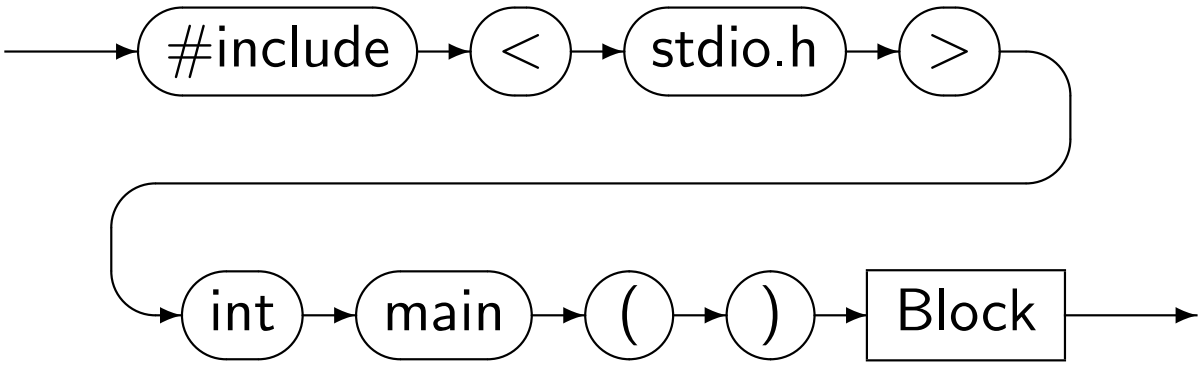
Syntaxgesteuerte Übersetzung (I)

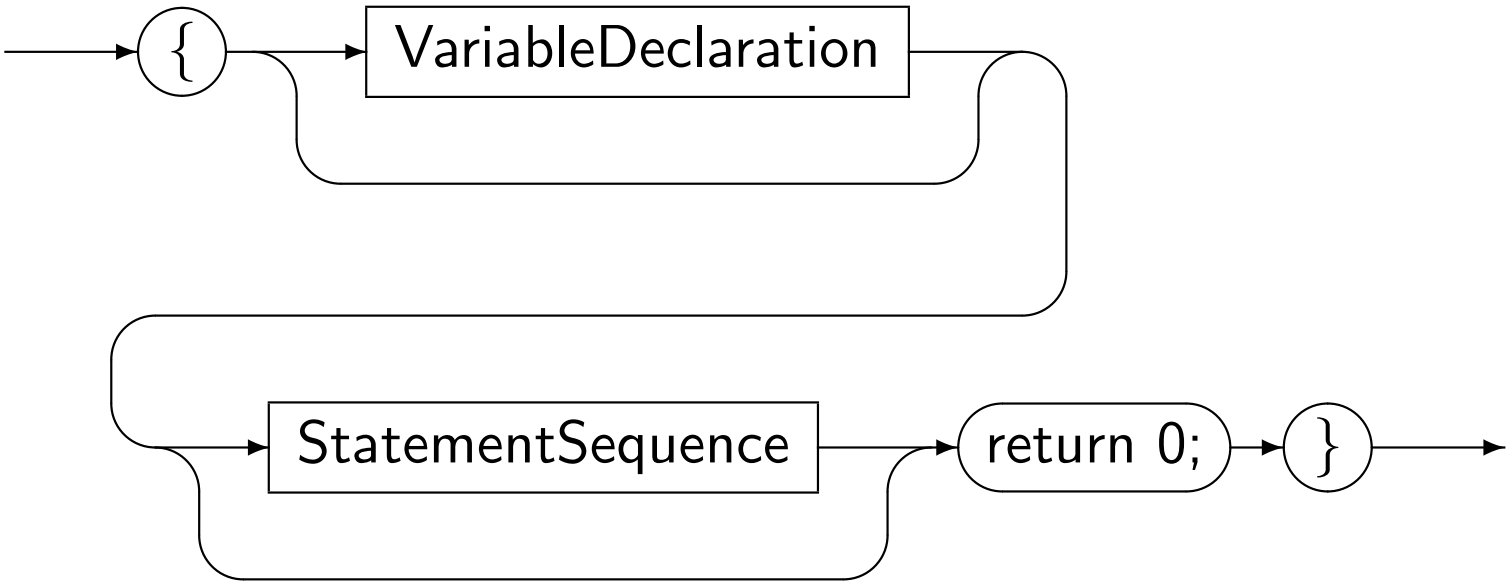
Program



$\rightsquigarrow$ $\underline{trans}$(**#include** <stdio.h> **int** main() $block)
$= \underline{blocktrans}(\$block)$
für alle $\$block \in W(\langle \text{Block} \rangle)$

