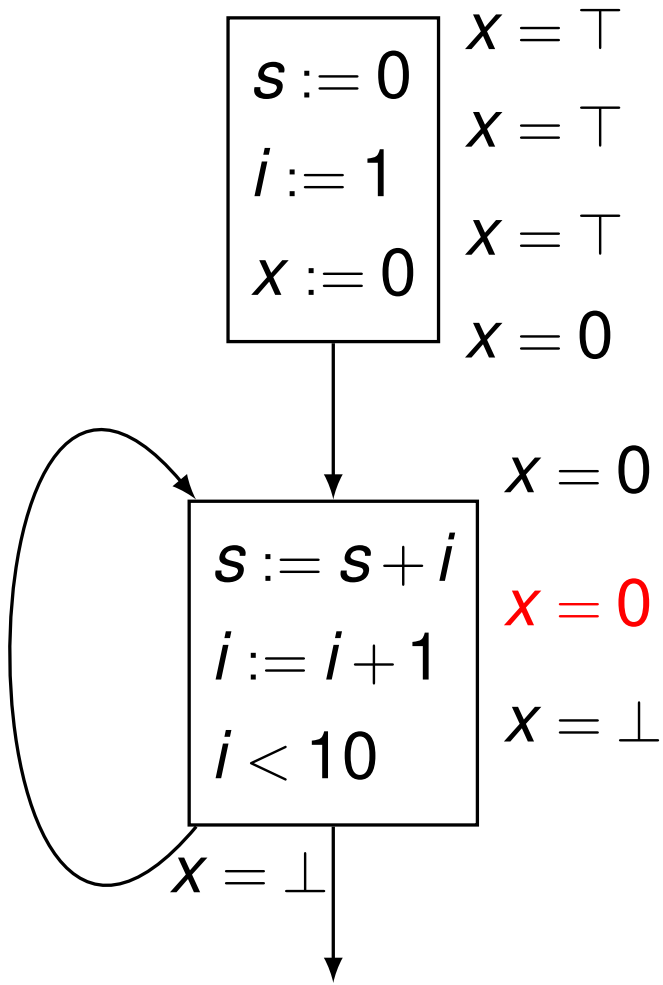
Utilitatea lui $\bot$ pentru ruperea ciclicității!

Analiza buclelor



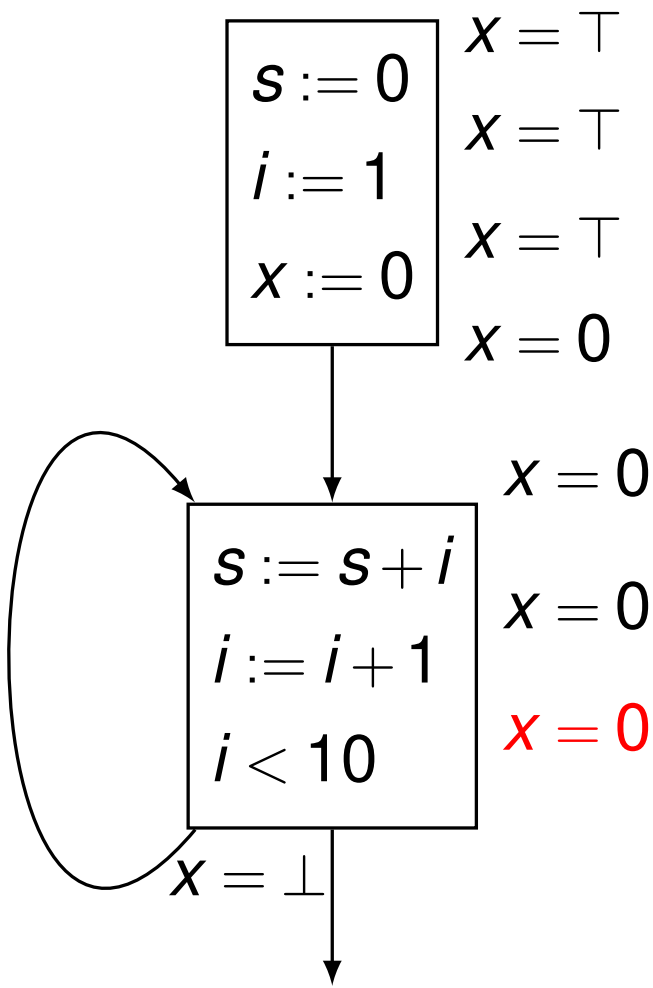
Utilitatea lui $\bot$ pentru ruperea ciclicității!

Analiza buclelor



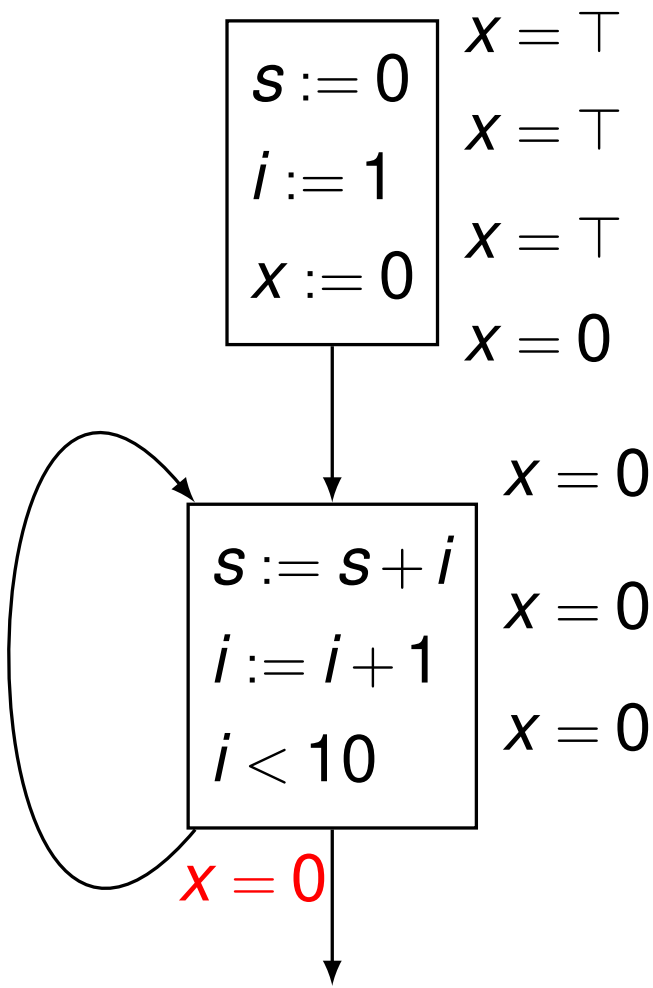
Utilitatea lui $\bot$ pentru ruperea ciclicității!

Analiza buclelor



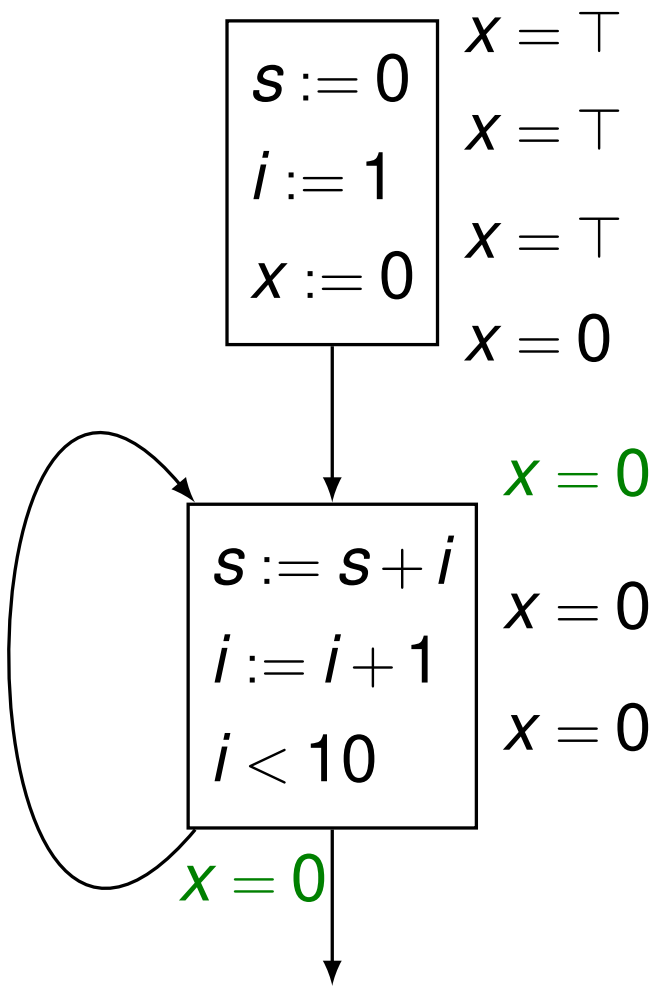
Utilitatea lui $\bot$ pentru ruperea ciclicității!

Analiza buclelor



Utilitatea lui $\perp$ pentru ruperea ciclicității!

Analiza buclelor



Utilitatea lui $\perp$ pentru ruperea ciclicității!

Ordonarea valorilor

- Valori **abstracte** utilizate: $\bot$, c, $\top$
- Ordonarea lor pe baza „cantității de **informație**”:

$$\bot < c < \top$$

$$c_1 \neq c_2 \;\Rightarrow\; c_1 \not< c_2 \wedge c_2 \not< c_1$$

(aceeași „cantitate de informație” în toate constantele)

- Regulile 1–4, rescrise folosind ***least upper bound***:

$C(s, x, in) = lub\ \{C(p, x, out) \mid p \text{ este predecesor al lui } s\}$

Terminarea algoritmului

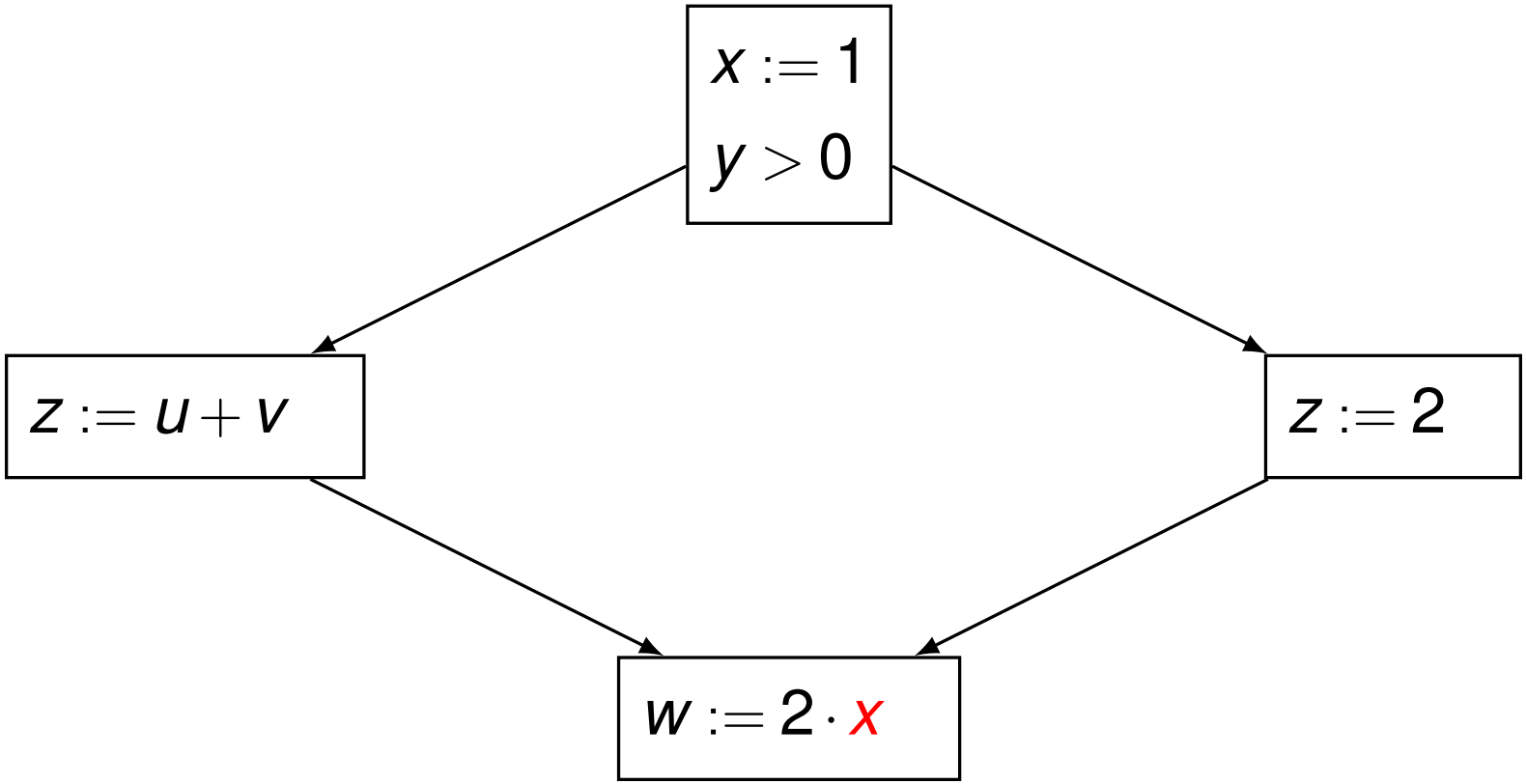
- Într-un pas, menținerea sau promovarea unei valori abstracte (niciodată retrogradarea)
- Numărul maxim de promovări ale unei valori: două
- În consecință, număr finit de pași

Complexitatea algoritmului

- Numărul de pași =
  Numărul de valori $C(\ldots)$ calculate $\times$ 2 =
  (maxim 2 promovări)
  Numărul de instrucțiuni $\times$ 4 (2 puncte, *in* și *out*)

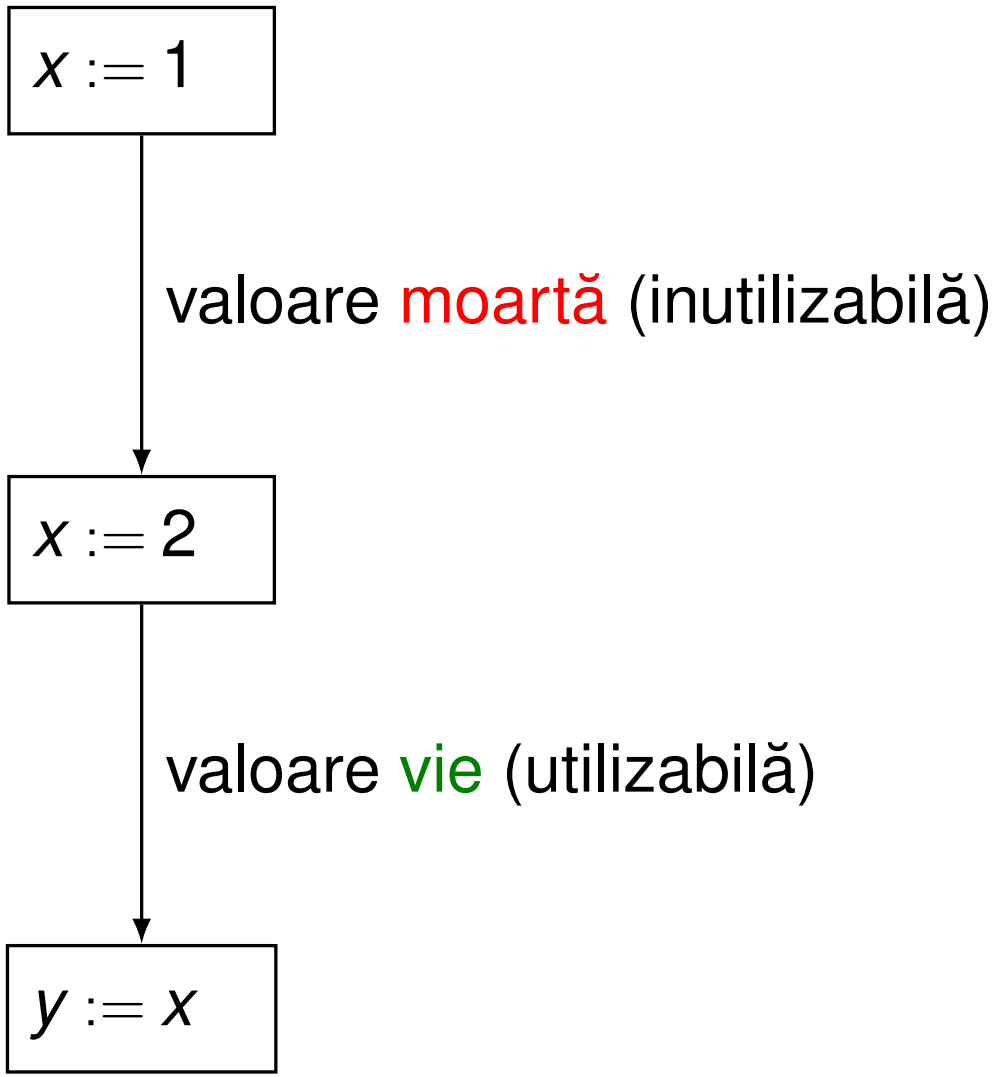
- Liniară în numărul de instrucțiuni

Analiza duratei de viață a valorilor



Odată ce am aplicat propagarea constantelor pentru x, devine instrucțiunea $x := 1$ cod mort?

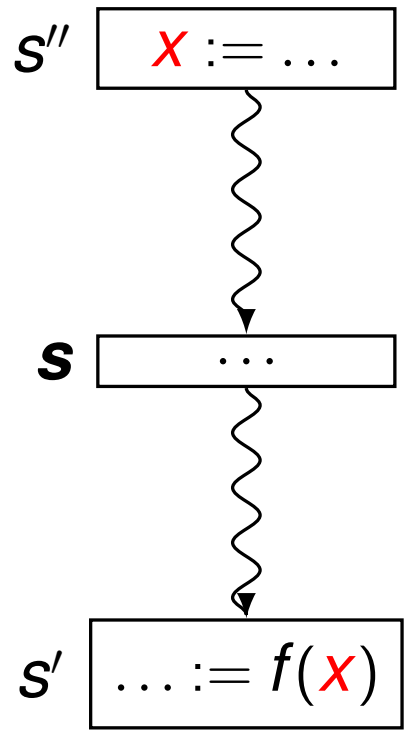
Intuiție


$x := 1$
valoare moartă (inutilizabilă)
$x := 2$
valoare vie (utilizabilă)
$y := x$

Durata de viață a valorilor

Variabila x, **vie** la instrucțiunea s, în condițiile:

- Existenței unei instrucțiuni s' care citește x
- Existenței unei căi de la s la s'
- Absenței altor atribuiri ale lui x pe acea cale



Eliminarea globală a codului mort

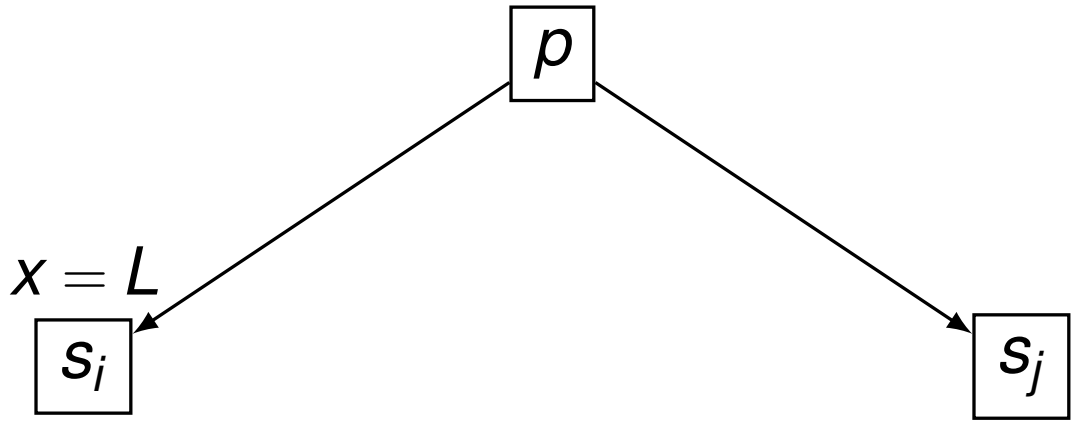
- Instrucțiunea $x := \ldots$, **moartă** dacă x este moartă după atribuire

