Script generated by TTT

Title: Simon: Compilerbau (13.05.2013)

Date: Mon May 13 14:21:18 CEST 2013

Duration: 83:21 min

Pages: 33

Lookahead Sets

For $\alpha \in (N \cup T)^*$ we are interested in the set:

$$\text{First}_1(\alpha) = \text{First}_1(\{w \in T^* \mid \alpha \rightarrow^* w\})$$

Idea: Treat ϵ separately: F_ϵ

- Let $\text{empty}(X) = \text{true}$ iff $X \rightarrow^* \epsilon$.
- $F_\epsilon(X_1 \dots X_m) = \bigcup_{i=1}^{j} F_\epsilon(X_i)$ if $\text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_{j-1})$

We characterize the ϵ-free First_1-sets with an inequality system:

$$\begin{array}{lllll} F_\epsilon(a) & = & \{a\} & \text{if} & a \in T \\ F_\epsilon(A) & \supseteq & F_\epsilon(X_j) & \text{if} & A \rightarrow X_1 \dots X_m \in P, \\ & & & & \text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_{j-1}) \end{array}$$

Lookahead Sets

for example...

$$\begin{array}{lllllll} E & \rightarrow & E+T & \mid & T & & \\ T & \rightarrow & T*F & \mid & F & & \\ F & \rightarrow & (\,E\,) & \mid & \text{name} & \mid & \text{int} \end{array}$$

with $\text{empty}(E) = \text{empty}(T) = \text{empty}(F) = \text{false}$

Lookahead Sets

For $\alpha \in (N \cup T)^*$ we are interested in the set:

$$\text{First}_1(\alpha) = \text{First}_1(\{w \in T^* \mid \alpha \rightarrow^* w\})$$

Idea: Treat ϵ separately: F_ϵ

- Let $\text{empty}(X) = \text{true}$ iff $X \rightarrow^* \epsilon$.
- $F_\epsilon(X_1 \dots X_m) = \bigcup_{i=1}^{j} F_\epsilon(X_i)$ if $\text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_{j-1})$

We characterize the ϵ-free First_1-sets with an inequality system:

$$\begin{array}{lllll} F_\epsilon(a) & = & \{a\} & \text{if} & a \in T \\ F_\epsilon(A) & \supseteq & F_\epsilon(X_j) & \text{if} & A \rightarrow X_1 \dots X_m \in P, \\ & & & & \text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_{j-1}) \end{array}$$

97 / 150

Lookahead Sets

for example...

$$
\begin{array}{lcllll}
E & \to & E+T & \mid & T & \\
T & \to & T*F & \mid & F & \\
F & \to & (\ E\ ) & \mid & \text{name} & \mid \ \text{int}
\end{array}
$$

with $\text{empty}(E) = \text{empty}(T) = \text{empty}(F) = \text{false}$

Fast Computation of Lookahead Sets

Observation:

- The form of each inequality of these systems is:

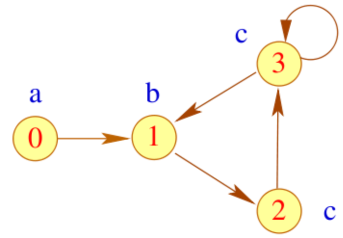
$$x \sqsupseteq y \qquad \text{resp.} \qquad x \sqsupseteq d$$

for variables x, y und $d \in \mathbb{D}$.

- Such systems are called pure unification problems
- Such problems can be solved in linear space/time.

for example: $\mathbb{D} = 2^{\{a,b,c\}}$

$$
\begin{array}{lll}
x_0 \supseteq \{a\} & & \\
x_1 \supseteq \{b\} & x_1 \supseteq x_0 & x_1 \supseteq x_3 \\
x_2 \supseteq \{c\} & x_2 \supseteq x_1 & \\
x_3 \supseteq \{c\} & x_3 \supseteq x_2 & x_3 \supseteq x_3
\end{array}
$$



Lookahead Sets

for example...

$$
\begin{array}{lcllll}
E & \to & E+T & \mid & T & \\
T & \to & T*F & \mid & F & \\
F & \to & (\ E\ ) & \mid & \text{name} & \mid \ \text{int}
\end{array}
$$

