TEHNICI DE COMPILARE – CURSUL 8

GRAMATICI DE TIP $LALR(1)$

Fie $G = (N, \Sigma, S, P)$ gramatică independentă de context de tip $LR(1)$, extensia sa $G' = (N \cup \{S'\}, \Sigma \cup \{\#\}, S', P \cup \{S' \to S\})$ cu $L(G) = L(G')$. Fie $C_G = \{I_0, I_1, \dots, I_n\}$ mulțimile canonice $LR(1)$ asociate, $I_0 = closure(\{S' \to .S; \#\})$.

Definiție. Definim nucleul unei mulțimi de configurații din C_G astfel:

- $S' \to .S$ este nucleul lui I_0
- Fie $I_j \in C_G, X \in N \cup \Sigma$ astfel încât $goto(I_j, X) \neq \emptyset$. Atunci
$$K = \{A \to \alpha X.\beta \mid \exists A \to \alpha.X\beta; a \in I_j\}$$
este nucleul multimii canonice $goto(I_j, X)$.

Definiție. Fie $G = (N, \Sigma, S, P)$ gramatică $LR(1)$, $C_G = \{I_0, I_1, \dots, I_n\}$, mulțimile canonice $LR(1)$ asociate. Partiționăm C_G în mulțimi disjuncte două câte două, $G_G = S_1 \cup \dots \cup S_m$, unde pentru orice $i, 1 \leq i \leq m, S_i$ conține toate mulțimile canonice $LR(1)$ din C_G care au același nucleu, K_i. Evident, $S_i \cap S_j = \emptyset, K_i \neq K_j, \forall i \neq j, 1 \leq i, j \leq m$.

Fie S_i o astfel de mulțime, $S_i = \{J_1, \dots, J_{s_i}\} \subseteq C_G$, unde $J_1, \dots, J_{s_i}$ au același nucleu, K_i. Pentru fiecare configurație $A \to \alpha \to .\beta \in K_i$ vom reuni simbolurile lookahead ale configurațiilor $LR(1)$ corespondente din fiecare mulțime $J_1, \dots, J_{s_i}$, apoi aplicăm aceeași unificare pentru toate celelalte configurații care nu sunt în nucleu și care au fost introduse în fiecare dintre mulțimile $J_1, \dots, J_{s_i}$ prin operația $closure$, plecând de la configurații din nucleul K_i.

Se obține astfel o nouă mulțime de configurații $LR(1)$, notată cu Z_i, având nucleul K_i și care va înlocui mulțimile $J_1, \dots, J_{s_i}$.

De asemenea, tranzițiile dintre mulțimile noi $Z_1, \dots Z_m$, date de $goto$, vor fi consistente cu tranzițiile inițiale dintre mulțimile $I_0, I_1, \dots, I_n$.

Dacă prin această operație de unificare în urma căreia se obțin mulțimile $Z_1, \dots, Z_m$ nu rezultă conflicte în tabela $action$, atunci spunem că gramatica G este de tip $LALR(1)$ (LA provine de la *Look Ahead*).

Observații.

- Aceasta este o metodă folosită de majoritatea parser-elor de tip LR pentru a reduce din numărul de intrări în tabela de analiză sintactică ($action, goto$).

- Programul $bison$, care este un generator de analozoare sintactice, primește la intrare o gramatică independentă de context, folosind o anumită specificație, și produce la ieșire o tabelă $LALR(1)$.
- Prin utilizarea tabelelor $LALR(1)$ numărul de intrări în tabela de analiză sintactică poate fi redus cu până la 30%.

Exemplu. Se dă gramatica G cu producțiile:

$1: S \to AA \quad 2: A \to aA \quad 3: A \to b$

Adăugăm producția $S' \to S$ și obținem mulțimile $LR(1)$

$$I_0 = \begin{bmatrix} S' \to .S; \# & \xrightarrow{goto(I_0,S)} I_1 \\ S \to .AA; \# & \xrightarrow{goto(I_0,A)} I_2 \\ A \to .aA; a|b & \xrightarrow{goto(I_0,a)} I_3 \\ A \to .b; \to a|b & \xrightarrow{goto(I_0,b)} I_4 \end{bmatrix}$$