- Posibilitatea eliminării instrucțiunilor moarte

Determinarea duratei de viață a valorilor

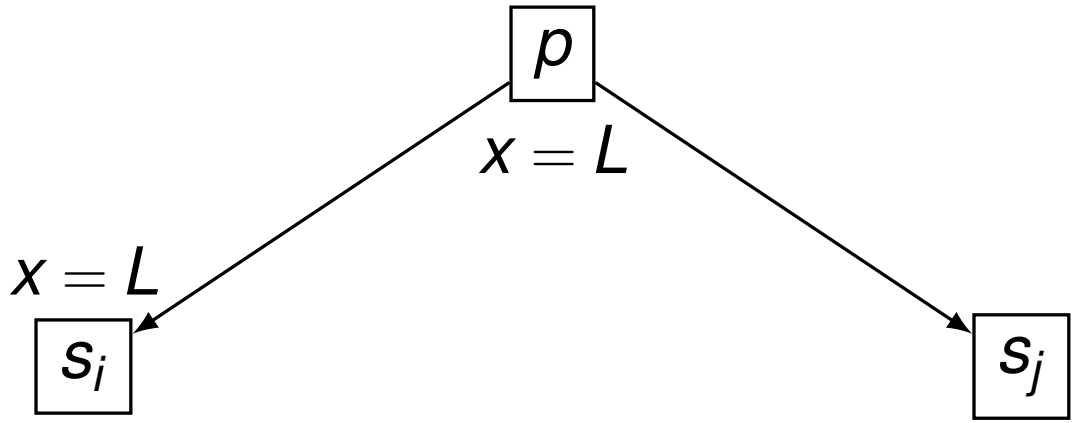
- Cum determinăm statutul unei variabile (vie sau moartă), în raport cu o intrucțiune?
- Numirea statutului lui x ($\underline{Live}/\underline{Dead}$) înainte și după instrucțiunea s (v. propagarea constantelor): $L(s, x, in)$, $L(s, x, out)$
- Specificarea modului în care starea este transferată inter- și intrainstrucțiune

Regula 1



Dacă $L(s_i, x, in) = L$,
atunci

Regula 1



Dacă $L(s_i, x, in) = L$,
atunci $L(p, x, out) = L$

Regula 2

$$\boxed{\ldots := f(x)}$$

Regula 2

$$\frac{x = L}{\boxed{\ldots := f(x)}}$$

$$L(\ldots := f(x), x, in) = L$$

Regula 3

$$\boxed{x := \ldots}$$

Dacă s nu referă x în partea dreaptă,
atunci

Regula 3

$$\frac{}{x := \ldots}\ x = D$$

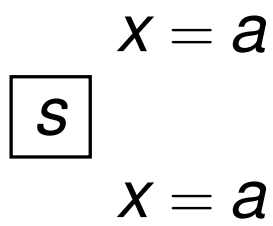
Dacă s nu referă x în partea dreaptă,
atunci $L(x := \ldots, x, in) = D$

Regula 4

$$\boxed{s}\quad x = a$$

Dacă s nu referă x,
atunci

Regula 4



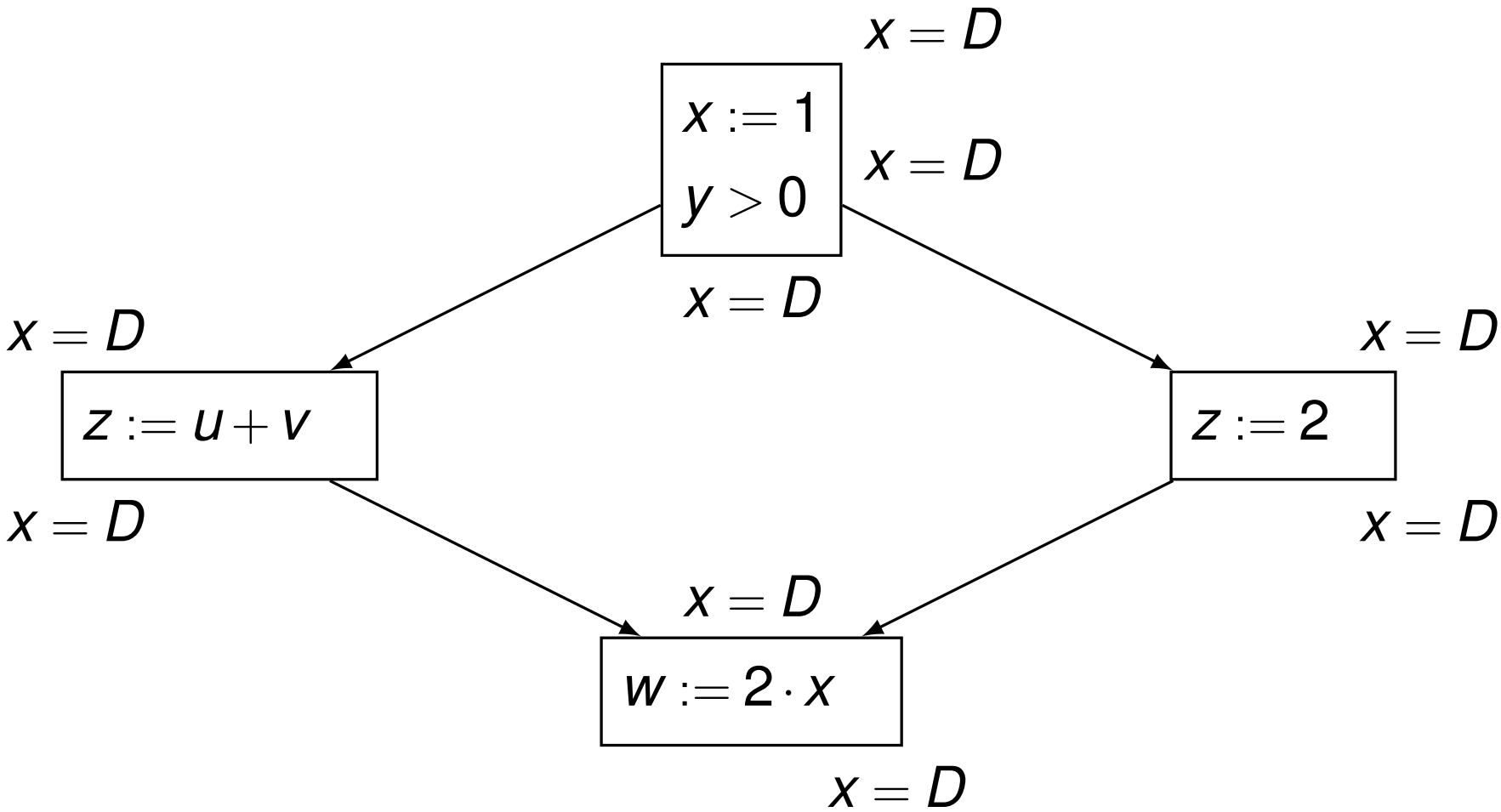
Dacă s nu referă x,
atunci $L(s, x, in) = L(s, x, out)$

Algoritmul

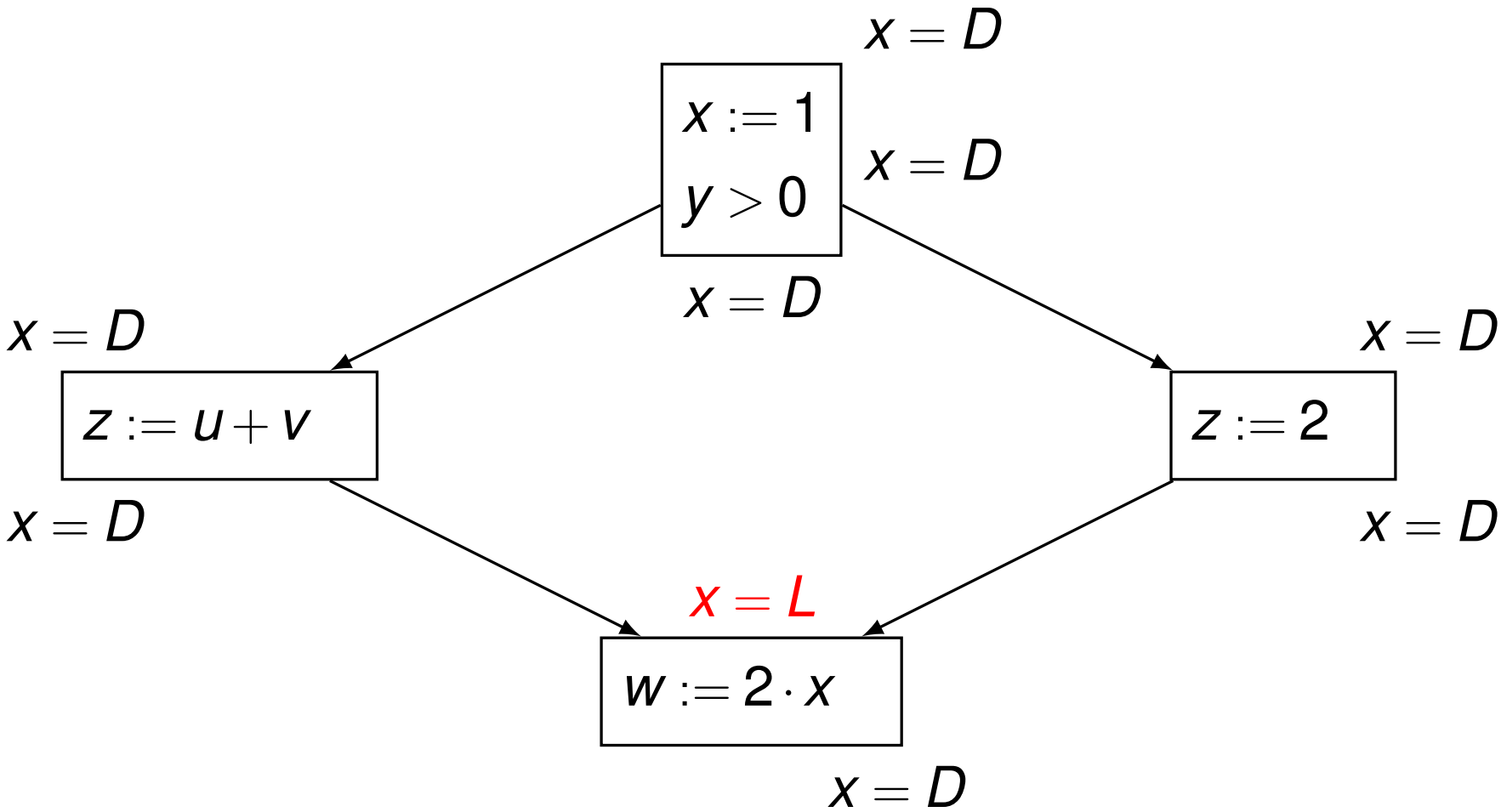
1. Inițial, $L(\ldots) = D$

2. Până la absența modificărilor, repetă alegerea unei instrucțiuni care încalcă una dintre regulile 1–4 și aplică acea regulă