Syntaxgesteuerte Übersetzung (II)

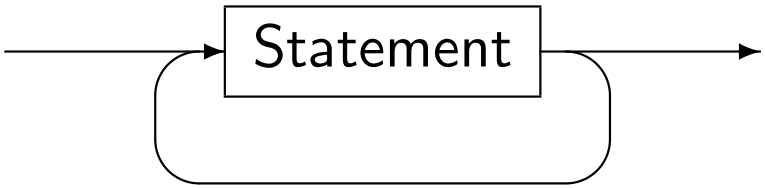
Block



$\rightsquigarrow$ $\underline{blocktrans}(\{\$vardecl\ \$statseq\ \textbf{return}\ 0;\})$
$= \underline{stseqtrans}(\$statseq, \cdots)$
für alle $\$vardecl \in \{\varepsilon\} \cup W(\langle \text{VariableDeclaration} \rangle)$
und $\$statseq \in \{\varepsilon\} \cup W(\langle \text{StatementSequence} \rangle)$

Syntaxgesteuerte Übersetzung (III)

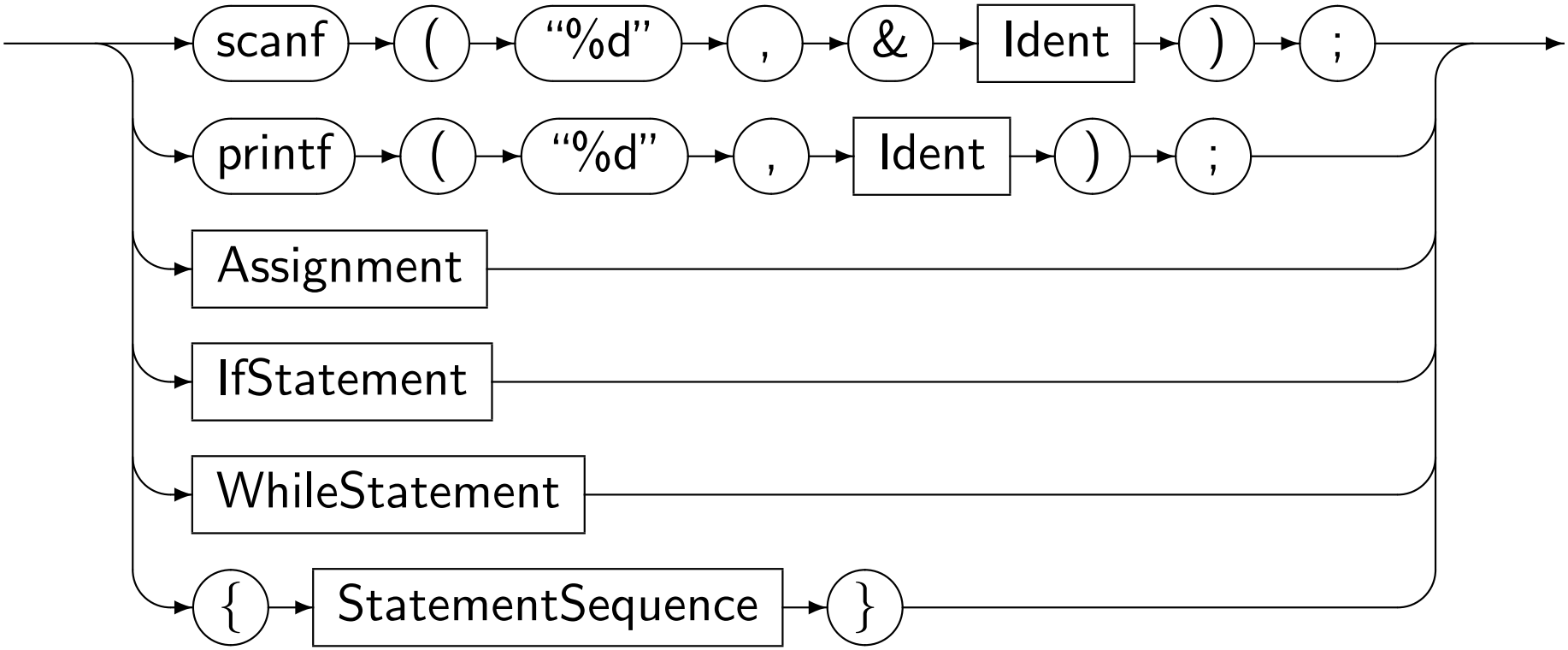
StatementSequence



$$\rightsquigarrow \underline{stseqtrans}(\$stat_1 \ \ldots \ \$stat_n, \cdots)$$
$$= \underline{sttrans}(\$stat_1, \cdots)$$
$$\vdots$$
$$\underline{sttrans}(\$stat_n, \cdots)$$
für alle $\$stat_1, \ldots, \$stat_n \in W(\langle \text{Statement} \rangle)$

