Script **generated by TTT**

Title: Petter: Compilerbau (06.07.2017)

Date: Thu Jul 06 14:15:10 CEST 2017

Duration: 83:52 min

Pages: 32

LR(2) to LR(1)

Example 2 finished:

With fresh nonterminals we get the final grammar

$$\begin{array}{lcl} S & \to & b\,S\,S^{0} \\ & | & a^{1} \\ & | & a\,a\,c^{2} \end{array} \quad \Rightarrow \quad \begin{array}{lcl} S & \to & b\,C\,A,^{0} \mid b\,S\,b\,B,^{1} \mid a^{2} \mid a\,a\,c^{3} \\ A & \to & \epsilon^{0} \mid a\,c^{1} \\ B & \to & C\,A^{0} \mid S\,b\,B^{1} \\ C & \to & b\,C\,D^{0} \mid b\,S\,b\,E^{1} \mid a\,a^{2} \mid a\,a\,c\,a^{3} \\ D & \to & a^{0} \mid a\,c\,a^{1} \\ E & \to & C\,D^{0} \mid S\,b\,E^{1} \end{array}$$

LR(2) to LR(1)

Algorithm:

For a Rule $A \to \alpha$, which is *reduce-conflicting* under terminal x

- $B \to \beta\,A$ is also considered *reduce-conflicting* under terminal x
- $B \to \beta\,A\,C\,\gamma$ is transformed by *right-context-extraction* on C:

$$B \to \beta\,A\,C\,\gamma \quad \Rightarrow \quad B \to \beta\,A\,x\,\langle x/C\rangle\,\gamma \quad \Big|_{y \in \mathsf{First}_1(C)\setminus x} \quad \beta\,A\,y\,\langle y/C\rangle\,\gamma$$

if $\epsilon \in \mathsf{First}_1(C)$ then consider $B \to \beta\,A\,\gamma$ for r.-c.-extraction

- $B \to \beta\,A\,x\,\gamma$ is transformed by *right-context-propagation* on A:

$$B \to \beta\,A\,x\,\gamma \quad \Rightarrow \quad B \to \beta\,\langle Ax\rangle\,\gamma$$

- The appropriate rules, created from introducing $\langle Ax\rangle \to \delta$ and $\langle x/B\rangle \to \eta$ are added to the grammar

LR(2) to LR(1)

Right-Context-Propagation Algorithm:

For $\langle Ax\rangle$ with $A \to \alpha_1 \mid \ldots \mid \alpha_k$, if α_i matches

- $\gamma\,A$ for some $\gamma \in (N \cup T)^*$, then $\langle Ax\rangle \to \gamma\,\langle Ax\rangle$ is added
- else $\langle Ax\rangle \to \alpha_i\,x$ is added

Right-Context-Extraction Algorithm:

For $\langle x/B\rangle$ with $B \to \alpha_1 \mid \ldots \mid \alpha_k$, if α_i matches

- $C\,\gamma$ for some $\gamma \in (N \cup T)^*$, then $\langle x/B\rangle \to \langle x/C\rangle\,\gamma$ is added
- $\boxed{x\,\gamma}$ for some $\gamma \in (N \cup T)^*$, then $\boxed{\langle x/B\rangle \to \gamma}$ is added
- $\boxed{y\,\gamma}$ for some $\gamma \in (N \cup T)^*$ and $y \neq x$, then nothing is added

LR(2) to LR(1)

Algorithm:

For a Rule $A \to \alpha$, which is *reduce-conflicting* under terminal x

- $B \to \beta\, A$ is also considered *reduce-conflicting* under terminal x
- $B \to \beta\, A\, C\, \gamma$ is transformed by *right-context-extraction* on C:

$$B \to \beta\, A\, C\, \gamma \quad\Rightarrow\quad B \to \beta\, A\, x\, \langle x/C\rangle\, \gamma \;\Big|_{y\in \text{First}_1(C)\setminus x}\; \beta\, A\, y\, \langle y/C\rangle\, \gamma$$

if $\epsilon \in \text{First}_1(C)$ then consider $B \to \beta\, A\, \gamma$ for r.-c.-extraction

