Programmiertechnik 1

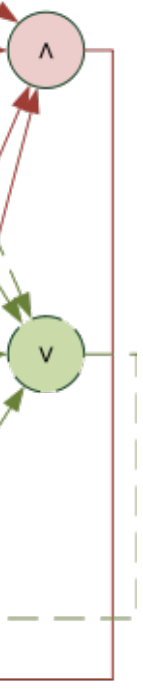
Übung 2

Feedback

- Aufgabenabgabesystem
  - Probleme?
  - Gruppen!
  - Kommentare lesen …
  - Name/Matrikel-Nr.

Normalformen

A	B	C	Ergebnis	Klausel
0	0	0	0	A ∨ B ∨ C
0	0	1	0	A ∨ B ∨ ¬C
0	1	0	1	¬A ∧ B ∧ ¬C
0	1	1	1	¬A ∧ B ∧ C
1	0	0	0	¬A ∨ B ∨ C
1	0	1	1	A ∧ ¬B ∧ C
1	1	0	0	¬A ∨ ¬B ∨ C
1	1	1	1	A ∧ B ∧ C



DNF: (¬A ∧ B ∧ ¬C) ∨ (¬A ∧ B ∧ C) ∨ (A ∧ ¬B ∧ C) ∨ (A ∧ B ∧ C)

KNF: (A ∨ B ∨ C) ∧ (A ∨ B ∨ ¬C) ∧ (¬A ∨ B ∨ C) ∧ (¬A ∨ ¬B ∨ C)

- Ableiten einer booleschen Funktion aus einer Wahrheitswerte-Tabelle
- **konjunktive Normalform**:
  - Konjunktion von Disjunktionstermen
  - Disjunktionsterme sind Disjunktionen von Literalen
  - Literale sind nichtnegierte oder negierte Variablen
- **disjunktiver Normalform:**
  - Disjunktion von Konjunktionstermen
  - Konjunktionsterme sind konjunktive Verknüpfungen von Literalen

Normalformen

- Erzeugen Sie die disjunktive Normalform DNF(f) und die konjunktive Normalform KNF(f) aus den folgenden Ausdrücken:

- $f(x, y, z) = (x \wedge y) \oplus (\neg x \vee z)$

- $g(x, y, z) = (x \wedge y) \vee z$

- $h(x, y, z) = x \oplus y \vee x$

/bin/sh

- Konfiguration
  - .bashrc
  - .login
  - .profile

- PATH-Variable
  - which
- HOME-Directory
- CWD-Variable
  - pwd
  - cd
  - cd –
  - OLDPWD

Erweiterte Backus Naur Form (EBNF)

- Extended BNF verbessert Lesbarkeit und Genauigkeit von BNF
- Kreuz (Plus +) - ein oder mehrere Elemente
  - <unsigned integer> ::= <digit>+
- Stern (*) – Folge aus Null oder mehr Elementen
  - <identifier> ::= <letter><alphanumeric>*
- Runde oder Geschweifte Klammern zur Gruppierung von Elementen
  - Auf diese Weise können Hilfsregeln eingespart werden
  - Wie die für <alphanumeric>
  - <identifier> ::= <letter>{<letter>|<digit>}*
- Eckige Klammern können optionale Elemente anzeigen
  - <integer> ::= [+ | -]<unsigned integer>
  - Kreuz oder Stern können nicht mit eckigen Klammern kombiniert werden
- BNF-Darstellung ist stets auch eine EBNF-Darstellung

EBNF-Regel

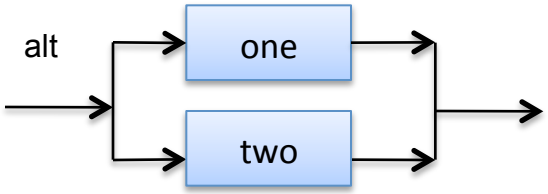
Verschiedene Notationen

- Hilfssymbole werden mit ::=-Regeln definiert
- Komposition von Regeln:
  - Hintereinanderschreiben von Hilfssymbolen und Terminalsymbolen
  - Alternative Regeln für ein Hilfssymbol: R1 | R2
  - Optionale Folge von Symbole: [S1 S2 ...]
  - beliebige Wiederholung (auch 0-mal): { S1 S2 ... } *
  - beliebige Wiederholung (wenigstens 1-mal): { S1 S2 ... }+
- Beispiel: Selection (if/switch)-Anweisung in C