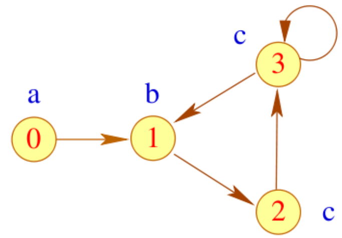
with $\text{empty}(E) = \text{empty}(T) = \text{empty}(F) = \text{false}$

... we obtain:

$$
\begin{array}{lcllcl}
F_\epsilon(S') & \supseteq & F_\epsilon(E) & F_\epsilon(E) & \supseteq & F_\epsilon(E) \\
F_\epsilon(E) & \supseteq & F_\epsilon(T) & F_\epsilon(T) & \supseteq & F_\epsilon(T) \\
F_\epsilon(T) & \supseteq & F_\epsilon(F) & F_\epsilon(F) & \supseteq & \{\,(\,,\text{name},\text{int}\}
\end{array}
$$

Fast Computation of Lookahead Sets

Frank DeRemer
& Tom Pennello

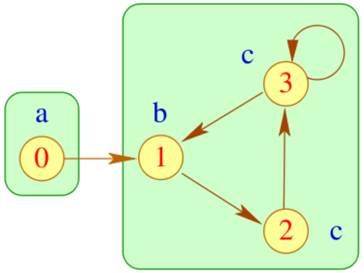


Proceeding:

- Create the Variable dependency graph for the inequality system.

Fast Computation of Lookahead Sets

Frank DeRemer
& Tom Pennello

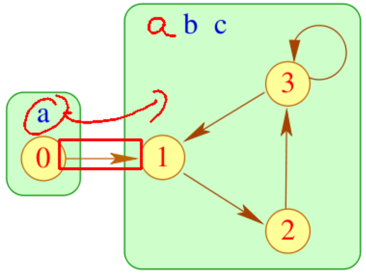


Proceeding:

- Create the Variable dependency graph for the inequality system.
- Whithin a strongly connected component (→ Tarjan) all variables have the same value

Fast Computation of Lookahead Sets

Frank DeRemer
& Tom Pennello

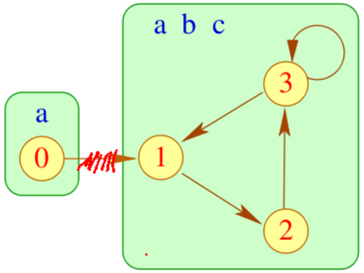


Proceeding:

- Create the Variable dependency graph for the inequality system.
- Whithin a strongly connected component (→ Tarjan) all variables have the same value
- Is there no ingoing edge for an SCC, its value is computed via the smallest upper bound of all values within the SCC

Fast Computation of Lookahead Sets

Frank DeRemer
& Tom Pennello



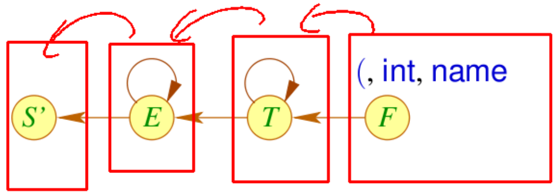
Proceeding:

- Create the Variable dependency graph for the inequality system.
- Whithin a strongly connected component (→ Tarjan) all variables have the same value
- Is there no ingoing edge for an SCC, its value is computed via the smallest upper bound of all values within the SCC
- In case of ingoing edges, their values are also to be considered for the upper bound

Fast Computation of Lookahead Sets

... for our example grammar:

$\boxed{\text{First}_1 :}$



Item Pushdown Automaton as LL(1)-Parser

First

ε a

back to the example: $S \to \epsilon \;\mid\; a\,S\,b$

The transitions in the according Item Pushdown Automaton:

0	$[S' \to \bullet\, S]$	ϵ	$[S' \to \bullet\, S]\,[S \to \bullet]$
1	$[S' \to \bullet\, S]$	ϵ	$[S' \to \bullet\, S]\,[S \to \bullet\, a\,S\,b]$
2	$[S \to \bullet\, a\,S\,b]$	a	$[S \to a \bullet S\,b]$
3	$[S \to a \bullet S\,b]$	ϵ	$[S \to a \bullet S\,b]\,[S \to \bullet]$
4	$[S \to a \bullet S\,b]$	ϵ	$[S \to a \bullet S\,b]\,[S \to \bullet\, a\,S\,b]$
5	$[S \to a \bullet S\,b]\,[S \to \bullet]$	ϵ	$[S \to a\,S \bullet b]$
6	$[S \to a \bullet S\,b]\,[S \to a\,S\,b \bullet]$	ϵ	$[S \to a\,S \bullet b]$
7	$[S \to a\,S \bullet b]$	b	$[S \to a\,S\,b \bullet]$
8	$[S' \to \bullet\, S]\,[S \to \bullet]$	ϵ	$[S' \to S \bullet]$
9	$[S' \to \bullet\, S]\,[S \to a\,S\,b \bullet]$	ϵ	$[S' \to S \bullet]$

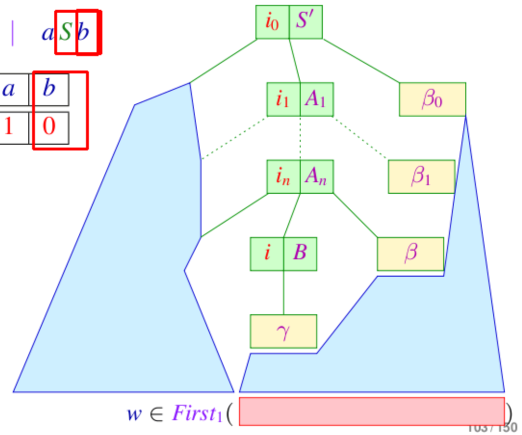
Conflicts arise between transations $(0, 1)$ or $(3, 4)$ resp..

Item Pushdown Automaton as LL(1)-Parser

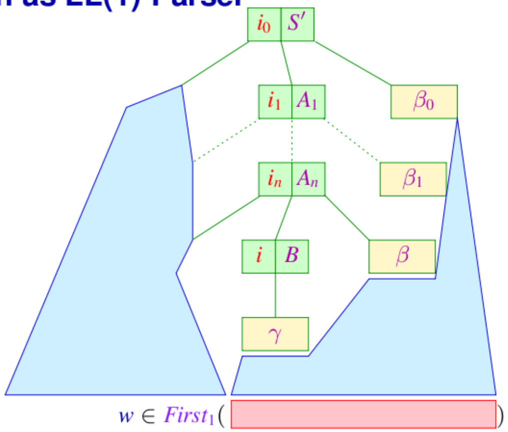
Is G an $LL(1)$-grammar, we can index a lookahead-table with items and nonterminals:
We set $M[B,\, w] \;=\; i$ exactly if (B, i) is the rule $B \to \gamma$ and:
$w \in \text{First}_1(\gamma) \odot \bigcup\{\text{First}_1(\beta) \mid S' \to^*_L u\,B\,\beta\}$.

... for example: $\boxed{S \to \epsilon} \;\mid\; a\,S\,\boxed{b}$

	ϵ	a	b
S	0	1	0



Item Pushdown Automaton as LL(1)-Parser



Inequality system for $\text{Follow}_1(B) = \bigcup\{\text{First}_1(\beta) \mid S' \to^*_L u\,B\,\beta\}$

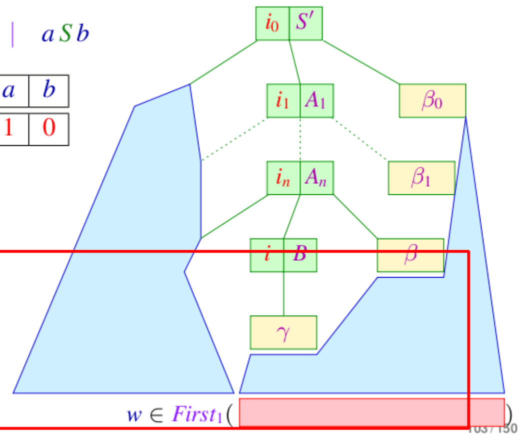
$$
\begin{array}{llll}
\text{Follow}_1(S) & \supseteq & \{\epsilon\} & \\
\text{Follow}_1(B) & \supseteq & F_\epsilon(X_j) & \text{if} \quad A \to \alpha\,B\,X_1 \ldots X_m \;\in P, \\
 & & & \quad \text{empty}(X_1) \wedge \ldots \wedge \text{empty}(X_{j-1}) \\
\text{Follow}_1(B) & \supseteq & \text{Follow}_1(A) & \text{if} \quad A \to \alpha\,B\,X_1 \ldots X_m \;\in P, \\
 & & & \quad \text{empty}(X_1) \wedge \ldots \wedge \text{empty}(X_m)
\end{array}
$$

