Compiler Construction

Lecture 21: Error Handling in Top-Down Parsing & Strongly Noncircular Attribute Grammars

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH Aachen University

noll@cs.rwth-aachen.de

http://www-i2.informatik.rwth-aachen.de/i2/cc10/

Winter semester 2010/11

Outline

Lookahead Sets

Definition (Lookahead set)

Given $\pi = A \to \beta \in P$,

$$\mathrm{la}(\pi) := \mathrm{fi}(\beta \cdot \mathrm{fo}(A)) \subseteq \Sigma_\varepsilon$$

is called the lookahead set of π (where $\mathrm{fi}(\Gamma) := \bigcup_{\gamma \in \Gamma} \mathrm{fi}(\gamma)$).

Corollary

1. *For all* $a \in \Sigma$,
$$a \in \mathrm{la}(A \to \beta) \text{ iff } a \in \mathrm{fi}(\beta) \text{ or } (\beta \Rightarrow^* \varepsilon \text{ and } a \in \mathrm{fo}(A))$$
2. $\varepsilon \in \mathrm{la}(A \to \beta)$ *iff* $\beta \Rightarrow^* \varepsilon$ *and* $\varepsilon \in \mathrm{fo}(A)$

Characterization of $LL(1)$

Theorem (Characterization of $LL(1)$)

$G \in LL(1)$ *iff for all pairs of rules* $A \to \beta \mid \gamma \in P$ *(where* $\beta \neq \gamma$*):*

$$\text{la}(A \to \beta) \cap \text{la}(A \to \gamma) = \emptyset.$$

Proof.

on the board □

Remark: the above theorem generally does not hold if $k > 1$ (cf. exercises)

The Deterministic Top-Down Automaton

Definition (Deterministic top-down parsing automaton)

Let $G = \langle N, \Sigma, P, S \rangle \in LL(1)$. The deterministic top-down parsing automaton of G, DTA(G), is defined by the following components.

- Input alphabet Σ, pushdown alphabet X, output alphabet $[p]$
- Configurations $\Sigma^* \times X^* \times [p]^*$, initial configuration (w, S, ε), final configurations $\{\varepsilon\} \times \{\varepsilon\} \times [p]^*$ (as NTA(G))
- Action function

$$\mathrm{act} : \Sigma_\varepsilon \times X_\varepsilon \to \{(\alpha, i) \mid \pi_i = A \to \alpha\} \cup \{\mathsf{pop}, \mathsf{accept}, \mathsf{error}\}$$

$$\begin{aligned} \text{with } \mathrm{act}(x, A) &:= (\alpha, i) \text{ if } \pi_i = A \to \alpha \text{ and } x \in \mathrm{la}(\pi_i) \\ \mathrm{act}(a, a) &:= \mathsf{pop} \\ \mathrm{act}(\varepsilon, \varepsilon) &:= \mathsf{accept} \\ \mathrm{act}(x, y) &:= \mathsf{error} \qquad \text{otherwise} \end{aligned}$$

- Transitions for $x \in \Sigma_\varepsilon$, $w \in \Sigma^*$, $Y \in X$, $\beta \in X^*$, and $z \in [p]^*$:

$$(xw, Y\beta, z) \vdash \begin{cases} (xw, \alpha\beta, zi) & \text{if } \mathrm{act}(x, Y) = (\alpha, i) \\ (w, \beta, z) & \text{if } \mathrm{act}(x, Y) = \mathsf{pop} \end{cases}$$

Outline

Error Handling

Error configurations of DTA(G):

- $(aw, A\alpha, z)$ where $a \notin \bigcup_{A \to \beta \in P} \mathrm{la}(A \to \beta)$ $(\implies \mathrm{act}(a, A) = \mathsf{error})$
- $(aw, b\alpha, z)$ where $a \neq b$ $(\implies \mathrm{act}(a, b) = \mathsf{error})$
- $(\varepsilon, A\alpha, z)$ where $\varepsilon \notin \bigcup_{A \to \beta \in P} \mathrm{la}(A \to \beta)$ $(\implies \mathrm{act}(\varepsilon, A) = \mathsf{error})$
- $(\varepsilon, b\alpha, z)$ $(\implies \mathrm{act}(\varepsilon, b) = \mathsf{error})$
- (aw, ε, z) $(\implies \mathrm{act}(a, \varepsilon) = \mathsf{error})$

Observation: correct prefix property of LL parsing, i.e., syntactic errors are detected at the earliest possible position (every input prefix which does not produce an error can be extended to a word $w \in L(G)$)

Does not mean: error is recognized at the position where it is caused!

