Syntaxgesteuerte Editoren und Baummaschinen

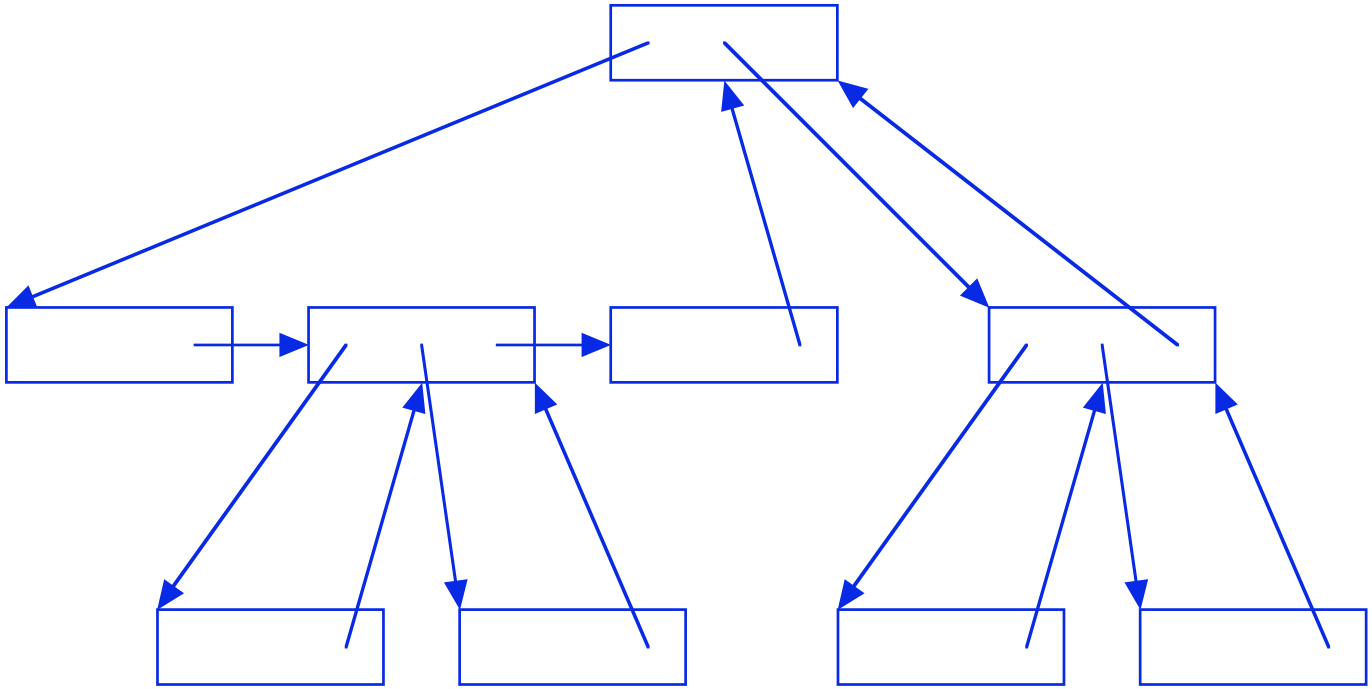
Texteingabe wird durch Schablonen gesteuert

```
PROGRAM <name>  (<namelist> );
    <declaration>
BEGIN
    <variable>  :=  <expression>
END.
```

Anforderungen an den Zwischencode

- bijektive Abbildung zwischen Quellprogramm und Zwischencode
- speichereffizient
- laufzeiteffizient
- Rückwertsausführung muß möglich sein
- Semantik leicht erkennbar
- einfaches Einfügen und Löschen von Code

hierarchischer Zwischencode
(abstrakter Syntaxbaum)



Besonderheiten des Zwischencodes

- Zeiger auf nächsten Knoten oder zurück zum Vaterknoten
- ermöglicht Abarbeitung des Baumes ohne Stapel
- ein Offsetfeld gibt die Position im Vaterknoten an
- Vorgängerknoten wird über Vaterknoten erreicht
- Zeiger können Sprungbefehle sein; Feld vor dem Sprungziel enthält dann den Offset zum Knotenanfang

Karel der Roboter

Computerspiel zum Erlernen des Programmierens

- Roboter bewegt sich in einem rechtwinkeligen Straßennetz
- besitzt einen Sack mit beeper
- er kann beeper wahrnehmen, auslegen und einsammeln
- er kann sich vorwärtsbewegen und nach links drehen
- er wird in einer einfachen Programmiersprache programmiert

Die Roboterprogrammiersprache

- keine Daten, rekursive Unterprogramme

Sprachelemente:

