Teoria Compilării

Drăgulici Dumitru Daniel

Facultatea de matematică și informatică,
Universitatea București

2011

Cuprins

Cuprins

Algoritmi de parsare bottom - up

Algoritmii de parsare bottom - up încearcă construirea arborelui de derivare de jos în sus (de la frontieră spre rădăcină): se pleacă de la cuvântul analizat, se încearcă "potrivirea" în el a membrului drept al unei producții, dacă se potrivește se înlocuiește porțiunea respectivă cu neterminalul din stânga producției, apoi se reia procedeul cu noul șir (format acum din terminale și neterminale).

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$$t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6 \quad t_7 \quad t_8 \quad t_9 \quad t_{10} \, t_{11} \, t_{12} \, t_{13} \, t_{14}$$

Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$$\underline{\boxed{t_1}}\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_8\ t_9\ t_{10}\ t_{11}\ t_{12}\ t_{13}\ t_{14}$$

Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$\underline{t_1 \mid t_2}\ t_3\ t_4\ t_5\ t_6\ t_7\ t_8\ t_9\ t_{10}\ t_{11}\ t_{12}\ t_{13}\ t_{14}$

Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

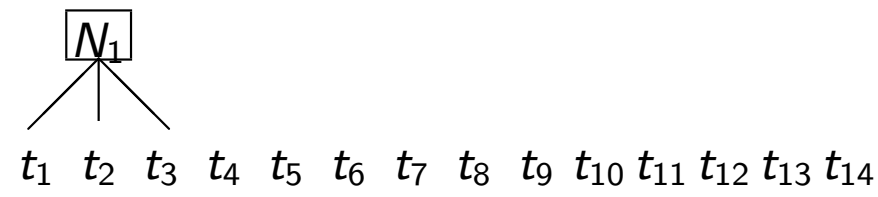
$\underline{t_1 \mid t_2 \mid t_3}\ t_4\ t_5\ t_6\ t_7\ t_8\ t_9\ t_{10}\ t_{11}\ t_{12}\ t_{13}\ t_{14}$

Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$N_1 \rightarrow t_1 t_2 t_3$

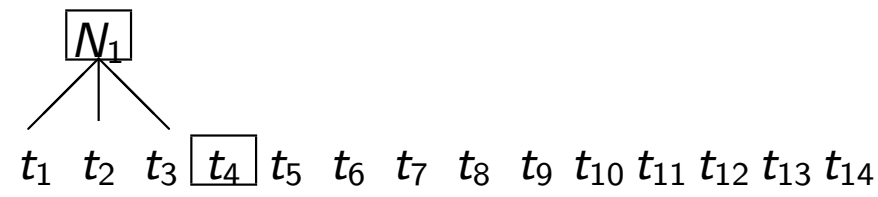


Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$N_1 \rightarrow t_1\, t_2\, t_3$

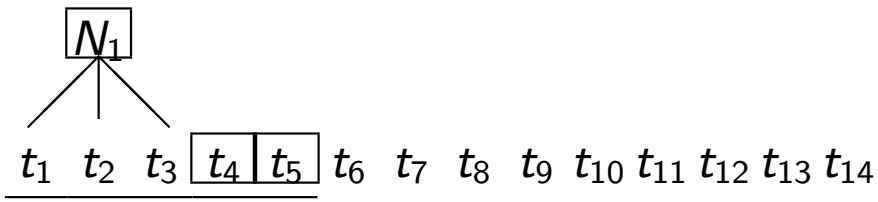


Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$N_1 \rightarrow t_1\, t_2\, t_3$

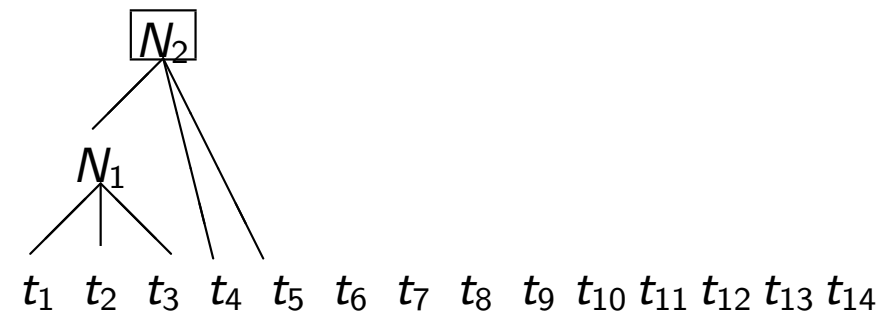


Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$N_2 \rightarrow N_1\, t_4\, t_5$
$N_1 \rightarrow t_1\, t_2\, t_3$

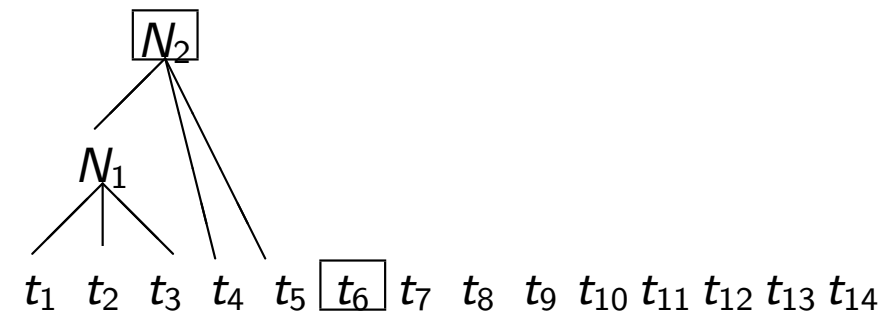


Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$N_2 \rightarrow N_1\, t_4\, t_5$
$N_1 \rightarrow t_1\, t_2\, t_3$

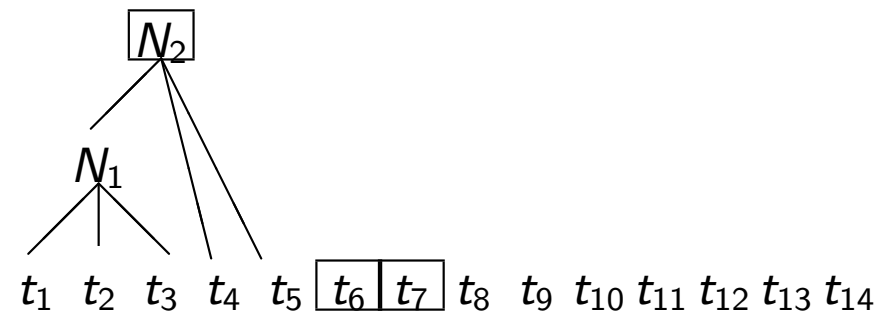


Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$N_2 \rightarrow N_1\, t_4\, t_5$
$N_1 \rightarrow t_1\, t_2\, t_3$

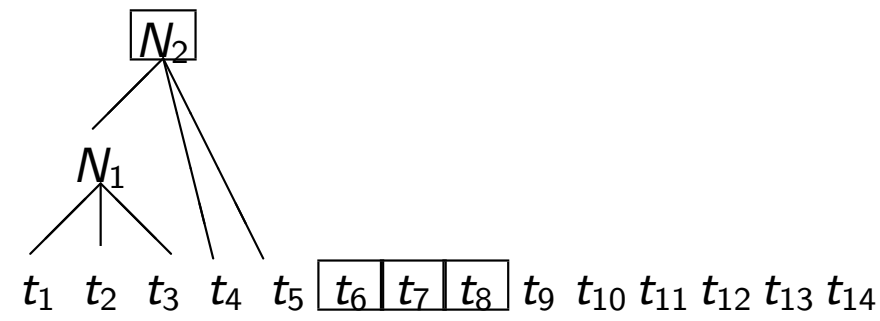


Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$N_2 \rightarrow N_1\, t_4\, t_5$
$N_1 \rightarrow t_1\, t_2\, t_3$

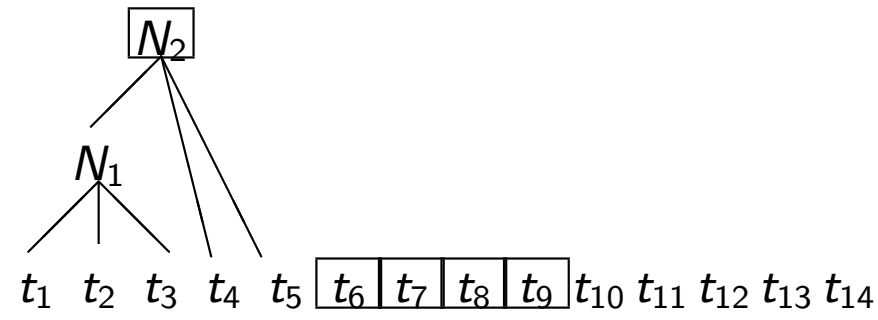


Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$N_2 \rightarrow N_1\, t_4\, t_5$
$N_1 \rightarrow t_1\, t_2\, t_3$

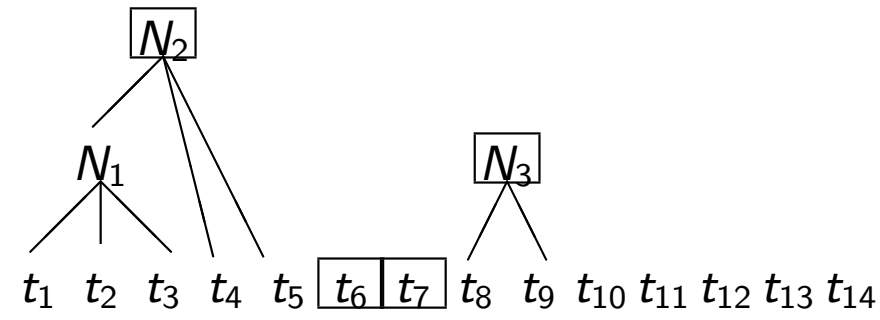


Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$N_3 \rightarrow t_8\, t_9$
$N_2 \rightarrow N_1\, t_4\, t_5$
$N_1 \rightarrow t_1\, t_2\, t_3$

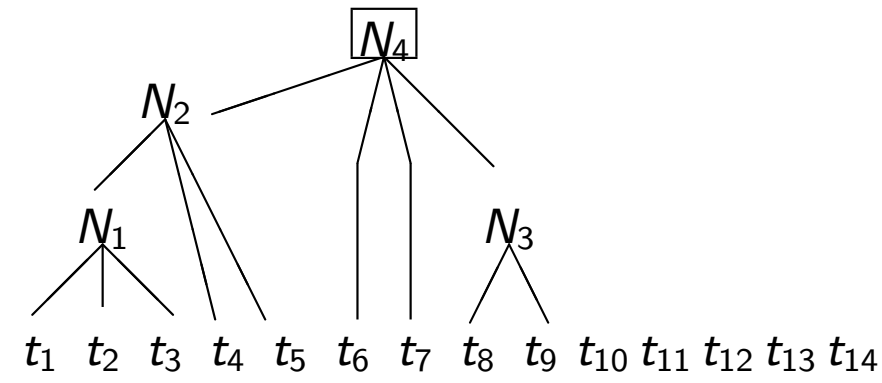


Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$N_4 \rightarrow N_2\, t_6\, t_7\, N_3$
$N_3 \rightarrow t_8\, t_9$
$N_2 \rightarrow N_1\, t_4\, t_5$
$N_1 \rightarrow t_1\, t_2\, t_3$