Syntaxgesteuerte Übersetzung (IV)

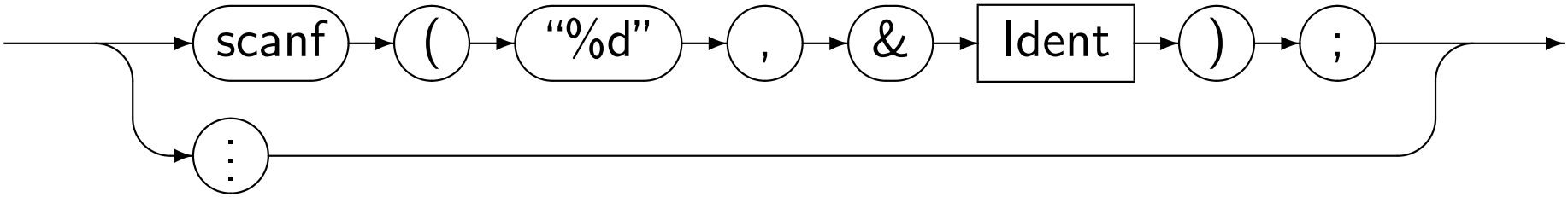
Statement



$\rightsquigarrow$ Fallunterscheidung,
zum Beispiel: $\underline{sttrans}(\text{scanf}(\text{"\%d"},\&\$id\,);,\cdots)$
$= \text{READ}\ ?$
für alle $\$id \in W(\langle\text{Ident}\rangle)$

Syntaxgesteuerte Übersetzung (IV)

Statement

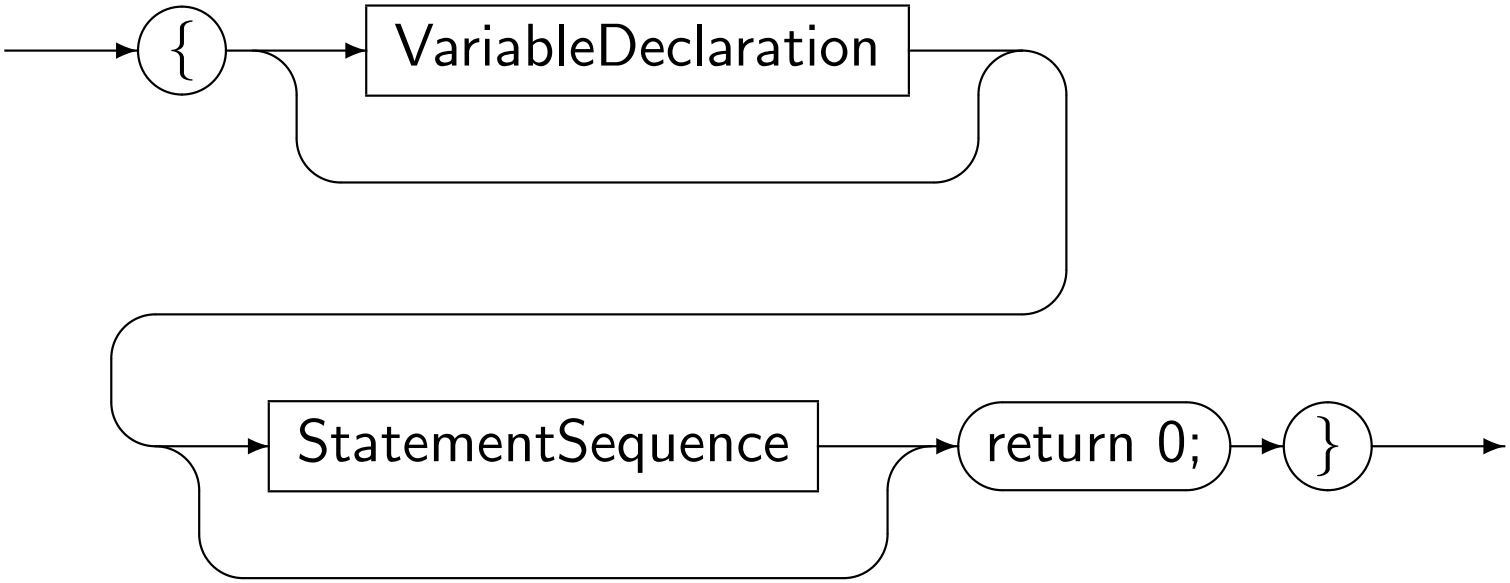


$\rightsquigarrow$ Fallunterscheidung,
zum Beispiel: $\underline{sttrans}(\text{scanf}(\text{"\%d"},\&\$id\,);,\cdots)$
$= \text{READ}\ ?$
für alle $\$id \in W(\langle\text{Ident}\rangle)$

Wir brauchen Informationen über die Zuordnung von Bezeichnern (im Programm) zu Speicherplätzen (im HS der AM)!