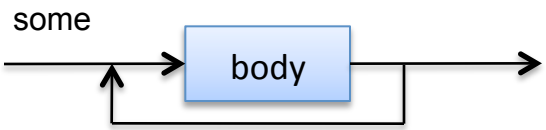
```
selection_stat      ::= 'if' '(' exp ')' stat
                    |   'if' '(' exp ')' stat 'else' stat
                    |   'switch' '(' exp ')' stat
```

EBNF Syntaxgraphen

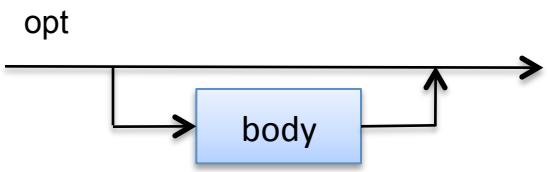
- Syntaxgraphen korrespondieren wesentlich intuitiver zu Regeln, wenn man sich auf Schreibweisen für Wiederholungen und standardisierte Graphen einigt



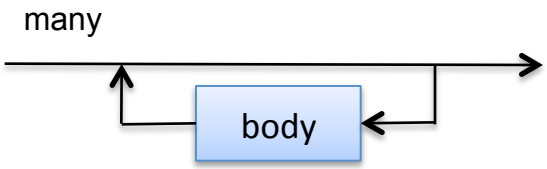
alt ::= (one | two) ;



some ::= body+ ;



opt ::= [body] ;



many ::= body* ;

- Analog zu den Syntaxgraphen sehen arithmetische Ausdrücke dann etwa so aus:

  sum ::= product { ('+'|'-') product }*;

  product ::= term { ('*'| ‘/') term }*;

  term ::= { Number | '(' sum ') ‘ } ;

- EBNF benötigt keine leeren Alternativen (anders als BNF)

BNF für C – ein Ausschnitt

```
stat  ::=       labeled_stat | exp_stat
          |     compound_stat | selection_stat
          |     iteration_stat | jump_stat;
labeled_stat::=   id ':' stat
                  | 'case' const_exp ':' stat
                  | 'default' ':' stat;
exp_stat::=     exp '; ‘
                | ';';
compound_stat::=     '{' decl_list stat_list '} ‘
                     | '{'stat_list '} ‘
                     | '{' decl_list'} ‘
                     | '{''}';
stat_list::=      stat
                | stat_list stat;
selection_stat::=      'if' '(' exp ')' stat
                     | 'if' '(' exp ')' stat 'else' stat
                     | 'switch' '(' exp ')' stat;
```

```
iteration_stat::=      'while' '(' exp ')' stat
                     | 'do' stat 'while' '(' exp ')' '; ‘
                     | 'for' '(' exp ';' exp ';' exp ')' stat
                     | 'for' '(' exp ';' exp ';'')' stat
                     | 'for' '(' exp ';'';' exp ')' stat
                     | 'for' '(' exp ';'';'')' stat
                     | 'for' '('';' exp ';' exp ')' stat
                     | 'for' '('';' exp ';'')' stat
                     | 'for' '('';'';' exp ')' stat
                     | 'for' '('';'';'')' stat
                    ;
```

Sätze und Syntaxgraphen

- X ::= a b { c }*
- Y ::= a* b* c*
- Z ::= X {X | Y}* [Z]

Kleines Programm: Rechnen

- Zwei per Kommandozeile übergebene ganze Zahlen werden:
  - Addiert
  - Multipliziert

- Nächster Schritt: Reelle Zahlen

Vi

- Beispiele:
  - :%s/unsigned int/float/g — Alle “unsigned int” durch “float” ersetzen
  - :wq — Speichern und Schließen
  - u, CTRL+R — Undo & Redo
  - r — Ersetze aktuelles Zeichen
  - . — Repeat
  - yy, p — Copy & Paste

- Für alles andere gibt es vimtutor

Ein weiteres Programm

- Fahrenheit-nach-Celsius-Converter
  - Eingabewert als Kommandozeilenparameter
  - Umrechnung soll in folgender Funktion geschehen:
    - float fahrenheit_to_celsius(float celsius);
  - Kleiner Tipp: °C = (°F - 32) * 5/9