Berry-Sethi Approach

... for example:

Powerset Construction

... for example:

Remarks:
- This construction is known as Berry-Sethi- or Glushkov-construction.
- It is used for XML to define Content Models
- The result may not be, what we had in mind...
**Theorem:**
For every non-deterministic automaton \( A = (Q, \Sigma, \delta, I, F) \) we can compute a deterministic automaton \( \mathcal{P}(A) \) with
\[
\mathcal{L}(A) = \mathcal{L}(\mathcal{P}(A))
\]

---

**Observation:**
There are exponentially many powersets of \( Q \)
- Idea: Consider only contributing powersets. Starting with the set \( Q_p = \{1\} \) we only add further states by need ...
- i.e., whenever we can reach them from a state in \( Q_p \)
- However, the resulting automaton can become enormously huge ...
  - which is (sort of) not happening in practice
Powerset Construction

... for example:

```
O2        O2 O3
a a b     a a a b
01         14
```

Observation:
There are exponentially many powersets of Q.

- Idea: Consider only contributing powersets. Starting with the set \( Q_p = \{ I \} \) we only add further states by need ...
- i.e., whenever we can reach them from a state in \( Q_p \)
- However, the resulting automaton can become enormously huge ...
- which is (sort of) not happening in practice

Remarks:

- For an input sequence of length \( u \), maximally \( O(u) \) sets are generated
- Once a set/edge of the DFA is generated, they are stored within a hash-table.
- Before generating a new transition, we check this table for already existing edges with the desired label.
Chapter 5: Scanner design

Implementation:

Idea:
- Create the DFA $P(A_e) = (Q, \Sigma, \delta, q_0, F_e)$ for the expression $e = e_1 \ldots e_k$.
- Define the sets:
  - $F_1 = \{ q \in F \mid \text{last}[e_1] \neq \emptyset \}$
  - $F_2 = \{ q \in (F \setminus F_1) \mid \text{last}[e_2] \neq \emptyset \}$
  - $\ldots$
  - $F_k = \{ q \in (F \setminus \bigcup F_{i-1}) \mid q \cap \text{last}[e_k] \neq \emptyset \}$
- For input $w$ we find: $\delta^*(q_0, w) \in F_i$ iff the scanner must execute action, for $w$

Extension: States

- Now and then, it is handy to differentiate between particular scanner states.
- In different states, we want to recognize different token classes with different precedences.
- Depending on the consumed input, the scanner state can be changed

Example: Comments

Within a comment, identifiers, constants, comments, ... are ignored
Basics of Contextfree Grammars

Chapter 1:

In general, parsers are not developed by hand, but generated from a specification:

Input (generalized):

- The start state is called (e.g. lex,JFlex) "YINITIAL".
- The state table (e.g. lex,JFlex) "YBEGIN(STATE); reads the current state..."

Remarks:

- For every state we generate the scanner respective state.
- Comments might be directly implemented as (similarly overcomplicated) token classes.
- Scanner states are especially handy for implementing preprocessors, expanding special fragments in regular programs.

Syntactic Analysis

Discussion:

Generator

Specification

Parser

For example:

...
Basics: Context-free Grammars

- Programs of programming languages can have arbitrary numbers of tokens, but only finitely many Token-classes.
- This is why we choose the set of Token-classes to be the finite alphabet of terminals $T$.
- The nested structure of program components can be described elegantly via context-free grammars...

Definition: Context-Free Grammar

A context-free grammar (CFG) is a 4-tuple $G = (N, T, P, S)$ with:
- $N$ the set of nonterminals,
- $T$ the set of terminals,
- $P$ the set of productions or rules, and
- $S \in N$ the start symbol.

Conventions

The rules of context-free grammars take the following form:

$$ A \rightarrow \alpha \quad \text{with} \quad A \in N, \quad \alpha \in (N \cup T)^* $$

... for example:

$$ S \rightarrow a S b $$
$$ S \rightarrow \epsilon $$

Specified language: $\{a^n b^n \mid n \geq 0\}$

Conventions:

In examples, we specify nonterminals and terminals in general implicitly:
- Nonterminals are: $A, B, C, ..., \text{(exp)}, \text{(stmt)}, ...$
- Terminals are: $a, b, c, ..., \text{int}, \text{name}, ...$
... a practical example:

\[ S \rightarrow (\text{stmt}) \]

\[ \text{stmt} \rightarrow (\text{if}) | (\text{while}) | (\text{exp}) \]

\[ (\text{if}) \rightarrow \text{if} (\text{exp}) (\text{stmt}) \text{ else } (\text{stmt}) \]

\[ (\text{while}) \rightarrow \text{while} (\text{exp}) (\text{stmt}) \]

\[ (\text{exp}) \rightarrow \text{int} | (\text{exp}) \text{ } (\text{exp}) = (\text{exp}) \text{ } \ldots \]

\[ (\text{exp}) \rightarrow \text{name} \text{ } \ldots \]

More 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 \( j \geq 0 \)).

\[ (\text{stmt}, 5) \]

Pair of grammars:

\[
\begin{array}{c|c|c|c}
E & \rightarrow & E + E & | \ E \ | \ (E) & \text{name} & \text{int} \\
E & \rightarrow & E * F & | \ F \\
T & \rightarrow & T * F & | \ F \\
F & \rightarrow & (E) & \text{name} & \text{int} \\
\end{array}
\]

Both grammars describe the same language.

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:

\[ \mathcal{L}(G) = \{ w \in T^* | S \rightarrow^* w \} \]

Derivation Tree

Derivations of a symbol are represented as derivation trees:

... for example:

\[
\begin{array}{c|c|c|c}
E & \rightarrow & E + T & | \ T + T \\
E & \rightarrow & T + T & | \ T \\
T & \rightarrow & T * F & | \ F \\
F & \rightarrow & (E) & \text{name} \text{ int} & \text{name} \text{ int} + T \\
F & \rightarrow & \text{name} \text{ int} + T \\
\end{array}
\]

A derivation tree for \( A \in N \):
- inner nodes: rule applications
- root: rule application for \( A \)
- leaves: terminals or \( \epsilon \)

The successors of \( (B,i) \) correspond to right hand sides of the rule.
Special Derivations

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

- These are called leftmost (or rather rightmost) derivations and are denoted with the index ℋ (or R respectively).
- Leftmost (or rightmost) derivations correspond 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:

独特的语法

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

... 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)

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 $\text{yield}(t) = w$.

... in our example:

\[
\begin{array}{c|c|c|c}
E & \rightarrow & E \times E^* & E \times E^* \\
E & \rightarrow & E \times T^* & T^* \\
T & \rightarrow & T \times F^* & F^* \\
F & \rightarrow & (E)^0 & \text{name}^1 \quad \text{int}^2
\end{array}
\]

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 interest.
- Derivation trees are one-to-one corresponding with leftmost derivations as well as (reverse) rightmost derivations.

Chapter 2:
Basics of Pushdown Automata