Erzeugung einer sogenannten Symboltabelle (I)

Block

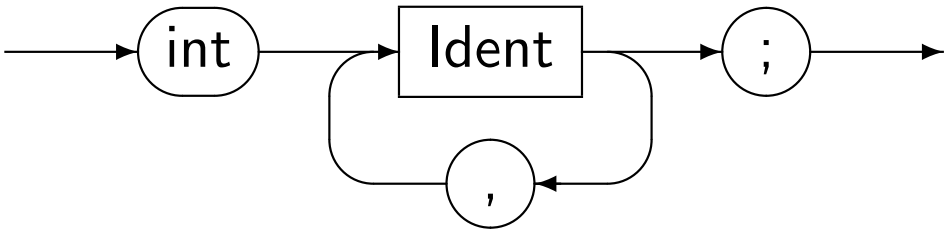


$\rightsquigarrow$ *blocktrans*({$*vardecl* $*statseq* **return** 0;})
$= \underline{\textit{stseqtrans}}(\$\textit{statseq}, \underline{\textit{mksymtab}}(\$\textit{vardecl}), \cdots)$
für alle $\$\textit{vardecl} \in \{\varepsilon\} \cup W(\langle\text{VariableDeclaration}\rangle)$
und $\$\textit{statseq} \in \{\varepsilon\} \cup W(\langle\text{StatementSequence}\rangle)$

Menge der Symboltabellen:
$\text{Tab} = \{tab \mid tab : W(\langle\text{Ident}\rangle) \rightarrow\!\!\!\!\!\rightarrow \mathbb{N}_+\}$

Erzeugung einer sogenannten Symboltabelle (II)

VariableDeclaration



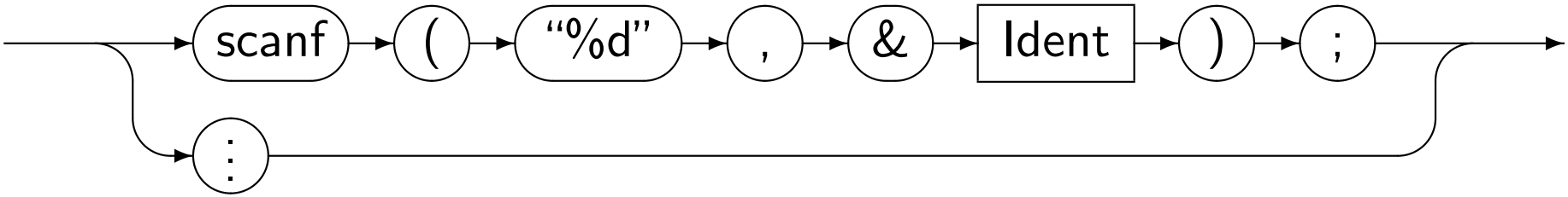
$\rightsquigarrow$ $\underline{mksymtab}(\varepsilon) = [\,]$ (leere Abbildung)
$\underline{mksymtab}(\textbf{int}\ \$id_1, \ldots, \$id_m;)$
$= [\$id_1/1, \ldots, \$id_m/m]$
für alle $\$id_1, \ldots, \$id_m \in W(\langle \text{Ident} \rangle)$

Die Symboltabelle wird von $\underline{stseqtrans}$ aus in weitere Übersetzungsfunktionen propagiert!

Zuordnung in der Symboltabelle ist eindeutig, wegen der ersten kontextsensitiven Nebenbedingung (keine Doppeldeklarationen).

Syntaxgesteuerte Übersetzung (IV)

Statement

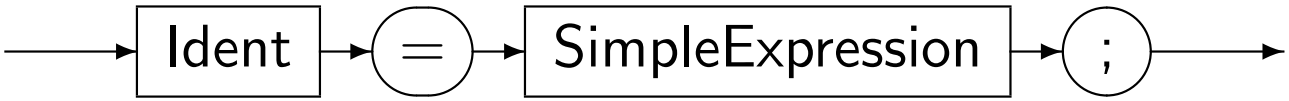


$\rightsquigarrow$ Fallunterscheidung,
zum Beispiel: $\underline{sttrans}(\text{scanf}(\text{"\%d"},\&\$id\,);, tab, \cdots)$
$=$ wenn $tab(\$id) = n$, dann READ n;
für alle $\$id \in W(\langle\text{Ident}\rangle)$ und $tab \in \text{Tab}$

Zugriff auf *tab* hier ist immer definiert, wegen der zweiten kontextsensitiven Nebenbedingung (nur deklarierte Bezeichner dürfen verwendet werden).