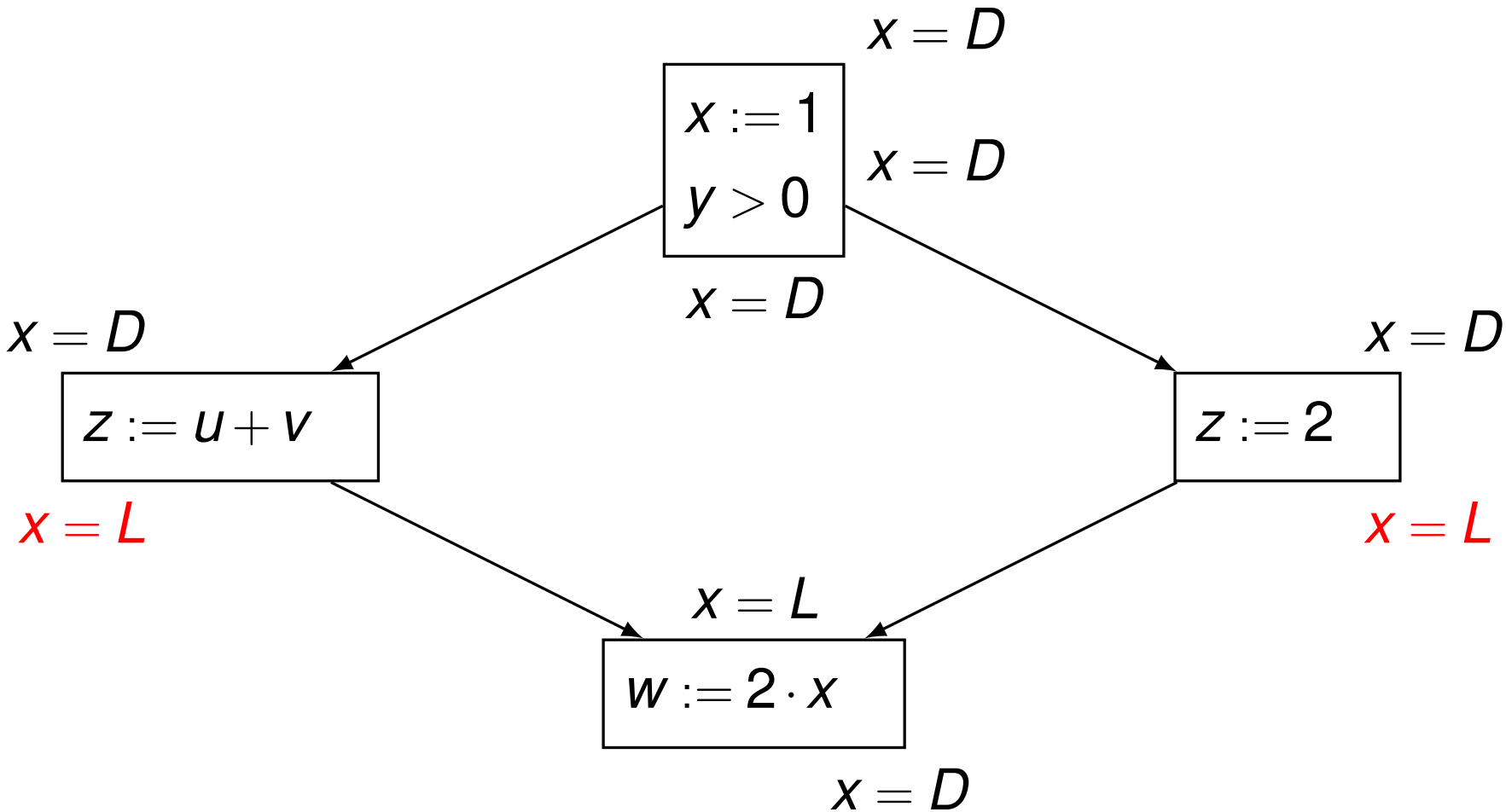
Exemplu reluat



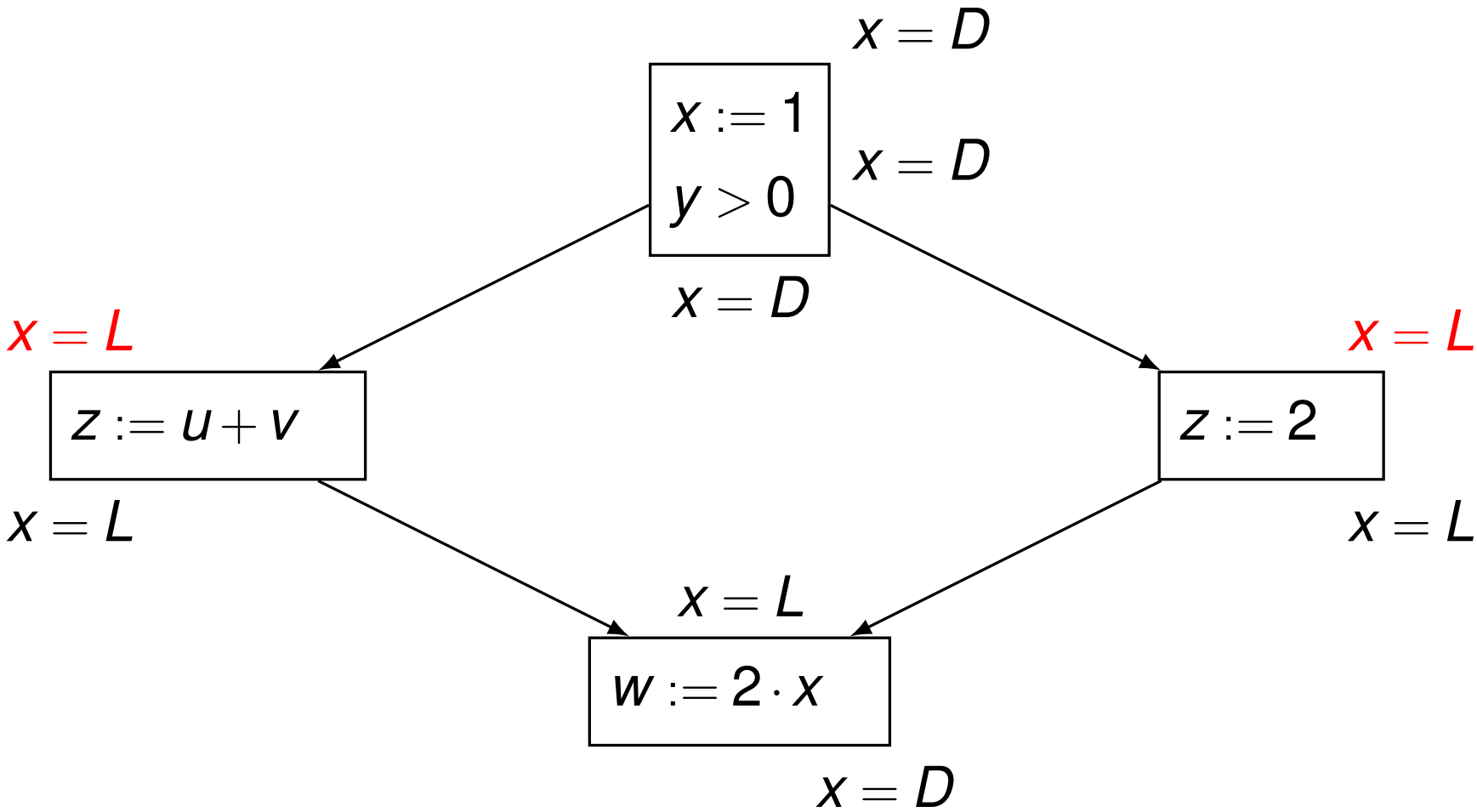
Exemplu reluat



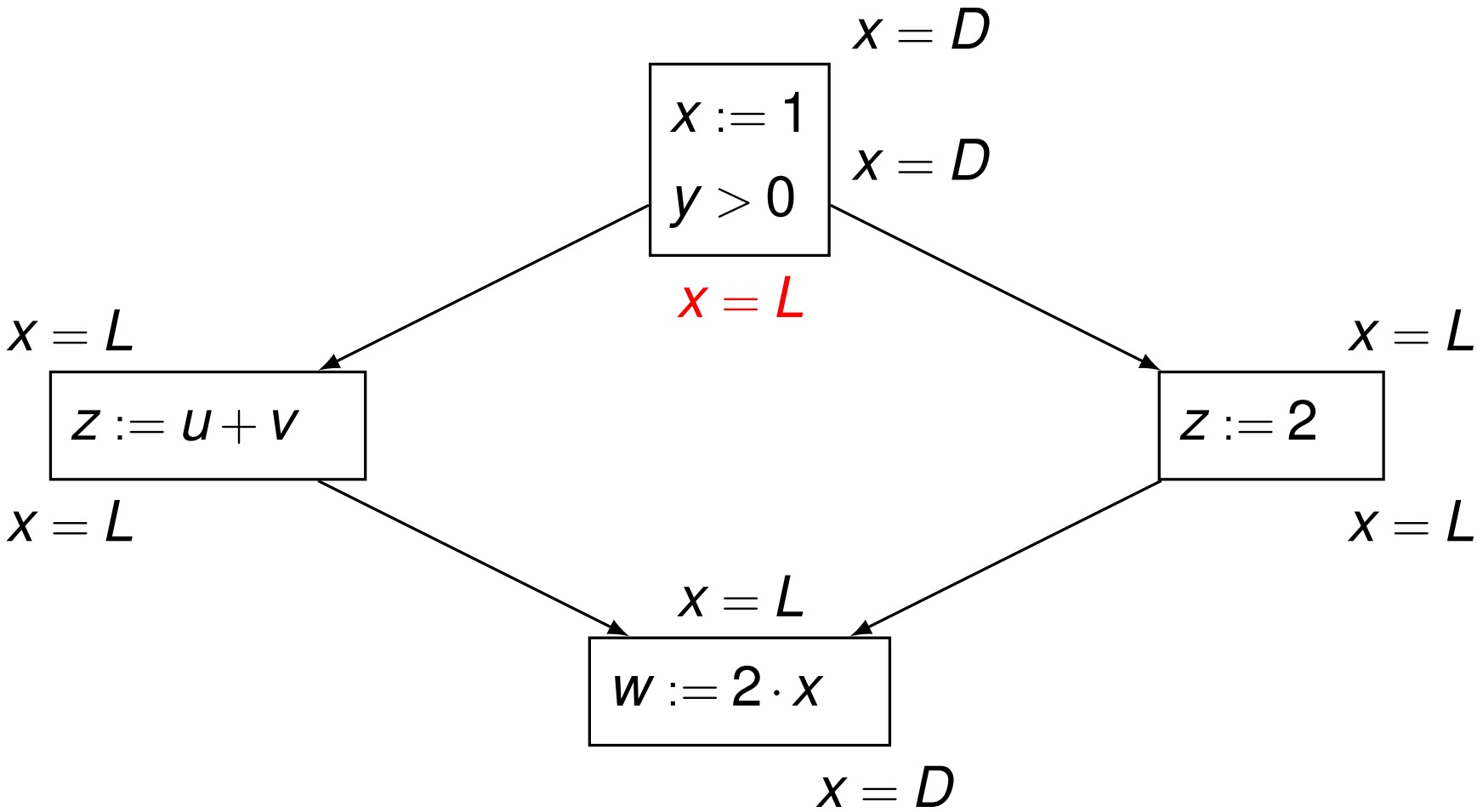
Exemplu reluat



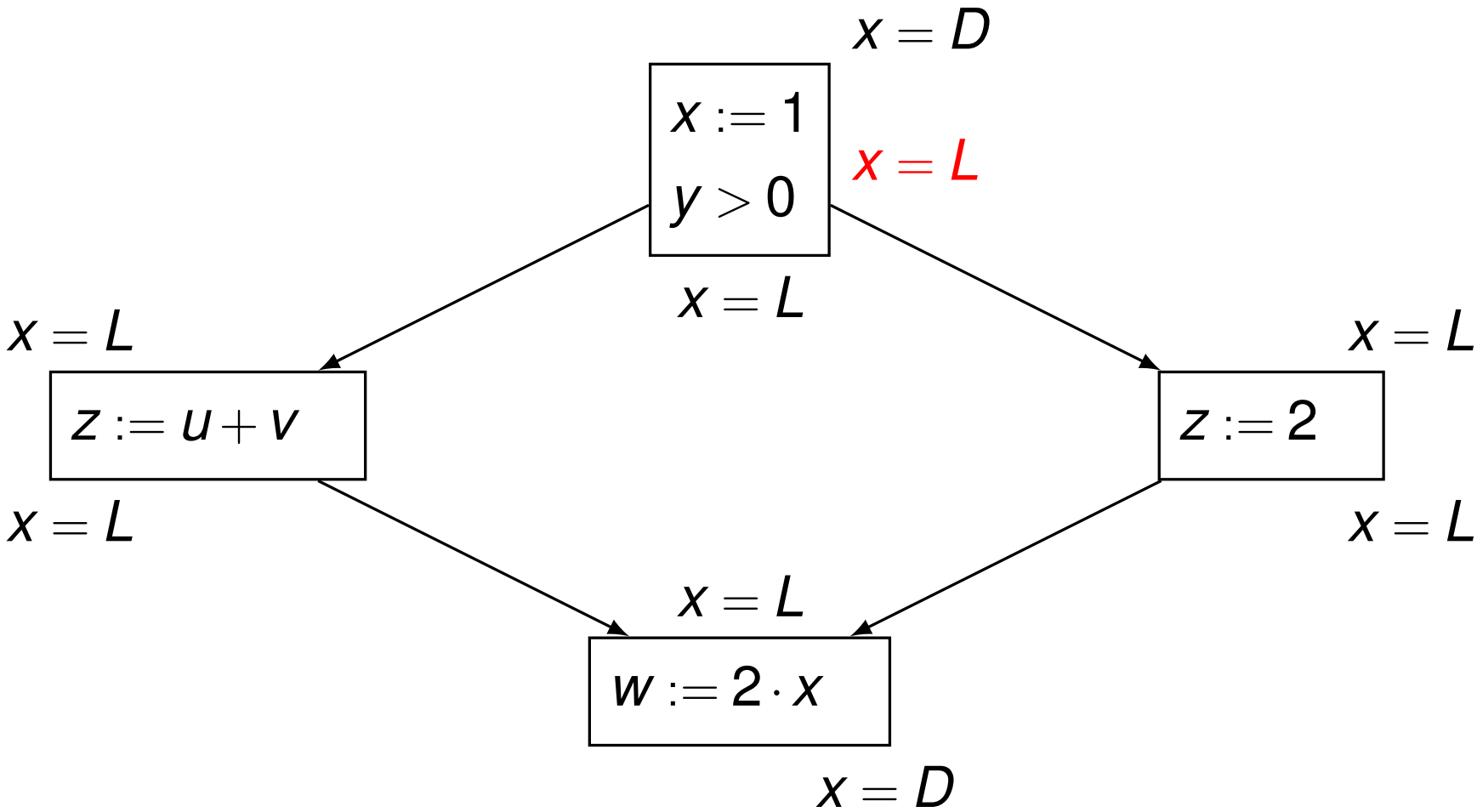
Exemplu reluat



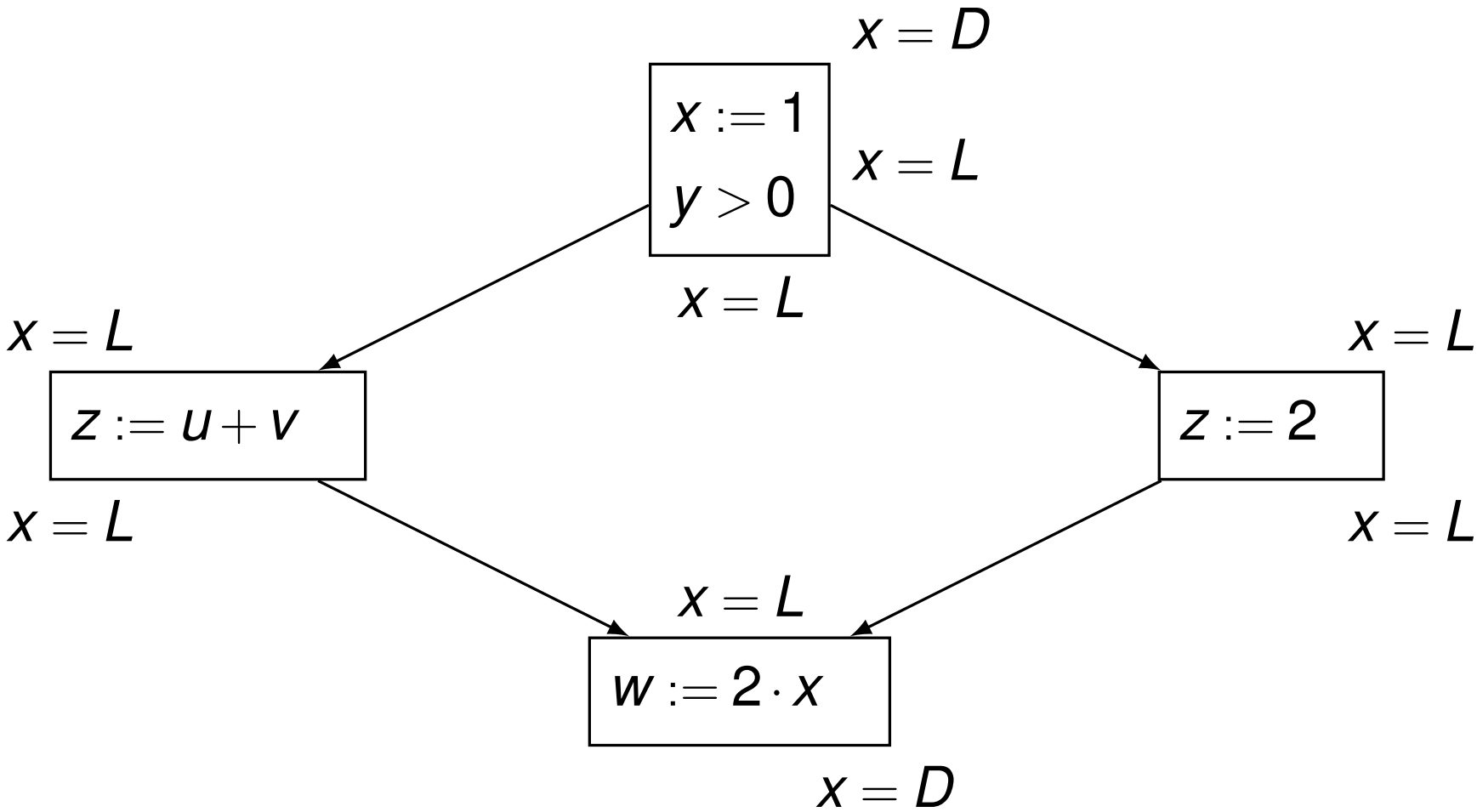
Exemplu reluat



Exemplu reluat



Exemplu reluat



Terminarea algoritmului

- Într-un pas, menținerea stării sau modificarea ei de la *D* la *L* (niciodată invers)
- Numărul maxim de modificări ale unui statut: una
- În consecință, număr finit de pași

Analiză *înainte* și *înapoi*

- Propagarea constantelor: analiză ***înainte***
  (cu transferul informației de la intrări la ieșiri)

- Durata de viață a valorilor: analiză ***înapoi***
  (cu transferul informației de la ieșiri la intrări)

Cuprins

Accesul la date

- Importanța gestiunii corespunzătoare a registrelor și a memoriei *cache* în scopul creșterii vitezei de acces la date
- Ideal, datele accesate frecvent, depuse în registre
- **Problemă**: număr infinit de registre temporare în codul intermediar, dar număr limitat pe arhitecturile fizice
- **Soluție**: alocarea mai multor registre temporare în același registru fizic, cu evitarea interferențelor

Exemplu

$$a := b + c$$
$$d := a + 1$$
$$e := d \cdot 2$$

Exemplu

$$a := b + c$$
$$d := a + 1$$
$$e := d \cdot 2$$

$$r_1 := r_2 + r_3$$
$$r_1 := r_1 + 1$$
$$r_1 := r_1 \cdot 2$$

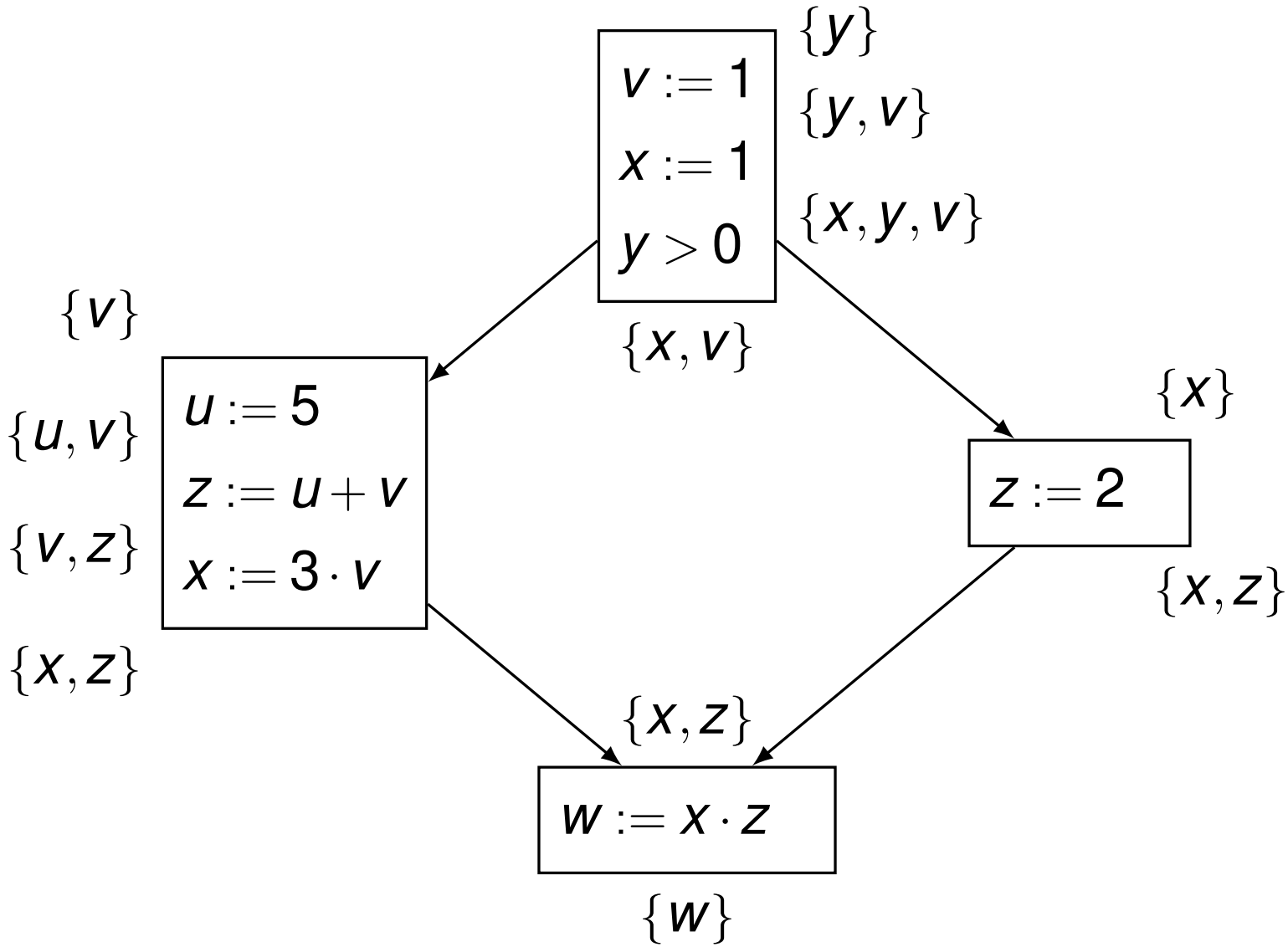
Posibilitatea alocării registrelor temporare a, d și e în același registru fizic, r_1, presupunând că a și d sunt *dead* după bloc.

Intuiția

- Posibilitatea ca registrele temporare t_1 și t_2 să împartă același registru fizic dacă și numai dacă, în orice punct din program, cel mult unul dintre ei este *live*

- Alternativ, imposibilitatea ca două registre temporare *live* în același timp să utilizeze același registru fizic

Exemplu

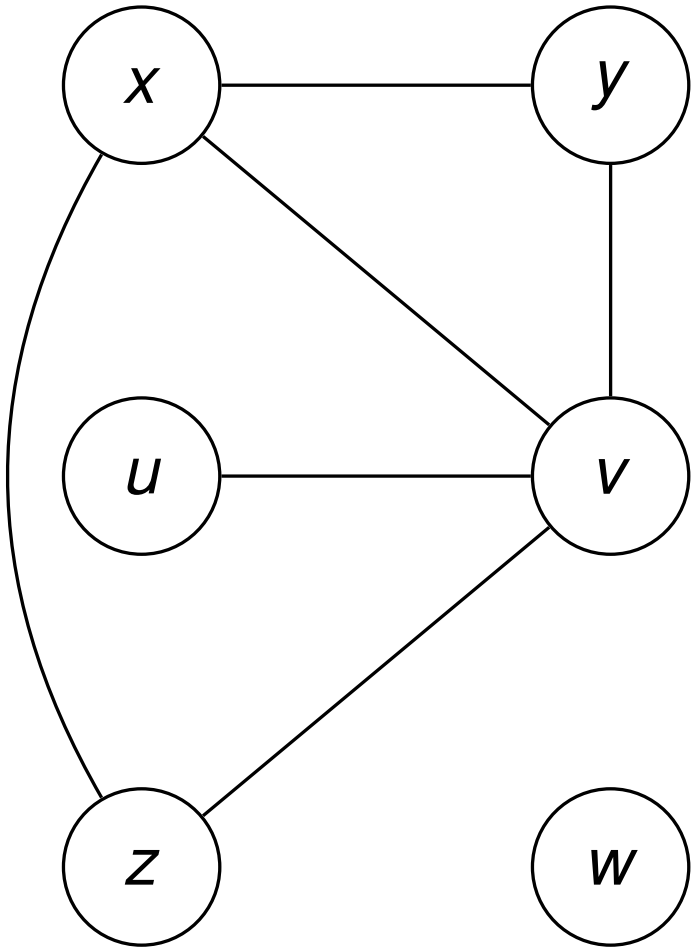


Mulțimile variabilelor *live* în fiecare punct din program

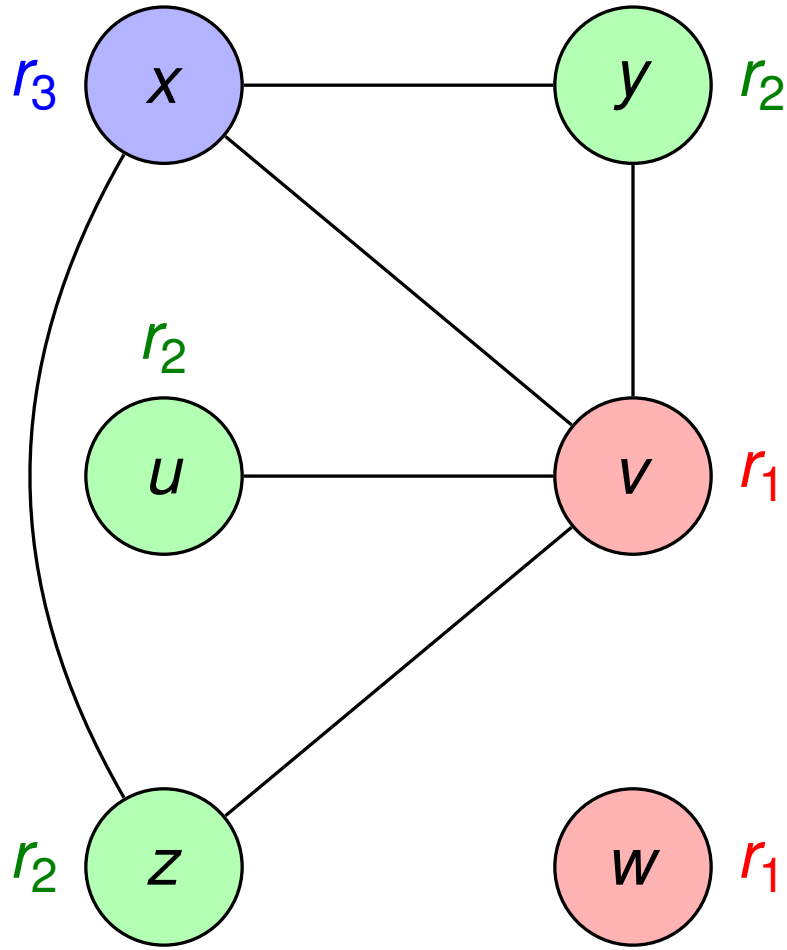
Graful de interferență a registrelor (RIG)

- **Noduri**: registrele temporare
- **Arce**: prezente dacă registrele temporare aferente sunt *live* în același timp
- **Proprietate**: RIG este *k*-colorabil dacă și numai dacă există o alocare cu cel mult *k* registre fizice

Exemplu reluat



Exemplu reluat



Problema colorării

- Problemă NP-completă — în practică, utilizare de euristici
- Posibilitatea absenței unei soluții — necesitatea salvării registrelor temporare în memorie

Observație despre k-colorare

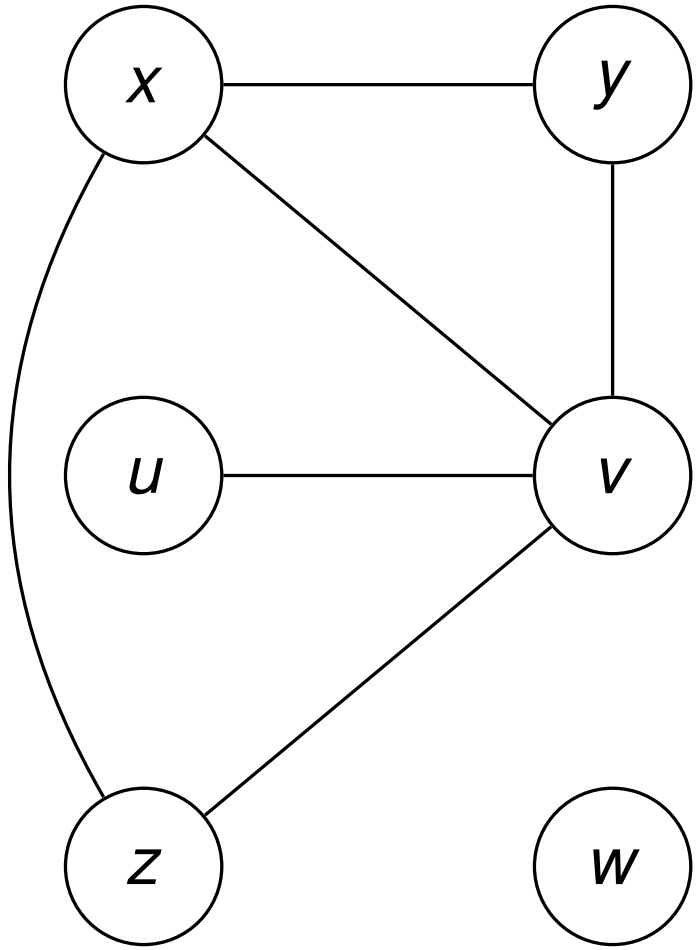
- Alegere nod t cu n vecini, $n < k$
- Eliminare t din graf, împreună cu arcele sale
- Graf rezultant k-colorabil dacă și numai dacă graful original este k-colorabil
- Explicație: cum $n < k$, posibilitatea găsirii unei noi culori pentru t, diferită de cele ale vecinilor