$I_1 = [S' \to S; \#]$

$$I_2 = \begin{bmatrix} S \to A.A; \# & \xrightarrow{goto(I_2,A)} I_5 \\ A \to .aA; \# & \xrightarrow{goto(I_2,a)} I_6 \\ A \to .b; \# & \xrightarrow{goto(I_2,b)} I_7 \end{bmatrix}$$

$$I_3 = \begin{bmatrix} A \to a.A; a|b & \xrightarrow{goto(I_3,A)} I_8 \\ A \to .aA; a|b & \xrightarrow{goto(I_3,a)} I_3 \\ A \to .b; a|b & \xrightarrow{goto(I_3,b)} I_4 \end{bmatrix}$$

$I_4 = [A \to b.; a|b]$

$I_5 = [S \to AA.; \#]$

$$I_6 = \begin{bmatrix} A \to a.A; \# & \xrightarrow{goto(I_6,A)} I_9 \\ A \to .aA; \# & \xrightarrow{goto(I_6,a)} I_6 \\ A \to .b; \# & \xrightarrow{goto(I_6,b)} I_7 \end{bmatrix}$$

$I_7 = [A \to b.; \#]$

$I_8 = [A \to aA.; a|b]$

$I_9 = [A \to aA.; \#]$

Observăm că I_3 și I_6, respectiv I_4, I_7 și I_8, I_9 au același nucleu, deci pot fi unificate. Se obțin astfel:

$$I_{36} = \begin{bmatrix} A \to a.A; a|b|\# & \xrightarrow{goto(I_{36},A)} I_{89} \\ A \to .aA; a|b|\# & \xrightarrow{goto(I_{36},a)} I_{36} \\ A \to .b; a|b|\# & \xrightarrow{goto(I_{36},b)} I_{47} \end{bmatrix}$$

$I_{47} = [A \to b.; a|b|\#] \qquad I_{89} = [A \to aA.; a|b|\#]$

Obținem tabela de simboluri:

	a	b	$\#$	S	A
0	$shift\ 36$	$shift\ 47$	$error$	1	2
1	$error$	$error$	$accept$	$error$	$error$
2	$shift\ 36$	$shift\ 47$	$error$	$error$	5
36	$shift\ 36$	$shift\ 47$	$error$	$error$	89
47	$reduce\ 3$	$reduce\ 3$	$reduce\ 3$	$error$	$error$
5	$error$	$error$	$reduce\ 1$	$error$	$error$
89	$reduce\ 2$	$reduce\ 2$	$reduce\ 2$	$error$	$error$

Tabela nou obținută nu are intrări multiple, rezultă că G este $LALR(1)$.

Un mod mai eficient de a calcula tabelele $LALR(1)$

Modul de construcție redat mai sus este destul de ineficient.

Există însă un mod mai eficient de a construi tabelele $LALR(1)$

- Ori de câte ori este creată o nouă stare (mulțime canonică) cu algoritmul $LR(1)$, se verifică imediat dacă se poate face operația de unificare.
- De obicei, operația de unificare presupune și propagarea simbolurilor $lookahead$ adăugate la starea nou construită la pasul de unificare.
  De exemplu, în exemplul anterior, în momentul când ar trebui construită I_6, aceasta are același nucleu cu I_3, deci cele două stări se reunesc, obținându-se I_{36}. Din I_3 cu b se ajunge în I_4, care era deja construită. Nu mai are sens să construim explicit I_7 (starea care s-ar fi obținut din I_6 trecând peste b, ci noile simboluri $lookahead$ aduse de I_6 (adică \$), vor fi propagate la I_4.
- Dacă mai este nevoie, această propagare se aplică în continuare.

În cele ce urmează vom considera $G = (N, \Sigma, S, P)$ gramatică independentă de context, extensia sa $G' = (N \cup \{S'\}, \Sigma \cup \{\$\}, S', P \cup \{S' \rightarrow S\})$

Definiție. Simboluri $lookahead$ propagate și simboluri $lookahead$ spontan generate. Fie I o mulțime de configurații $LR(1), B \rightarrow \gamma.C\delta; b \in I, X \in N \cup \Sigma$. Fie K nucleul lui I astfel încât $B \rightarrow \gamma.C\delta \in K$. Presupunem că $C \overset{*}{\Rightarrow}_d A\eta, A \rightarrow X\beta \in P$. Atunci $A \rightarrow X.\beta; a$ este în $goto(I, X)$ pentru anumite simboluri $a \in \Sigma \cup \{\$\}$.

Problemă: pentru ce simboluri $a \in \Sigma \cup \{\$\}, A \rightarrow X.\beta; a \in goto(I, X)$?