Syntaxgesteuerte Übersetzung (V)

Assignment



$\rightsquigarrow \underline{sttrans}(\$id = \$exp;, tab, \cdots)$
$= $ wenn $tab(\$id) = n$, dann:
$\underline{simpleexptrans}(\$exp, tab)$
STORE n;
für alle $\$id \in W(\langle\text{Ident}\rangle)$, $\$exp \in W(\langle\text{SimpleExpression}\rangle)$
und $tab \in \text{Tab}$

Geht auf, wenn (weil!) $\underline{simpleexptrans}(\$exp, tab)$ zu einem Stack mit dem Berechnungsergebnis an oberster Position führt (wobei nicht tiefer in den Datenkeller eingegriffen wird).

Einschub: Prinzip der Berechnungsübersetzung

Jede Rechnung in C_0 wird in AM mittels der sogenannten „reverse polish notation“ (Postfix-Notation) umgesetzt, d.h. Operatoren stehen hinter den Operanden.

Beispiel: $1 + 2 \Rightarrow 1\ 2\ +$

Der Vorteil ist, dass keine Klammerung mehr nötig ist, da jeder Operator nur so viel Operanden konsumiert, wie er benötigt.

Beispiel: $2 * (1 + 3 - 2) \Rightarrow 2\ 1\ 3\ +\ 2\ -\ *$

Legt man Operanden auf den Stack und führt Operatoren jeweils direkt aus, dann ergeben sich die Zwischen- und Endergebnisse.

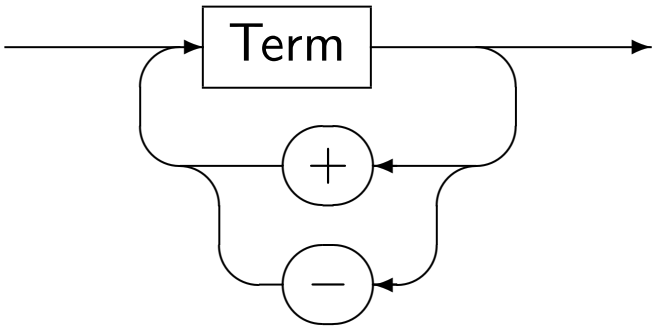
Beispiel:

$$2 * (1 + 3 - 2) \Rightarrow 2\ 1\ 3\ + \Rightarrow 2\ 4 \Rightarrow 2\ 4\ 2\ - \Rightarrow 2\ 2 \Rightarrow 2\ 2\ * \Rightarrow 4$$

Wir wissen nun auch: Jede Berechnung nimmt nie mehr vom Stack als sie drauflegt.

Syntaxgesteuerte Übersetzung (VI)

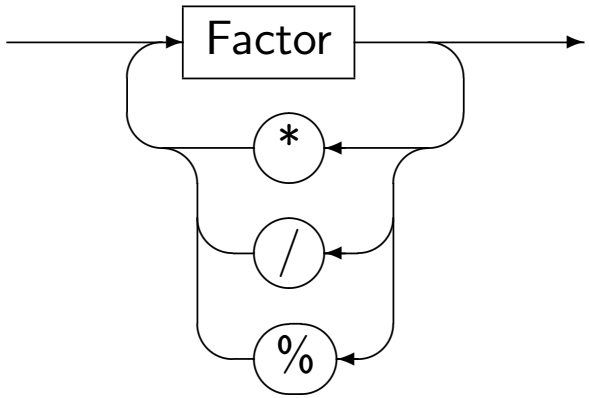
SimpleExpression



$\rightsquigarrow$ $\underline{simpleexptrans}(\$t_1\ \$op_2\ \$t_2\ \ldots\ \$op_n\ \$t_n, tab)$
$= \underline{termtrans}(\$t_1, tab)$
$\underline{termtrans}(\$t_2, tab)$
OP_2;
$\vdots$
$\underline{termtrans}(\$t_n, tab)$
OP_n;
für alle $\$t_1, \ldots, \$t_n \in W(\langle \mathrm{Term} \rangle)$,
$\$op_2, \ldots, \$op_n \in \{+, -\}$ und $tab \in \mathrm{Tab}$,
wobei $\mathrm{OP}_i = \mathrm{ADD}$, falls $\$op_i = +$
$\mathrm{OP}_i = \mathrm{SUB}$, falls $\$op_i = -$

