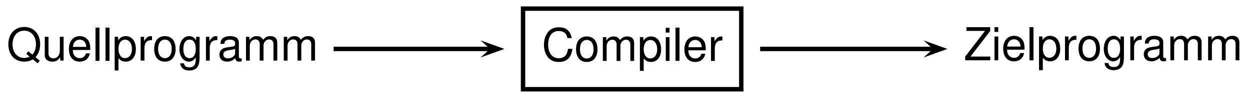
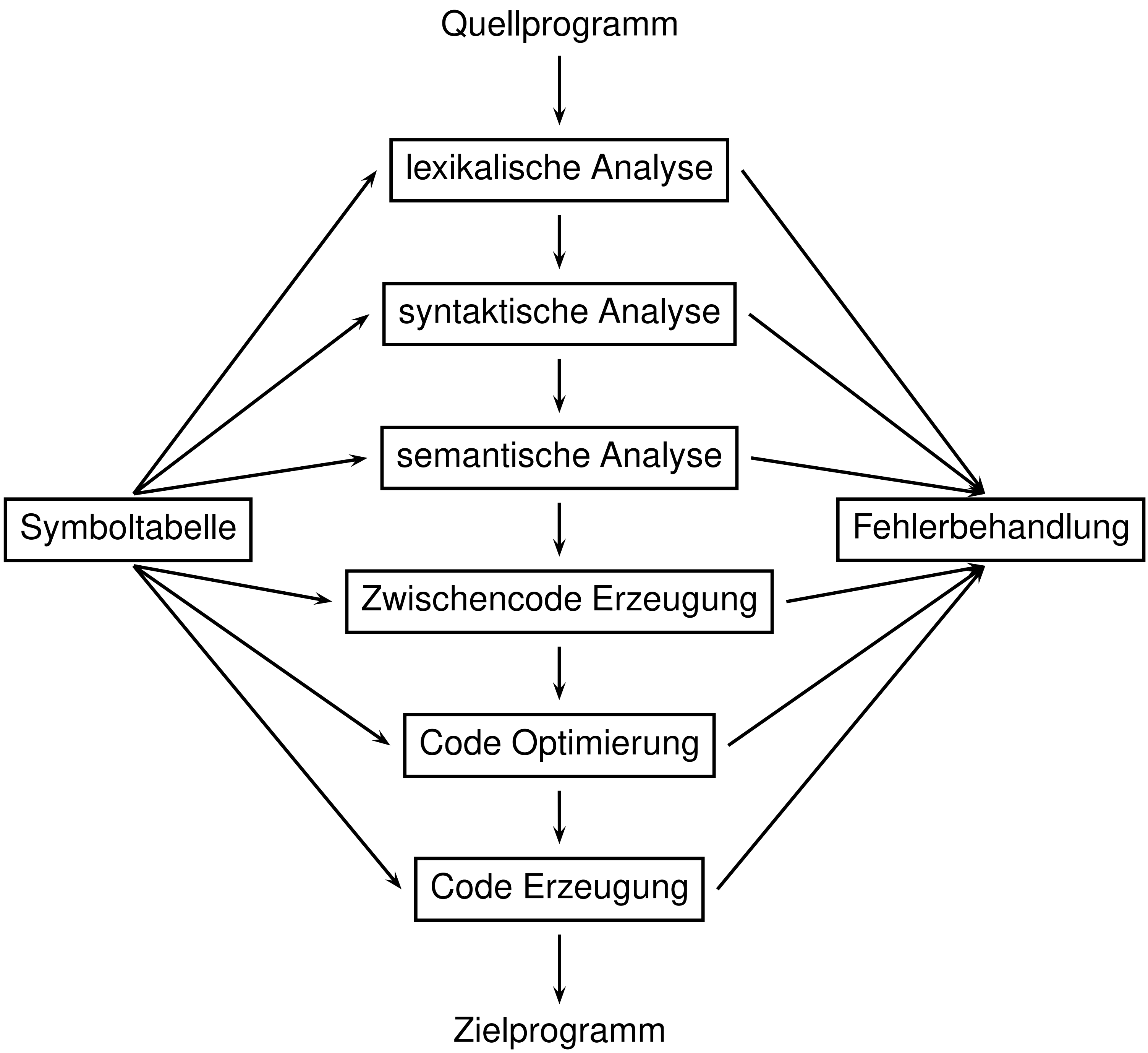
Compiler



Phasen eines Compilers:



Lexikalische Analyse: Scanner

Aufgabe: Extrahiere aus dem Eingabestring die nächste "Einheit", z.B. Namen einer Variablen, eine Zahl, reserviertes Wort (wie z.B. if, while, etc.), "+"-Zeichen, usw.