Euristică pentru k-colorare

1. **Ordonare** noduri
   1.1 Alegere nod t cu n vecini, $n < k$
   1.2 Eliminare t din graf și adăugarea lui într-o stivă
   1.3 Repetare până când rămâne un singur nod

2. **Colorare** noduri
   2.1 Extragere nod din vârful stivei
   2.2 Asocierea unei culori diferite de cele ale vecinilor deja colorați
   2.3 Repetare până la golirea stivei

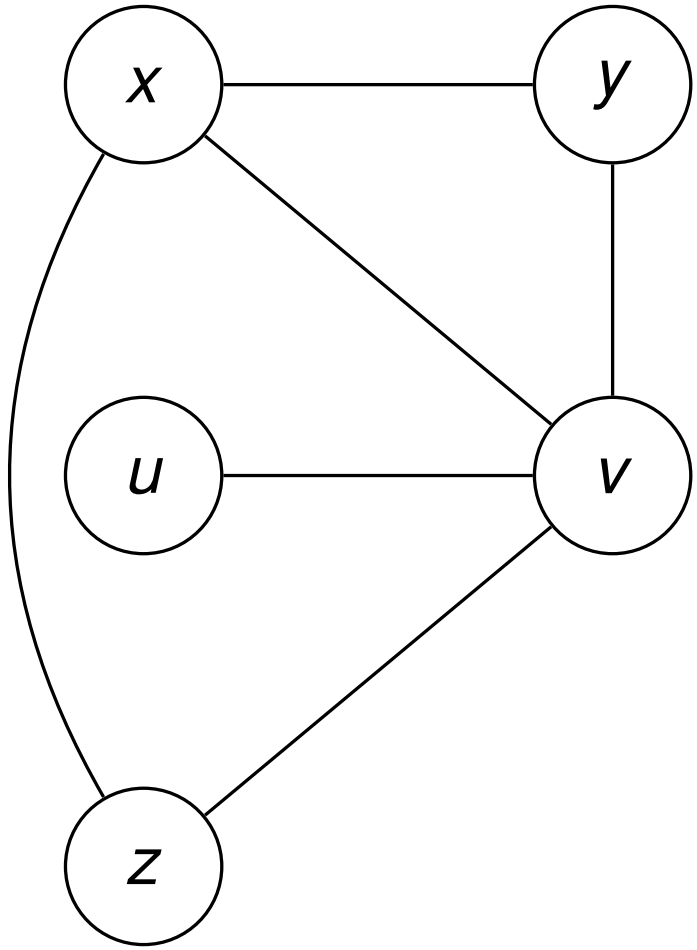
Exemplu reluat ($k = 3$)



Stiva:

{}

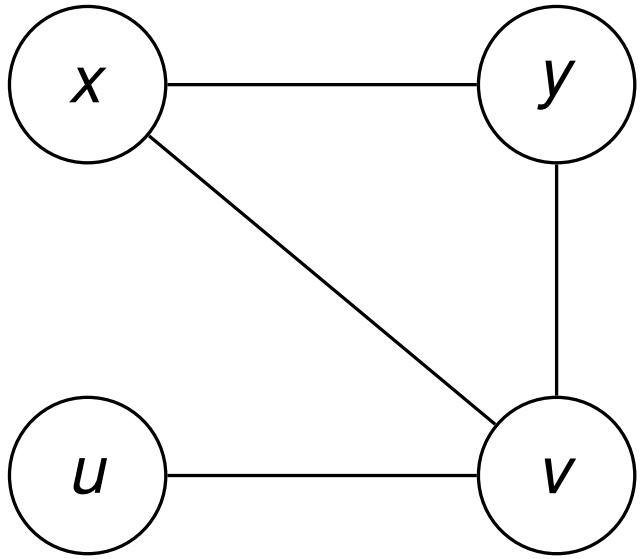
Exemplu reluat ($k = 3$)



Stiva:

$\{w\}$

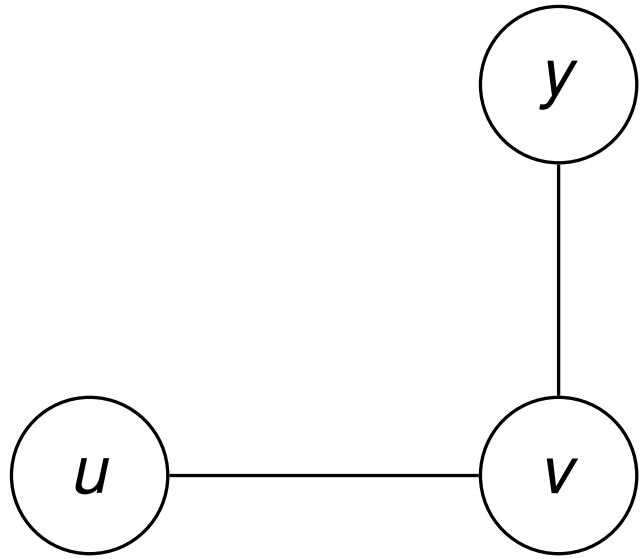
Exemplu reluat ($k = 3$)



Stiva:

$\{z, w\}$

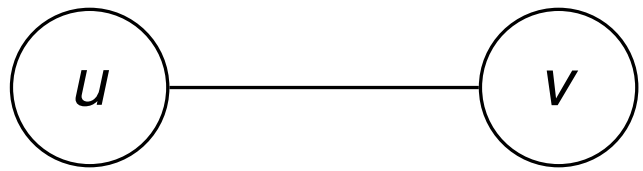
Exemplu reluat ($k = 3$)



Stiva:

$\{x, z, w\}$

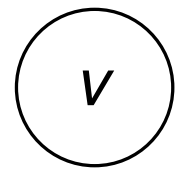
Exemplu reluat ($k = 3$)



Stiva:

$\{y, x, z, w\}$

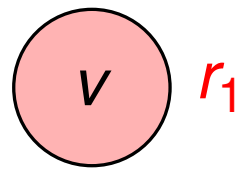
Exemplu reluat ($k = 3$)



Stiva:

$\{u, y, x, z, w\}$

Exemplu reluat ($k = 3$)



Stiva:

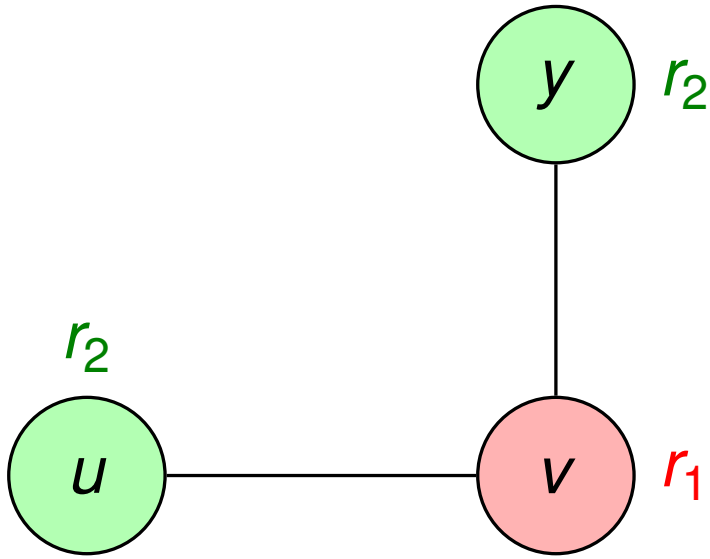
$\{u, y, x, z, w\}$

Exemplu reluat ($k = 3$)

Stiva:

$\{y, x, z, w\}$

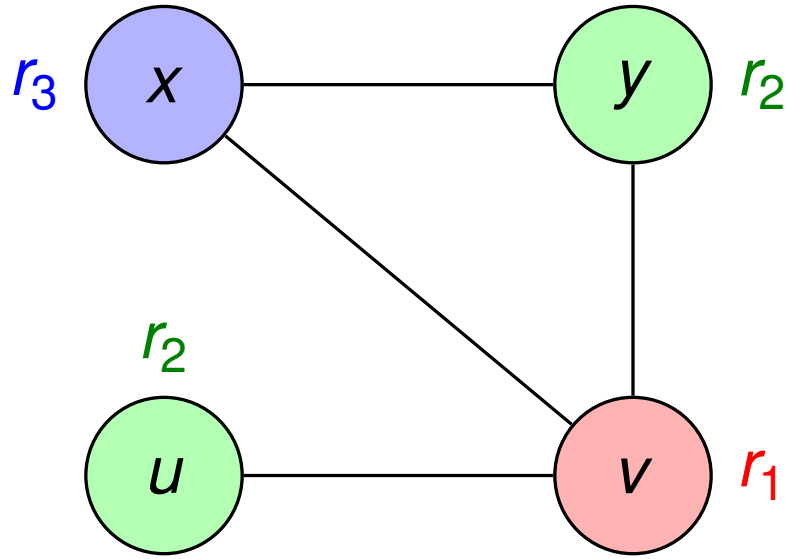
Exemplu reluat ($k = 3$)



Stiva:

$\{x, z, w\}$

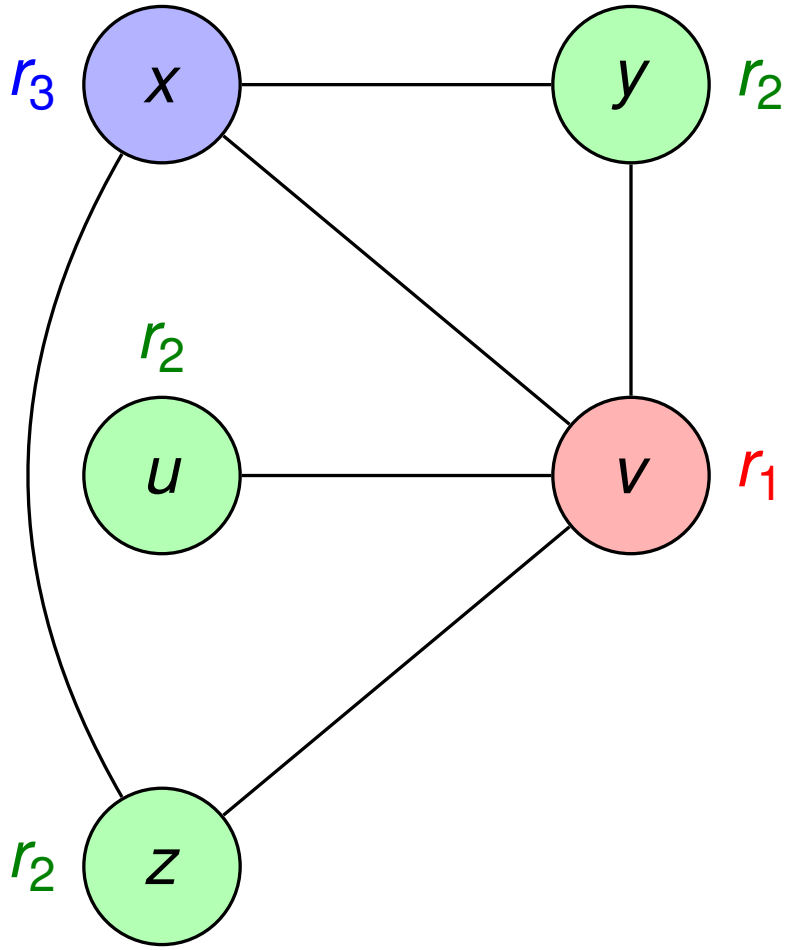
Exemplu reluat ($k = 3$)



Stiva:

$\{z, w\}$

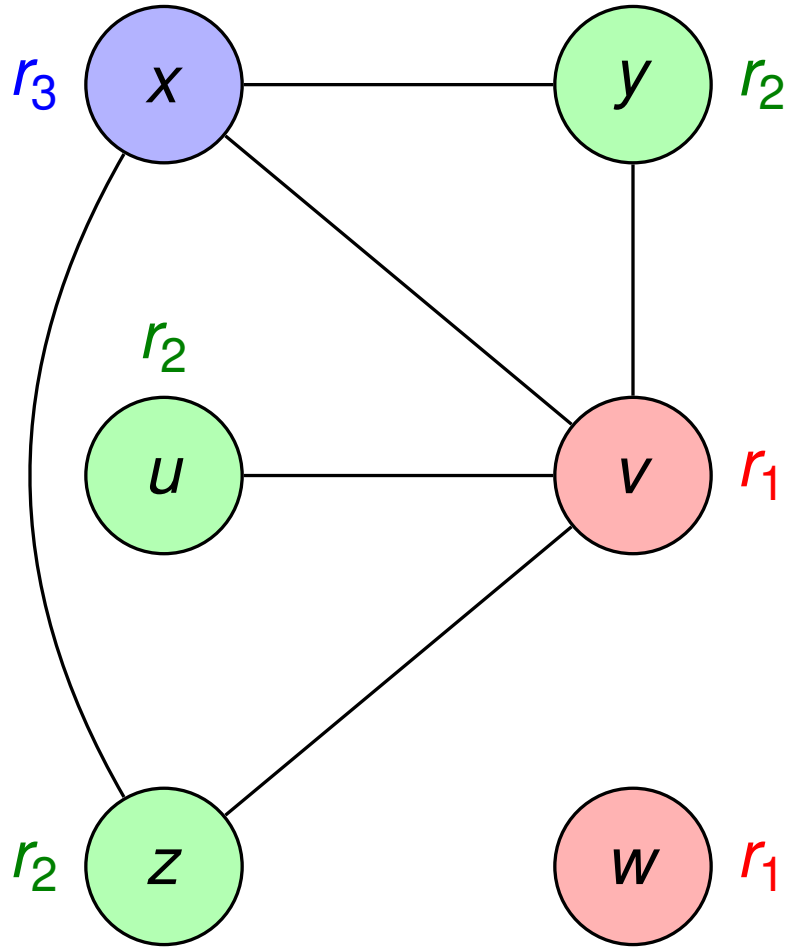
Exemplu reluat ($k = 3$)



Stiva:

$\{w\}$

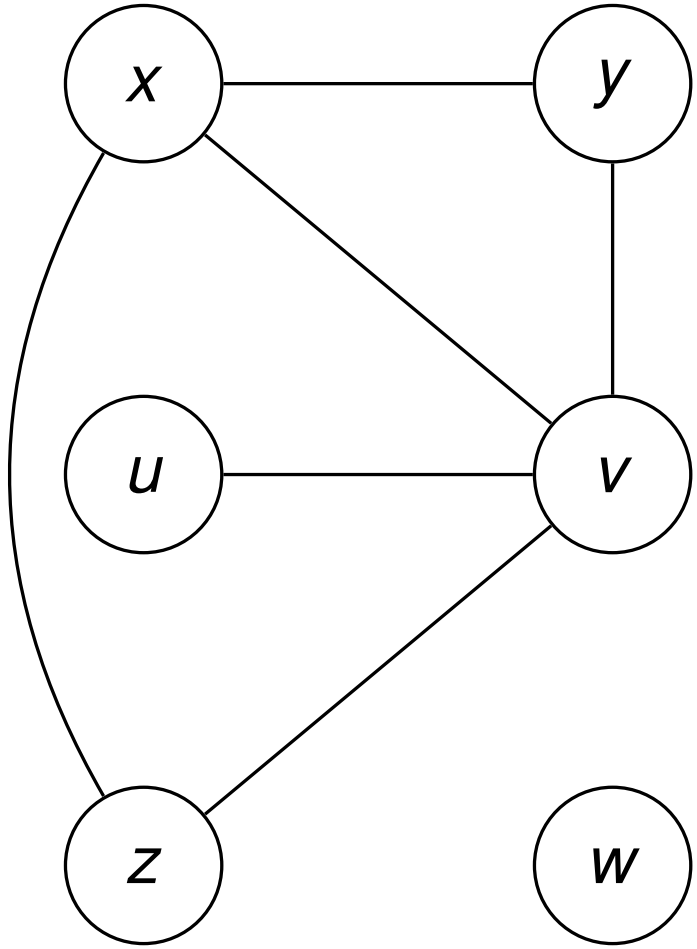
Exemplu reluat ($k = 3$)



Stiva:

{}

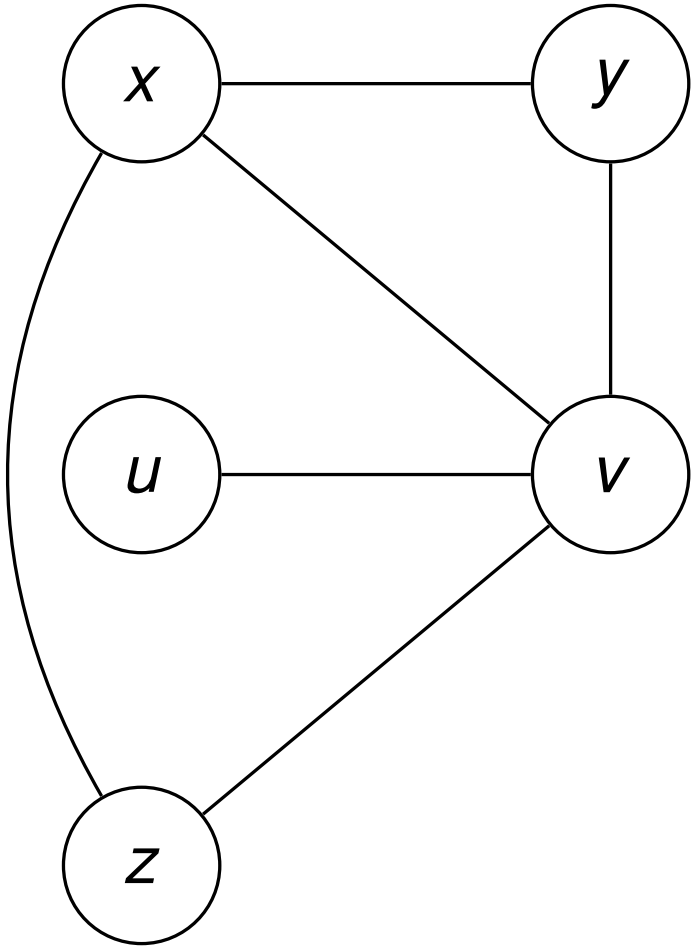
Exemplu reluat ($k = 2$)



Stiva:

{}

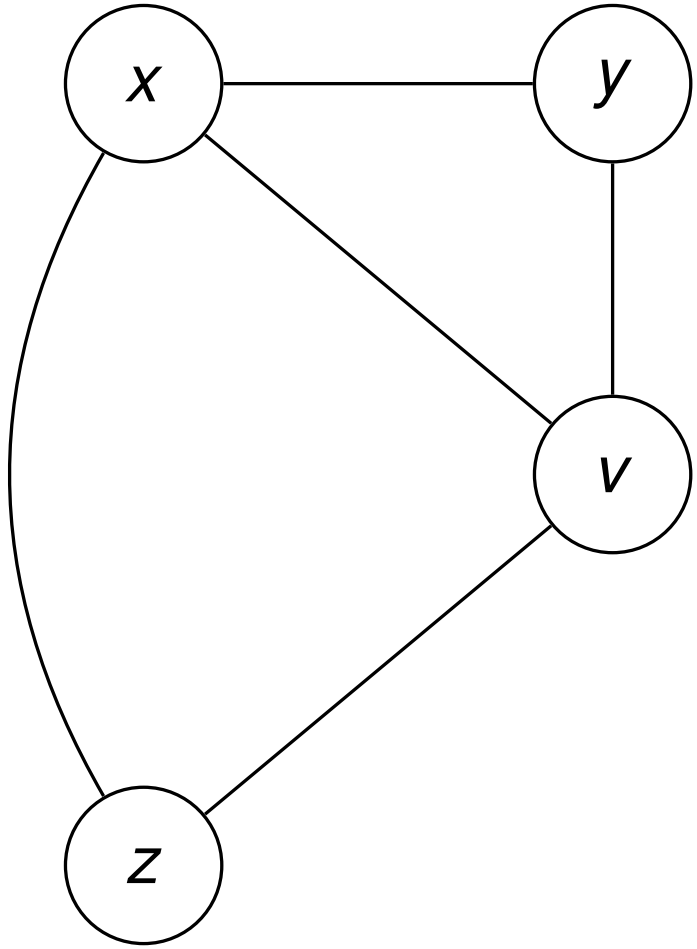
Exemplu reluat ($k = 2$)