Example: assignment `a := b * c - (d + e));`
Possible corrections:

- remove closing bracket: `a := b * c - (d + e);`
- insert opening bracket: `a := b * (c - (d + e));`

The General Problem

- Let $w = xy \in \Sigma^*$ be the input word such that x is the longest prefix of a word in $L(G)$ (i.e., the error is detected at the first symbol of y) and $w \notin L(G)$.
- Parser makes assumption about error type and corrects w accordingly:
  - Assumes prefix x' of x to be correct
  - Correct prefix property
    $\implies$ there exists $z \in \Sigma^*$ such that $x'z \in L(G)$
  - Parser chooses prefix z' of z and suffix y' of y
  - Parsing resumed with input $w' := x'z'y'$ (at first symbol of z') (error recovery)
- Desirable properties of correction:
  - At least one symbol of y' can be processed before next error occurs (if $y' \neq \varepsilon$)
  - Preserve as many symbols of w as possible (i.e., x' and y' "long" and z' "short")
- $x' \neq x$ hard to implement, therefore usually $x' := x$

Aspects of Error Handling

Further criteria for "good" error handling:

- Continuation of parsing in any case, independent of severity of error
- High probability of correct error diagnosis
- Suppression of subsequent errors
- Complexity of analyzing correct inputs not impaired

Observation: no "best method" available

- correction not unique
- experience of programmer
- peculiarities of (programming) language

$\Longrightarrow$ employ heuristics

Panic Mode

Simplest form of error handling: panic mode

Upon occurrence of an error,

- skip input symbols ...
- until a token in a selected set of "separating" or "closing" tokens appears (synchronizing tokens)

Example: suitable synchronizing tokens in imperative languages for

- assignments: "`;`"
- declarations: "`;`" or "`,`"
- control structures: `fi` or `od`
- blocks: `end`

Challenge: choose set of synchronizing tokens such that

- parser recovers quickly from errors that are likely to occur and
- not too much input is overread

(see Aho/Lam/Sethi/Ullman: *Compilers: Principles, Techniques, and Tools*, 2nd ed., pp. 228)

Error Handling Example I

Example 21.1 (cf. Example 9.3)

$$G'_{AE}: \begin{array}{lll} E & \to TE' & (1) \\ E' & \to +TE' \mid \varepsilon & (2,3) \\ T & \to FT' & (4) \\ T' & \to *FT' \mid \varepsilon & (5,6) \\ F & \to (E) \mid a \mid b & (7,8,9) \end{array}$$

$A \in N$	fo(A)
E	$\{\varepsilon,)\}$
E'	$\{\varepsilon,)\}$
T	$\{+, \varepsilon,)\}$
T'	$\{+, \varepsilon,)\}$
F	$\{*, +, \varepsilon,)\}$

With synchronizing tokens from fo sets:

act : $\Sigma_\varepsilon \times X_\varepsilon \to \{(\alpha, i) \mid \pi_i = A \to \alpha\} \cup \{$pop, accept, error, sync$\}$ (empty = error)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Error Handling Example II

Example 21.1 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(\mathtt{a}, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(\mathtt{b}, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Meaning of table entries:

- $\text{act}(x, Y) = \text{sync}$
  $\Longrightarrow$ pop Y and resume parsing
- $\text{act}(x, A) = \text{error}$
  $\Longrightarrow$ skip x and resume parsing