- $B \to \beta\, A\, x\, \gamma$ is transformed by *right-context-propagation* on A:

$$B \to \beta\, A\, x\, \gamma \quad\Rightarrow\quad B \to \beta\, \langle A x\rangle\, \gamma$$

- The appropriate rules, created from introducing $\langle A x\rangle \to \delta$ and $\langle x/B\rangle \to \eta$ are added to the grammar

LR(2) to LR(1)

Right-Context-Propagation Algorithm:

For $\langle A x\rangle$ with $A \to \alpha_1 \mid \ldots \mid \alpha_k$, if α_i matches

- $\gamma\, A$ for some $\gamma \in (N \cup T)^*$, then $\langle A x\rangle \to \gamma\, \langle A x\rangle$ is added
- else $\langle A x\rangle \to \alpha_i\, x$ is added

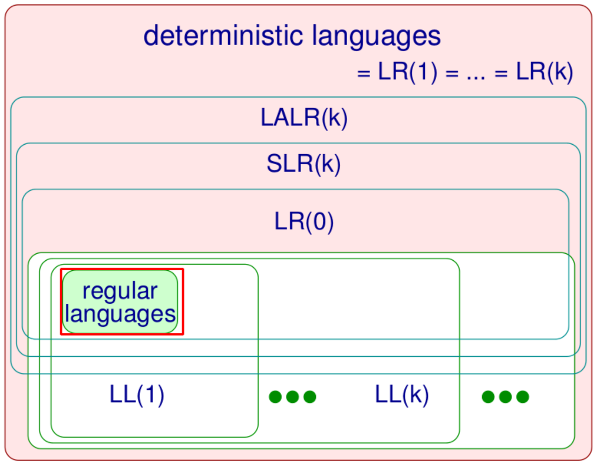
Right-Context-Extraction Algorithm:

For $\langle x/B\rangle$ with $B \to \alpha_1 \mid \ldots \mid \alpha_k$, if α_i matches

- $C\,\gamma$ for some $\gamma \in (N \cup T)^*$, then $\langle x/B\rangle \to \langle x/C\rangle\, \gamma$ is added
- $x\,\gamma$ for some $\gamma \in (N \cup T)^*$, then $\langle x/B\rangle \to \gamma$ is added
- $y\,\gamma$ for some $\gamma \in (N \cup T)^*$ and $y \neq x$, then nothing is added

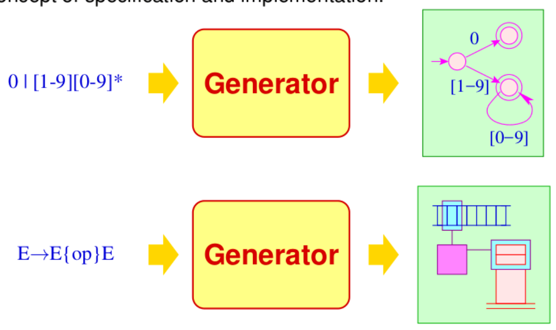
Chapter 5:
Summary

Parsing Methods



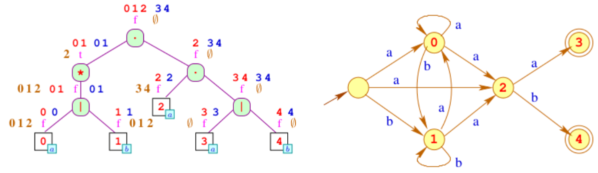
Lexical and Syntactical Analysis:

Concept of specification and implementation:

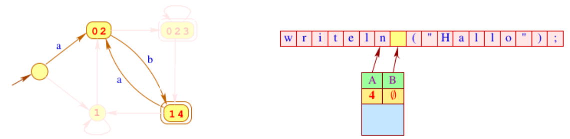


Lexical and Syntactical Analysis:

From Regular Expressions to Finite Automata

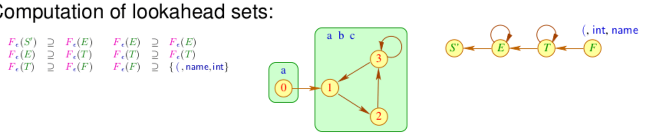


From Finite Automata to Scanners

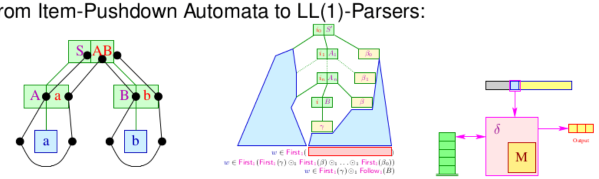


Lexical and Syntactical Analysis:

Computation of lookahead sets:

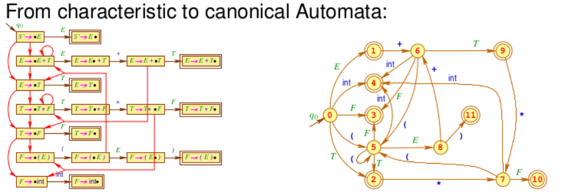


From Item-Pushdown Automata to LL(1)-Parsers:

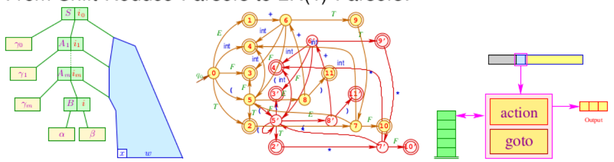


Lexical and Syntactical Analysis:

From characteristic to canonical Automata:



From Shift-Reduce-Parsers to LR(1)-Parsers:



Topic:

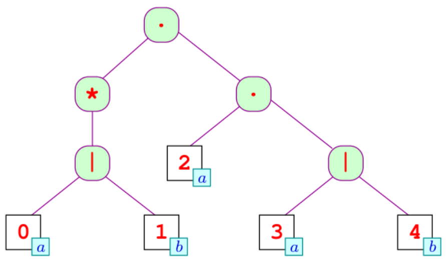
Semantic Analysis

Chapter 1:

Attribute Grammars

Example: Computation of the empty$[r]$ Attribute

Consider the syntax tree of the regular expression (a|b)*a(a|b):



Implementation Strategy

- attach an attribute empty to every node of the syntax tree
- compute the attributes in a *depth-first* post-order traversal:
  - at a leaf, we can compute the value of empty without considering other nodes
  - the attribute of an inner node only depends on the attribute of its children
- the empty attribute is a *synthetic* attribute
- The local dependencies between the attributes are dependent on the *type* of the node

Implementation Strategy

- attach an attribute empty to every node of the syntax tree
- compute the attributes in a *depth-first* post-order traversal:
  - at a leaf, we can compute the value of empty without considering other nodes
  - the attribute of an inner node only depends on the attribute of its children
- the empty attribute is a *synthetic* attribute
- The local dependencies between the attributes are dependent on the *type* of the node

in general:

Definition

An attribute is called

- synthetic if its value is always propagated upwards in the tree (in the direction leaf $\rightarrow$ root)
- inherited if its value is always propagated downwards in the tree (in the direction root $\rightarrow$ leaf)

Attribute Equations for empty

In order to compute an attribute *locally*, we need to specify attribute equations for each node.
These equations depend on the *type* of the node:

for leaves: $r \equiv \boxed{i \mid x}$ we define $\text{empty}[r] = (x \equiv \epsilon)$.

otherwise:

$$\begin{array}{lcl}
\text{empty}[r_1 \mid r_2] & = & \text{empty}[r_1] \vee \text{empty}[r_2] \\
\text{empty}[r_1 \cdot r_2] & = & \text{empty}[r_1] \wedge \text{empty}[r_2] \\
\text{empty}[r_1^*] & = & t \\
\text{empty}[r_1?] & = & t
\end{array}$$

Specification of General Attribute Systems

General Attribute Systems

In general, for establishing attribute systems we need a flexible way to *refer to parents and children*:

$\leadsto$ We use consecutive indices to refer to neighbouring attributes

$\text{attribute}_k[0]$:	the attribute of the current root node
$\text{attribute}_k[i]$:	the attribute of the i-th child $(i > 0)$

Observations

- the *local* attribute equations need to be evaluated using a *global* algorithm that knows about the dependencies of the equations
- in order to construct this algorithm, we need
  1. a sequence in which the nodes of the tree are visited
  2. a sequence within each node in which the equations are evaluated
- this *evaluation strategy* has to be compatible with the *dependencies* between attributes