Syntaxgesteuerte Übersetzung (VII)

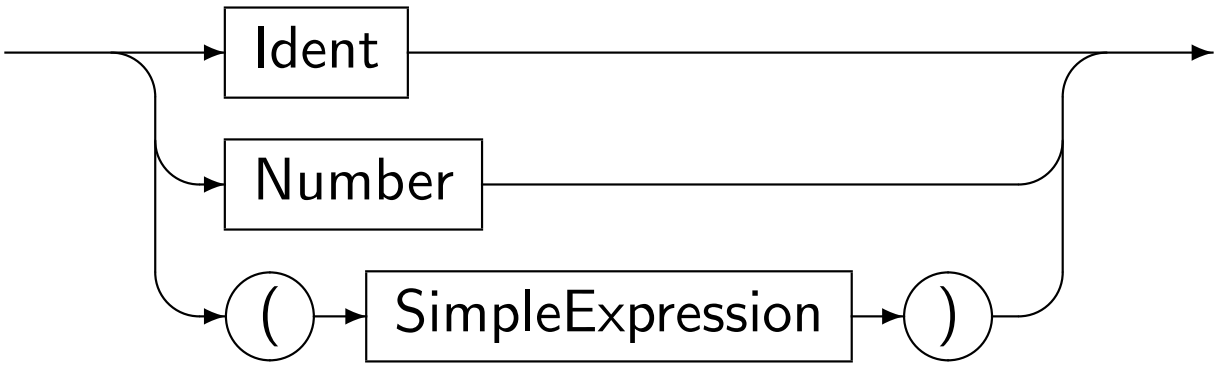
Term



$\rightsquigarrow$ $\underline{termtrans}(\$f_1\ \$op_2\ \$f_2\ \ldots\ \$op_n\ \$f_n, tab)$
$= \underline{factortrans}(\$f_1, tab)$
$\underline{factortrans}(\$f_2, tab)$
OP_2;
$\vdots$
$\underline{factortrans}(\$f_n, tab)$
OP_n;
für alle $\$f_1, \ldots, \$f_n \in W(\langle \mathrm{Factor} \rangle)$,
$\$op_2, \ldots, \$op_n \in \{*, /, \%\}$ und $tab \in \mathrm{Tab}$,
wobei $\mathrm{OP}_i = \mathrm{MUL}$, falls $\$op_i = *$
$\mathrm{OP}_i = \mathrm{DIV}$, falls $\$op_i = /$
$\mathrm{OP}_i = \mathrm{MOD}$, falls $\$op_i = \%$

Syntaxgesteuerte Übersetzung (VIII)

Factor



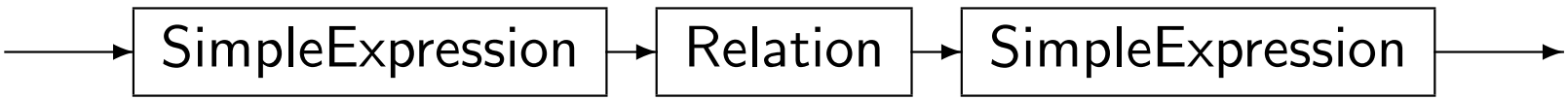
$\rightsquigarrow$ $\underline{factortrans}(\$id, tab)$
$=$ wenn $tab(\$id) = n$, dann LOAD n;
für alle $\$id \in W(\langle \text{Ident} \rangle)$ und $tab \in \text{Tab}$

$\underline{factortrans}(\$z, tab)$
$=$ LIT $\$z$;
für alle $\$z \in W(\langle \text{Number} \rangle)$ und $tab \in \text{Tab}$

$\underline{factortrans}((\$se), tab)$
$= \underline{simpleexptrans}(\$se, tab)$
für alle $\$se \in W(\langle \text{SimpleExpression} \rangle)$ und $tab \in \text{Tab}$

Syntaxgesteuerte Übersetzung (IX)

BoolExpression



$\rightsquigarrow$ $\underline{boolexptrans}(\$se_1\ \$rel\ \$se_2, tab)$
$=$ $\underline{simpleexptrans}(\$se_1, tab)$
$\underline{simpleexptrans}(\$se_2, tab)$
REL;
für alle $\$se_1, \$se_2 \in W(\langle \text{SimpleExpression} \rangle)$,
$\$rel \in \{==, !=, <, >, <=, >=\}$ und $tab \in \text{Tab}$,
wobei REL $=$ EQ, falls $\$rel = ==$
REL $=$ NE, falls $\$rel = !=$
REL $=$ LT, falls $\$rel = <$
$\ldots$