$$
\begin{array}{ll}
 & (\mathtt{+a*+b},\ E,\ \varepsilon) \\
\vdash & (\mathtt{a*+b},\ E,\ \varepsilon) \\
\vdash & (\mathtt{a*+b},\ TE',\ 1) \\
\vdash & (\mathtt{a*+b},\ FT'E',\ 14) \\
\vdash & (\mathtt{a*+b},\ \mathtt{a}T'E',\ 148) \\
\vdash & (\mathtt{*+b},\ T'E',\ 148) \\
\vdash & (\mathtt{*+b},\ \mathtt{*}FT'E',\ 1485) \\
\vdash & (\mathtt{+b},\ FT'E',\ 1485) \\
\vdash & (\mathtt{+b},\ T'E',\ 1485) \\
\vdash & (\mathtt{+b},\ E',\ 14856) \\
\vdash & (\mathtt{+b},\ \mathtt{+}TE',\ 148562) \\
\vdash & (\mathtt{b},\ TE',\ 148562) \\
\vdash & (\mathtt{b},\ FT'E',\ 1485624) \\
\vdash & (\mathtt{b},\ \mathtt{b}T'E',\ 14856249) \\
\vdash & (\varepsilon,\ T'E',\ 14856249) \\
\vdash & (\varepsilon,\ E',\ 148562496) \\
\vdash & (\varepsilon,\ \varepsilon,\ 1485624963)
\end{array}
$$

Outline

Formal Definition of Attribute Grammars

Definition (Attribute grammar)

Let $G = \langle N, \Sigma, P, S \rangle \in CFG_\Sigma$ with $X := N \uplus \Sigma$.

- Let $Att = Syn \uplus Inh$ be a set of (synthesized or inherited) attributes, and let $V = \bigcup_{\alpha \in Att} V^\alpha$ be a union of value sets.
- Let $\mathrm{att} : X \to 2^{Att}$ be an attribute assignment, and let $\mathrm{syn}(Y) := \mathrm{att}(Y) \cap Syn$ and $\mathrm{inh}(Y) := \mathrm{att}(Y) \cap Inh$ for every $Y \in X$.
- Every production $\pi = Y_0 \to Y_1 \ldots Y_r \in P$ determines the set
  $$Var_\pi := \{\alpha.i \mid \alpha \in \mathrm{att}(Y_i), i \in \{0, \ldots, r\}\}$$
  of attribute variables of π with the subsets of inner and outer variables:
  $$In_\pi := \{\alpha.i \mid (i = 0, \alpha \in \mathrm{syn}(Y_i)) \text{ or } (i \in [r], \alpha \in \mathrm{inh}(Y_i))\}$$
  $$Out_\pi := Var_\pi \setminus In_\pi$$
- A semantic rule of π is an equation of the form
  $$\alpha.i = f(\alpha_1.i_1, \ldots, \alpha_n.i_n)$$
  where $n \in \mathbb{N}$, $\alpha.i \in In_\pi$, $\alpha_j.i_j \in Out_\pi$, and $f : V^{\alpha_1} \times \ldots \times V^{\alpha_n} \to V^\alpha$.
- For each $\pi \in P$, let E_π be a set with exactly one semantic rule for every inner variable of π, and let $E := (E_\pi \mid \pi \in P)$.

Then $\mathfrak{A} := \langle G, E, V \rangle$ is called an attribute grammar: $\mathfrak{A} \in AG$.

Attribution of Syntax Trees I

Definition (Attribution of syntax trees)

Let $\mathfrak{A} = \langle G, E, V \rangle \in AG$, and let t be a syntax tree of G with the set of nodes K.

- K determines the set of attribute variables of t:
$$Var_t := \{\alpha.k \mid k \in K \text{ labelled with } Y \in X, \alpha \in \text{att}(Y)\}.$$
- Let $k_0 \in K$ be an (inner) node where production $\pi = Y_0 \to Y_1 \dots Y_r \in P$ is applied, and let $k_1, \dots, k_r \in K$ be the corresponding successor nodes. The attribute equation system E_{k_0} of k_0 is obtained from E_π by substituting every attribute index $i \in \{0, \dots, r\}$ by k_i.
- The attribute equation system of t is given by
$$E_t := \bigcup \{E_k \mid k \text{ inner node of } t\}.$$

Circularity of Attribute Grammars

Goal: unique solvability of equation system
$\Longrightarrow$ avoid cyclic dependencies

Definition (Circularity)

An attribute grammar $\mathfrak{A} = \langle G, E, V \rangle \in AG$ is called circular if there exists a syntax tree t such that the attribute equation system E_t is recursive (i.e., some attribute variable of t depends on itself). Otherwise it is called noncircular.

Remark: because of the division of Var_π into In_π and Out_π, cyclic dependencies cannot occur at production level (see Corollary 16.8).