$B \rightarrow \gamma.C\delta; b \in I \quad B \rightarrow \gamma.C\delta \in K$
$C \overset{*}{\Rightarrow}_d A\eta$
$A \rightarrow X\beta$

Rezultă $A \rightarrow .X\beta; a \in I,\ a \in First(\eta\delta b)$.

- Dacă $a \in First(\eta\delta)$ atunci $A \rightarrow X.\beta; a \in goto(I, X)$. Simbolul a se numește *spontan generat*
- Dacă $\eta\delta \overset{*}{\Rightarrow} \lambda$ atunci $A \rightarrow X.\beta; b \in goto(I, X)$. Spunem că b este simbol lookahead propagat de la $B \rightarrow \gamma.C\delta$ la $A \rightarrow X.\beta$

ALGORITM 1: determinarea simbolurilor $lookahead$ propagate și spontan generate pentru G

INPUT: Nucleul K al unei mulțimi canonice $LR(0), I; X \in N \cup \Sigma$.

OUTPUT: Simbolurile $lookahead$ spontan generate de configurațiile din nucleul K în $goto(I, X)$, configurațiile din I de la care sunt propagate ulterior simbolurile $lookahead$ în nucleul lui $goto(I, X)$.

for (fiecare $B \rightarrow \gamma.\delta \in K$)

{ $J \leftarrow closure(\{B \rightarrow \gamma.\delta; \$\})$;

$if(A \rightarrow \alpha.X\beta; a \in J$ && $a \neq \$)$

a este spontan generat pentru $A \rightarrow \alpha X.\beta$ in $goto(I, X)$

$if(A \rightarrow \alpha.X\beta; \$ \in J)$ $simbolurile\ \ lookahead$ asociate lui $B \rightarrow \gamma.\delta$ din I

vor fi propagate la $A \rightarrow \alpha X.\beta$ din $goto(I, X)$

}

ALGORITM 2: calcularea eficientă a configurațiilor $LALR(1)$

INPUT: $G = (N, \Sigma, S, P)$ gramatică independentă de context, extensia sa $G' = (N \cup \{S'\}, \Sigma \cup \{\$\}, S', P \cup \{S' \rightarrow S\})$

OUTPUT: mulțimile canonice $LALR(1)$ pentru G

1. Se construiesc mulțimile canonice $LR(0)$ $C_G = \{I_0, \dots, I_n\}, I_0 = closure(\{S' \rightarrow .S\})$.
2. Se aplica algoritmul 1 pentru nucleul fiecărei mulțimi de configurații $LR(0)$ $I \in C_G$. Se determină astfel simbolurile lookahead spontan generate pentru configurațiile din nucleul lui $goto(I, X)$. Se determină de asemenea

configurațiile din I ale căror simboluri lookahead sunt propagate la configurațiile din nucleul lui $goto(I, X)$.

3. Se crează o tabelă care indică pentru fiecare nucleu de configurații simbolurile lookahead asociate. Inițial, fiecare configurație are asociate simbolurile lookahead spontan generate, determinate la pasul 2.
4. Se fac treceri repetate peste nucleele de configurații cât timp sunt adăugate noi simboluri lookahead. Atunci când se vizitează o mulțime I se determină nucleul către care I își va propaga mulțimea de simboluri lookahead asociate, utilizând 2. Mulțimea curentă de simboluri lookahead asociate lui I este adăugată la toate nucleele determinate la pasul 2.

Exemplu. Fie gramatica extinsă

$$S' \to S$$
$$S \to L = R \mid R$$
$$L \to * R \mid id$$
$$R \to L$$

Setul complet de multimi $LR(0)$

$$I_0 = \begin{bmatrix} S' \to .S \longrightarrow I_1 \\ S \to .L = R \longrightarrow I_2 \\ S \to .R \longrightarrow I_3 \\ L \to .* R \longrightarrow I_4 \\ L \to .id \longrightarrow I_5 \\ R \to .L \longrightarrow I_2 \end{bmatrix}$$

$$I_1 = [S' \to S.]$$

$$I_2 = \begin{bmatrix} S \to L. = R \longrightarrow I_6 \\ R \to L. \end{bmatrix}$$

$$I_3 = [S \to R.]$$

$$I_4 = \begin{bmatrix} L \to *.R \longrightarrow I_7 \\ R \to .L \longrightarrow I_8 \\ L \to .* R \longrightarrow I_4 \\ L \to .id \longrightarrow I_5 \end{bmatrix}$$