Syntaxgesteuerte Übersetzung (X)

WhileStatement

$\rightsquigarrow$ *sttrans*(**while** ($*exp*) $*stat*, *tab*, $\cdots$)
$=$ *boolexptrans*($*exp*, *tab*)
JMC ?;
sttrans($*stat*, *tab*, $\cdots$)
JMP ?;
für alle $*exp* $\in W(\langle$BoolExpression$\rangle)$,
$*stat* $\in W(\langle$Statement$\rangle)$ und *tab* $\in$ Tab

Problem:
- keine konkreten Adressen bekannt
- hängen unter anderem von Länge des übersetzten Codes für $*exp* und $*stat* ab

Lösung:
- zunächst nur abstrakte Adressen, später Nachbearbeitung
- „baumstrukturierte Adressen“: Listen über natürlichen Zahlen (Notation 3.2.4.1)

Syntaxgesteuerte Übersetzung (X)

$\underline{sttrans}$(**while** ($exp) $stat, $tab, $a)
= a.2: $\underline{boolexptrans}$($exp, $tab)
JMC a;
$\underline{sttrans}$($stat, $tab, $a.1)
JMP a.2;
a:

für alle $exp $\in W(\langle \text{BoolExpression} \rangle)$,
$stat $\in W(\langle \text{Statement} \rangle)$, $tab $\in$ Tab und $a \in \mathbb{N}^*$

Syntaxgesteuerte Übersetzung (XI)

$\underline{blocktrans}(\{\$vardecl\ \$statseq\ \textbf{return}\ 0;\})$
$= \underline{stseqtrans}(\$statseq, \underline{mksymtab}(\$vardecl), 1)$
für alle $\$vardecl \in \{\varepsilon\} \cup W(\langle \text{VariableDeclaration} \rangle)$
und $\$statseq \in \{\varepsilon\} \cup W(\langle \text{StatementSequence} \rangle)$

$\underline{stseqtrans}(\$stat_1 \ \ldots \ \$stat_n, tab, a)$
$= \underline{sttrans}(\$stat_1, tab, a.1)$
$\ldots$
$\underline{sttrans}(\$stat_n, tab, a.n)$
für alle $\$stat_1, \ldots, \$stat_n \in W(\langle \text{Statement} \rangle)$, $tab \in \text{Tab}$ und $a \in \mathbb{N}^*$

Noch einige Fälle offen ...

Syntaxgesteuerte Übersetzung (XII)

$\underline{sttrans}$(**if** ($exp) $stat, tab, a)
= $\underline{boolexptrans}$($exp, tab)
JMC a;
$\underline{sttrans}$($stat, tab, a.1)
a:
für alle $exp $\in W(\langle$BoolExpression$\rangle)$, $stat $\in W(\langle$Statement$\rangle)$,
$tab \in$ Tab und $a \in \mathbb{N}^*$

$\underline{sttrans}$(**if** ($exp) $stat$_1$ **else** $stat$_2, tab, a)$
= $\underline{boolexptrans}$($exp, tab)
JMC a;
$\underline{sttrans}$($stat$_1, tab, a.1)$
JMP a.3;
a: $\underline{sttrans}$($stat$_2, tab, a.2)$
a.3:
für alle $exp $\in W(\langle$BoolExpression$\rangle)$,
$stat$_1, $stat$_2 \in W(\langle$Statement$\rangle)$, $tab \in$ Tab und $a \in \mathbb{N}^*$

Syntaxgesteuerte Übersetzung (XIII)

$\underline{sttrans}(\text{printf}\,(\texttt{"\%d"},\$id\,);\,, tab, a)$
$= \text{wenn } tab(\$id) = n, \text{ dann WRITE } n;$
für alle $\$id \in W(\langle\text{Ident}\rangle)$, $tab \in \text{Tab}$ und $a \in \mathbb{N}^*$

$\underline{sttrans}(\{\$stat_1 \ \ldots \ \$stat_n\}, tab, a)$
$= \underline{stseqtrans}(\$stat_1 \ \ldots \ \$stat_n, tab, a)$
für alle $\$stat_1, \ldots, \$stat_n \in W(\langle\text{Statement}\rangle)$,
$tab \in \text{Tab}$ und $a \in \mathbb{N}^*$

Übersetzungsfunktionen kompakt

Zusammenfassung (I)

$\underline{trans}$(**#include** <stdio.h> **int** main() $\$block$)
$= \underline{blocktrans}(\$block)$