BEISPIEL: **identifier** := letter(letter|digit)*

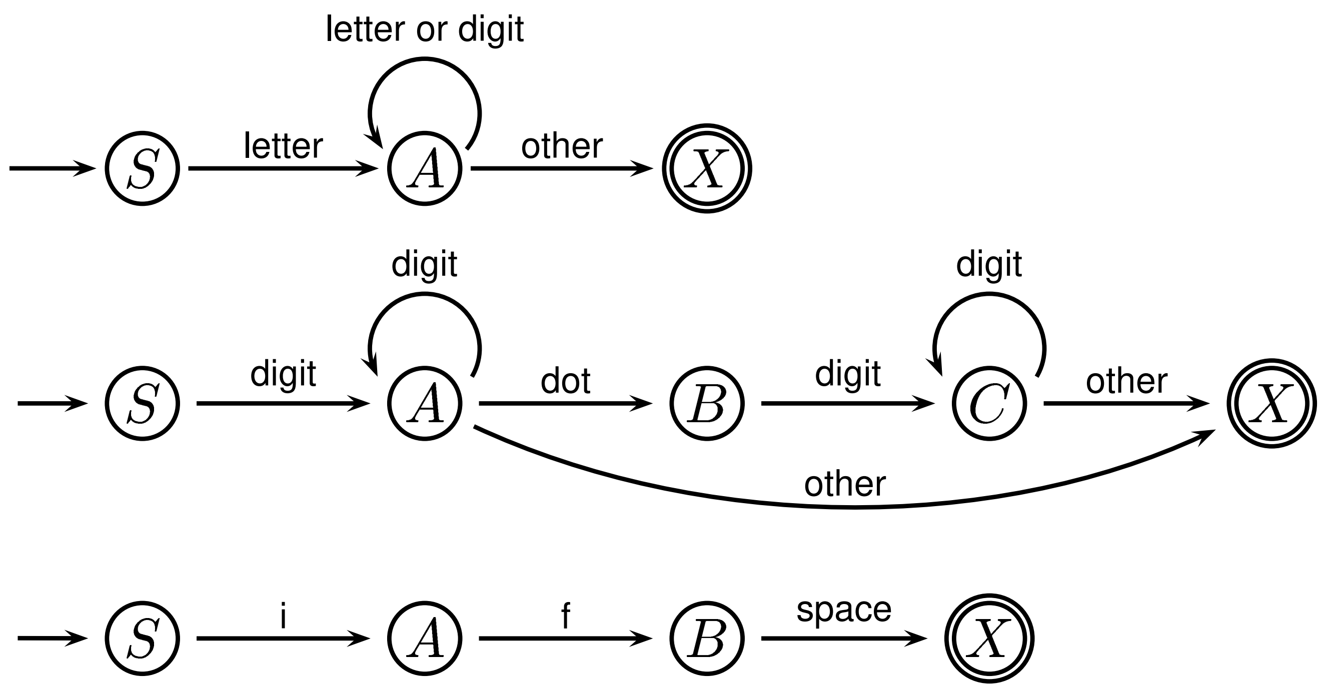
number := digit$^+$ | digit$^+$.(digit)$^+$

if := if

Anwendung der Erkenntnisse der Vorlesung:

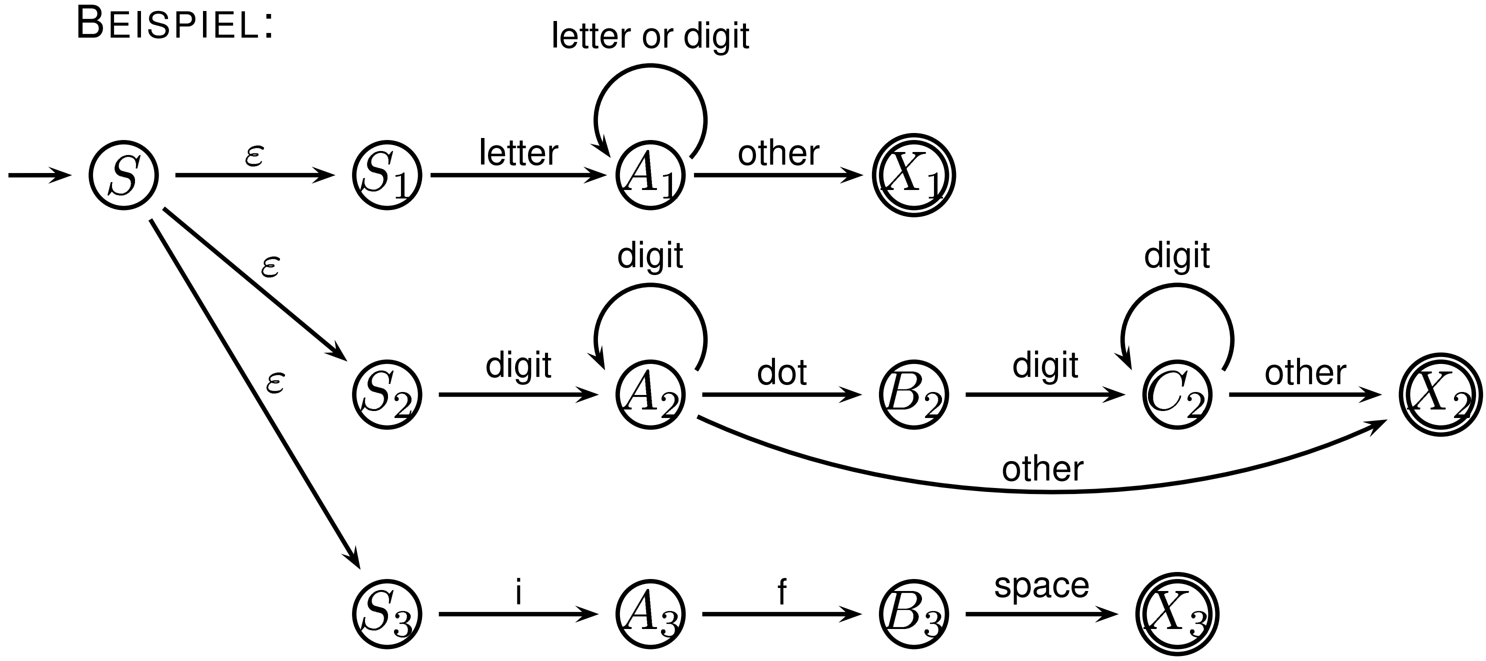
Zu jedem regulären Ausdruck gibt es einen endlichen Automaten, der genau die Wörter aus dieser Sprache erkennt.

BEISPIEL:



Für die lexikalische Ananlyse verbindet man diese endlichen Automaten zu einem einzigen *nichtdeterministischen* Automaten.

BEISPIEL:



Anwendung der Erkenntnisse der Vorlesung:

Mit Hilfe der *Potenzmengenkonstruktion* kann man daraus wieder einen deterministischen endlichen Automaten bauen ... und aus diesem kann man dann relativ einfach ein C oder Java Programm erzeugen.

Tools: Es gibt fertige Programme, die aus regulären Ausdrücken den zugehörigen Parser erzeugen.

Klassisch: **lex** (A Lexical Analyzer Generator)

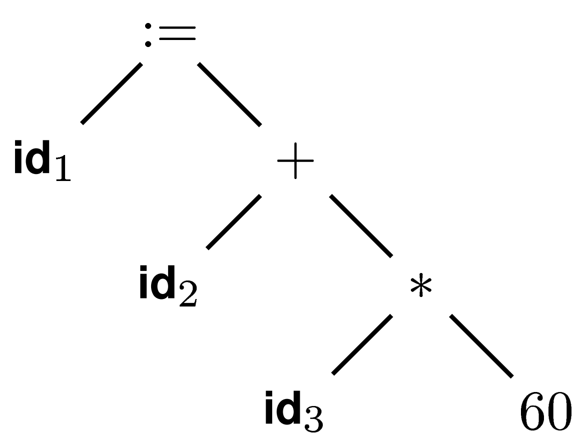
Gnu-Version: **flex**

More info: http://www.combo.org/lex_yacc_page/

Syntaktische Analyse: Parser

Aufgabe: Extrahiere aus der vom Scanner bereitgestellten Eingabe die logische Struktur.

BEISPIEL: Aus **id**$_1$:= **id**$_2$ + **id**$_3$ * 60 soll werden:



Ansatz: Die zulässige Syntax eines Programms wird durch eine (möglichst deterministische) kontextfreie Grammatik beschrieben.

Anwendung der Erkenntnisse der Vorlesung:

- Mit Hilfe des CYK-Algorithmus kann man in $\mathcal{O}(n^3)$ Zeit aus einem Wort die zugehörige Ableitung rekonstruieren.
- Ist die Grammatik deterministisch kontext-frei geht dies sogar in linearer Zeit.

Tools: Es gibt fertige Programme, die aus einer (geeignet spezifizierten) Grammatik einen zugehörigen (effizienten) Scanner erzeugen.

Klassisch: **yacc** (Yet Another Compiler-Compiler)
Gnu-Version: **bison**

More info: http://www.combo.org/lex_yacc_page/