$$I_5 = [L \to id.]$$

$$I_6 = \begin{bmatrix} S \to L = .R \longrightarrow I_9 \\ R \to .L \longrightarrow I_8 \\ L \to .* R \longrightarrow I_4 \\ L \to .id \longrightarrow I_5 \end{bmatrix}$$

$$I_7 = [L \to * R.]$$

$$I_8 = [R \to L.]$$

$$I_9 = [S \to L = R.]$$

Nucleele acestor multimi sunt:

$$K_0 = [S' \to .S]$$

$$K_1 = [S' \to S.]$$

$$K_2 = \begin{bmatrix} S \to L. = R \\ R \to L. \end{bmatrix}$$

$$K_3 = [S \to R.]$$

$K_4 = [L \to *. R]$

$K_5 = [L \to id.]$

$K_6 = [S \to L =. R]$

$K_7 = [L \to * R.]$

$K_8 = [R \to L.]$

$K_9 = [S \to L = R.]$

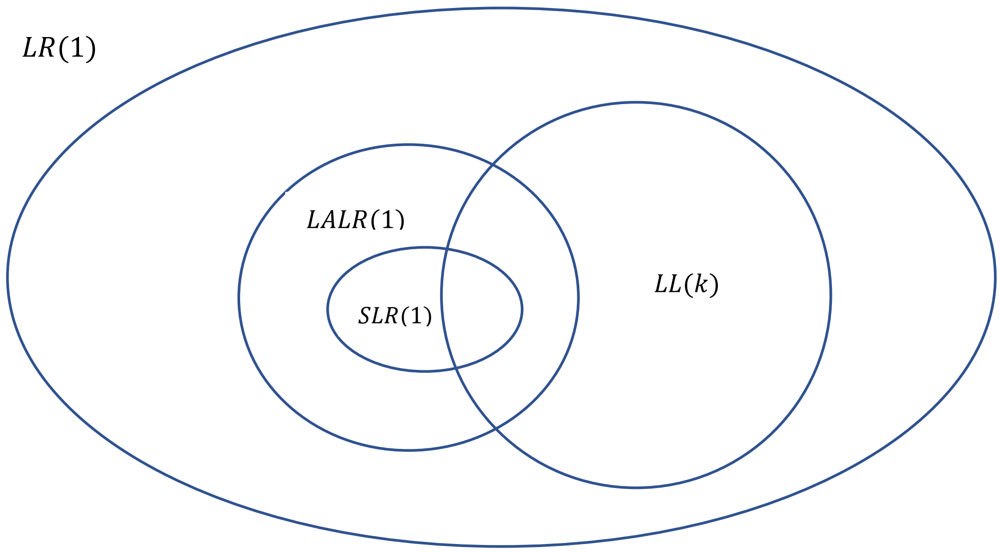
Calculam $closure(K_0; \$), closure(I_4; \$), closure(I_6; \$)$. Obtinem:

$$\begin{bmatrix} S' \to . S; \$ \\ S \to . L = R; \$ \\ S \to . R; \$ \\ L \to .* R; \$| = \\ L \to . id; \$| = \\ R \to . L; \$ \end{bmatrix} \quad \begin{bmatrix} L \to *. R; \$ \\ R \to . L; \$ \\ L \to .* R; \$ \\ L \to . id; \$ \end{bmatrix} \quad \begin{bmatrix} S \to L =. R; \$ \\ R \to L; \$ \\ L \to . R; \$ \\ L \to . id; \$ \end{bmatrix}$$

FROM		TO	
I_0:	$S' \to \cdot S$	I_1:	$S' \to S\cdot$
		I_2:	$S \to L\cdot = R$
		I_2:	$R \to L\cdot$
		I_3:	$S \to R\cdot$
		I_4:	$L \to *\cdot R$
		I_5:	$L \to \mathbf{id}\cdot$
I_2:	$S \to L\cdot = R$	I_6:	$S \to L = \cdot R$
I_4:	$L \to *\cdot R$	I_4:	$L \to *\cdot R$
		I_5:	$L \to \mathbf{id}\cdot$
		I_7:	$L \to *R\cdot$
		I_8:	$R \to L\cdot$
I_6:	$S \to L = \cdot R$	I_4:	$L \to *\cdot R$
		I_5:	$L \to \mathbf{id}\cdot$
		I_8:	$R \to L\cdot$
		I_9:	$S \to L = R\cdot$