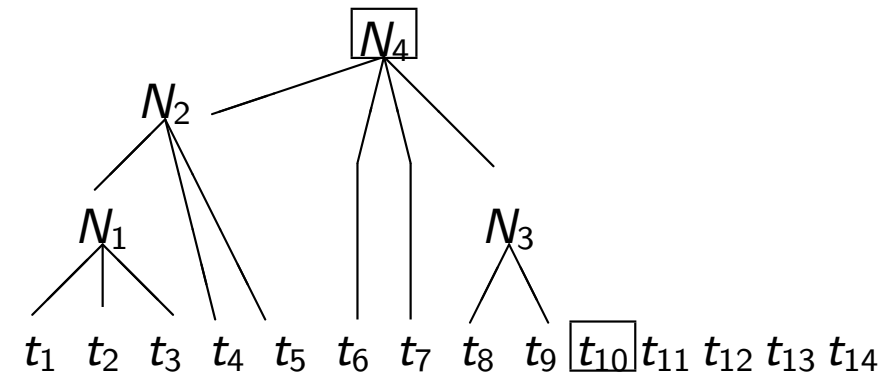


Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$N_4 \rightarrow N_2\, t_6\, t_7\, N_3$
$N_3 \rightarrow t_8\, t_9$
$N_2 \rightarrow N_1\, t_4\, t_5$
$N_1 \rightarrow t_1\, t_2\, t_3$

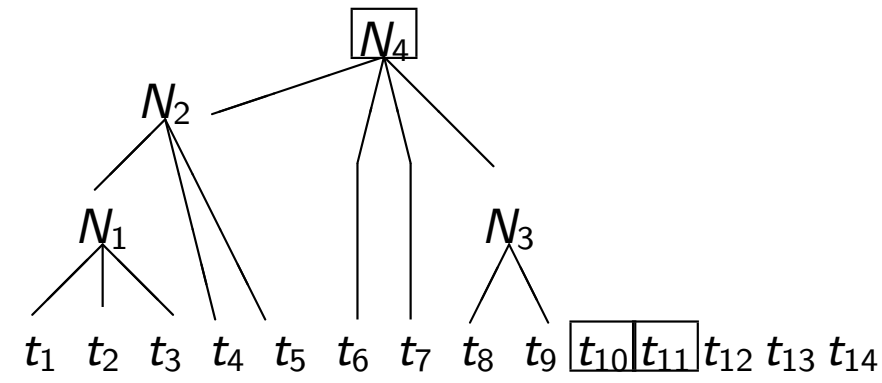


Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$N_4 \rightarrow N_2\, t_6\, t_7\, N_3$
$N_3 \rightarrow t_8\, t_9$
$N_2 \rightarrow N_1\, t_4\, t_5$
$N_1 \rightarrow t_1\, t_2\, t_3$

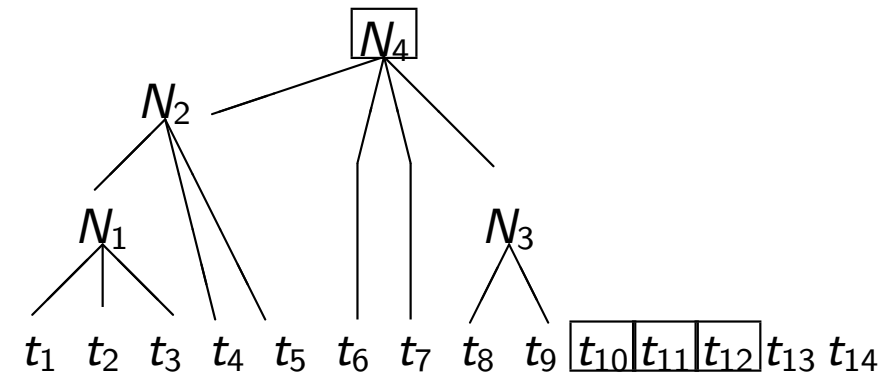


Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$N_4 \rightarrow N_2\, t_6\, t_7\, N_3$
$N_3 \rightarrow t_8\, t_9$
$N_2 \rightarrow N_1\, t_4\, t_5$
$N_1 \rightarrow t_1\, t_2\, t_3$

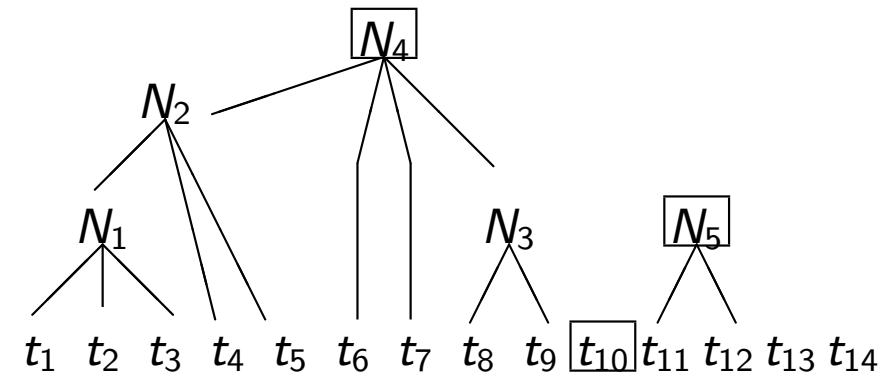


Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$N_5 \rightarrow t_{11}\, t_{12}$
$N_4 \rightarrow N_2\, t_6\, t_7\, N_3$
$N_3 \rightarrow t_8\, t_9$
$N_2 \rightarrow N_1\, t_4\, t_5$
$N_1 \rightarrow t_1\, t_2\, t_3$

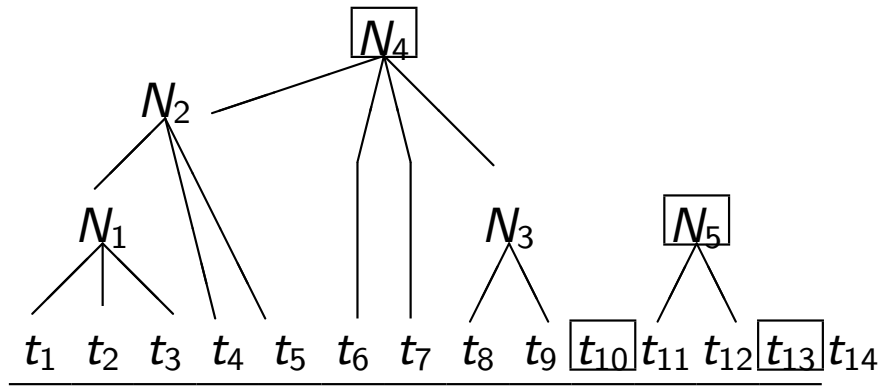


Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$N_5 \rightarrow t_{11}\, t_{12}$
$N_4 \rightarrow N_2\, t_6\, t_7\, N_3$
$N_3 \rightarrow t_8\, t_9$
$N_2 \rightarrow N_1\, t_4\, t_5$
$N_1 \rightarrow t_1\, t_2\, t_3$

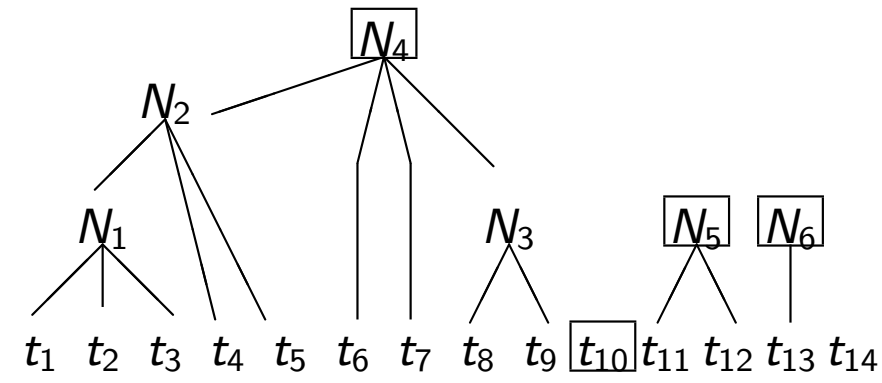


Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$N_6 \rightarrow t_{13}$
$N_5 \rightarrow t_{11}\, t_{12}$
$N_4 \rightarrow N_2\, t_6\, t_7\, N_3$
$N_3 \rightarrow t_8\, t_9$
$N_2 \rightarrow N_1\, t_4\, t_5$
$N_1 \rightarrow t_1\, t_2\, t_3$

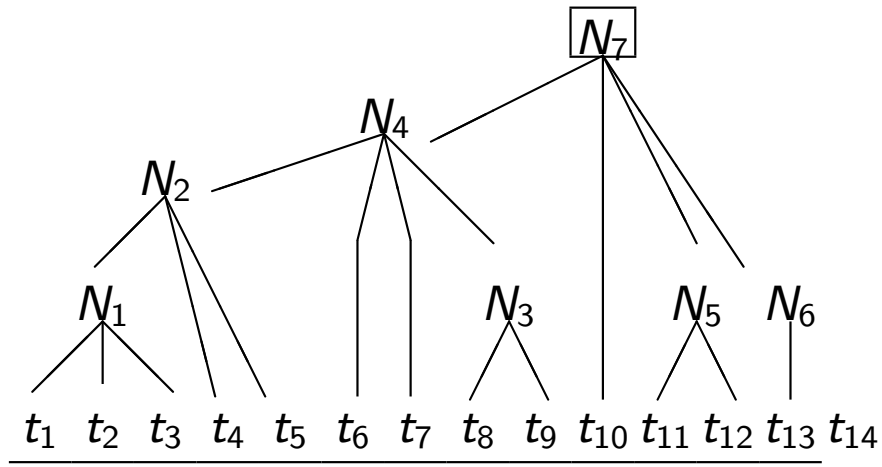


Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$N_7 \rightarrow N_4\, t_{10}\, N_5\, N_6$
$N_6 \rightarrow t_{13}$
$N_5 \rightarrow t_{11}\, t_{12}$
$N_4 \rightarrow N_2\, t_6\, t_7\, N_3$
$N_3 \rightarrow t_8\, t_9$
$N_2 \rightarrow N_1\, t_4\, t_5$
$N_1 \rightarrow t_1\, t_2\, t_3$