Attribute Dependency Graphs and Circularity I

Observation: a cycle in the dependency graph D_t of a given syntax tree t is caused by the occurrence of a "cover" production $\pi = A_0 \rightarrow w_0 A_1 w_1 \ldots A_r w_r \in P$ in a node k_0 of t such that

- the dependencies in E_{k_0} yield the "upper end" of the cycle and
- for at least one $i \in [r]$, some attributes in $\mathrm{syn}(A_i)$ depend on attributes in $\mathrm{inh}(A_i)$.

Example

on the board

To identify such "critical" situations we need to determine for each $i \in [r]$ the possible ways in which attributes in $\mathrm{syn}(A_i)$ can depend on attributes in $\mathrm{inh}(A_i)$.

Attribute Dependency Graphs and Circularity II

Definition (Attribute dependence)

Let $\mathfrak{A} = \langle G, E, V \rangle \in AG$ with $G = \langle N, \Sigma, P, S \rangle$.

- If t is a syntax tree with root label $A \in N$ and root node k, $\alpha \in \mathrm{syn}(A)$, and $\beta \in \mathrm{inh}(A)$ such that $\beta.k \to_t^+ \alpha.k$, then α is dependent on β below A in t (notation: $\beta \overset{A}{\hookrightarrow} \alpha$).
- For every syntax tree t with root label $A \in N$,
$$is(A,t) := \{(\beta, \alpha) \in \mathrm{inh}(A) \times \mathrm{syn}(A) \mid \beta \overset{A}{\hookrightarrow} \alpha \text{ in } t\}.$$
- For every $A \in N$,
$$IS(A) := \{is(A,t) \mid t \text{ syntax tree with root label A}\} \subseteq 2^{Inh \times Syn}.$$

Remark: it is important that $IS(A)$ is a system of attribute dependence sets, not a union (later: strong noncircularity).

Example

on the board

Outline

Simplifying the Circularity Test

Idea: to simplify the circularity test, do not distinguish between attribute dependences which are caused by different syntax trees

Definition 21.2 (Attribute dependence (modified))

Let $\mathfrak{A} = \langle G, E, V \rangle \in AG$ with $G = \langle N, \Sigma, P, S \rangle$.

- Reminder: if t is a syntax tree with root label $A \in N$ and root node k, $\alpha \in \text{syn}(A)$, and $\beta \in \text{inh}(A)$ such that $\beta.k \to_t^+ \alpha.k$, then α is dependent on β below A in t (notation: $\beta \overset{A}{\hookrightarrow} \alpha$).
- For every $A \in N$,

$$\begin{aligned} IS'(A) &:= \{(\beta, \alpha) \mid \beta \overset{A}{\hookrightarrow} \alpha \text{ in some syntax tree with root label A}\} \\ &\subseteq Inh \times Syn \end{aligned}$$

The Strong Circularity Test

Algorithm 21.3 (Strong circularity test for attribute grammars)

Input: $\mathfrak{A} = \langle G, E, V \rangle \in AG$ *with* $G = \langle N, \Sigma, P, S \rangle$

Procedure:

1. *for every* $A \in N$, *iteratively construct* $IS'(A)$ *as follows:*
   1. *if* $\pi = A \to w \in P$, *then* $is[\pi] \subseteq IS'(A)$
   2. *if* $\pi = A \to w_0 A_1 w_1 \dots A_r w_r \in P$, *then* $is[\pi; IS'(A_1), \dots, IS'(A_r)] \subseteq IS'(A)$
2. *test whether there exists* $\pi = A \to w_0 A_1 w_1 \dots A_r w_r \in P$ *such that the following relation is cyclic:*

   $$\to_\pi \cup \bigcup_{i=1}^{r} \{(\beta.p_i, \alpha.p_i) \mid (\beta, \alpha) \in IS'(A_i)\}$$

   (where $p_i := \sum_{j=1}^{i} |w_{j-1}| + i$*)*

Output: *"yes" or "no"*

Example 21.4

on the board

Strongly Noncircular Attribute Grammars

Definition 21.5 (Strong noncircularity)

An attribute grammar is called strongly noncircular if Algorithm 21.3 yields the answer "no".

Lemma 21.6

The time complexity of the strong circularity test is polynomial in the size of the attribute grammar (= maximal length of right-hand sides of productions).

Proof.

omitted □