SET	ITEM	LOOKAHEADS			
		INIT	PASS 1	PASS 2	PASS 3
I_0:	$S' \to \cdot S$	\$	\$	\$	\$
I_1:	$S' \to S\cdot$		\$	\$	\$
I_2:	$S \to L\cdot = R$		\$	\$	\$
	$R \to L\cdot$		\$	\$	\$
I_3:	$S \to R\cdot$		\$	\$	\$
I_4:	$L \to *\cdot R$	=	=/\$	=/\$	=/\$
I_5:	$L \to \mathbf{id}\cdot$	=	=/\$	=/\$	=/\$
I_6:	$S \to L = \cdot R$			\$	\$
I_7:	$L \to *R\cdot$		=	=/\$	=/\$
I_8:	$R \to L\cdot$		=	=/\$	=/\$
I_9:	$S \to L = R\cdot$				\$

Teoremă. Există următoarele incluziuni (stricte) între familiile de limbaje $LL(k), SLR(1), LALR(1), LR(1)$:



ANALIZA SEMANTICĂ

Există construcții specifice limbajelor de programare care nu pot fi descrise cu ajutorul gramaticilor independente de context. Iată câteva exemple:

1. Fie limbajul $L_1 = \{w\#w|w \in \Sigma^*\}, \Sigma$ alfabet finit, $|\Sigma| \geq 2,\ \# \notin \Sigma$. Acest limbaj abstractizează condiția ca un identificator-variabilă să fie declarat înainte de utilizare. Aici w reprezintă numele variabilei.
   Limbajul L_1 nu este un limbaj independent de context, deci această condiție nu poate fi formalizată cu o gramatică independentă de context.
2. Considerăm limbajul $L_2 = \{a^m b^m a^m c^m | n, m \geq 1\}$ care, de asemenea, nu este independent de context. L_2 abstractizează corespondența între declarația unei funcții cu parametri și utilizarea acesteia cu același număr și aceleași tipuri de parametri actuali ca cei formali

Din exemplele de mai sus desucem că prin analiza sintactică (la baza căreia avem gramatici independente de context/ translatoare stivă) nu se pot verifica condițiile din 1) și 2). Acestea vor fi rezolvate în etapa de analiză semantică.

Pentru un limbaj de programare, verificările semantice constau din:

– verificarea condiției ca un identificator să fie declarat înainte de utilizare;

– verificarea corespondenței între numărul și tipurile parametrilor actuali cu numărul și tipurile parametrilor formali;
– verificarea compatibilității tipurilor operanzilor într-o expresie etc.

Există și în aceste cazuri un model formal care descrie aceste condiții: gramaticile atributate (cu atribute).

GRAMATICI ATRIBUTATE

Definiție. O gramatică atributată (cu atribute) are o structură de forma $GA = (G, A, R, B)$, unde:

- $G = (N, \Sigma, S, P)$ este o gramatică independentă de context, numită gramatică de bază (care abstractizează sintaxa limbajului)
- $A = \bigcup_{X \in \Sigma \cup N} A(X),\ A(X)$ fiind o mulțime de atribute asociate lui X. Dacă $a \in A(X)$, atunci vom folosi notația clasică $X.a$ sau $X \uparrow a$ pentru atributele sintetizate și $X \downarrow a$ pentru atributele moștenite.
- $R = \bigcup_{p \in P} R(p), R(p)$ fiind o mulțime finită de reguli de atributare asociate producției p.
- $B = \bigcup_{q \in P} B(q), B(q)$ fiind o mulțime finită de predicate (condiții semantice) →În plus, dacă $A(X) \cap A(Y) \neq \emptyset \Longrightarrow X = Y$. Pentru fiecare apariție a lui X într-un arbore sintactic, se poate aplica cel mult o regulă pentru a calcula fiecare atribut $a \in A(X)$.

Definiție. Pentru fiecare producție $p: X_0 \to X_1 \dots X_n \in P$ mulțimea atributelor definite de p este:

$$A(p) = \{X_i.a | 0 \leq i \leq n, \exists X_i \leftarrow f(\dots) \in R(p)\}$$

unde f este o funcție ale cărei argumente sunt alte atribute.

Atributul $X.a$ este numit <u>sintetizat</u> sau <u>derivat</u> dacă există $p: X \to \alpha \in P$ și $X.a \in A(p)$. Notăm acest atribut cu $X \uparrow a$, care sugerează modul cum este obținută valoarea sa.