Stiva:

$\{w\}$

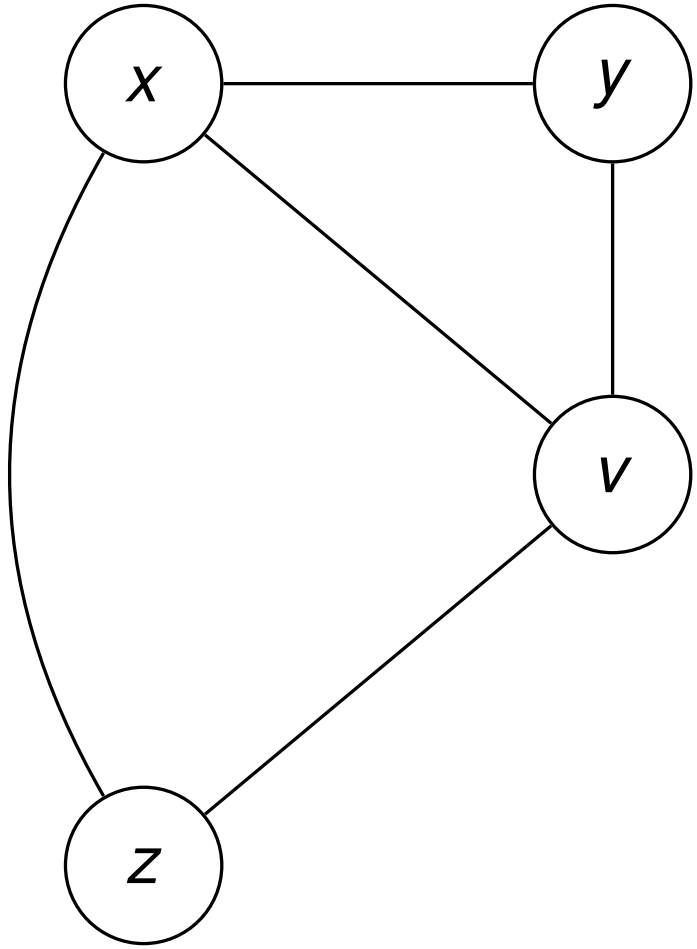
Exemplu reluat ($k = 2$)



Stiva:

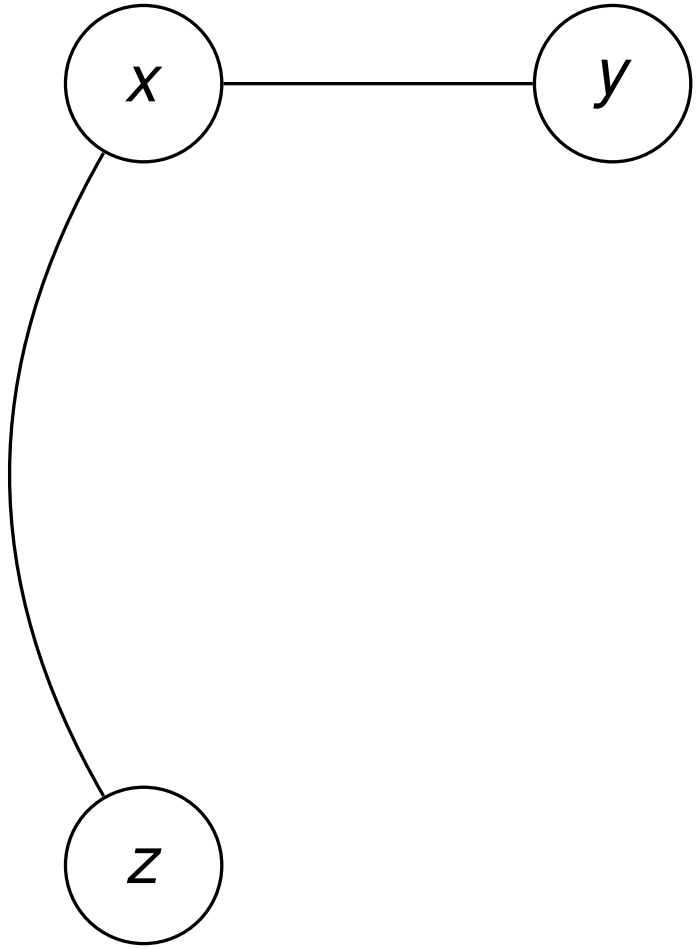
$\{u, w\}$

Exemplu reluat ($k = 2$)



Imposibilitatea alegerii unui nod cu grad ≤ 1!
Soluție: alegerea lui v drept candidat
pentru „vărsarea” în memorie (*spilling*)

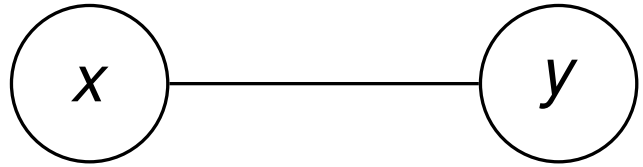
Exemplu reluat ($k = 2$)



Stiva:

$\{v, u, w\}$

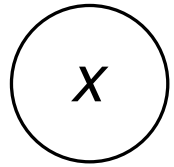
Exemplu reluat ($k = 2$)



Stiva:

$\{z, v, u, w\}$

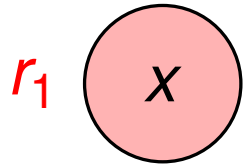
Exemplu reluat ($k = 2$)



Stiva:

$\{y, z, v, u, w\}$

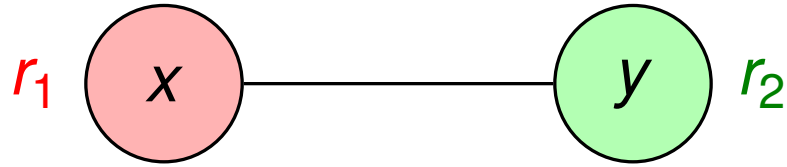
Exemplu reluat ($k = 2$)



Stiva:

$\{y, z, v, u, w\}$

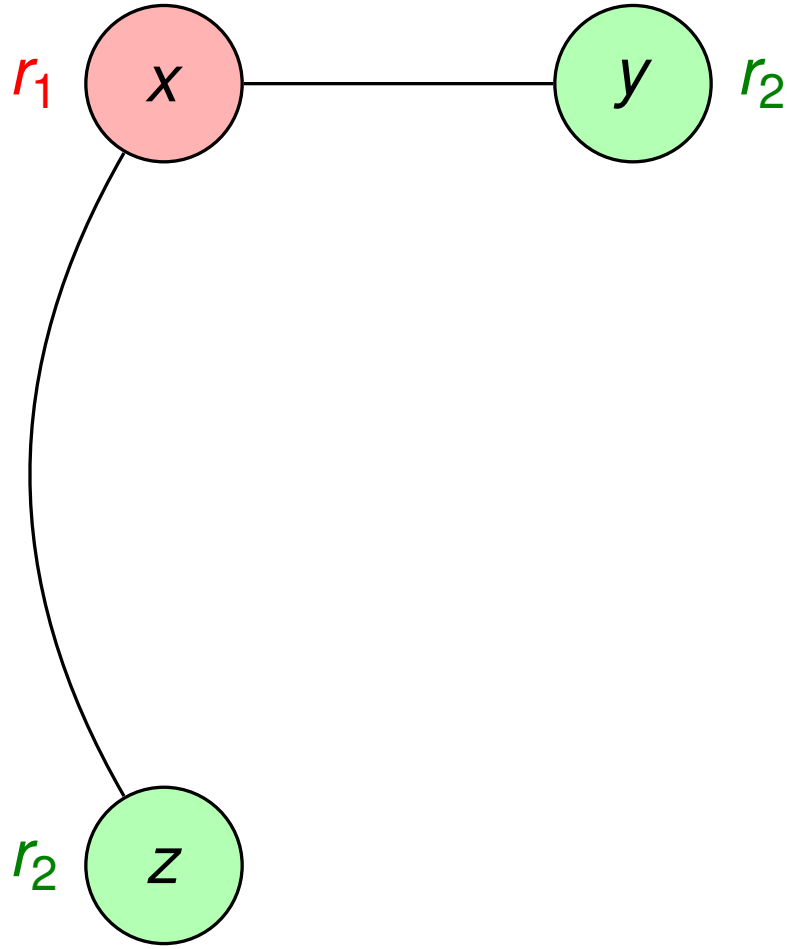
Exemplu reluat ($k = 2$)



Stiva:

$\{z, v, u, w\}$

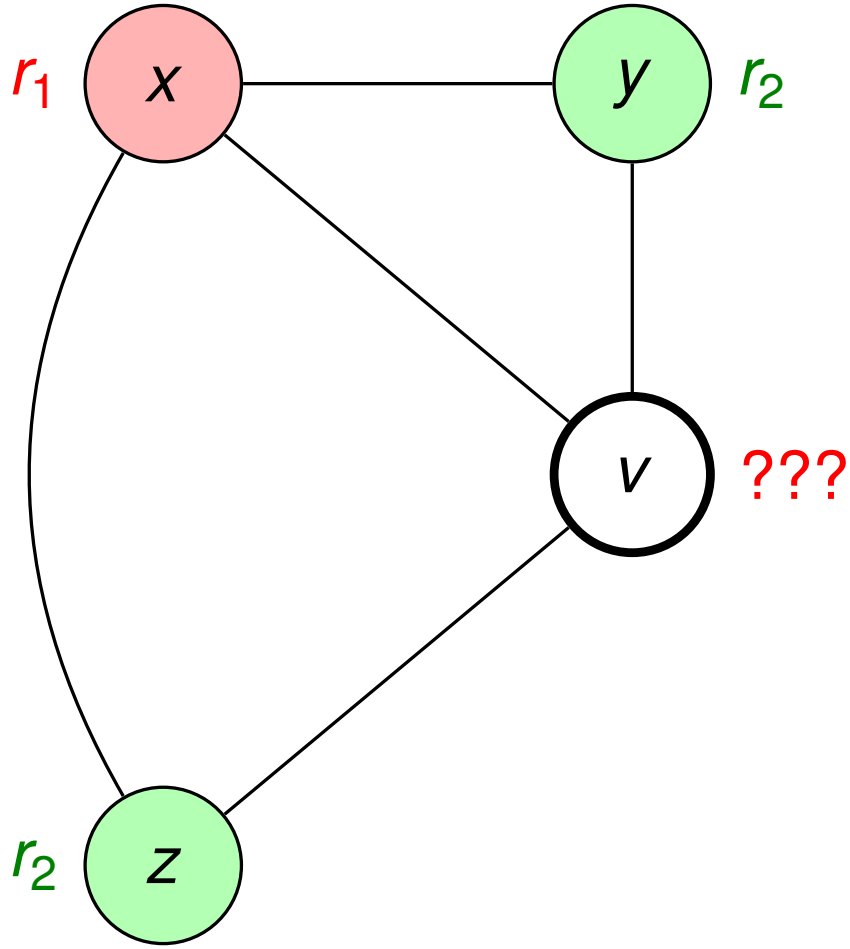
Exemplu reluat ($k = 2$)



Stiva:

$\{v, u, w\}$

Exemplu reluat ($k = 2$)



Colorare **optimistă**: speranța că vecinii lui v nu au epuizat toate culorile.
Eșec: imposibilitatea colorării lui v!

„Vărsarea” în memorie

În caz de eșec al colorării optimiste pentru v:

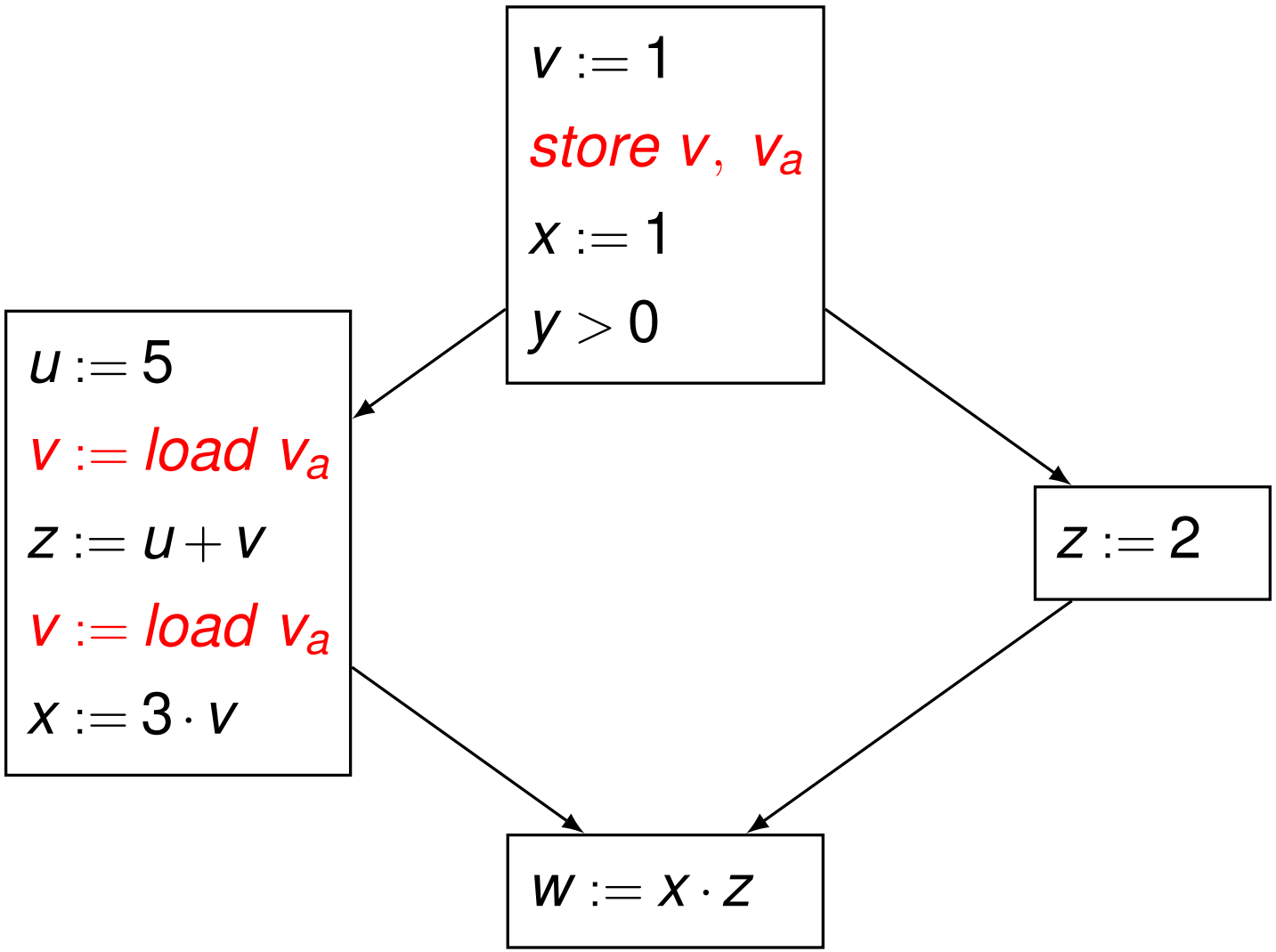
- Rezervarea unei locații de memorie pentru v, numită v_a (de obicei, în AR curent)
- Înaintea fiecărei **citiri** a lui v, introducerea operației:

$$v := load\ v_a$$

- După fiecare **scriere** a lui v, introducerea operației:

$$store\ v,\ v_a$$

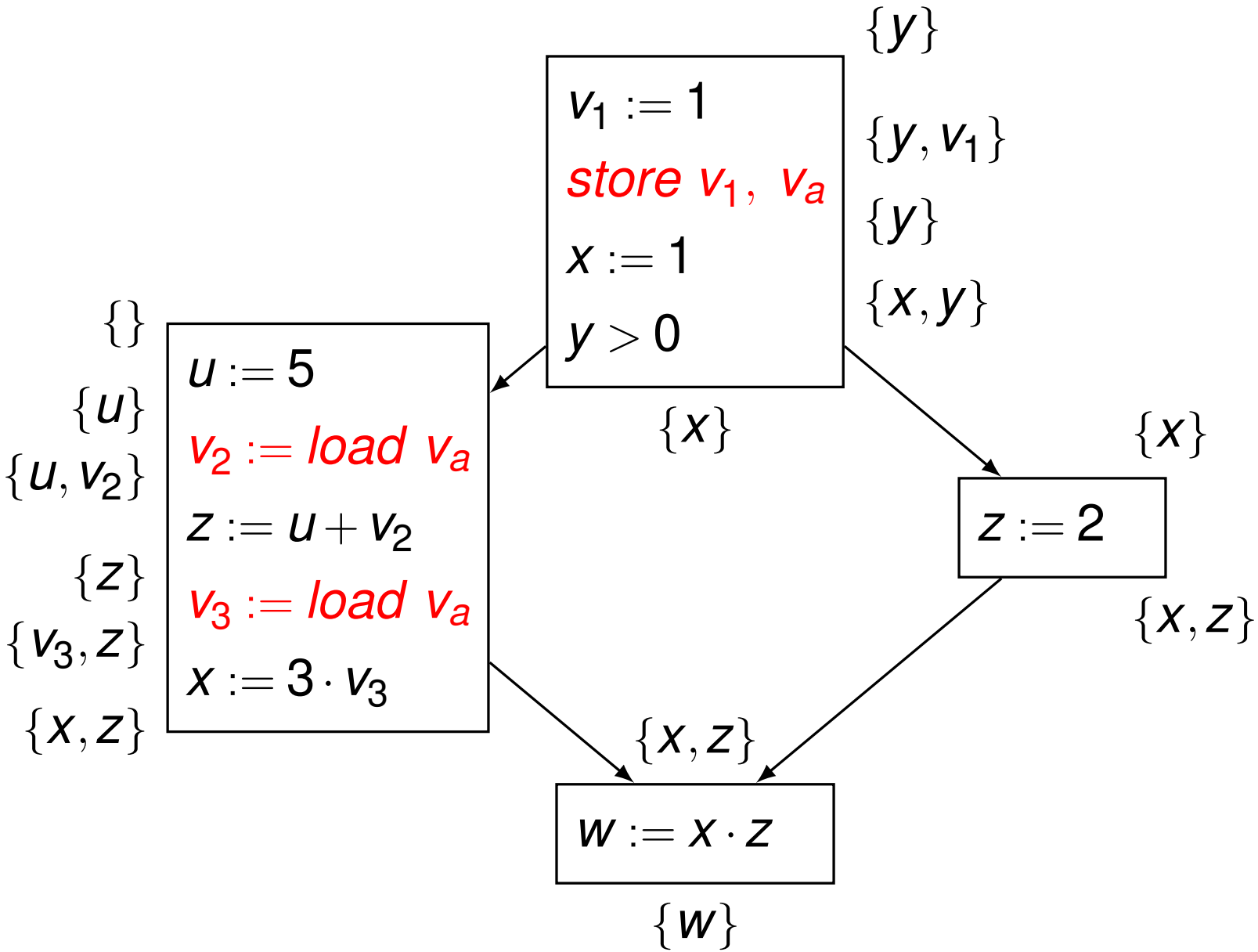
Exemplu reluat



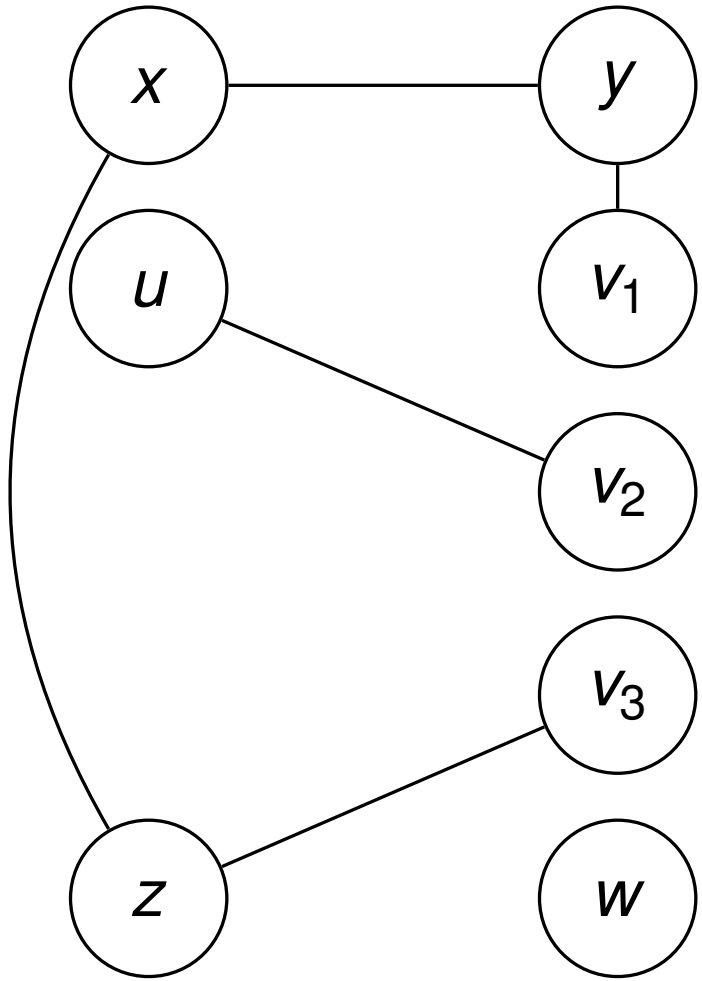
Discuție

- Reducerea regiunilor în care v este *live* între:
  - Operația $v := load\ v_a$ și instrucțiunea următoare
  - Operația $store\ v,\ v_a$ și instrucțiunea anterioară
- Prin urmare, reducerea interferențelor dintre registrele temporare
- Totuși, forțarea alocării aceluiași registru fizic pentru toate mențiunile lui v, datorită utilizării aceluiași nume peste tot
- Ideal, posibilitatea varierii registrului fizic alocat, prin utilizarea de nume diferite: v_1, v_2 etc.

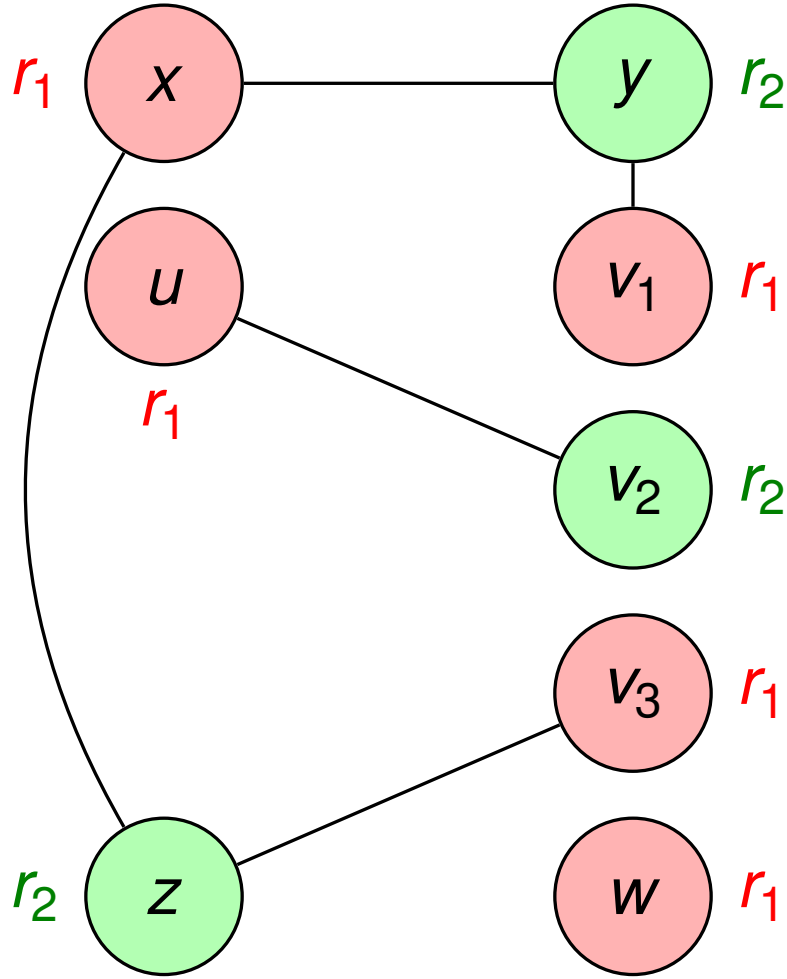
Exemplu reluat



Exemplu reluat ($k = 2$)



Exemplu reluat ($k = 2$)



Posibilitatea utilizării de registre fizice diferite pentru registrele temporare derivate din v!

Alegerea registrelor temporare de „vărsat”

- Posibilitatea „vărsării” mai multor registre temporare înainte de identificarea unei colorări

- Euristici pentru alegerea registrului temporar de „vărsat”:
  - Număr mare de conflicte
  - Număr mic de citiri și scrieri
  - Neapartenența la bucle

Partea IX

Gestiunea automată a memoriei

Cuprins

Cuprins

Probleme cu gestiunea manuală a memoriei

- Neeliberarea memoriei deși nu mai este utilizată
- Pierderea referinței la o zonă de memorie
- Dereferențierea unui pointer după eliberare
- Depășirea limitelor unei structuri de date
- Posibilitatea manifestării efectelor acestor probleme mult mai târziu

Gestiunea automată a memoriei

- La **crearea** unui obiect: alocare de spațiu disponibil
- La **epuizarea** spațiului disponibil sau uneori mai devreme: eliberarea spațiului utilizat de obiectele care (ideal) nu vor mai fi folosite
- Cum determinăm obiectele de mai sus?

Determinarea obiectelor „neutilizate în viitor”

- De obicei, imposibil de determinat exact
- Euristică: concentrare pe obiectele care pot fi referite în cadrul programului
- Exemplu în COOL de pierdere a referinței la obiectul proaspăt alocat:

```
1  {
2        x <- new A;
3        x <- y;
4  }
```

Identificabilitate

- Un obiect este **identificabil** (*reachable*) dacă:
  - este accesibil prin intermediul unui registru SAU
  - este referit de un alt obiect identificabil

- Posibilitatea determinării tuturor obiectelor identificabile, pornind din registre și urmând toate referințele

- Imposibilitatea utilizării ulterioare a unui obiect neidentificabil (*garbage*)

Discuție

```
1  {
2      x <- new A;
3      x <- y;
4
5      -- Itereaza pana la sfarsitul executiei
6      while (...) loop
7          -- Nu refera x
8          ...
9      pool;
10 }
```

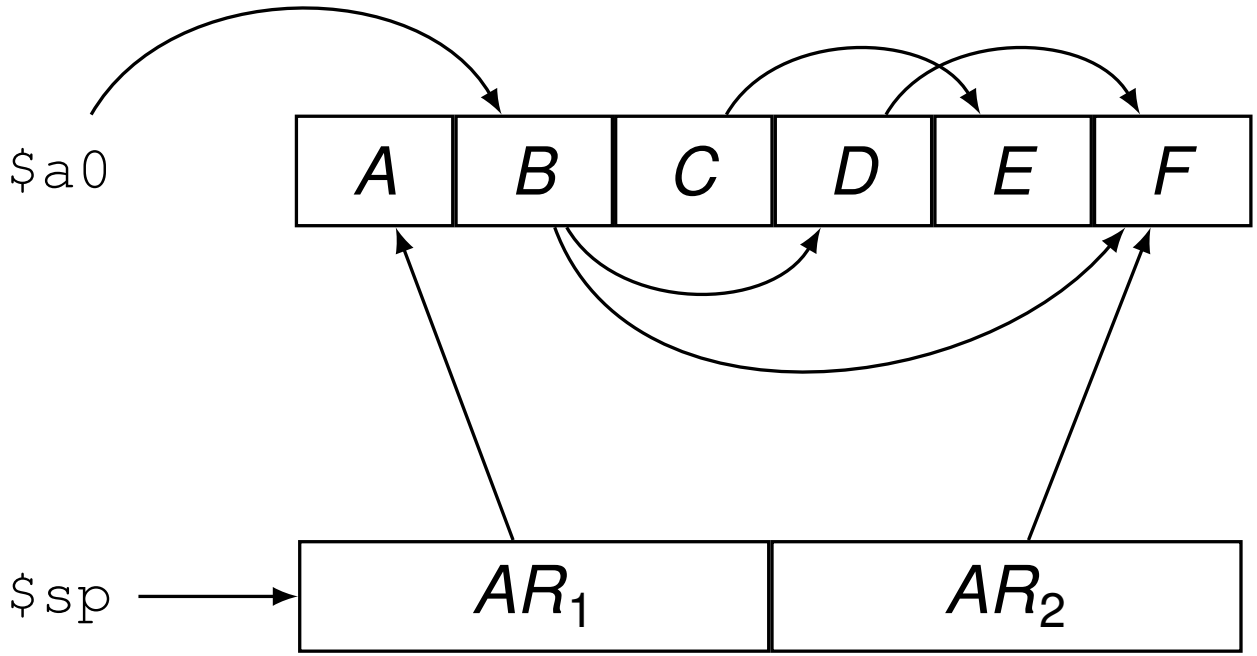
- `x` identificabil, cu toate că nu mai este folosit!
- Neidentificabil $\Longrightarrow$ neutilizat în viitor $\ddot{\smile}$
- Identificabil $\not\Longrightarrow$ utilizat în viitor! $\ddot{\frown}$

Identificabilitate în compilatorul Cool

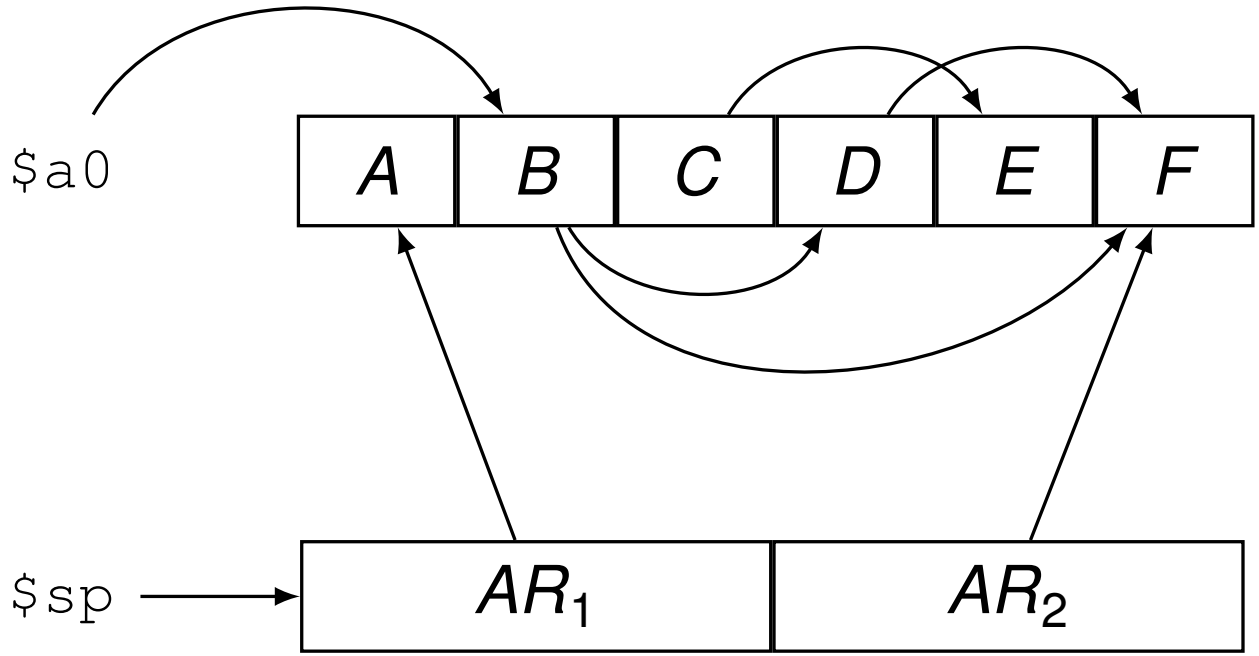
Pornind de la **rădăcini** (*roots*):

- **Acumulatorul** (registrul `$a0`)
- **Stiva**: prezența referințelor la obiecte în AR-uri — necesitatea de a distinge referințele de alte informații (e.g. adresa de revenire) pe baza organizării AR-urilor

Exemplu



Exemplu



Obiectele *C* și *E*, neidentificabile!

Eliberarea automată a memoriei neutilizate

- Determinarea obiectelor identificabile (cf. definiției anterioare)
- Eliberarea memoriei aferente obiectelor neidentificabile
- Posibilitatea activării mecanismului la umplerea memoriei sau mai devreme

Cuprins

Descriere

- Etapa ***mark***: determinarea obiectelor identificabile
- Etapa ***sweep***: eliberarea memoriei aferente obiectelor neidentificabile
- Prezența unui marcaj (*mark bit*) în cadrul reprezentării obiectelor, inițial 0, și modificat la 1 în etapa *mark* dacă obiectul este identificabil

Structuri utilizate

- Lista ***todo***: folosită pentru marcarea obiectelor indentificabile, în etapa *mark*

- Lista ***free***: utilizată pentru reținerea zonelor disponibile de memorie, adăugate în etapa *sweep*

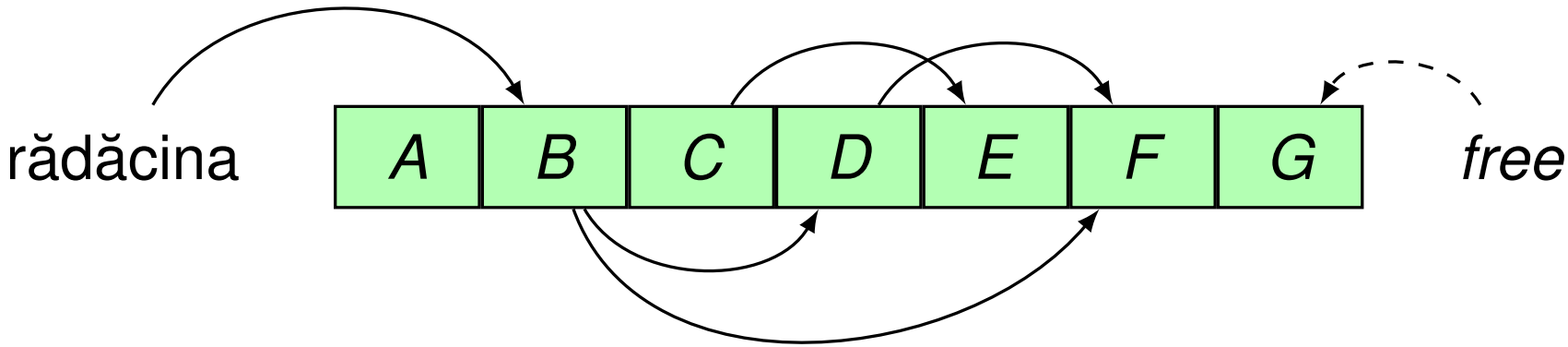
Etapa *mark*

1 $todo \leftarrow \{\text{radacinile}\}$
2 **cât timp** $todo \neq \emptyset$
3 alege $x \in todo$
4 $todo \leftarrow todo \setminus \{x\}$
5 **dacă** $mark(x) = 0$ **atunci**
6 $mark(x) = 1$
7 $todo \leftarrow todo \cup \{x_i \mid x_i \text{ este referit de } x\}$

Etapa *sweep*

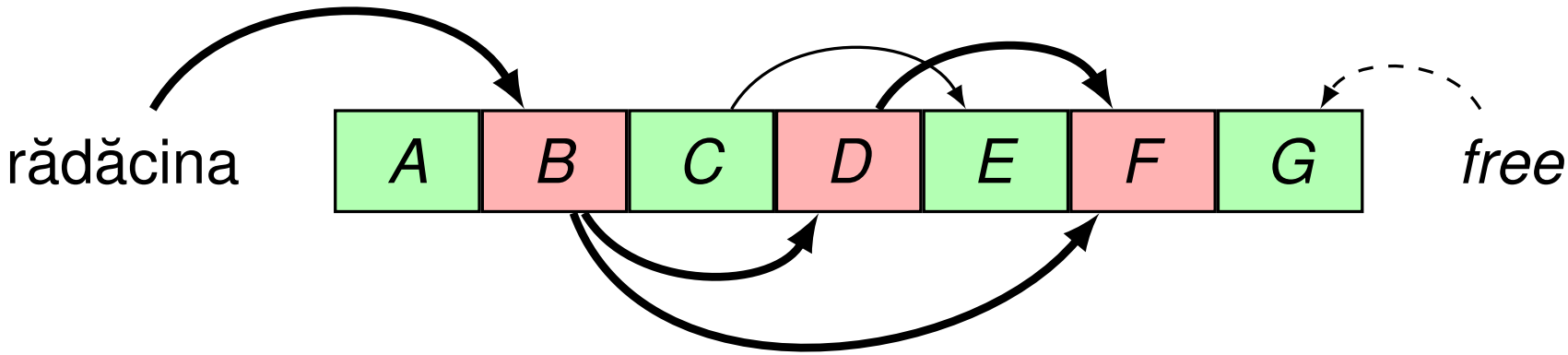
```
1 p ← bottom(heap)
2 cât timp p < top(heap)
3     dacă mark(p) = 1 atunci
4         mark(p) = 0
5     altfel
6         free ← free ∪ {p...(p + sizeof(p) − 1)}
7     p ← p + sizeof(p)
```

Exemplu



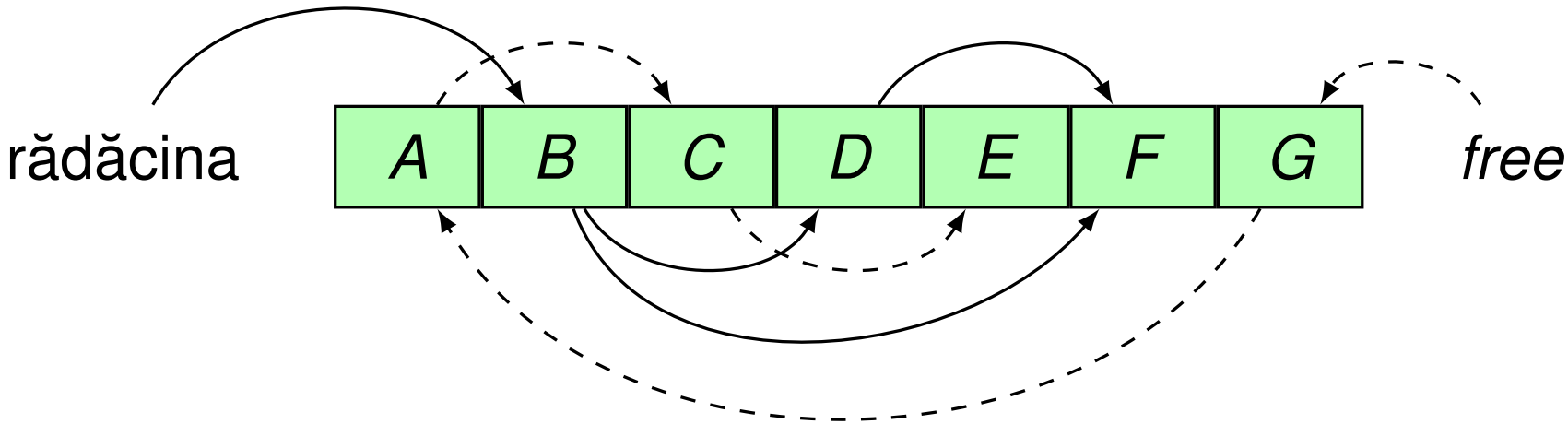
Inițial

Exemplu



Mark

Exemplu

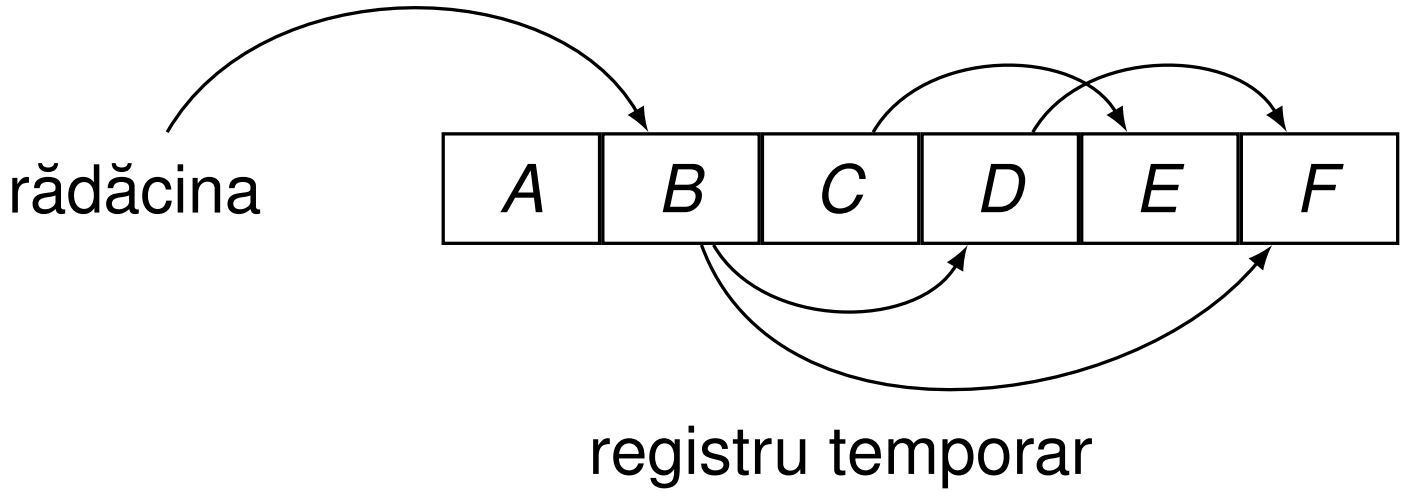


Sweep (*A*, *C*, *E*, adăugate la lista *free*)