Item Pushdown Automaton as LL(1)-Parser

Is G an $LL(1)$-grammar, we can index a lookahead-table with items and nonterminals:
We set $M[B,\, w] \;=\; i$ exactly if (B, i) is the rule $B \to \gamma$ and:
$w \in \text{First}_1(\gamma) \odot \bigcup\{\text{First}_1(\beta) \mid S' \to^*_L u\,B\,\beta\}$.

... for example: $S \to \epsilon \;\mid\; a\,S\,b$

	ϵ	a	b
S	0	1	0

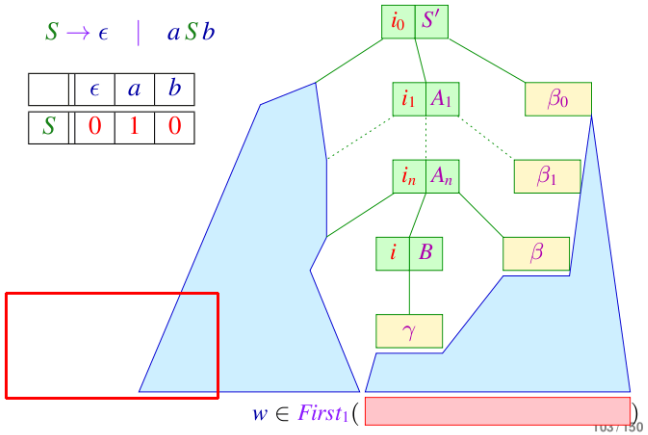


Item Pushdown Automaton as LL(1)-Parser

Is G an $LL(1)$-grammar, we can index a lookahead-table with items and nonterminals:
We set $M[B,\ w] = i$ exactly if (B, i) is the rule $B \to \gamma$ and:
$w \in \text{First}_1(\gamma) \odot \bigcup \{\text{First}_1(\beta) \mid S' \to_L^* u\,B\,\beta\}$.

... for example: $S \to \epsilon \quad | \quad a\,S\,b$

	ϵ	a	b
S	0	1	0



Topdown-Parsing

Discussion

- A practical implementation of an $LL(1)$-parser via recursive Descent is a straight-forward idea
- However, only a subset of the deterministic contextfree languages can be read this way.

Topdown-Parsing

Discussion

- A practical implementation of an $LL(1)$-parser via recursive Descent is a straight-forward idea
- However, only a subset of the deterministic contextfree languages can be read this way.
- Solution: Going from $LL(1)$ to $LL(k)$
- The size of the occuring sets is rapidly increasing with larger k
- Unfortunately, even $LL(k)$ parsers are not sufficient to accept all deterministic contextfree languages.
- In practical systems, this often motivates the implementation of $k = 1$ only ...

Chapter 4:
Bottom-up Analysis

Ende der Präsentation. Klicken Sie zum Schließen.

Bottom-up Analysis

Attention:

Many grammars are not $LL(k)$!

A reason for that is:

Definition

Grammar G is called left-recursive, if

$$A \to^{+} A\,\beta \qquad \text{for an} \quad A \in N,\ \beta \in (T \cup N)^{*}$$

Example:

$$\begin{array}{lcl}
E & \to & E+T \quad | \quad T \\
T & \to & T*F \quad | \quad F \\
F & \to & (\,E\,) \quad | \quad \text{name} \quad | \quad \text{int}
\end{array}$$

... is left-recursive

Bottom-up Analysis

Theorem:

Let a grammar G be reduced and left-recursive, then G is not $LL(k)$ for any k.