Observations

- in order to infer an evaluation strategy, it is not enough to consider the *local* attribute dependencies at each node
- the evaluation strategy must also depend on the *global* dependencies, that is, on the information flow between nodes
- the global dependencies thus change with each new syntax tree
- in the example, the parent node is always depending on children only
  $\leadsto$ a depth-first post-order traversal is possible
- in general, variable dependencies can be much *more complex*

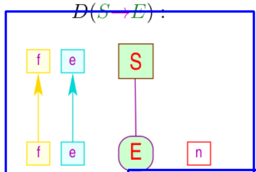
Simultaneous Computation of Multiple Attributes

Computing empty, first, next from regular expressions:

$$\boxed{S \to E:} \quad : \quad \begin{array}{lcl} \text{empty}[0] & := & \text{empty}[1] \\ \text{first}[0] & := & \text{first}[1] \\ \text{next}[1] & := & \emptyset \end{array}$$

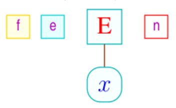
$$\boxed{E \to x} \quad : \quad \begin{array}{lcl} \text{empty}[0] & := & (x \equiv \epsilon) \\ \text{first}[0] & := & \{x \mid x \neq \epsilon\} \\ & & // \quad \text{(no equation for next)} \end{array}$$

$D(S \to E)$:



$D(S \to E) = \{ \ (empty[1], empty[0]), \ (first[1], first[0]) \}$

$D(E \to x)$:

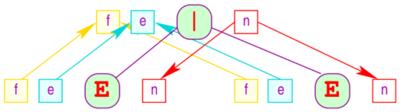


$D(E \to x) = \{ \quad \}$

Regular Expressions: Rules for Alternative

$$\boxed{E \to E|E} \quad : \quad \begin{array}{lcl} \text{empty}[0] & := & \text{empty}[1] \vee \text{empty}[2] \\ \text{first}[0] & := & \text{first}[1] \cup \text{first}[2] \\ \text{next}[1] & := & \text{next}[0] \\ \text{next}[2] & := & \text{next}[0] \end{array}$$

$D(E \to E|E)$:

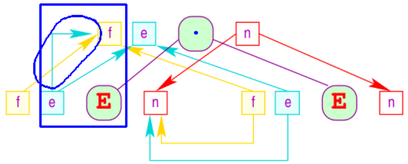


$$D(E \to E|E) = \{ \begin{array}{l} (empty[1], empty[0]), \\ (empty[2], empty[0]), \\ (first[1], first[0]), \\ (first[2], first[0]), \\ (next[0], next[2]), \\ (next[0], next[1]) \} \end{array}$$

Regular Expressions: Rules for Concatenation

$$\boxed{E \to E \cdot E} \quad : \quad \begin{array}{lcl} \text{empty}[0] & := & \text{empty}[1] \wedge \text{empty}[2] \\ \text{first}[0] & := & \text{first}[1] \cup (\text{empty}[1] \ ? \ \text{first}[2] : \emptyset) \\ \text{next}[1] & := & \text{first}[2] \cup (\text{empty}[2] \ ? \ \text{next}[0] : \emptyset) \\ \text{next}[2] & := & \text{next}[0] \end{array}$$

$D(E \to E \cdot E)$:



$$D(E \to E \cdot E) = \{ \begin{array}{l} (empty[1], empty[0]), \\ (empty[2], empty[0]), \\ (empty[2], next[1]), \\ (empty[1], first[0]), \\ (first[1], first[0]), \\ (first[2], first[0]), \\ (first[2], next[1]), \\ (next[0], next[2]), \\ (next[0], next[1]) \} \end{array}$$

Regular Expressions: Kleene-Star and '?'

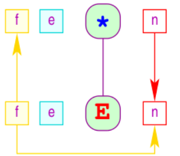
$\boxed{E \to E*}$:

$$\begin{array}{lcl} \text{empty}[0] & := & t \\ \text{first}[0] & := & \text{first}[1] \\ \text{next}[1] & := & \text{first}[1] \cup \text{next}[0] \end{array}$$

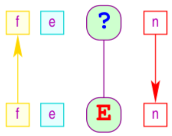
$\boxed{E \to E?}$:

$$\begin{array}{lcl} \text{empty}[0] & := & t \\ \text{first}[0] & := & \text{first}[1] \\ \text{next}[1] & := & \text{next}[0] \end{array}$$

$D(E \to E*)$:



$D(E \to E?)$:



$D(E \to E*) = \{ (first[1], first[0]), (first[1], next[2]), (next[0], next[1]) \}$

$D(E \to E?) = \{ (first[1], first[0]), (next[0], next[1]) \}$

Challenges for General Attribute Systems

Static evaluation

Is there a static evaluation strategy, which is generally applicable?