Implementare

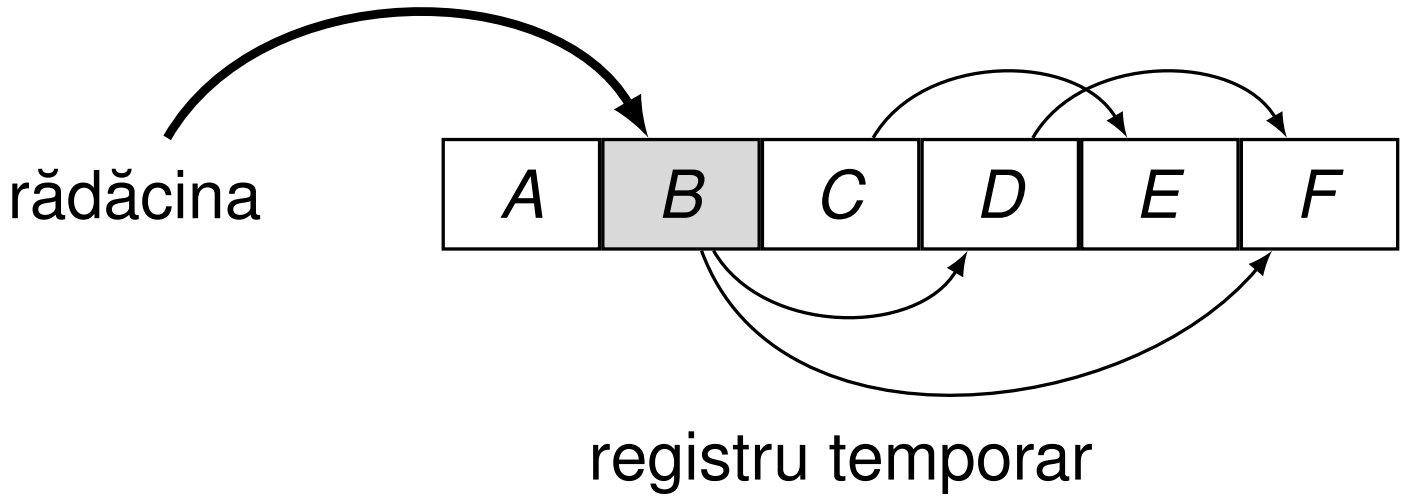
- **Problemă**: imposibilitatea alocării de spațiu pentru listele *todo* și *free*, din moment ce algoritmul rulează la epuizarea memoriei disponibile
- Soluție neviabilă: imposibilitatea prealocării de spațiu pentru lista *todo*, având dimensiune nemărginită
- **Soluție** corectă: reprezentarea listelor utilizând câmpuri din obiectele însele!

Implementarea listei *todo*



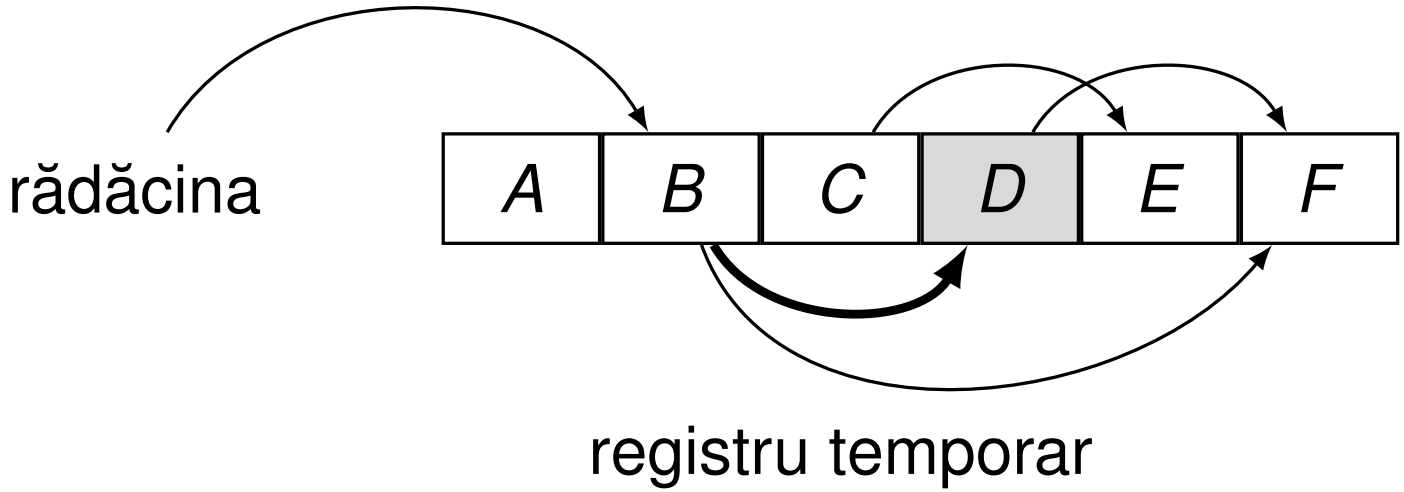
- **Inversarea** pointerilor: stocarea temporară a adresei predecesorului în pointer-ul traversat către urmaș
- Utilizarea unui registru temporar pentru a stoca adresa ultimului nod parcurs
- Reprezentarea implicită a stivei aferente unei DFS

Implementarea listei *todo*



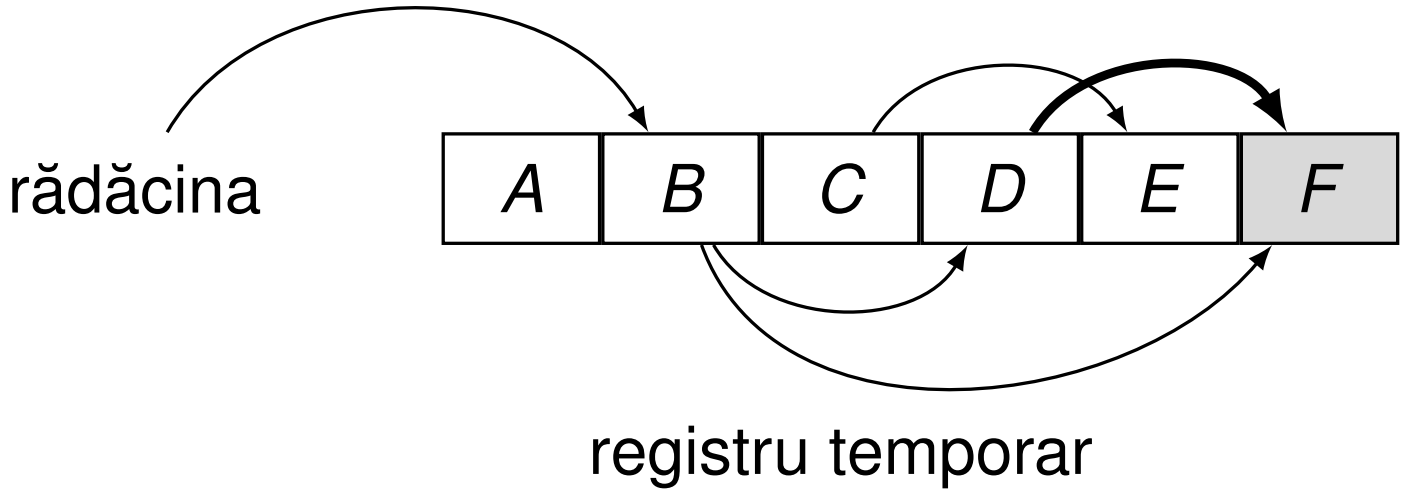
- **Inversarea** pointerilor: stocarea temporară a adresei predecesorului în pointer-ul traversat către urmaș
- Utilizarea unui registru temporar pentru a stoca adresa ultimului nod parcurs
- Reprezentarea implicită a stivei aferente unei DFS

Implementarea listei *todo*



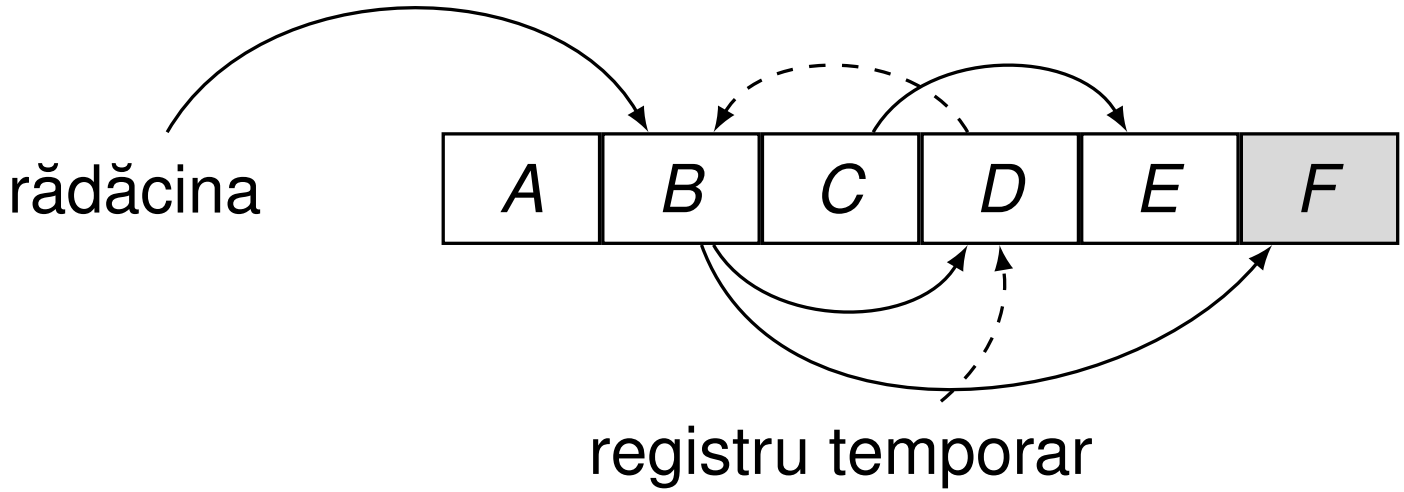
- **Inversarea** pointerilor: stocarea temporară a adresei predecesorului în pointer-ul traversat către urmaș
- Utilizarea unui registru temporar pentru a stoca adresa ultimului nod parcurs
- Reprezentarea implicită a stivei aferente unei DFS

Implementarea listei *todo*



- **Inversarea** pointerilor: stocarea temporară a adresei predecesorului în pointer-ul traversat către urmaș
- Utilizarea unui registru temporar pentru a stoca adresa ultimului nod parcurs
- Reprezentarea implicită a stivei aferente unei DFS

Implementarea listei *todo*



- **Inversarea** pointerilor: stocarea temporară a adresei predecesorului în pointer-ul traversat către urmaș
- Utilizarea unui registru temporar pentru a stoca adresa ultimului nod parcurs
- Reprezentarea implicită a stivei aferente unei DFS

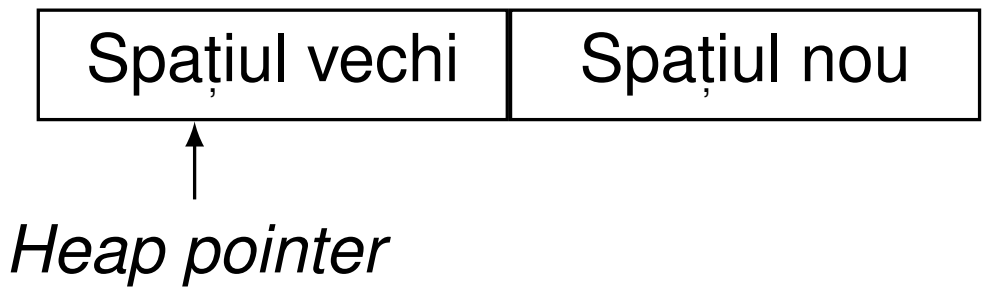
Aspecte generale

- La **alocarea** unui obiect, parcurgerea listei *free* până la întâlnirea unui bloc suficient de încăpător, cu depunerea porțiunii neutilizate înapoi în listă
- Pericolul fragmentării excesive a memoriei, când lista *free* conține un număr mare de blocuri foarte mici, neîncăpătoare pentru obiecte noi
- Soluție: agregarea blocurilor adiacente într-un singur bloc
- Evitarea deplasării obiectelor în memorie, importantă în limbaje ca C/C++, care expun adresele în program

Cuprins

Descriere

- Organizarea memoriei în două zone:
  - **Spațiul vechi** (*old space*): utilizat pentru alocare
  - **Spațiul nou** (*new space*): rezervat pentru algoritm

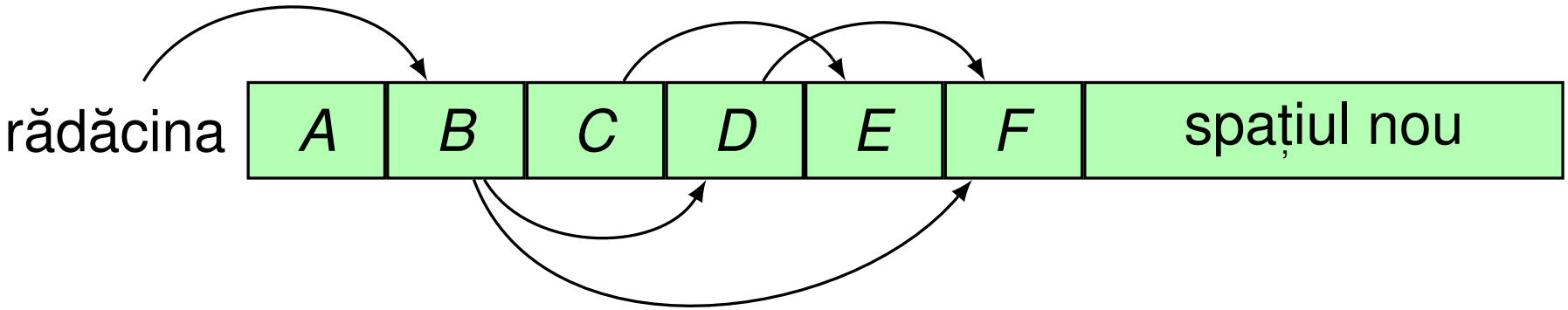


- *Heap pointer*: primul cuvânt liber din spațiul vechi
- **Alocare** = deplasare *heap pointer*

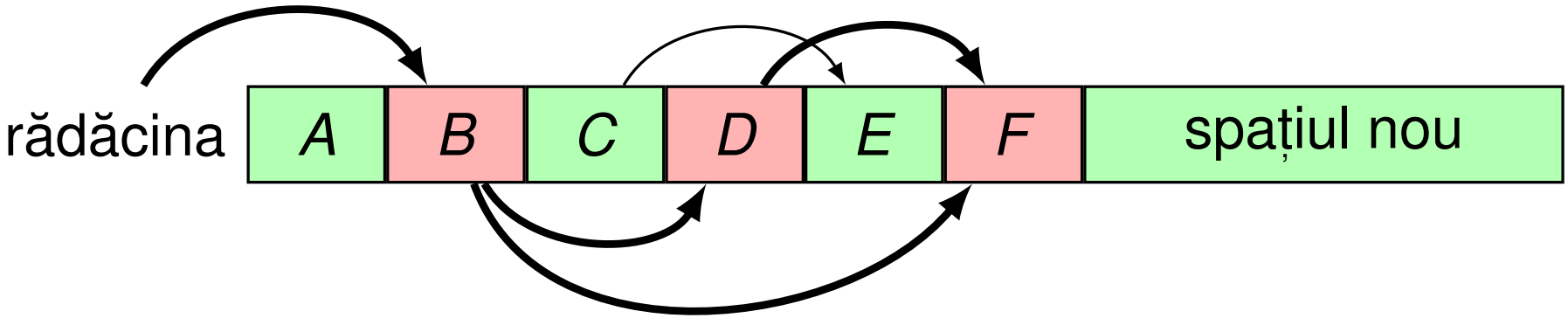
Descriere (cont.)

- Rularea algoritmului la umplerea spațiului vechi
- Determinarea și copierea obiectelor identificabile din spațiul vechi în cel nou, lăsând celelalte obiecte în urmă
- În final, inversarea rolurilor celor două spații

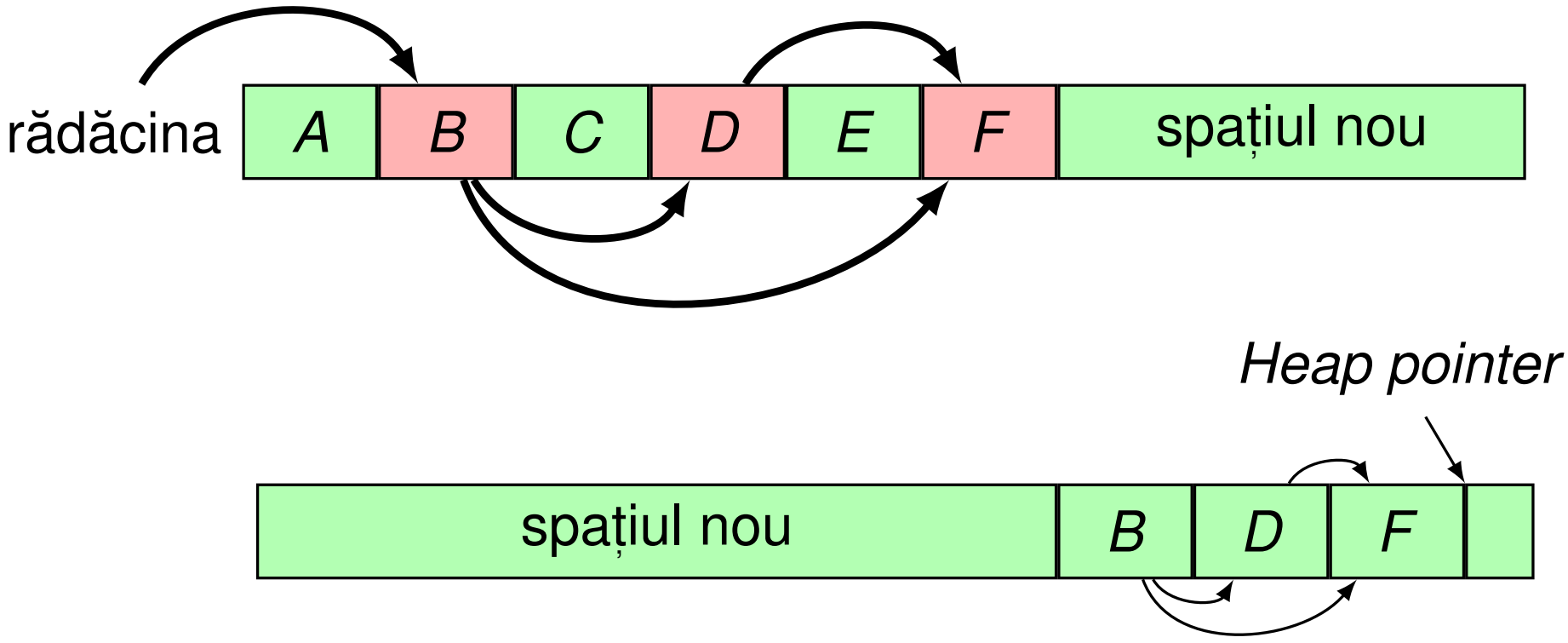
Exemplu



Exemplu



Exemplu

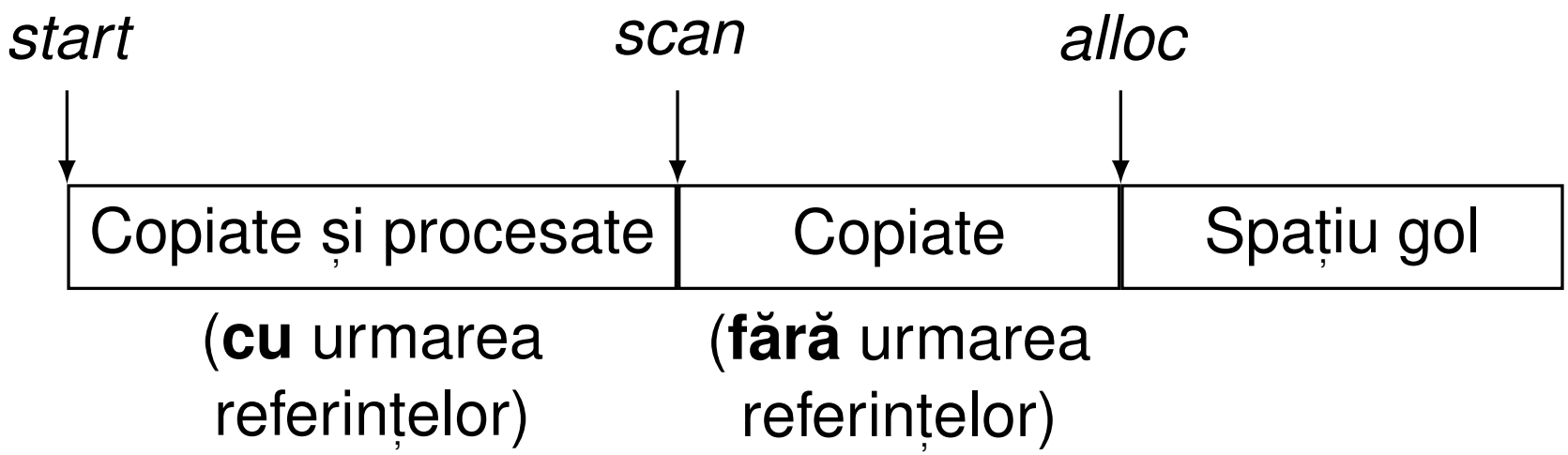


Implementare

- Necesitatea determinării obiectelor identificabile, la fel ca la *Mark and Sweep*
- **Problemă**: modificarea adresei unui obiect în urma copierii, de unde necesitatea actualizării referințelor către acestea
- **Soluție**: stocarea unui **pointer de redirectare** către copie în cadrul obiectului copiat (*forwarding pointer*)

Implementare (cont.)