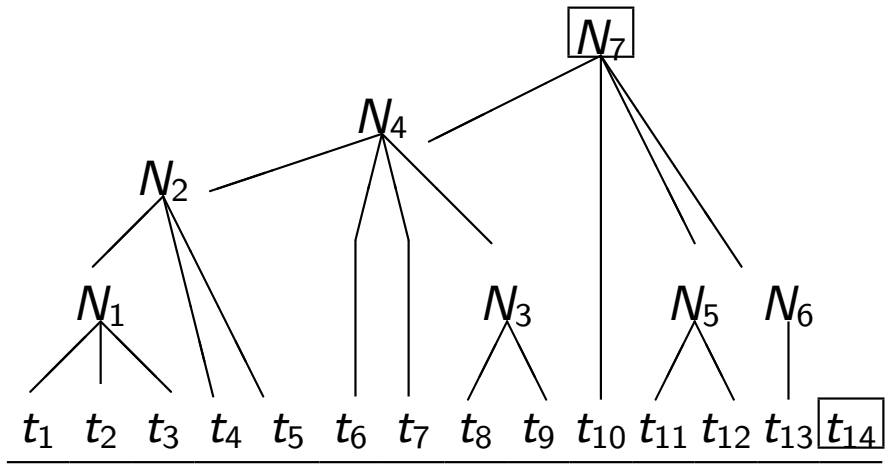


Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$N_7 \rightarrow N_4\, t_{10}\, N_5\, N_6$
$N_6 \rightarrow t_{13}$
$N_5 \rightarrow t_{11}\, t_{12}$
$N_4 \rightarrow N_2\, t_6\, t_7\, N_3$
$N_3 \rightarrow t_8\, t_9$
$N_2 \rightarrow N_1\, t_4\, t_5$
$N_1 \rightarrow t_1\, t_2\, t_3$

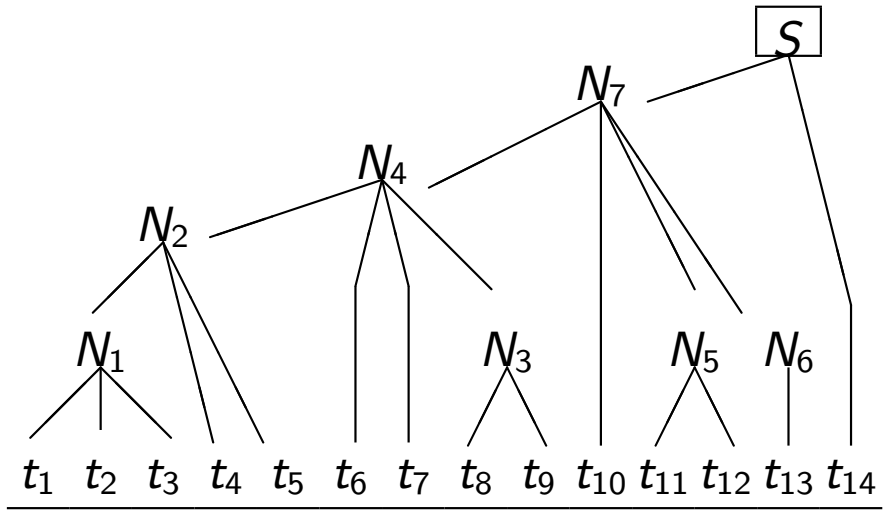


Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Pentru a elimina mai repede variantele greșite, cuvântul inițial se parcurge de la stânga la dreapta și se încearcă potrivirea membrilor drepți ai producțiilor peste un sufix al prefixului parcurs în el sau al șirurilor obținute după înlocuire - astfel, hotărâtoare în alegerea producțiilor sunt următoarele terminale parcurse în cuvântul inițial:

$S \to N_7\, t_{14}$
$N_7 \to N_4\, t_{10}\, N_5\, N_6$
$N_6 \to t_{13}$
$N_5 \to t_{11}\, t_{12}$
$N_4 \to N_2\, t_6\, t_7\, N_3$
$N_3 \to t_8\, t_9$
$N_2 \to N_1\, t_4\, t_5$
$N_1 \to t_1\, t_2\, t_3$



Putem să nu cerem condiții suplimentare asupra gramaticii și atunci după un număr de potriviri/înlocuiri putem fi nevoiți să ne întoarcem și să încercăm altele - algoritmul general bottom - up este exponențial (face backtracking), sau putem cere niște condiții suplimentare asupra gramaticii, de ex. să fie suficientă consultarea a k terminale înainte în cuvântul analizat pentru a determina direct acțiunea corectă (ex. producția corectă) și atunci să avem un algoritm liniar.

Algoritmi de parsare bottom - up

Obs. că deși parcurgerea cuvântului inițial pentru a determina producțiile se face de la stânga la dreapta, când ne punem problema cum se obține acel cuvânt din S folosind producțiile respective constatăm că de fiecare dată se derivează cel mai din dreapta neterminal, adică s-a obținut o derivare dreaptă (S, N_7, N_6, N_5, ...).

De aceea gramaticile cu proprietăți suplimentare care ne permit să determinăm direct, în mod unic, acțiunea/producția corectă consultând k simboluri înainte îndeplinesc niște condiții diferite de cele din cazul $LL(k)$ și s.n. gramatici **de tip** $LR(k)$ (denumirea LR, i.e. **left-right**, provine de la faptul că cuvântul analizat este parcurs de la stânga la dreapta ("left"), iar în final se obține o derivare dreaptă a sa ("right")).

Algoritmii de parsare bottom-up sunt construiți după principiul deplasare - reducere: la fiecare pas fie avansez cu un simbol în cuvântul analizat, fie înlocuiesc un sufix al șirului acumulat cu o producție; reducerile sunt încercate înaintea deplasărilor.

Cuprins

Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăți suplimentare ale GIC, dar este exponențial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T \quad 2 : E \to T \quad 3 : T \to T \star F \quad 4 : T \to F \quad 5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:

$$a \qquad \star \qquad a$$

Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăţi suplimentare ale GIC, dar este exponenţial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T \quad 2 : E \to T \quad 3 : T \to T \star F \quad 4 : T \to F \quad 5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:

$$\underline{\boxed{a}} \qquad \star \qquad a$$

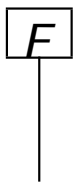
Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăți suplimentare ale GIC, dar este exponențial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T$ $2 : E \to T$ $3 : T \to T \star F$ $4 : T \to F$ $5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:



$5 : F \to a$ $\underline{a}$ $\star$ a

Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăţi suplimentare ale GIC, dar este exponenţial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T \quad 2 : E \to T \quad 3 : T \to T \star F \quad 4 : T \to F \quad 5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:

$4 : T \to F$
$5 : F \to a$

$\star \qquad a$

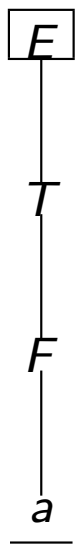
Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăți suplimentare ale GIC, dar este exponențial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T$ $2 : E \to T$ $3 : T \to T \star F$ $4 : T \to F$ $5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:



$\star$ a

$2 : E \to T$
$4 : T \to F$
$5 : F \to a$

Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

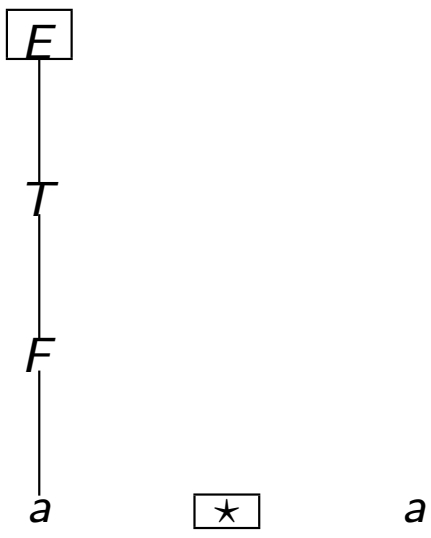
Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăţi suplimentare ale GIC, dar este exponenţial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T$ $2 : E \to T$ $3 : T \to T \star F$ $4 : T \to F$ $5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:

$2 : E \to T$
$4 : T \to F$
$5 : F \to a$



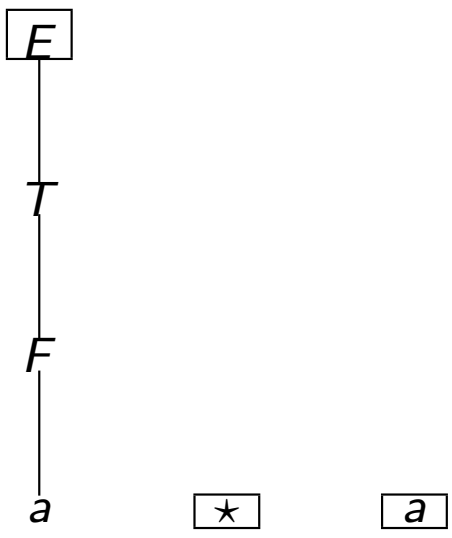
Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăţi suplimentare ale GIC, dar este exponenţial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T$ $2 : E \to T$ $3 : T \to T \star F$ $4 : T \to F$ $5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:



$2 : E \to T$
$4 : T \to F$
$5 : F \to a$

Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

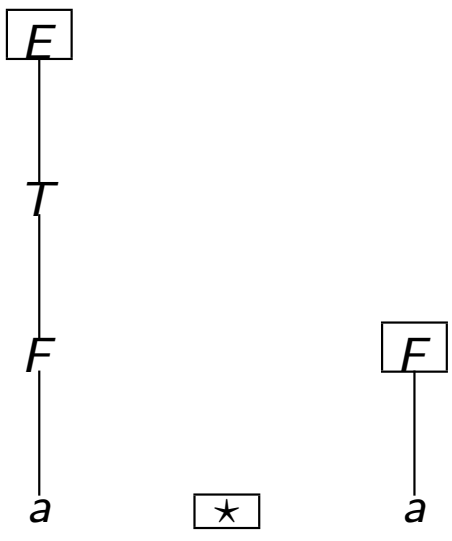
Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăți suplimentare ale GIC, dar este exponențial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T$ $2 : E \to T$ $3 : T \to T \star F$ $4 : T \to F$ $5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:

$5 : F \to a$
$2 : E \to T$
$4 : T \to F$
$5 : F \to a$



Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

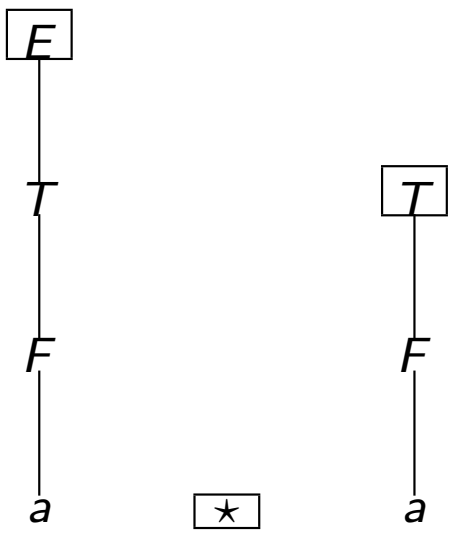
Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăți suplimentare ale GIC, dar este exponențial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T \quad 2 : E \to T \quad 3 : T \to T \star F \quad 4 : T \to F \quad 5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:

$4 : T \to F$
$5 : F \to a$
$2 : E \to T$
$4 : T \to F$
$5 : F \to a$



Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

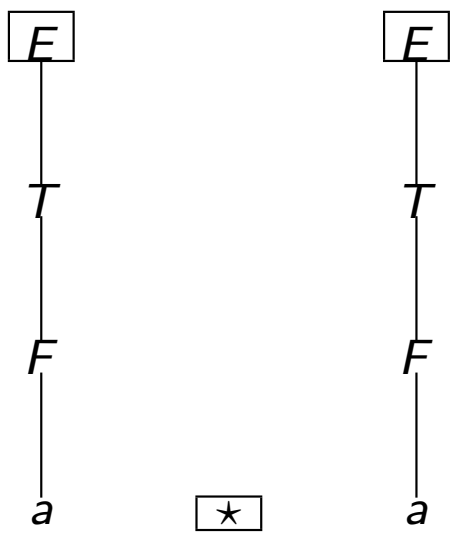
Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăţi suplimentare ale GIC, dar este exponenţial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T$ $2 : E \to T$ $3 : T \to T \star F$ $4 : T \to F$ $5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:

$2 : E \to T$
$4 : T \to F$
$5 : F \to a$
$2 : E \to T$
$4 : T \to F$
$5 : F \to a$



Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

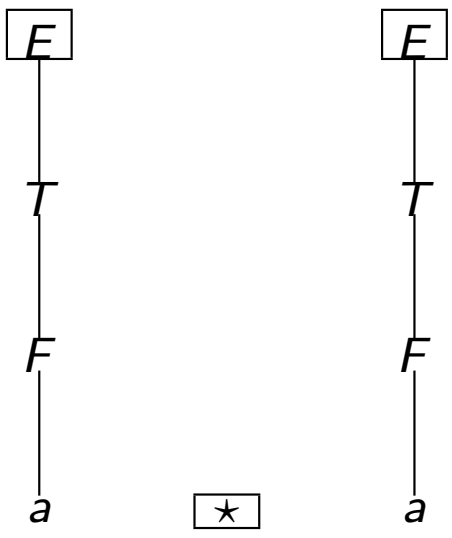
Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăți suplimentare ale GIC, dar este exponențial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T \quad 2 : E \to T \quad 3 : T \to T \star F \quad 4 : T \to F \quad 5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:

$2 : E \to T$
$4 : T \to F$
$5 : F \to a$
$2 : E \to T$
$4 : T \to F$
$5 : F \to a$



Revenire

Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

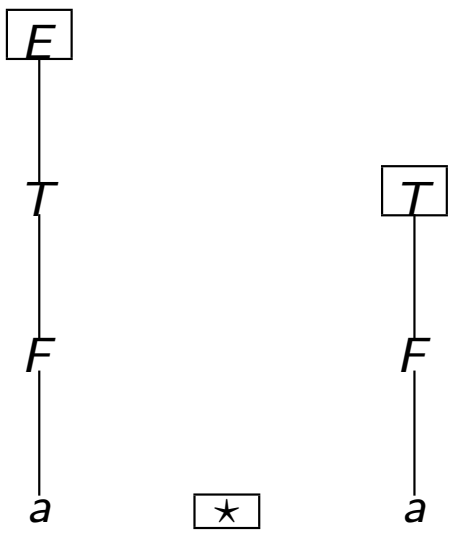
Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăţi suplimentare ale GIC, dar este exponenţial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T \quad 2 : E \to T \quad 3 : T \to T \star F \quad 4 : T \to F \quad 5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:

$4 : T \to F$
$5 : F \to a$
$2 : E \to T$
$4 : T \to F$
$5 : F \to a$



Revenire

Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

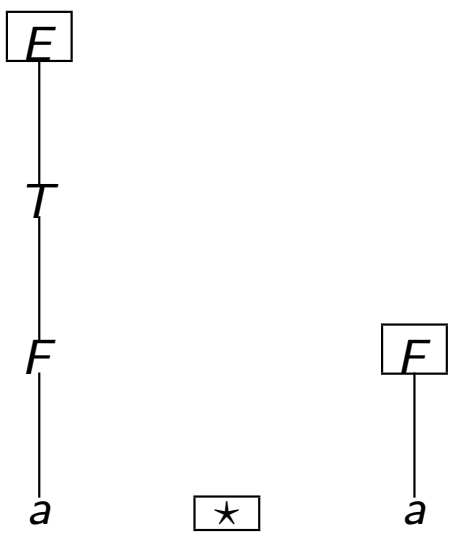
Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăţi suplimentare ale GIC, dar este exponenţial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T \quad 2 : E \to T \quad 3 : T \to T \star F \quad 4 : T \to F \quad 5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:

$5 : F \to a$
$2 : E \to T$
$4 : T \to F$
$5 : F \to a$



Revenire

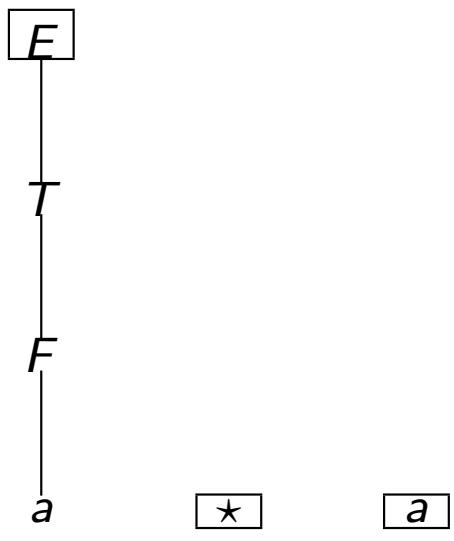
Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăţi suplimentare ale GIC, dar este exponenţial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T$ $2 : E \to T$ $3 : T \to T \star F$ $4 : T \to F$ $5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:



Revenire

$2 : E \to T$
$4 : T \to F$
$5 : F \to a$

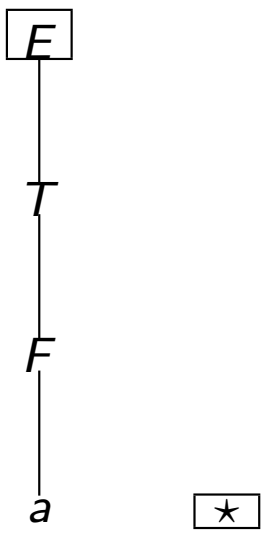
Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăţi suplimentare ale GIC, dar este exponenţial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T$ $2 : E \to T$ $3 : T \to T \star F$ $4 : T \to F$ $5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:



Revenire

$2 : E \to T$
$4 : T \to F$
$5 : F \to a$

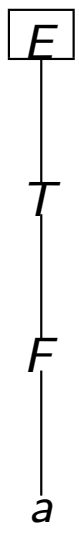
Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăţi suplimentare ale GIC, dar este exponenţial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T$ $2 : E \to T$ $3 : T \to T \star F$ $4 : T \to F$ $5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:



Revenire

$2 : E \to T$
$4 : T \to F$
$5 : F \to a$

$\star$ a

Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

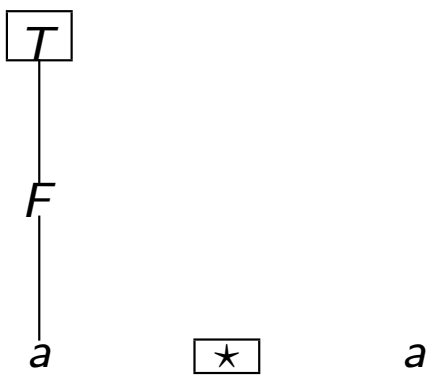
Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăți suplimentare ale GIC, dar este exponențial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T$ $2 : E \to T$ $3 : T \to T \star F$ $4 : T \to F$ $5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:

$4 : T \to F$
$5 : F \to a$



Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

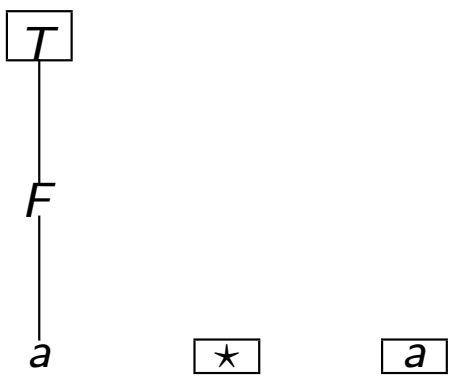
Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăţi suplimentare ale GIC, dar este exponenţial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T$ $2 : E \to T$ $3 : T \to T \star F$ $4 : T \to F$ $5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:

$4 : T \to F$
$5 : F \to a$



Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

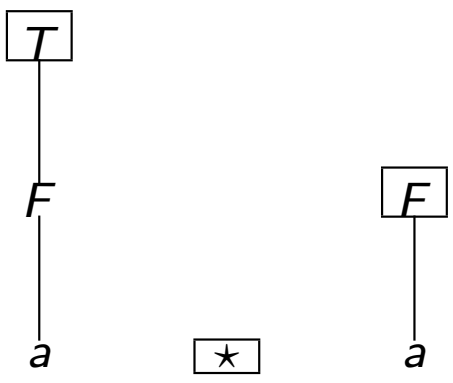
Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăți suplimentare ale GIC, dar este exponențial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T$ $2 : E \to T$ $3 : T \to T \star F$ $4 : T \to F$ $5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:

$5 : F \to a$
$4 : T \to F$
$5 : F \to a$



Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

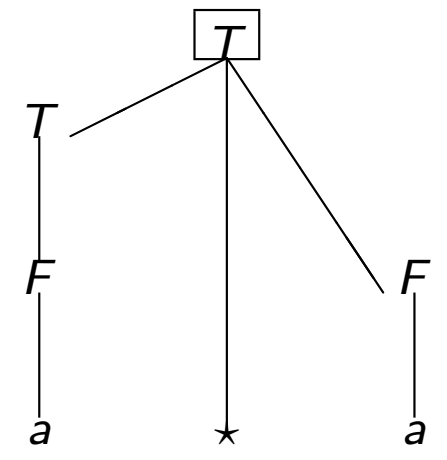
Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăţi suplimentare ale GIC, dar este exponenţial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T \quad 2 : E \to T \quad 3 : T \to T \star F \quad 4 : T \to F \quad 5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:

$3 : T \to T \star F$
$5 : F \to a$
$4 : T \to F$
$5 : F \to a$



Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

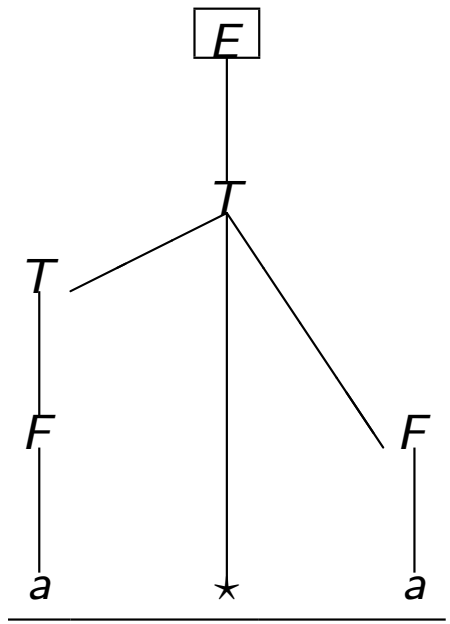
Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up nu necesită proprietăți suplimentare ale GIC, dar este exponențial (face backtracking).