Proof:

Let $A \to A\,\beta \mid \alpha \quad \in P$
and A be reachable from S

Assumption: G is $LL(k)$

Bottom-up Analysis

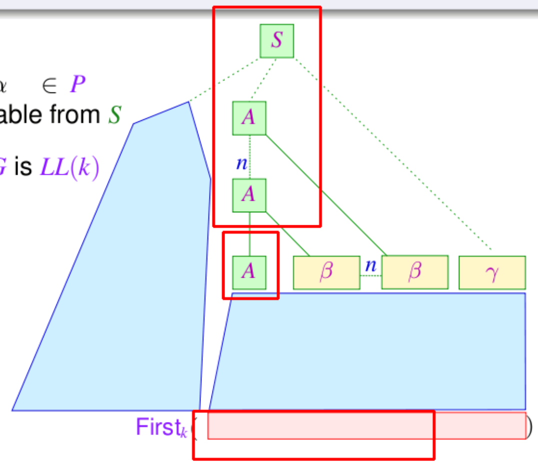
Theorem:

Let a grammar G be reduced and left-recursive, then G is not $LL(k)$ for any k.

Proof:

Let $A \to A\,\beta \mid \alpha \quad \in P$
and A be reachable from S

Assumption: G is $LL(k)$



Bottom-up Analysis

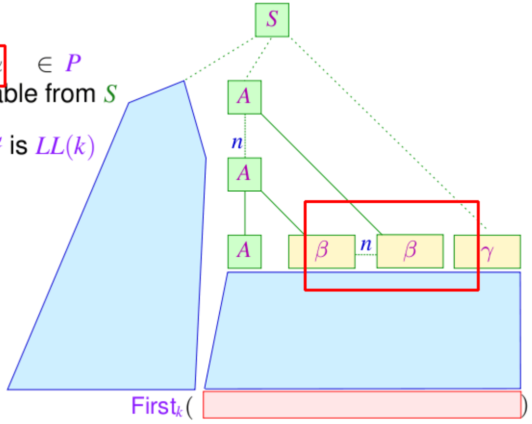
Theorem:

Let a grammar G be reduced and left-recursive, then G is not $LL(k)$ for any k.

Proof:

Let $\boxed{A} \rightarrow \boxed{A\,\beta} \mid \boxed{\alpha} \quad \in P$
and A be reachable from S

Assumption: G is $LL(k)$



Bottom-up Analysis

Theorem:

Let a grammar G be reduced and left-recursive, then G is not $LL(k)$ for any k.

Proof:

Let $A \rightarrow A\,\beta \mid \alpha \quad \in P$
and A be reachable from S

Assumption: G is $LL(k)$

$\Rightarrow \text{First}_k\boxed{(\alpha\,\beta^n\,\gamma)} \cap \text{First}_k\boxed{(\alpha\,\beta^{n+1}\,\gamma)} = \emptyset$

Bottom-up Analysis

Theorem:

Let a grammar G be reduced and left-recursive, then G is not $LL(k)$ for any k.

Proof:

Let $A \rightarrow A\,\beta \mid \alpha \quad \in P$
and A be reachable from S

Assumption: G is $LL(k)$

$\Rightarrow \text{First}_k(\alpha\,\beta^n\,\gamma) \cap \text{First}_k(\alpha\,\beta^{n+1}\,\gamma) = \emptyset$

Case 1: $\beta \rightarrow^* \epsilon$ — Contradiction !!!
Case 2: $\beta \rightarrow^* w \neq \epsilon \Longrightarrow \text{First}_k(\alpha\,\beta^k\,\gamma) \cap \text{First}_k(\alpha\,\beta^{k+1}\,\gamma) \neq \emptyset$

Shift-Reduce Parser

Donald Knuth

Idee:

We delay the decision whether to reduce until we know, whether the input matches the right-hand-side of a rule!