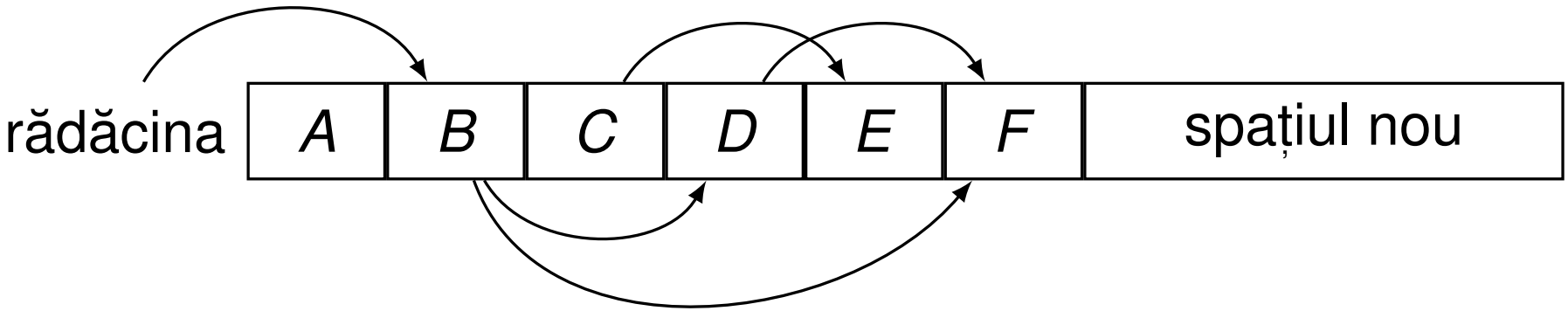
- **Problemă**: evitarea alocării de spațiu suplimentar pentru copierea obiectelor și actualizarea referințelor
- **Soluție**: partiționarea **spațiului nou** în trei zone:



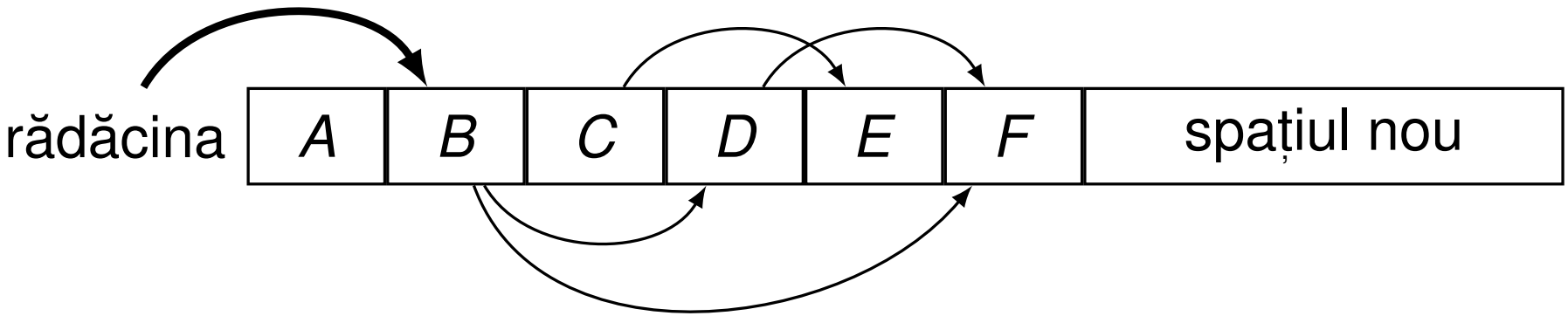
Algoritmul

1 Copiază obiectele indicate de rădăcini și actualizează pointerii lor de redirectare
2 **cât timp** $scan \neq alloc$
3 $O \leftarrow$ obiectul indicat de *scan*
4 **pentru fiecare** pointer p din O
5 $O_1 \leftarrow$ obiectul referit de p
6 **dacă** O_1 nu are pointer de redirectare **atunci**
7 $O_2 \leftarrow$ copiază O_1 în spațiul nou, actualizând *alloc*
8 actualizează pointerul de redirectare din O_1 pentru a indica O_2
9 actualizează p cu pointer-ul de redirectare
10 actualizează *scan* pentru a indica următorul obiect

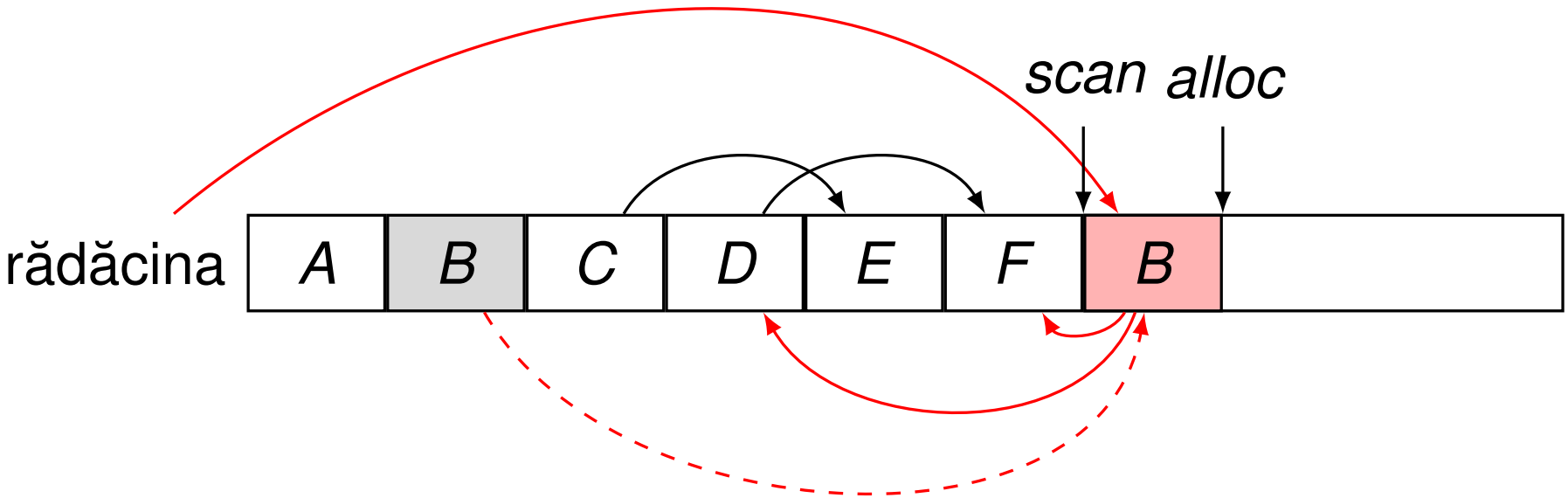
Exemplu



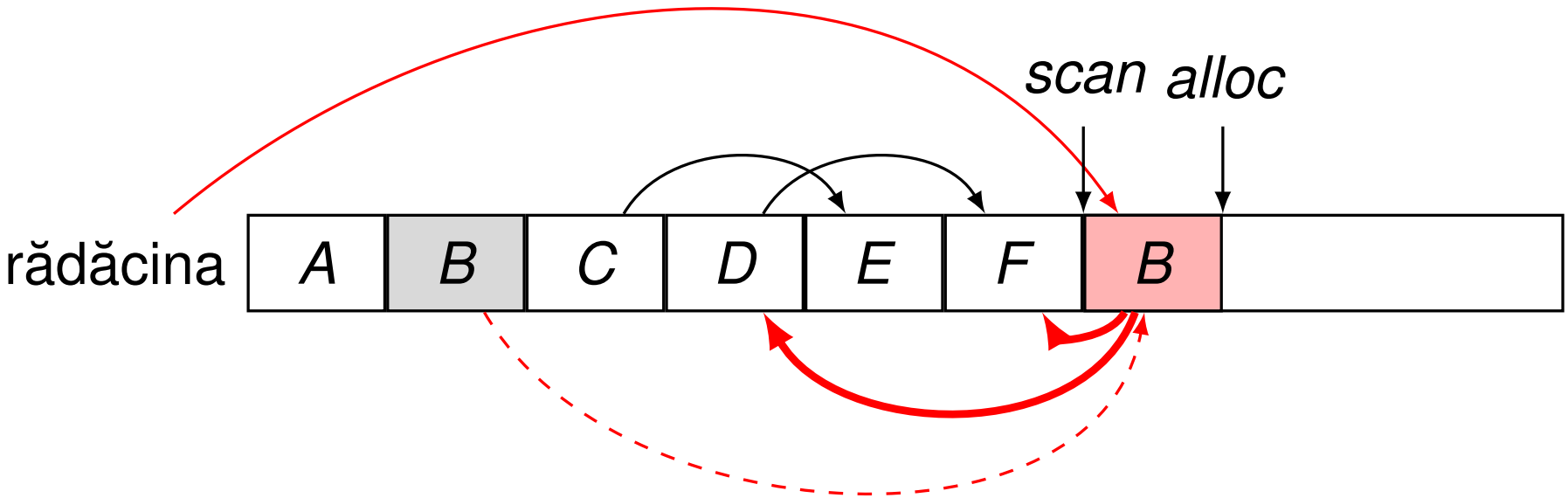
Exemplu



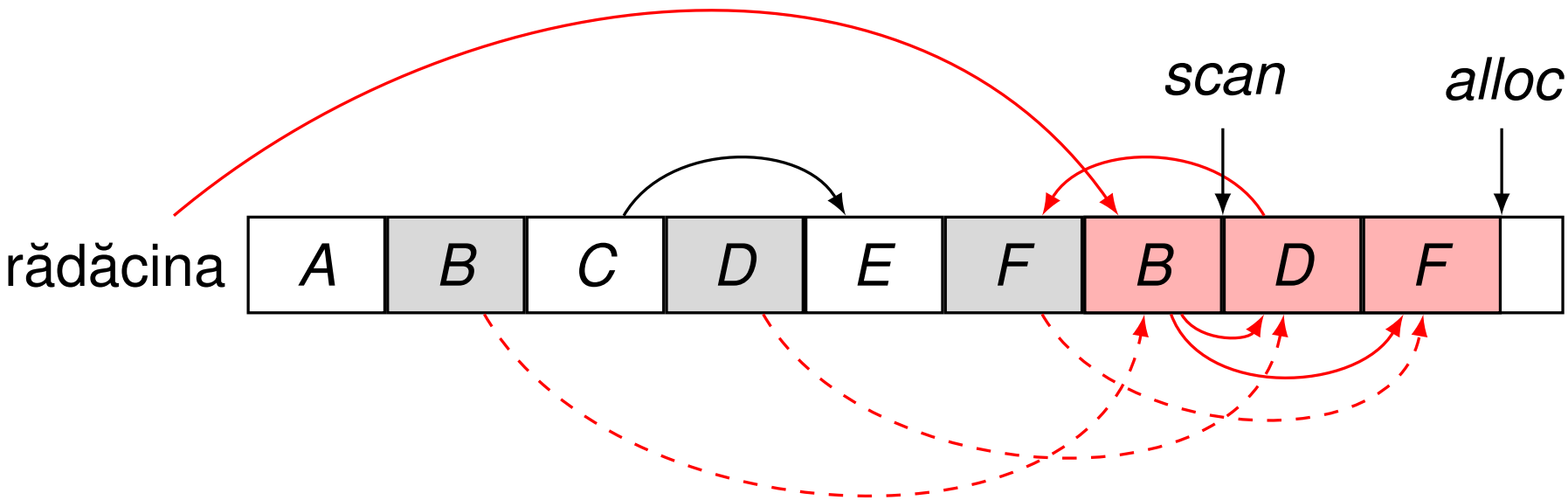
Exemplu



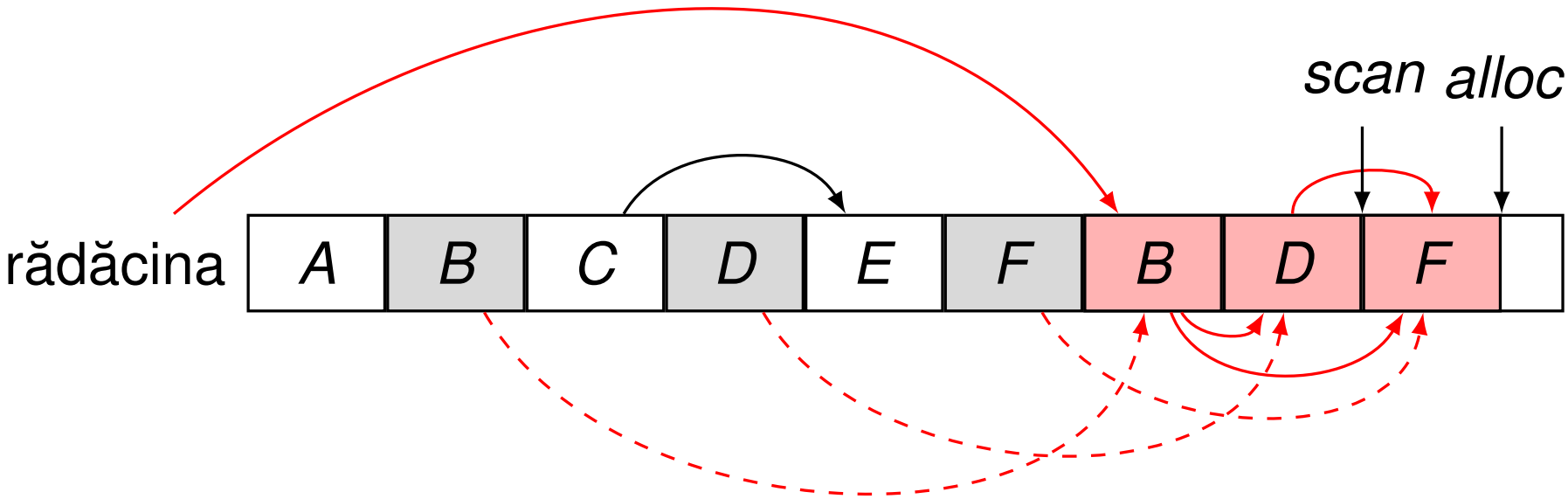
Exemplu



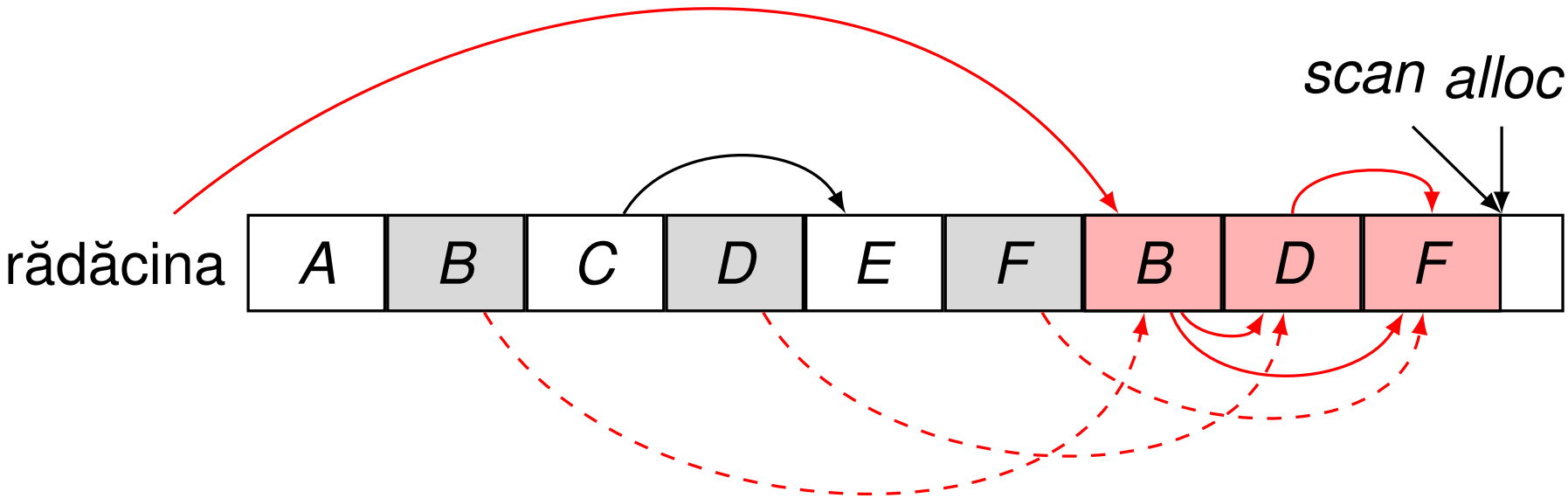
Exemplu



Exemplu



Exemplu



Aspecte generale

- Necesitatea cunoașterii dimensiunii unui obiect și a poziției referințelor din cadrul acestuia
- Necesitatea copierii obiectelor referite pe stivă și a actualizării referințelor pe stivă (operație potențial costisitoare)
- Considerată cea mai rapidă strategie
- Alocare ieftină computațional (deplasarea *heap pointer*)
- Colectare de obicei ieftină, în special dacă sunt puține obiecte identificabile

Cuprins

Descriere

- Eliberarea memoriei aferente unui obiect în momentul în care nu mai există referințe la el
- Stocarea numărului de referințe în cadrul reprezentării obiectului
- Actualizarea numărului de referințe în cazul atribuirilor și al eliberării altor obiecte care îl referă pe cel curent

Implementare

- **Numărul de referințe** (*reference count*) ale obiectului o: $rc(o)$
- **Construirea** de către `new` a unui obiect cu numărul de referințe 1
- Înlocuirea fiecărei **atribuiri** $x \leftarrow y$, unde cele două variabile denotă obiectele o, respectiv p, cu secvența:

1 $rc(o) \leftarrow rc(o) - 1$
2 $rc(p) \leftarrow rc(p) + 1$
3 **dacă** $rc(o) = 0$ **atunci** eliberează o
4 $x \leftarrow y$

- La **colectarea** unui obiect, decrementarea numărului de referințe ale obiectelor referite de acesta

Aspecte generale

- **Avantaje**:
  - Facil de implementat
  - Absența pauzelor mari în execuție datorită colectării incrementale
- **Dezavantaje**:
  - Imposibilitatea colectării structurilor circulare
  - Cost mare al atribuirilor, din cauza instrucțiunilor adăugate

Gestiunea automată a memoriei

- Utilă în prevenirea unor erori grave
- Reducerea controlului programatorului
- Probleme în aplicațiile de timp real, din cauza întârzierilor introduse de colectare
- Posibilitate de *memory leaks* (e.g. din cauza referințelor circulare)
- Concepte mai avansate: colectare **concurentă**, **paralelă**, **generațională**