Exemplu:

Fie G GIC dată de: $V_N = \{E, T, F\}$, $V_T = \{a, +, \star\}$, simbolul de start: E, P: $1 : E \to E + T \quad 2 : E \to T \quad 3 : T \to T \star F \quad 4 : T \to F \quad 5 : F \to a$ (am numerotat producţiile). Pentru $w = a \star a$ avem succesiv:

$2 : E \to T$
$3 : T \to T \star F$
$5 : F \to a$
$4 : T \to F$
$5 : F \to a$

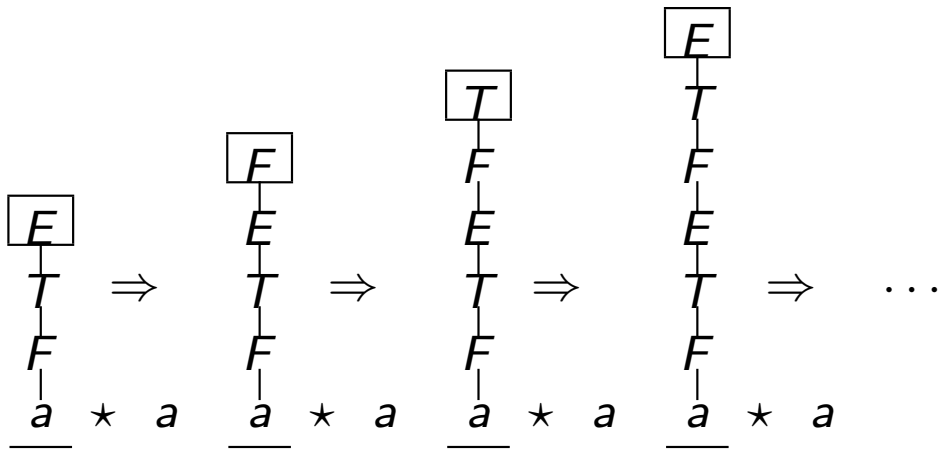


Am subliniat prefixul parcurs în cuvântul iniţial şi am încadrat simbolurile din şirul curent obţinut după eventuala reducere a unor sufixe.

Algoritmul de parsare general bottom - up

Exemplul anterior arată că la algoritmii bottom - up nu mai este necesar ca gramatica să nu fie recursivă la stânga, ca în cazul algoritmilor top -down.

Am putea avea însă probleme dacă ar exista ciclicități de forma $A \stackrel{\star}{\Rightarrow} A$, $A \in V_N$. Dacă în exemplul anterior am fi avut și producția $F \to E$, am fi intrat în recursie infinită (reamintim că reducerile se încearcă înaintea deplasărilor):



De aceea vom cere ca gramatica să nu aibe λ-producții și nici redenumiri, sau mai general să fie proprie - aceste cerințe nu restrâng generalitatea, deoarece am văzut că orice GIC cu limbaj nevid se poate transforma într-una echivalentă proprie.

Algoritmul de parsare general bottom - up

Algoritmul de parsare general bottom - up se poate formaliza astfel:

Intrare:
- $G = \langle V_N, V_T, S, P \rangle$ o GIC proprie, cu producțiile numerotate 1, ..., $|P|$;
- $w = a_1 \ldots a_n \in V_T^+$ (deci $n \geq 1$);

Ieșire:
- dacă $w \in L(G)$, atunci o analiză dreaptă a lui w (i.e. șirul numerelor producțiilor folosite într-o derivare dreaptă a sa din S);
- dacă $w \notin L(G)$, atunci eroare.

Algoritmul de parsare general bottom - up

Notații: Lucrăm cu **configurații** de forma $(s;\ i;\ \alpha;\ \beta)$, unde:

- $s \in \{q, r, t, e\}$ reprezintă starea algoritmului (stare curentă, revenire, terminare, resp. eroare);
- i este poziția curentă în șirul de intrare, care este $\$w$, \$ simbol nou; deci \$ este pe poziția 1, terminalele din w pe pozițiile 2, ..., $n+1$, iar i are valori 1, ..., $n+1$;
- $\alpha\_$ este o stivă cu vârful la dreapta, ce reține șirul curent de simboluri din V_G din care se poate deriva porțiunea din $\$w$ de la stânga lui i (inclusiv);
- $\_\beta$ este o stivă cu vârful la stânga, ce reține un istoric al deplasărilor și reducerilor care au condus la α curent.

Algoritmul de parsare general bottom - up

Configurația inițială este: $(q; 1; \$; \lambda)$.

Trecerea de la o configurație la alta se face în baza următoarelor reguli (încercate în ordinea asta):

1. (Reducere) Se aplică cât timp este posibil:
   $(q; i; \alpha\beta; \gamma) \vdash (q; i; \alpha A; j\gamma)$, unde $j : A \to \beta \in P$ este prima producție al
   cărei membru drept este sufix al lui $\alpha\beta$;
2. (Deplasare): Dacă $i < n+1$ atunci: $(q; i; \alpha; \gamma) \vdash (q; i+1; \alpha a_i; d\gamma)$ și salt la (1);
3. (Acceptare): $(q; n+1; \$S; \gamma) \vdash (t; n+1; \$S; \gamma)$ și STOP; avem $w \in L(G)$
   iar la ieșire se emite $g(\gamma)$, unde g este morfismul de monoizi (i.e.
   extinderea la șiruri) generat de funcția $g(d) = \lambda$, $g(j) = j$ $(j \in \{1, \ldots, |P|\})$;
4. (Revenire): $(q; n+1; \alpha; \gamma) \vdash (r; n+1; \alpha; \gamma)$;
5. Se aplică una din (5a) - (5e) (încercate în ordinea asta):

5a. $(r; i; \alpha A; j\gamma) \vdash (q; i; \alpha' B; k\gamma)$ și salt la (1),
   unde $j : A \to \beta$, $k : B \to \beta'$, $j < k$ și $\alpha\beta = \alpha'\beta'$;

5b. $(r; n+1; \alpha A; j\gamma) \vdash (r; n+1; \alpha\beta; \gamma)$ și salt la (5),
   unde $j : A \to \beta$;

5c. $(r; i; \alpha A; j\gamma) \vdash (q; i+1; \alpha\beta a_i; d\gamma)$ și salt la (1),
   unde $i < n+1$, $j : A \to \beta$;

5d. $(r; i; \alpha a_{i-1}; d\gamma) \vdash (r; i-1; \alpha; \gamma)$ și salt la (5), unde $i \neq 1$;

5e. $(r; 1; \$; \lambda) \vdash (e; 1; \$; \lambda)$ și se emite eroare.

Algoritmul de parsare general bottom - up

Exemplu:

Să aplicăm algoritmul pentru aceeaşi gramatică şi cuvânt ca în exemplul ilustrat grafic mai devreme:

Fie G GIC dată de: $V_N = \{E, T, F\}$
$V_T = \{a, +, \star\}$
simbolul de start: E
P: $1 : E \to E + T$
$2 : E \to T$
$3 : T \to T \star F$
$4 : T \to F$
$5 : F \to a$
(am scris în stânga numărul ca producţie).

Fie $w = a \star a$, deci avem de analizat $\begin{matrix} \$ & a & \star & a \\ & 1 & 2 & 3 & 4 \end{matrix}$

(am notat dedesubt poziţiile simbolurilor în şirul de intrare).

Atunci vom avea (am notat deasupra fiecărei "$\vdash$" numărul regulii aplicate):

Algoritmul de parsare general bottom - up

Exemplu:

$1: E \to E + T, \quad 2: E \to T, \quad 3: T \to T \star F, \quad 4: T \to F, \quad 5: F \to a, \quad \$w = \underset{1}{\$}\ \underset{2}{a}\ \underset{3}{\star}\ \underset{4}{a}$

$$(q;\ 1;\ \$;\ \lambda) \overset{2}{\vdash}$$

$$(q;\ 2;\ \$a;\ d) \overset{1}{\vdash}$$

$$(q;\ 2;\ \$F;\ 5d) \overset{1}{\vdash}$$

$$(q;\ 2;\ \$T;\ 45d) \overset{1}{\vdash}$$

$$(q;\ 2;\ \$E;\ 245d) \overset{2}{\vdash}$$

$$(q;\ 3;\ \$E\star;\ d245d) \overset{2}{\vdash}$$

$$(q;\ 4;\ \$E \star a;\ dd245d) \overset{1}{\vdash}$$

$$(q;\ 4;\ \$E \star F;\ 5dd245d) \overset{1}{\vdash}$$

$$(q;\ 4;\ \$E \star T;\ 45dd245d) \overset{1}{\vdash}$$

$$(q;\ 4;\ \$E \star E;\ 245dd245d) \overset{4}{\vdash}$$

$$(r;\ 4;\ \$E \star E;\ 245dd245d) \overset{5b}{\vdash}$$

$$(r;\ 4;\ \$E \star T;\ 45dd245d) \overset{5b}{\vdash}$$

Algoritmul de parsare general bottom - up

Exemplu:

$1 : E \to E + T$, $2 : E \to T$, $3 : T \to T \star F$, $4 : T \to F$, $5 : F \to a$, $\$w = \underset{1}{\$}\ \underset{2}{a}\ \underset{3}{\star}\ \underset{4}{a}$

$(r;\ 4;\ \$E \star F;\ 5dd245d) \overset{5b}{\vdash}$

$(r;\ 4;\ \$E \star a;\ dd245d) \overset{5d}{\vdash}$

$(r;\ 3;\ \$E\star;\ d245d) \overset{5d}{\vdash}$

$(r;\ 2;\ \$E;\ 245d) \overset{5c}{\vdash}$

$(q;\ 3;\ \$T\star;\ d45d) \overset{2}{\vdash}$

$(q;\ 4;\ \$T \star a;\ dd45d) \overset{1}{\vdash}$

$(q;\ 4;\ \$T \star F;\ 5dd45d) \overset{1}{\vdash}$

$(q;\ 4;\ \$T;\ 35dd45d) \overset{1}{\vdash}$ (merge şi producţia 4, dar prima care merge este 3)

$(q;\ 4;\ \$E;\ 235dd45d) \overset{3}{\vdash}$

$(t;\ 4;\ \$E;\ 235dd45d)$ accept

deci $a \star a \in L(G)$ iar o derivare dreaptă a sa este $g(235dd45d) = 23545$.