Atributul $Y.b$ este numit <u>moștenit</u> dacă există $q: X \to \alpha Y \beta \in P$ și $Y.b \in A(q)$. Notăm acest atribut cu $Y \downarrow b$.

Atributele sintetizate asociate unui neterminal $X \in N$ se calculează pornind de la atributele asociate nodurilor subarborelui de rădăcină X. Atributele asociate unor terminali (tokeni) sunt întotdeauna sintetizate și reprezintă anumite valori intrinseci asociate acelor tokeni, cum ar fi valoarea unei constante, și de regulă sunt furnizate de analizorul lexical.

Atributele moștenite asociate lui $X \in N$ se calculează top-down, în funcție de atributele asociate nodurilor unui subarbore în care apare X ca descendent, direct sau indirect.

Exemple de atribute:

- pentru identificatori-variabilă: tip, adresă, nume
- pentro o constantă numerică: valoarea
- pentru o expresie: tipul său

Definiție. Limbajul generat de gramatica atributată $GA = (G, A, R.B)$:

$L(GA) = \{w \in \Sigma^* | w \in L(G)$, pentru orice arbore de derivare T al lui w toate atributele pot fi evaluate și toate predicatele asociate sunt evaluate la $true\}$

Cu alte cuvinte, $L(GA)$ conține toate șirurile corecte din punct de vedere sintactic ($w \in L(G)$) și corecte din punct de vedere semantic.

Exemple.

1) $Exp \rightarrow Exp"+"Exp$
   $Exp \rightarrow Exp"OR"Exp$
   $Exp \rightarrow num$
   $Exp \rightarrow "TRUE"$
   $Exp \rightarrow "FALSE"$

   Se atributează gramatica de mai sus astfel încât neterminalul Exp are asociat un atribut $type$ ce poate lua două valori, $type \in \{int, bool\}$ și astfel încât să existe compatibilitate între tipurile operanzilor.

$$
\begin{aligned}
Exp \uparrow type \rightarrow\ & Exp \uparrow type1"+"Exp \uparrow type2 \\
& [type1 == type2; type1 == int; type \leftarrow type1] \\
\rightarrow\ & Exp \uparrow type1"\text{OR}"Exp \uparrow type2 \\
& [type1 == type2; type1 == bool; type \leftarrow type1] \\
\rightarrow\ & num \uparrow val\ [type \leftarrow int] \\
\rightarrow\ & "\text{TRUE}"[type \leftarrow bool] \\
\rightarrow\ & "FALSE"\ [type \leftarrow bool]
\end{aligned}
$$

2) Se dă gramatica cu producțiile:
   $S \rightarrow N"."N$
   $N \rightarrow NB$
   $N \rightarrow B$
   $B \rightarrow 0$
   $B \rightarrow 1$

   Vom atributa S, N, B astfel încât S să aibă asociat un atribut sintetizat v a cărui valoare va reprezenta valoarea în zecimal a numărului în virgulă mobilă din baza 2 generat de S. Astfel:

$S \upharpoonleft v$ $\quad v - valoarea\ in\ zecimal$
$N \downharpoonleft f \upharpoonleft v \upharpoonleft l$ $\quad f - factorul\ de\ scala\ (departarea\ fata\ de\ punctul\ zecimal)$
$B \downharpoonleft f \upharpoonleft v$ $\quad l - lungimea\ sirului\ de\ biti\ generat\ de\ N$

$S \upharpoonleft v \rightarrow N \downharpoonleft f1 \upharpoonleft v1 \upharpoonleft l1 "." N \downharpoonleft f2 \upharpoonleft v2 \upharpoonleft l2$

$[v \leftarrow v1 + v2; f1 \leftarrow 1; f2 \leftarrow 2^{-l2}]$

$N \downharpoonleft f \upharpoonleft v \upharpoonleft l \rightarrow N \downharpoonleft f1 \upharpoonleft v1 \upharpoonleft l1\ B \downharpoonleft f2 \upharpoonleft v2$

$[f1 \leftarrow 2 * f; f2 \leftarrow f; v \leftarrow v1 + v2; l \leftarrow l1 + 1]$

$B \downharpoonleft f1 \upharpoonleft v1\ [f1 \leftarrow f; v \leftarrow v1; l \leftarrow 1]$

$B \downharpoonleft f \upharpoonleft v \rightarrow 0\ [v \leftarrow 0]$

$\rightarrow 1\ [v \leftarrow f]$