Konstruktion: Shift-Reduce parser M_G^R

- The input is shifted successively to the pushdown.
- Is there a complete right-hand side (a handle) atop the pushdown, it is replaced (reduced) by the corresponding left-hand side

Shift-Reduce Parser

Example:

$$
\begin{array}{lcl}
S & \to & A\,B \\
A & \to & a \\
B & \to & b
\end{array}
$$

The pushdown automaton:

States: q_0, f, a, b, A, B, S;
Start state: q_0
End state: f

q_0	a	$q_0\,a$
a	ϵ	A
A	b	$A\,b$
b	ϵ	B
$A\,B$	ϵ	S
$q_0\,S$	ϵ	f

Shift-Reduce Parser

Construction:

In general, we create an automaton $M_G^R = (Q, T, \delta, q_0, F)$ with:

- $Q = T \cup N \cup \{q_0, f\}$ (q_0, f fresh);
- $F = \{f\}$;
- Transitions:

$$
\begin{array}{lll}
\delta = & \{(q, x, q\,x) \mid q \in Q, x \in T\} \;\cup & // \text{ Shift-transitions} \\
 & \{(q\,\alpha, \boxed{\epsilon}, q\,A) \mid q \in Q, A \to \alpha \in P\} \;\cup & // \text{ Reduce-transitions} \\
 & \{(q_0\,S, \epsilon, f)\} & // \text{ finish}
\end{array}
$$

Shift-Reduce Parser

Construction:

In general, we create an automaton $M_G^R = (Q, T, \delta, q_0, F)$ with:

- $Q = T \cup N \cup \{q_0, f\}$ (q_0, f fresh);
- $F = \{f\}$;
- Transitions:

$$
\begin{array}{lll}
\delta = & \{(q, x, q\,x) \mid q \in Q, x \in T\} \;\cup & // \text{ Shift-transitions} \\
 & \{(q\,\alpha, \epsilon, q\,A) \mid q \in Q, A \to \alpha \in P\} \;\cup & // \text{ Reduce-transitions} \\
 & \{(q_0\,S, \epsilon, f)\} & // \text{ finish}
\end{array}
$$

Example-computation:

$$
\begin{array}{llll}
\boxed{(q_0, \; a\,b)} & \vdash \; (q_0\,a, \; b) & \vdash & (q_0\,A, \; b) \\
 & \vdash \; (q_0\,A\,b, \; \epsilon) & \vdash & (q_0\,A\,B, \; \epsilon) \\
 & \vdash \; (q_0\,S, \; \epsilon) & \vdash & (f, \; \epsilon)
\end{array}
$$

Shift-Reduce Parser

Construction:

In general, we create an automaton $M_G^R = (Q, T, \delta, q_0, F)$ with:

- $Q = T \cup N \cup \{q_0, f\}$ (q_0, f fresh);
- $F = \{f\}$;
- Transitions:

$$
\begin{array}{lll}
\delta = & \{(q, x, q\,x) \mid q \in Q, x \in T\} \;\cup & // \text{ Shift-transitions} \\
 & \{(q\,\alpha, \epsilon, q\,A) \mid q \in Q, A \to \alpha \in P\} \;\cup & // \text{ Reduce-transitions} \\
 & \{(q_0\,S, \epsilon, f)\} & // \text{ finish}
\end{array}
$$

Shift-Reduce Parser

Observation:

- The sequence of reductions correpsonds to a reverse rightmost-derivation for the input
- To prove correctnes, we have to prove:

$$(\epsilon, w) \vdash^* (A, \epsilon) \qquad \text{gdw.} \qquad A \to^* w$$

- The shift-reduce pushdown automaton M_G^R is in general also non deterministic
- For a deterministic parsing-algorithm, we have to identify spots for reduction

$\Longrightarrow$ LR-Parsing

Bottom-up Analysis

Idea: We reconstruct reverse rightmost-derivations!

Therfore we try to identify the reduction spots for the shift-reduce parser M_G^R ...

Consider the computations of this pushdown automaton:

$$(q_0\, \alpha\, \gamma,\ v) \vdash (q_0\, \alpha\, B,\ v) \vdash^* (q_0\, S,\ \epsilon)$$

We call $\alpha\,\gamma$ a viable prefix for the complete item $[B \to \gamma\bullet]$.