Să aplicăm algoritmul şi pentru $w = a+$

Algoritmul de parsare general bottom - up

Exemplu:

$1 : E \to E + T, \quad 2 : E \to T, \quad 3 : T \to T \star F, \quad 4 : T \to F, \quad 5 : F \to a, \quad \$w = \underset{1}{\$}\ \underset{2}{a}\ \underset{3}{+}$

$(q;\ 1;\ \$;\ \lambda) \overset{2}{\vdash}$

$(q;\ 2;\ \$a;\ d) \overset{1}{\vdash}$

$(q;\ 2;\ \$F;\ 5d) \overset{1}{\vdash}$

$(q;\ 2;\ \$T;\ 45d) \overset{1}{\vdash}$

$(q;\ 2;\ \$E;\ 245d) \overset{2}{\vdash}$

$(q;\ 3;\ \$E+;\ d245d) \overset{4}{\vdash}$

$(r;\ 3;\ \$E+;\ d245d) \overset{5d}{\vdash}$

$(r;\ 2;\ \$E;\ 245d) \overset{5c}{\vdash}$

$(q;\ 3;\ \$T+;\ d45d) \overset{4}{\vdash}$

$(r;\ 3;\ \$T+;\ d45d) \overset{5d}{\vdash}$

$(r;\ 2;\ \$T;\ 45d) \overset{5c}{\vdash}$

$(q;\ 3;\ \$F+;\ d5d) \overset{4}{\vdash}$

Algoritmul de parsare general bottom - up

Exemplu:

$1 : E \to E + T$, $2 : E \to T$, $3 : T \to T \star F$, $4 : T \to F$, $5 : F \to a$, $\$w = \underset{1}{\$}\ \underset{2}{a}\ \underset{3}{+}$

$(r;\ 3;\ \$F+;\ d5d) \overset{5d}{\vdash}$

$(r;\ 2;\ \$F;\ 5d) \overset{5c}{\vdash}$

$(q;\ 3;\ \$a+;\ dd) \overset{4}{\vdash}$

$(r;\ 3;\ \$a+;\ dd) \overset{5d}{\vdash}$

$(r;\ 2;\ \$a;\ d) \overset{5d}{\vdash}$

$(r;\ 1;\ \$;\ \lambda) \overset{5e}{\vdash}$

$(e;\ 1;\ \$;\ \lambda)$ error

deci $a+ \notin L(G)$.

Cuprins

Gramatici de tip LR

Fie $G = \langle V_N, V_T, S, P \rangle$ o GIC.

Definiție:

G este **de tip** $LR(k)$, $k \in \mathbf{N}$, *dacă:*

- *S nu apare în membrul drept al nici unei producții (aceasta se poate rezolva ușor, introducând un nou simbol de start S' și producția $S' \to S$);*
- *pentru orice derivări drepte* $\begin{cases} S \stackrel{\star}{\Rightarrow}_d \alpha A w \Rightarrow_d \alpha\beta w \\ S \stackrel{\star}{\Rightarrow}_d \gamma B x \Rightarrow_d \alpha\beta y \end{cases}$ *a.î.* $w, x, y \in V_T^\star$ *și*

$First_k(w) = First_k(y)$, *avem* $\begin{cases} \alpha = \gamma \\ A = B \\ x = y \end{cases}$

Obs: deci, dacă am redus un prefix al unui cuvânt din limbajul generat la un șir (ce poate fi format din terminale și neterminale) $\alpha\beta$, rămânând încă neparcurse în cuvânt niște terminale (din w sau y), atunci e suficient să privim k terminale înainte printre acestea ca să știm exact dacă avem de făcut o reducere și cu ce producție.

Gramatici de tip LR

Următoarea propoziție dă un criteriu de a determina mai ușor dacă o gramatică este de tip $LR(k)$:

Fie G o GIC a.î. S nu apare în membrul drept al nici unei producții.
Numerotăm producțiile cu 1, ..., $|P|$.

Pentru orice producție $i : A \to \beta$ și orice $u \in V_T^\star$ notăm:

$$R_k(i,u) = \{\alpha\beta u \,:\, S \Rightarrow_d^\star \alpha A w \Rightarrow_d^i \alpha\beta w,\ w \in V_T^\star,\ \{u\} = First_k(w)\}$$

($\Rightarrow_d^i$ înseamnă derivare dreaptă directă cu producția i; se poate demonstra că $R_k(i,u)$ este o mulțime regulată).

Propoziție:

G este de tip $LR(k)$ d.d.

1. *S nu apare în membrul drept al nici unei producții.*
2. *pentru orice u, v, y, z, dacă $u \in R_k(i,y)$ și $uv \in R_k(j,z)$, atunci $v = \lambda$ și $i = j$.*

Gramatici de tip LR

Următoarea propoziție condensează alte câteva proprietăți ale gramaticilor de tip $LR(k)$:

Propoziție:

Fie G o GIC.

- *G este de tip $LR(k)$, $k \geq 1$ d.d. G este de tip $LR(1)$;*
- *dacă G este de tip $LR(0)$ atunci G este de tip $LR(1)$ (nu este valabil și reciproc);*
- *dacă G este de tip $LL(k)$ atunci G este de tip $LR(k)$ ($k \in \mathbf{N}$);*
- *dacă G este de tip $LR(k)$ atunci G este neambiguă ($k \in \mathbf{N}$).*

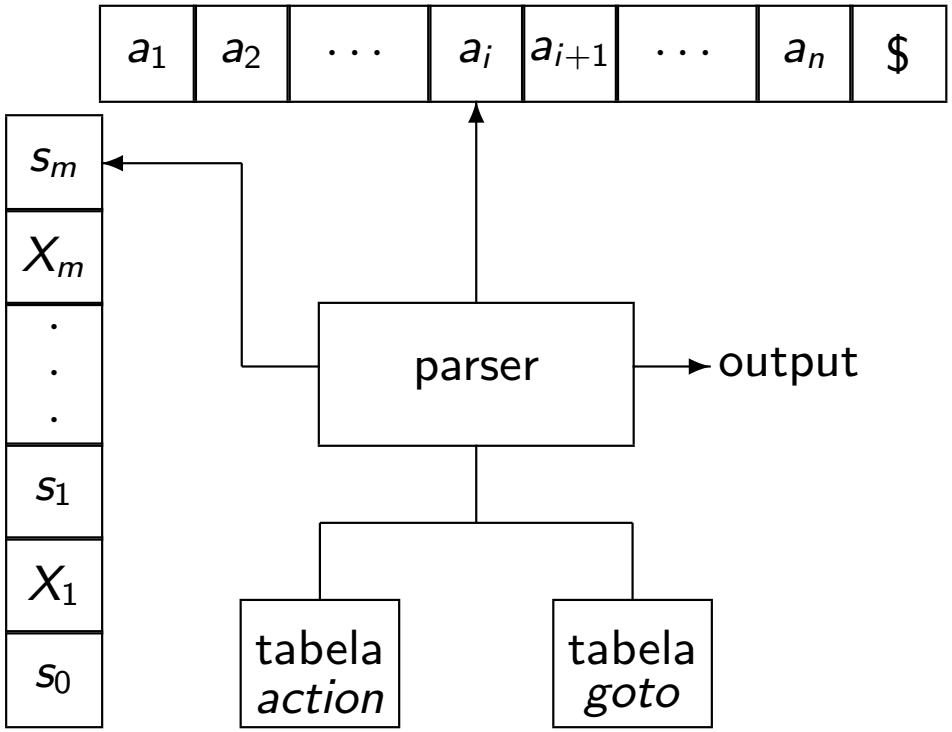
Observații: din propoziția de mai sus rezultă următoarele:

• în studiul gramaticilor de tip $LR(k)$ este suficient să ne limităm la cazurile $k = 0$ și $k = 1$, spre deosebire de gramaticile de tip $LL(k)$, unde aveam câte un caz relevant pentru fiecare k;

• condiția LR este mai slabă decât condiția LL și deci mai multe gramatici se pot încadra aici - știm să facem parsarea pentru mai multe gramatici.

Gramatici de tip LR

Pentru a ilustra ideea algoritmului de parsare pentru gramatici de tip $LR(k)$ (e suficient să considerăm $k = 0$ sau $k = 1$), imaginăm o mașină:



Mașina (algoritmul) funcționează astfel:

1. Stiva se inițializează cu starea inițială 0; st. curentă este mereu cea din vf. stivei.
2. St. curentă s_m și terminalul curent din șirul de intrare a_i se caută în tabela *action*, iar $action(s_m, a_i)$ dă acțiunea ce se va face în continuare; ea poate fi:
   - o deplasare ds_k (d =deplasare, s_k este o stare); în acest caz: se avansează un terminal în șirul de intrare ($i \to i+1$) iar în stivă se adaugă a_i și apoi s_k;
   - o reducere rm (r =reducere, m este nr. unei producții $m : A \to X_1 \dots X_n$); în acest caz:

numărul m este scris la ieșire (output), apoi din vârful stivei sunt scoase primele n stări și simbolurile de sub ele (vom vedea că mereu aceste simboluri vor fi exact X_1, ..., X_n (i.e. mb. drept al prod. m), ceea ce ne-ar permite să reținem în stivă doar stări), apoi perechea (s, A) (unde s este starea rămasă în vârful stivei) se caută în tabela *goto* care va da o nouă stare $s' = goto(s, A)$, iar in final se adaugă în stivă A și apoi s'.

Cuprins

Gramatici de tip LR(0)

Construcția tabelelor *action* și *goto* în cazul $LR(0)$ se face astfel:

Numim **configurație** $LR(0)$ (sau **item** $LR(0)$, pe scurt **item**) orice producție a gramaticii considerate, cu un caracter special "." inserat în membrul drept.

Exemplu:
producția $E \to E + T$ are 4 itemuri: $E \to .E + T$
$E \to E. + T$
$E \to E + .T$
$E \to E + T.$
producția $A \to \lambda$ are un singur item: $A \to .$

Cu asemenea itemuri definim stările parserului; de ex. itemul $E \to E. + T$ spune că parserul a recunoscut în șirul de intrare un șir generat din E, iar acum se așteaptă să citească un "+", urmat de un șir generat de T (E și T fiind neterminale iar $+$ un terminal) - atunci totul se va reduce cu această producție în E.
Apar următoarele probleme:

Gramatici de tip LR(0)

• De obicei nu este posibil să caracterizăm starea parserului cu un singur item, deoarece s-ar putea să nu știm dinainte ce producție se va aplica pentru reducere.

De exemplu dacă există și producția $E \to E \star T$ ($\star$ este un terminal), atunci ambele itemuri $E \to E. + T$, $E \to E. \star T$ vor fi aplicabile după ce parserul a citit un șir generat de E. De aceea starea respectivă a parserului va fi descrisă de o mulțime de itemuri, în acest caz $\{E \to E. + T, E \to E. \star T\}$.

Gramatici de tip LR(0)

• Dacă într-o stare avem un item cu "." în fața unui neterminal, de ex. $E \to E + .B$ (E, B neterminale, $+$ terminal), acesta arată că parserul se așteaptă să citească un șir generat de B; ca să știm însă dacă și cum putem aplica itemul ar trebui să știm cum începe un asemenea șir - să-i știm primul terminal; atunci, la starea respectivă trebuie adăugate și itemurile care permit procesarea în continuare a lui B.