- an evaluation strategy can only exist, if for *any* derivation tree the dependencies between attributes are acyclic
- it is *DEXPTIME*-complete to check for cyclic dependencies [Jazayeri, Odgen, Rounds, 1975]

Subclass: Strongly Acyclic Attribute Dependencies

Idea: For all nonterminals X compute a set $\mathcal{R}(X)$ of relations between its attributes, as an *overapproximation of the global dependencies* between root attributes of every production for X.

Describe $\mathcal{R}(X)$s as sets of relations, similar to $D(p)$ by

- setting up each production $X \mapsto X_1 \ldots X_k$ s effect on the relations of $\mathcal{R}(X)$
- compute effect on all so far accumulated evaluations of each rhs X_i's $\mathcal{R}(X_i)$
- iterate until stable

Subclass: Strongly Acyclic Attribute Dependencies

The 2-ary operator $L[i]$ re-decorates relations from L

$$L[i] = \{(a[i], b[i]) \mid (a, b) \in L\}$$

π_0 projects only onto relations between root elements only

$$\pi_0(S) = \{(a, b) \mid (a[0], b[0]) \in S\}$$

Subclass: Strongly Acyclic Attribute Dependencies

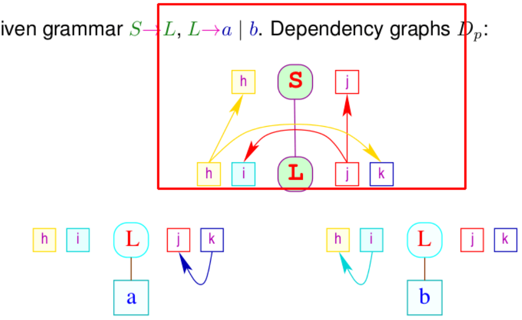
Strongly Acyclic Grammars

If all $D(p) \cup \mathcal{R}^\star(X_1)[1] \cup \ldots \cup \mathcal{R}^\star(X_k)[k]$ are acyclic for all $p \in G$, G is strongly acyclic.

Idea: we compute the least solution $R^\star(X)$ of $R(X)$ by a fixpoint computation, starting from $R(X) = \emptyset$.

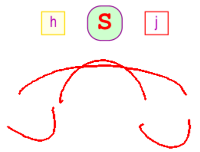
Example: Strong Acyclic Test

Given grammar $S\to L$, $L\to a \mid b$. Dependency graphs $\mathcal{D}_p$:



Example: Strong Acyclic Test

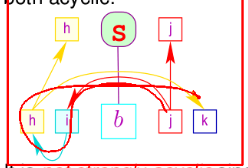
Continue with $\mathcal{R}(S) = [\![S\to L]\!]^\sharp(\mathcal{R}(L))$:

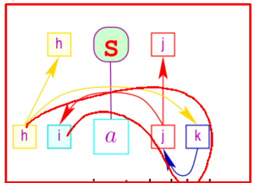


1. re-decorate and embed $\mathcal{R}(L)[1]$
2. transitive closure of all relations $(D(S\to L) \cup \{(k[1], j[1])\} \cup \{(i[1], h[1])\})^+$
3. apply π_0
4. $\mathcal{R}(S) = \{\}$

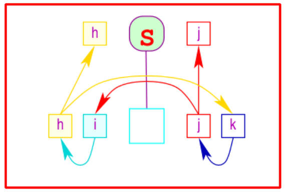
Strong Acyclic and Acyclic

The grammar $S\to L$, $L\to a \mid b$ has only two derivation trees which are both acyclic:





It is *not strongly acyclic* since the over-approximated global dependence graph for the non-terminal L contributes to a cycle when computing $\mathcal{R}(S)$:



From Dependencies to Evaluation Strategies

Possible strategies: