Script generated by TTT

Title: Simon: Compilerbau (28.04.2014)

Date: Mon Apr 28 15:12:39 CEST 2014

Duration: 40:49 min

Pages: 37

... further examples:

S	$\rightarrow$	⟨stmt⟩
⟨stmt⟩	$\rightarrow$	⟨if⟩ | ⟨while⟩ | ⟨rexp⟩;
⟨if⟩	$\rightarrow$	if (⟨rexp⟩) ⟨stmt⟩ else ⟨stmt⟩
⟨while⟩	$\rightarrow$	while (⟨rexp⟩) ⟨stmt⟩
⟨rexp⟩	$\rightarrow$	int | ⟨lexp⟩ | ⟨lexp⟩ = ⟨rexp⟩ | ...
⟨lexp⟩	$\rightarrow$	name | ...

Further conventions:

- For every nonterminal, we collect the right hand sides of rules and list them together.
- The j-th rule for A can be identified via the pair (A, j) (with $j \geq 0$).

further grammars:

E	$\rightarrow$	$E+E$	$E*E$	(E)	name	int

E	$\rightarrow$	$E+T$	T	
T	$\rightarrow$	$T*F$	F	
F	$\rightarrow$	(E)	name	int

Both grammars describe the same language

further grammars:

E	$\rightarrow$	$E+E\ ^0$	$E*E\ ^1$	$(E)\ ^2$	name ^{3}	int ^{4}

E	$\rightarrow$	$E+T\ ^0$	$T\ ^1$	
T	$\rightarrow$	$T*F\ ^0$	$F\ ^1$	
F	$\rightarrow$	$(E)\ ^0$	name ^{1}	int ^{2}

Both grammars describe the same language

Derivation

Grammars are term rewriting systems. The rules offer feasible rewriting steps. A sequence of such rewriting steps $\alpha_0 \to \ldots \to \alpha_m$ is called derivation.

... for example: $\underline{E}$

Derivation

Grammars are term rewriting systems. The rules offer feasible rewriting steps. A sequence of such rewriting steps $\alpha_0 \to \ldots \to \alpha_m$ is called derivation.

... for example: $\underline{E} \to \underline{E} + T$

Derivation

Grammars are term rewriting systems. The rules offer feasible rewriting steps. A sequence of such rewriting steps $\alpha_0 \to \ldots \to \alpha_m$ is called derivation.

... for example:

$$\begin{array}{lll} \underline{E} & \to & \underline{E} + T \\ & \to & \underline{T} + T \end{array}$$

Derivation

Grammars are term rewriting systems. The rules offer feasible rewriting steps. A sequence of such rewriting steps $\alpha_0 \to \ldots \to \alpha_m$ is called derivation.

... for example:

$$\begin{array}{lll} \underline{E} & \to & \underline{E} + T \\ & \to & \underline{T} + T \\ & \to & T * \underline{F} + T \\ & \to & \underline{T} * \text{int} + T \end{array}$$

Derivation

Grammars are term rewriting systems. The rules offer feasible rewriting steps. A sequence of such rewriting steps $\alpha_0 \to \ldots \to \alpha_m$ is called derivation.

... for example:

$$\begin{array}{lll} \underline{E} & \to & \underline{E} + T \\ & \to & \underline{T} + T \\ & \to & T * \underline{F} + T \\ & \to & \underline{T} * \text{int} + T \\ & \to & \underline{F} * \text{int} + T \end{array}$$

Derivation

Grammars are term rewriting systems. The rules offer feasible rewriting steps. A sequence of such rewriting steps $\alpha_0 \to \ldots \to \alpha_m$ is called derivation.

... for example:

$$\begin{array}{lll} \underline{E} & \to & \underline{E} + T \\ & \to & \underline{T} + T \\ & \to & T * \underline{F} + T \\ & \to & \underline{T} * \text{int} + T \\ & \to & \underline{F} * \text{int} + T \\ & \to & \text{name} * \text{int} + \underline{T} \\ & \to & \text{name} * \text{int} + \underline{F} \\ & \to & \text{name} * \text{int} + \text{int} \end{array}$$

Definition

A derivation $\to$ is a relation on words over $N \cup T$, with

$$\alpha \to \alpha' \quad \text{iff} \quad \alpha = \alpha_1\ A\ \alpha_2 \ \wedge\ \alpha' = \alpha_1\ \beta\ \alpha_2 \quad \text{for an } A \to \beta \in P$$

Derivation

Grammars are term rewriting systems. The rules offer feasible rewriting steps. A sequence of such rewriting steps $\alpha_0 \to \ldots \to \alpha_m$ is called derivation.

... for example:

$$\begin{array}{lll} \underline{E} & \to & \underline{E} + T \\ & \to & \underline{T} + T \\ & \to & T * \underline{F} + T \\ & \to & \underline{T} * \text{int} + T \\ & \to & \underline{F} * \text{int} + T \\ & \to & \text{name} * \text{int} + \underline{T} \\ & \to & \text{name} * \text{int} + \underline{F} \\ & \to & \text{name} * \text{int} + \text{int} \end{array}$$

Definition

A derivation $\to$ is a relation on words over $N \cup T$, with

$$\alpha \to \alpha' \quad \text{iff} \quad \alpha = \alpha_1\ A\ \alpha_2 \ \wedge\ \alpha' = \alpha_1\ \beta\ \alpha_2 \quad \text{for an } A \to \beta \in P$$

The reflexive and transitive closure of $\to$ is denoted as: $\to^*$

Derivation

Remarks:

- The relation $\to$ depends on the grammar
- In each step of a derivation, we may choose:
  - $*$ a spot, determining where we will rewrite.
  - $*$ a rule, determining how we will rewrite.
- The language, specified by G is:

$$\mathcal{L}(G) = \{w \in T^* \mid S \to^* w\}$$

Derivation

Remarks:

- The relation $\rightarrow$ depends on the grammar
- In each step of a derivation, we may choose:
  - $*$ a spot, determining where we will rewrite.
  - $*$ a rule, determining how we will rewrite.
- The language, specified by G is:

$$\mathcal{L}(G) = \{w \in T^* \mid S \rightarrow^* w\}$$

Attention:
The order, in which disjunct fragments are rewritten is not relevant.

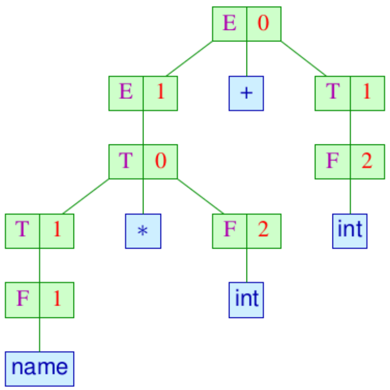
Special Derivations

Attention:
In contrast to arbitrary derivations, we find special ones, always rewriting the leftmost (or rather rightmost) occurance of a nonterminal.

- These are called leftmost (or rather rightmost) derivations and are denoted with the index L (or R respectively).
- Leftmost (or rightmost) derivations correspondt to a left-to-right (or right-to-left) preorder-DFS-traversal of the derivation tree.
- Reverse rightmost derivations correspond to a left-to-right postorder-DFS-traversal of the derivation tree

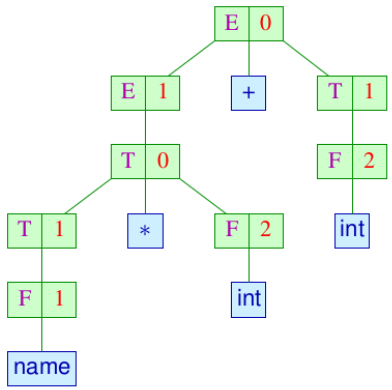
Special Derivations

... for example:



Special Derivations

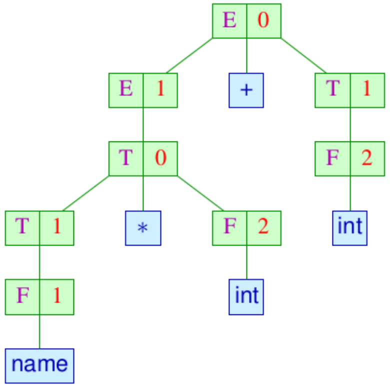
... for example:



Leftmost derivation: $(E,0)\,(E,1)\,(T,0)\,(T,1)\,(F,1)\,(F,2)\,(T,1)\,(F,2)$
Rightmost derivation: $(E,0)\,(T,1)\,(F,2)\,(E,1)\,(T,0)\,(F,2)\,(T,1)\,(F,1)$

Special Derivations

... for example:

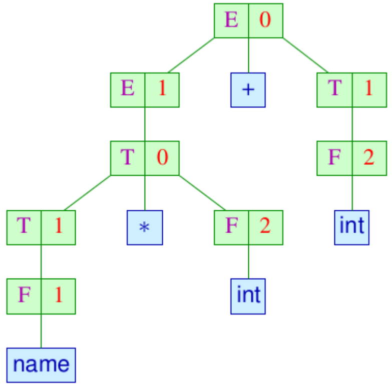


Leftmost derivation: $(E,0)\,(E,1)\,(T,0)\,(T,1)\,(F,1)\,(F,2)\,(T,1)\,(F,2)$
Rightmost derivation: $(E,0)\,(T,1)\,(F,2)\,(E,1)\,(T,0)\,(F,2)\,(T,1)\,(F,1)$
Reverse rightmost derivation:
$(F,1)\,(T,1)\,(F,2)\,(T,0)\,(E,1)\,(F,2)\,(T,1)\,(E,0)$

Unique grammars

The concatenation of leaves of a derivation tree t are often called yield(t) .

... for example:



gives rise to the concatenation: name $*$ int $+$ int .

Unique grammars

Definition:

Grammar G is called unique, if for every $w \in T^*$ there is maximally one derivation tree t of S with yield$(t) = w$.

... in our example:

$E \to E{+}E^{\,0} \mid E{*}E^{\,1} \mid (\,E\,)^{\,2} \mid \text{name}^{\,3} \mid \text{int}^{\,4}$

$E \to E{+}T^{\,0} \mid T^{\,1}$
$T \to T{*}F^{\,0} \mid F^{\,1}$
$F \to (\,E\,)^{\,0} \mid \text{name}^{\,1} \mid \text{int}^{\,2}$

The first one is ambiguous, the second one is unique

Conclusion:

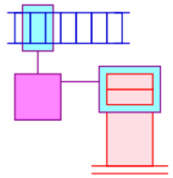
- A derivation tree represents a possible hierarchical structure of a word.
- For programming languages, only those grammars with a unique structure are of intrerest.
- Derivation trees are one-to-one corresponding with leftmost derivations as well as (reverse) rightmost derivations.
- Leftmost derivations correspond to a top-down reconstruction of the syntax tree.
- Reverse rightmost derivations correspond to a bottom-up reconstruction of the syntax tree.

Chapter 2:

Basics of Pushdown Automata

Basics of Pushdown Automata

Languages, specified by context free grammars are accepted by Pushdown Automata:



The pushdown is used e.g. to verify correct nesting of braces.

Example:

States: $0, 1, 2$
Start state: 0
Final states: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

Example:

States: $0, 1, 2$
Start state: 0
Final states: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

Conventions:

- We do not differentiate between pushdown symbols and states
- The rightmost / upper pushdown symbol represents the state
- Every transition consumes / modifies the upper part of the pushdown

Pushdown Automata


Friedrich Bauer


Klaus Samelson

Definition:

A pushdown automaton (PDA) is a tuple $M = (Q, T, \delta, q_0, F)$ with:

- Q a finite set of states;
- T an input alphabet;
- $q_0 \in Q$ the start state;
- $F \subseteq Q$ the set of final states and
- $\delta \subseteq Q^+ \times (T \cup \{\epsilon\}) \times Q^*$ a finite set of transitions

Pushdown Automata


Friedrich Bauer


Klaus Samelson

Definition:

A pushdown automaton (PDA) is a tuple $M = (Q, T, \delta, q_0, F)$ with:

- Q a finite set of states;
- T an input alphabet;
- $q_0 \in Q$ the start state;
- $F \subseteq Q$ the set of final states and
- $\delta \subseteq Q^+ \times (T \cup \{\epsilon\}) \times Q^*$ a finite set of transitions

We define computations of pushdown automata with the help of transitions; a particular computation state (the current configuration) is a pair:

$$(\gamma, w) \in Q^* \times T^*$$

consisting of the pushdown content and the remaining input.

... for example:

States: 0, 1, 2
Start state: 0
Final states: 0, 2

0	a	11
1	a	11
11	b	2
12	b	2

... for example:

States: 0, 1, 2
Start state: 0
Final states: 0, 2

0	a	11
1	a	11
11	b	2
12	b	2

$$(0, \; a\,a\,a\,b\,b\,b) \quad \vdash \quad (1\,1, \; a\,a\,b\,b\,b)$$

... for example:

States: $0, 1, 2$
Start state: 0
Final states: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

$$\begin{array}{lll} (0, \; aaabbb) & \vdash & (11, \; aabbb) \\ & \vdash & (111, \; abbb) \end{array}$$

... for example:

States: $0, 1, 2$
Start state: 0
Final states: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

$$\begin{array}{lll} (0, \; aaabbb) & \vdash & (11, \; aabbb) \\ & \vdash & (111, \; abbb) \\ & \vdash & (1111, \; bbb) \end{array}$$

... for example:

States: $0, 1, 2$
Start state: 0
Final states: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

$$\begin{array}{lll} (0, \; aaabbb) & \vdash & (11, \; aabbb) \\ & \vdash & (111, \; abbb) \\ & \vdash & (1111, \; bbb) \\ & \vdash & (112, \; bb) \end{array}$$

... for example:

States: $0, 1, 2$
Start state: 0
Final states: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

$$\begin{array}{lll} (0, \; aaabbb) & \vdash & (11, \; aabbb) \\ & \vdash & (111, \; abbb) \\ & \vdash & (1111, \; bbb) \\ & \vdash & (112, \; bb) \\ & \vdash & (12, \; b) \end{array}$$

A computation step is characterized by the relation
$\vdash \;\subseteq\; (Q^* \times T^*)^2$ with

$$(\alpha\,\gamma,\, x\,w) \vdash (\alpha\,\gamma',\, w) \quad \text{for} \quad (\gamma,\, x,\, \gamma') \in \delta$$

A computation step is characterized by the relation
$\vdash \;\subseteq\; (Q^* \times T^*)^2$ with

$$(\alpha\,\gamma,\, x\,w) \vdash (\alpha\,\gamma',\, w) \quad \text{for} \quad (\gamma,\, x,\, \gamma') \in \delta$$

Remarks:

- The relation $\vdash$ depends of the pushdown automaton M
- The reflexive and transitive closure of $\vdash$ is called $\vdash^*$
- Then, the language, accepted by M, is

$$\mathcal{L}(M) \;=\; \{w \in T^* \mid \exists f \in F :\; (q_0, w) \vdash^* (f, \epsilon)\}$$

A computation step is characterized by the relation
$\vdash \;\subseteq\; (Q^* \times T^*)^2$ with

$$(\alpha\,\gamma,\, x\,w) \vdash (\alpha\,\gamma',\, w) \quad \text{for} \quad (\gamma,\, x,\, \gamma') \in \delta$$

Remarks:

- The relation $\vdash$ depends of the pushdown automaton M
- The reflexive and transitive closure of $\vdash$ is called $\vdash^*$
- Then, the language, accepted by M, is

$$\mathcal{L}(M) \;=\; \{w \in T^* \mid \exists f \in F :\; (q_0, w) \vdash^* (f, \epsilon)\}$$

We accept with a final state together with empty input.

Deterministic Pushdown Automaton

Definition:
The pushdown automaton M is deterministic, if every configuration has maximally one successor configuration.

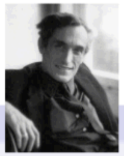
This is exactly the case if for distinct transitions
$(\gamma_1, x, \gamma_2)\,,\; (\gamma_1', x', \gamma_2') \in \delta$ we can assume:
Is γ_1 a suffix of γ_1', then $x \neq x' \;\wedge\; x \neq \epsilon \neq x'$ is valid.

... for example:

0	a	11
1	a	11
11	b	2
12	b	2

... this obviously holds

Pushdown Automata


M. Schützenberger


A. Öttinger

Theorem:

For each context free grammar $G = (N, T, P, S)$
a pushdown automaton M with $\mathcal{L}(G) = \mathcal{L}(M)$ can be built.

The theorem is so important for us, that we take a look at two constructions for automata, motivated by both of the special derivations:

- M_G^L to build Leftmost derivations
- M_G^R to build reverse Rightmost derivations

Chapter 3:
Top-down Parsing