Astfel, dacă într-o mulțime de itemuri există vreunul de forma $A \to v.Bw$, $B \in V_N$, iar în gramatică există o producție de forma $B \to w'$, atunci la mulțime trebuie adăugat și itemul $B \to .w'$; dacă w' începe cu un neterminal, atunci trebuie adăugate și itemurile similare corespunzătoare acestuia, ș.a.m.d., până nu se mai adaugă noi itemuri.

Obținem ceea ce s.n. **închiderea** (**closure**) mulțimii de itemuri (extensia minimală care se poate obține așa). Aceste închideri vor fi de fapt stările parserului.

Nu vom genera însă toate stările posibile ci doar pe cele accesibile (i.e. care se pot atinge efectiv din starea inițială).

În acest moment avem tot suportul intuitiv necesar pentru a descrie algoritmul de parsare.

Gramatici de tip LR(0)

Observație:

În cazul $LR(1)$ în definiţia itemurilor se ţine cont şi de un simbol "**lookahead**", care se presupune a fi consultat înainte.

Exemplu:

$$A \to \alpha.\beta; a$$

Gramatici de tip LR(0)

Algoritmul de parsare pentru gramatici de tip $LR(0)$ se poate formaliza astfel:

Intrare: • $G = \langle V_N, V_T, S, P \rangle$ o GIC de tip $LR(0)$;
• $w \in V_T^\star$;

Ieșire: • dacă $w \in L(G)$, atunci o analiză dreaptă a lui w;
• dacă $w \notin L(G)$, atunci eroare.

• Adăugăm un simbol de start nou S', producția $S' \to S$ și un terminal nou \$, care va fi pus la sfârșitul șirului de intrare.

Gramatici de tip LR(0)

- Folosim funcția *closure*(I) pentru a obține închiderea mulțimii de itemuri I:

```
function closure(I)
begin
    J ← I;
    repeat
        for [fiecare A → α.Bβ ∈ J] do
            for [fiecare B → γ ∈ P] do
                if B → .γ ∉ J then
                    J ← J ∪ {B → .γ}
    until [nu se mai adaugă itemuri];
    closure ← J
end;
```

(obs. că $closure(closure(I)) = closure(I)$)

Gramatici de tip LR(0)

Folosim funcția $go(I, X)$ pentru a determina mulțimea de itemuri în care se ajunge dacă din mulțimea de itemuri I procesez simbolul $X \in V_G$:

function $go(I, X)$
begin
    $go \leftarrow closure(\{A \rightarrow \alpha X.\beta \ : \ A \rightarrow \alpha.X\beta \in I\})$
end;

(deocamdată nu am generat stări, am descris niște instrumente cu care vom genera)

Gramatici de tip LR(0)

- Calculăm **mulțimile canonice** de tip $LR(0)$:

$C \leftarrow \{closure(\{S' \rightarrow .S\})\}$;
repeat
  for [fiecare $I \in C$] do
    for [fiecare $X \in V_G$] do
      if $go(I, X) \neq \emptyset$ and $go(I, X) \notin C$ then
        $C \leftarrow C \cup \{go(I, X)\}$
until [nu se mai adaugă noi mulțimi]

Gramatici de tip LR(0)

- Calculăm tabelele *action* și *goto*:

1. Notăm $C = \{I_0, \ldots, I_n\}$ a.î. $I_0 = closure(\{S' \to .S\})$, și pentru fiecare $I_k \in C$ considerăm starea $k \in \{0, \ldots, n\}$;
   Starea inițială este 0 (și deci corespunde lui $closure(\{S' \to .S\})$).
   Numerotăm producțiile sub forma $m : A \to \alpha$, $m \in \{0, \ldots, |P|\}$, a.î. $0 : S' \to S$ (deci producțiile din P au numere ≥ 1).
2. Efectuăm:

for [fiecare $k \in \{0, \ldots, n\}$] do begin
  for [fiecare $A \to \alpha.a\beta \in I_k$ a.î. $a \in V_T$ (deci fără \$) și $go(I_k, a) = I_j$] do
    $action(k, a) \leftarrow dj$; (deplasare j)
  for [fiecare $A \to \alpha. \in I_k$ cu $m : A \to \alpha \in P$ și $m > 0$ (deci fără $S' \to S$)] do
    for [fiecare $a \in V_T \cup \{\$\}$] do
      $action(k, a) = rm$; (reducere cu producția m)
  if $S' \to S. \in I_k$ then
    $action(k, \$) \leftarrow accept$;
  for [fiecare $A \to \alpha.X\beta \in I_k$ cu $X \in V_N$ și $go(I_k, X) = I_j$] do
    $goto(k, X) \leftarrow j$
end

3. Toate intrările pentru *action* și *goto* nesetate la pasul 2 se setează cu *error*.

Gramatici de tip LR(0)

Propoziție:

O GIC G este de tip LR(0) *d.d. făcând pentru ea construcțiile de mai sus, tabela action nu are intrări multiple.*

Domeniul de definiție al celor două tabele este următorul:

$$action : \text{Stări} \times (V_T \cup \{\$\}), \quad goto : \text{Stări} \times V_N$$

Gramatici de tip LR(0)

- Lucrăm cu **configurații** de forma $(\beta;\ \alpha;\ \pi)$, unde:
  - β_ este o stivă cu vârful la dreapta, ce conține stări intercalate cu simboluri (un istoric al deplasărilor și reducerilor), de forma $s_0X_1s_1 \ldots X_ms_m$;
  - _α este o stivă cu vârful la stânga, ce conține șirul de analizat completat la dreapta cu \$;
  - _π este o stivă cu vârful la stânga, ce va conține o analiză dreaptă (se va construi spre stânga și se va citi spre dreapta).

- **Configurația inițială este:** $(0;\ w\$;\ \lambda)$.

- **Trecerea de la o configurație la alta** se face în baza următoarelor reguli (în fiecare moment se poate aplica doar una dintre ele):

1. $(0X_1s_1 \ldots X_ms_m;\ a_ia_{i+1} \ldots a_n\$;\ \pi) \vdash (0X_1s_1 \ldots X_ms_ma_is';\ a_{i+1} \ldots a_n\$;\ \pi)$, dacă $action(s_m, a_i) = ds'$;
2. $(0X_1s_1 \ldots s_{j-1}X_js_j \ldots X_ms_m;\ a_ia_{i+1} \ldots a_n\$;\ \pi) \vdash$
   $(0X_1s_1 \ldots s_{j-1}As;\ a_ia_{i+1} \ldots a_n\$;\ k\pi)$, dacă $\begin{cases} action(s_m, a_i) = rk \\ k : A \to X_j \ldots X_m \\ goto(s_{j-1}, A) = s \end{cases}$
3. $(0Ss_1;\ \$;\ \pi) \vdash accept$ (i.e. $action(s_1, \$) = accept$); analiza dreaptă se obține citind π de la stânga la dreapta;
4. $(0X_1s_1 \ldots X_ms_m;\ a_ia_{i+1} \ldots a_n\$;\ \pi) \vdash error$, altfel (i.e. $action(s_m, a_i) = error$).

Gramatici de tip LR(0)

Exemplu:

Să aplicăm algoritmul pentru următoarea GIC care generează expresii aritmetice binare:

$1 : E \to E \star B$, $2 : E \to E + B$, $3 : E \to B$, $4 : B \to 0$, $5 : B \to 1$
(E şi B sunt neterminale, E simbol de start, $+$, $\star$, 0 şi 1 sunt terminale).

şi pentru cuvântul $w = 1 + 1$.

Adăugăm noul simbol de start S şi noul terminal de încheiere \$.

Vom calcula simultan C, *go*, *action*, *goto* şi vom da numere stărilor pe măsură ce le vom obţine.
De asemenea, la scrierea unei stări, itemurile generatoare (din care se determină celelalte prin închidere) le vom scrie mai la dreapta.

Pentru a nu confunda terminalele 0 şi 1 cu stările 0, 1, vom scrie terminalele subliniat: $\underline{0}$, $\underline{1}$.

Gramatici de tip LR(0)

Exemplu:

$1 : E \to E \star B$, $2 : E \to E + B$, $3 : E \to B$, $4 : B \to \underline{0}$, $5 : B \to \underline{1}$

Starea 0 (mulţimea I_0):

$S \to .E$

$E \to .E \star B$

$E \to .E + B$

$E \to .B$

$B \to .\underline{0}$

$B \to .\underline{1}$

În starea 0 "." apare în faţa terminalelor $\underline{0}$ şi $\underline{1}$ şi a neterminalelor E şi B, deci doar cu acestea este definită funcţia go:

$go(I_0, \underline{0}) = closure(\{B \to \underline{0}.\}) \stackrel{not}{=} I_1$ (o construim şi obs. că e o stare nouă);

$go(I_0, \underline{1}) = closure(\{B \to \underline{1}.\}) \stackrel{not}{=} I_2$;

$action(0, \underline{0}) = d1$; $action(0, \underline{1}) = d2$;

$go(I_0, E) = closure(\{S \to E., E \to E. \star B, E \to E. + B\}) \stackrel{not}{=} I_3$;

$go(I_0, B) = closure(\{E \to B.\}) \stackrel{not}{=} I_4$.

$goto(0, E) = 3$; $goto(0, B) = 4$;

Nu avem itemuri terminate cu ".", deci nu avem reduceri, şi nici itemul $S \to E.$ deci nici $accept$.

Gramatici de tip LR(0)

Exemplu:

$1 : E \to E \star B$, $2 : E \to E + B$, $3 : E \to B$, $4 : B \to \underline{0}$, $5 : B \to \underline{1}$

Starea 1 (mulţimea I_1):
$B \to \underline{0}.$

Din starea 1 nu avem *go*, dar avem un item terminat cu ".", anume $B \to \underline{0}.$, ce corespunde producţiei 4 deci avem reduceri:

$action(1, \star) = action(1, +) = action(1, \underline{0}) = action(1, \underline{1}) = action(1, \$) = r4$

nu avem itemul $S \to E.$ (pentru *accept*).