$\underline{blocktrans}$({$\$vardecl$ $\$statseq$ **return** 0;})
$= \underline{stseqtrans}(\$statseq, \underline{mksymtab}(\$vardecl), 1)$

$\underline{mksymtab}(\varepsilon) = [\,]$
$\underline{mksymtab}(\textbf{int}\ \ \$id_1, \ldots, \$id_m;) = [\$id_1/1, \ldots, \$id_m/m]$

$\underline{stseqtrans}(\$stat_1\ \ \ldots\ \ \$stat_n, tab, a)$
$= \underline{sttrans}(\$stat_1, tab, a.1)$
$\ldots$
$\underline{sttrans}(\$stat_n, tab, a.n)$

Zusammenfassung (II)

$\underline{sttrans}(\$id = \$exp;, tab, a)$
= wenn $tab(\$id) = n$, dann:
  $\underline{simpleexptrans}(\$exp, tab)$
  STORE n;

$\underline{sttrans}(\textbf{if}\ (\$exp)\ \$stat, tab, a)$
= $\underline{boolexptrans}(\$exp, tab)$
  JMC a;
  $\underline{sttrans}(\$stat, tab, a.1)$
  a:

$\underline{sttrans}(\textbf{if}\ (\$exp)\ \$stat_1\ \textbf{else}\ \$stat_2, tab, a)$
= $\underline{boolexptrans}(\$exp, tab)$
  JMC a;
  $\underline{sttrans}(\$stat_1, tab, a.1)$
  JMP a.3;
  a: $\underline{sttrans}(\$stat_2, tab, a.2)$
  a.3:

Zusammenfassung (III)

$\underline{sttrans}$(**while** ($exp) $stat, tab, a)
= a.2: $\underline{boolexptrans}$($exp, tab)
JMC a;
$\underline{sttrans}$($stat, tab, a.1)
JMP a.2;
a:

$\underline{sttrans}$(scanf("%d",&$id);, tab, a)
= wenn tab($id) = n, dann READ n;

$\underline{sttrans}$(printf ("%d",$id);, tab, a)
= wenn tab($id) = n, dann WRITE n;

$\underline{sttrans}$({$stat_1 \ldots \ $stat_n$}, tab, a)
= $\underline{stseqtrans}$($stat_1 \ldots \ $stat_n$, tab, a)

Zusammenfassung (IV)

$\underline{boolexptrans}(\$se_1\ \$rel\ \$se_2, tab)$
$= \underline{simpleexptrans}(\$se_1, tab)$
$\underline{simpleexptrans}(\$se_2, tab)$
REL;
wobei REL $=$ EQ, falls $\$rel == $
$\ldots$

$\underline{simpleexptrans}(\$t_1\ \$op_2\ \$t_2\ \ldots\ \$op_n\ \$t_n, tab)$
$= \underline{termtrans}(\$t_1, tab)$
$\underline{termtrans}(\$t_2, tab)$
OP_2;
$\ldots$
$\underline{termtrans}(\$t_n, tab)$
OP_n;
wobei OP_i $=$ ADD, falls $\$op_i = +$
OP_i $=$ SUB, falls $\$op_i = -$

Zusammenfassung (V)

$\underline{termtrans}(\$f_1\ \$op_2\ \$f_2\ \ldots\ \$op_n\ \$f_n, tab)$
$= \underline{factortrans}(\$f_1, tab)$
$\underline{factortrans}(\$f_2, tab)$
OP_2;
$\ldots$
$\underline{factortrans}(\$f_n, tab)$
OP_n;
wobei $\mathrm{OP}_i = \mathrm{MUL}$, falls $\$op_i = *$
$\ldots$

$\underline{factortrans}(\$id, tab)$
$=$ wenn $tab(\$id) = n$, dann LOAD n;

$\underline{factortrans}(\$z, tab) = \mathrm{LIT}\ \$z$;

$\underline{factortrans}((\$se), tab) = \underline{simpleexptrans}(\$se, tab)$

Zusammenfassung (VI)

C_0 wird auf AM abgebildet, indem:

- „atomare Befehle“ (wie scanf oder printf) direkt umgesetzt/übersetzt werden,
- Kontrollstrukturen in (geeignet arrangierte) Sprünge übersetzt werden, wobei
  - wegen syntaktischer Schachtelung (Blockstruktur, komplexe Statements als Teile anderer Statements) die Übersetzung in flach strukturierten AM-Code nicht (bzw. schwer) möglich ist, daher ein Umweg über baumstrukturierte Adressen gegangen wird,
- Schachtelung bei Ausdrücken mittels Stackprinzip umgesetzt wird.