- Unterprogrammdefinition
- Block-Anweisung
- WHILE-Schleife
- Zählschleife (ITERATE)
- IF-THEN-ELSE -Anweisung
- fixer Satz an Bedingungen
- Unterprogrammaufruf
- move, turnleft, pickbeeper, putbeeper, turnoff

Knotendefinition

```
abstract class Node {
    int offset;
    Node next;

    abstract void print(int level);
    abstract int exec_step();
    abstract int back_step();
}
```

Knotenarten

ProgramNode	{DefineNode define; ExecutionNode execution};
DefineNode	{Name name; Node instruction};
ExecutionNode	{Node instruction};
BlockNode	{Node instruction};
WhileNode	{int test; Node instruction};
IterateNode	{int count; Node instruction};
IfThenElseNode	{int test; Node then_stmt; Node else_stmt};
CallNode	{Name name};
BasicInstrNode	{int instruction};
Name	{String name; DefineNode define; Name next};

Implementierung des Interpreters

- Befehlszähler besteht aus Knotenzeiger und Offset
- Stapel für Rücksprungadressen und ITERATE-Zähler

Rückwärtsausführung:

- einfache Befehle sind umkehrbar
- Vorgängerknoten wird über Vaterknoten erreicht

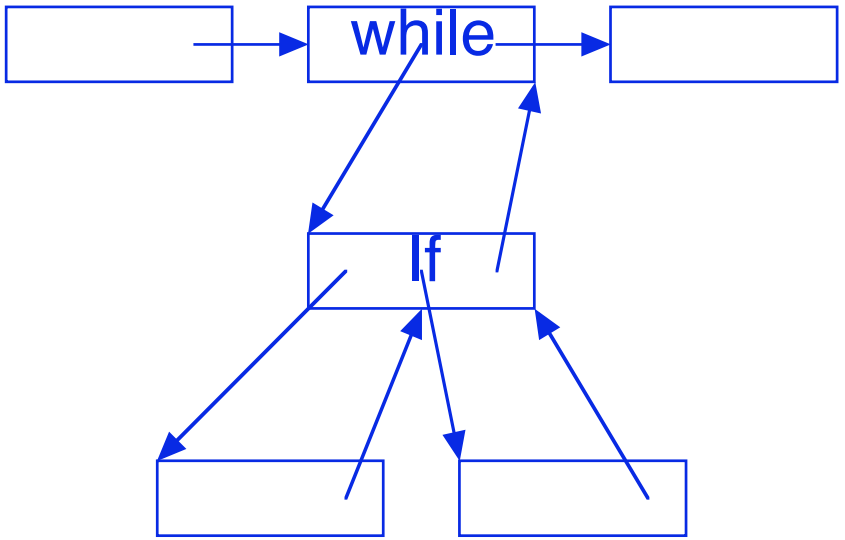
Rückwärtsausführung von Schleifen und Verzweigungen

Bedingungen werden nach der Befehlsausführung in einem Stapel gespeichert

vorwärts →

FTTFT

rückwärts ←



Analyse der Befehle erfolgt tabellengesteuert

Indizes: Zustand des Programmbaumes, eingegebene Zeichen

Inhalt: Unterprogramm mit optionalen Parameter

Ausgabe des Programmtextes:

- Schablonen stehen in Tabelle (mit Zeileninformation)
- Einrücktiefe ist die Tiefe des Syntaxbaums

Programmablauf des Editors

- Initialisierung
- Eingabeaufforderung
- Einlesen des Befehls
- Analyse und Auswahl des Befehls
- Ausführen des Befehls
- Ausgabe des Programmtextes
- solange Name oder Nummer
  - Einlesen von Name oder Nummer
  - Ausgabe des Programmtextes
- bis Befehl gleich Quit

Syntaxgesteuerte Editoren, Interpreter und Compiler

Mentor: Arbeiten an diesem Projekt seit 1975, Strukturediteor, Mental = Editordefinitionssprache, Softwareentwicklungs-umgebung, INRIA, Frankreich

Gandalf (ALOE): A Language Oriented Editor Editorgenerator, Softwareentwicklungsumgebung, CMU

COPS: Cornel Program Synthesizer, Editorgenerator mit Semantiküberprüfung (attributierte Grammatik), Interpreter und Debugger für Syntaxbaum, Cornell Univ.

PSG: Programm System Generator, TH Darmstadt

Aktuelle Systeme

XText: ANTLR (LL(k) Parsergenerator) + Softwareentwicklungs-umgebunggenerator

Truffle: mit Graal (in Java geschriebener JIT-Compiler für Java) Spezialisierer für Interpreter

PyPy: JIT- Compiler für Python, Spezialisierer für Interpreter