Starea 2 (mulţimea I_2):
$B \to \underline{1}.$

$action(2, \star) = action(2, +) = action(2, \underline{0}) = action(2, \underline{1}) = action(2, \$) = r5$

Gramatici de tip LR(0)

Exemplu:

$1 : E \to E \star B$, $2 : E \to E + B$, $3 : E \to B$, $4 : B \to \underline{0}$, $5 : B \to \underline{1}$

Starea 3 (mulţimea I_3):
$S \to E.$
$E \to E. \star B$
$E \to E. + B$

(conţine doar itemurile generatoare, deoarece "." nu apare în faţa vreunui neterminal)

$go(I_3, \star) = closure(\{E \to E \star .B\}) \stackrel{not}{=} I_5$;
$action(3, \star) = d5$;
$go(I_3, +) = closure(\{E \to E + .B\}) \stackrel{not}{=} I_6$;
$action(3, +) = d6$;

În starea 3 nu avem reduceri - chiar dacă avem un item terminat cu ".", el corespunde producţiei $S \to E$ care nu este din gramatica iniţială; avem însă o acceptare:

$action(3, \$) = accept$;

Gramatici de tip LR(0)

Exemplu:

$1 : E \to E \star B,\ 2 : E \to E + B,\ 3 : E \to B,\ 4 : B \to \underline{0},\ 5 : B \to \underline{1}$

Starea 4 (mulţimea I_4):
$E \to B.$

$action(4, \star) = action(4, +) = action(4, \underline{0}) = action(4, \underline{1}) = action(4, \$) = r3$

Starea 5 (mulţimea I_5):
$E \to E \star .B$
$B \to .\underline{0}$
$B \to .\underline{1}$

$go(I_5, \underline{0}) = closure(\{B \to \underline{0}.\}) = I_1;$
$action(5, \underline{0}) = d1;$
$go(I_5, \underline{1}) = closure(\{B \to \underline{1}.\}) = I_2;$
$action(5, \underline{1}) = d2;$
$go(I_5, B) = closure(\{E \to E \star B.\}) \overset{not}{=} I_7;$
$goto(5, B) = 7;$

Gramatici de tip LR(0)

Exemplu:

$1 : E \to E \star B,\ 2 : E \to E + B,\ 3 : E \to B,\ 4 : B \to \underline{0},\ 5 : B \to \underline{1}$

Starea 6 (mulţimea I_6):

$E \to E + .B$

$B \to .\underline{0}$

$B \to .\underline{1}$

$go(I_6, \underline{0}) = closure(\{B \to \underline{0}.\}) = I_1;\ action(6, \underline{0}) = d1;$

$go(I_6, \underline{1}) = closure(\{B \to \underline{1}.\}) = I_2;\ action(6, \underline{1}) = d2;$

$go(I_6, B) = closure(\{E \to E + B.\}) \overset{not}{=} I_8;\ goto(6, B) = 8;$

Starea 7 (mulţimea I_7):

$E \to E \star B.$

$action(7, \star) = action(7, +) = action(7, \underline{0}) = action(7, \underline{1}) = action(7, \$) = r1$

Starea 8 (mulţimea I_8):

$E \to E + B.$

$action(8, \star) = action(8, +) = action(8, \underline{0}) = action(8, \underline{1}) = action(8, \$) = r2$

Gramatici de tip LR(0)

Exemplu:

Avem deci tabelele *action* şi *goto* următoare (intrările necompletate sunt *error*):

	action					*goto*	
Stare	$\star$	$+$	$\underline{0}$	$\underline{1}$	\$	E	B
0			$d1$	$d2$		3	4
1	$r4$	$r4$	$r4$	$r4$	$r4$		
2	$r5$	$r5$	$r5$	$r5$	$r5$		
3	$d5$	$d6$			acc		
4	$r3$	$r3$	$r3$	$r3$	$r3$		
5			$d1$	$d2$			7
6			$d1$	$d2$			8
7	$r1$	$r1$	$r1$	$r1$	$r1$		
8	$r2$	$r2$	$r2$	$r2$	$r2$		

Gramatici de tip LR(0)

Exemplu:

Observaţii:

• Nu au rezultat intrări multiple, ceea ce arată că gramatica este de tip $LR(0)$.

• În stările din care se fac reduceri apare acelaşi *rm* pe toată linia, astfel că reducerea folosită nu depinde de terminalul citit înainte în cuvântul analizat - asta înseamnă $LR(0)$: consultă 0 terminale înainte pentru a decide reducerea folosită.

Dacă gramatica necesită consultarea unui terminal înainte pentru a nu avea reduceri ambigue, tabelul ar avea *rm*-uri diferite în coloane diferite, iar algoritmul de mai sus nu poate produce asemenea linii de tabel - algoritmul trebuie rafinat.

Cu asemenea algoritmi se pot parsa mai multe gramatici decât cu cel din cazul $LR(0)$.

Gramatici de tip LR(0)

Exemplu:

- De exemplu putem înlocui:

  for [fiecare $A \to \alpha. \in I_k$ cu $m : A \to \alpha \in P$ şi $m > 0$] do
  for [fiecare $a \in V_T \cup \{\$\}$] do
  $action(k, a) = rm$;

  cu:

  for [fiecare $A \to \alpha. \in I_k$ cu $m : A \to \alpha \in P$ şi $m > 0$] do
  for [fiecare $a \in Follow_1(A)$] do
  $action(k, a) = rm$;

  iar gramaticile pentru care acest algoritm produce tabela *action* fără intrări multiple s.n. **gramatici de tip** $SLR(1)$ (**simple** $LR(1)$).

  Se demonstrează că orice gramatică de tip $LR(0)$ este şi de tip $SLR(1)$, iar orice gramatică de tip $SLR(1)$ este şi de tip $LR(1)$.

Gramatici de tip LR(0)

Exemplu:

$1 : E \to E \star B$, $2 : E \to E + B$, $3 : E \to B$, $4 : B \to \underline{0}$, $5 : B \to \underline{1}$

	action					*goto*	
Stare	$\star$	$+$	$\underline{0}$	$\underline{1}$	\$	E	B
0			$d1$	$d2$		3	4
1	$r4$	$r4$	$r4$	$r4$	$r4$		
2	$r5$	$r5$	$r5$	$r5$	$r5$		
3	$d5$	$d6$			acc		
4	$r3$	$r3$	$r3$	$r3$	$r3$		
5			$d1$	$d2$			7
6			$d1$	$d2$			8
7	$r1$	$r1$	$r1$	$r1$	$r1$		
8	$r2$	$r2$	$r2$	$r2$	$r2$		

Să parsăm cuvântul $w = \underline{1} + \underline{1}$ (am scris deasupra fiecărui $\vdash$ regula folosită):

$(0;\ \underline{1} + \underline{1}\$;\ \lambda) \overset{1}{\vdash}$ $\quad (action(0, \underline{1}) = d2)$

$(0\underline{1}2;\ +\underline{1}\$;\ \lambda) \overset{2}{\vdash}$ $\quad (action(2, +) = r5,\ 5 : B \to \underline{1},\ goto(0, B) = 4)$

$(0B4;\ +\underline{1}\$;\ 5) \overset{2}{\vdash}$

$(0E3;\ +\underline{1}\$;\ 35) \overset{1}{\vdash}$

$(0E3 + 6;\ \underline{1}\$;\ 35) \overset{1}{\vdash}$

Gramatici de tip LR(0)

Exemplu:

$1 : E \to E \star B$, $2 : E \to E + B$, $3 : E \to B$, $4 : B \to \underline{0}$, $5 : B \to \underline{1}$

	action					*goto*	
Stare	$\star$	$+$	$\underline{0}$	$\underline{1}$	\$	E	B
0			$d1$	$d2$		3	4
1	$r4$	$r4$	$r4$	$r4$	$r4$		
2	$r5$	$r5$	$r5$	$r5$	$r5$		
3	$d5$	$d6$			acc		
4	$r3$	$r3$	$r3$	$r3$	$r3$		
5			$d1$	$d2$			7
6			$d1$	$d2$			8
7	$r1$	$r1$	$r1$	$r1$	$r1$		
8	$r2$	$r2$	$r2$	$r2$	$r2$		

$(0E3 + 6\underline{1}2;\ \$;\ 35) \overset{2}{\vdash}$

$(0E3 + 6B8;\ \$;\ 535) \overset{2}{\vdash}$

$(0E3;\ \$;\ 2535) \overset{3}{\vdash} accept \qquad (action(3, \$) = accept)$

iar analiza dreaptă este: 2535

Exerciţiu: desenaţi arborele de derivare pornind de la această analiză dreaptă.

Gramatici de tip LR(0)

Am spus că o GIC este de tip $LR(0)$ d.d. tabela *action* nu are intrări multiple (**conflicte**).

Riscul să apară conflicte există doar la adăugarea reducerilor - pot apărea conflicte *dr* sau *rr*.

Uneori însă (ca și în cazul LL) conflictele pot fi rezolvate. Prezentăm în continuare tipurile posibile de conflicte și modul lor de rezolvare:

Gramatici de tip LR(0)

- **Conflicte** *dr* - exemplu:

Dacă gramatica este: $1 : E \to aE$
$2 : E \to a$
(simbol de start E, terminal a)

atunci avem:

starea 0 :
$S \to .E$
$E \to .aE$
$E \to .a$

starea 1 :
$E \to a.E$
$E \to a.$
$E \to .aE$
$E \to .a$

Din starea 0 avem tranziții *go* cu a și E; cu a obținem starea 1.
Din starea 1 avem tranziții *go* cu a și E, iar cu a ajungem tot în starea 1; astfel, $action(1, a) = d1$.
Totodată, în starea 1 avem și itemul $E \to a.$ care determină setarea întregii linii a lui 1 din tabela *action* cu r2; în particular, $action(1, a) = r2$.

Gramatici de tip LR(0)

- **Conflicte** *rr* - exemplu:

Dacă gramatica este:

1 : $E \to Aa$

2 : $E \to Bb$

3 : $A \to a$

4 : $B \to a$

(neterminale E, A, B, simbol de start E, terminale a, b)

atunci avem:

starea 0 :

$S \to .E$

$E \to .Aa$

$E \to .Bb$

$A \to .a$

$B \to .a$

starea 1 :

$A \to a.$

$B \to a.$

Din starea 0 avem tranziții *go* cu E, A, B și a; cu a obținem starea 1.
Starea 1 conține două itemuri terminate cu " .", care conduc la setarea liniei lui 1 din tabela *action* atât cu $r3$ cât și cu $r4$.

Gramatici de tip LR(0)

Ambele tipuri de conflicte (*dr* și *rr*) se pot rezolva dacă permitem parserului să consulte mulțimile $Follow_1(A)$, $A \in V_N$, pentru a decide dacă va aplica o reducere cu una din alternativele lui A.
El va face o reducere cu producția $A \to \alpha$ doar dacă următorul simbol din șirul analizat este în $Follow_1(A)$.

Această soluție conduce la așa-numitele **Simple LR parsers** (pentru gramaticile $SLR(1)$ - ele sunt o specie de gramatici $LR(1)$ mai constrânsă decât $LR(1)$ general dar mai relaxată decât $LR(0)$).

Analiza sintactică bottom - up

Teme laborator:

1. Implementați algoritmul de parsare general bottom - up.
Programul va citi o GIC (presupusă a fi proprie), apoi într-un ciclu va citi diverse cuvinte și va determina dacă fac parte din limbajul generat de gramatică; în caz afirmativ, va afișa și producțiile aplicate într-o derivare dreaptă a sa.

2. Implementați algoritmul de parsare general bottom - up folosind o procedură/funcție recursivă, care la fiecare apel să facă o derivare. Celelalte detalii sunt ca la problema 1.

3. Implementați algoritmul de parsare pentru gramatici de tip $LR(0)$.
Programul va citi o GIC, va construi tabelele *action* și *goto*, determinând dacă gramatica este $LR(0)$, apoi într-un ciclu va citi diverse cuvinte și va determina dacă fac parte din limbajul generat de gramatică; în caz afirmativ, va afișa și producțiile aplicate într-o derivare dreaptă a sa.

4. Implementați algoritmul de parsare pentru gramatici de tip $SLR(1)$. Detaliile sunt ca la problema 3.