Überblick über die Vorlesung

1. Definition: Formale Sprachen
2. Reguläre Ausdrücke: Spezifikation von Strings
   wichtig in Skriptsprachen
   Tcl, *Perl*, *Python*, *Ruby*, ···
3. Anwendungen regulärer Ausdrücke
   `flex`: *fast lexical analyser generator*.
   Scanner-Generator
4. Endliche Automaten
5. Kontextfreie Sprachen
6. Antlr, *Bison*: Parser-Generatoren
7. Keller-Automaten
8. Theorie der LL(k)-Sprachen
   Grundlage von Antlr

Überblick, Teil II

1. Theorie der LL(∗)-Sprachen (Antlr)
2. Grenzen kontextfreier Sprachen
3. Interpreter
4. Theorie der LALR-Sprachen
   Grundlage von *Bison*
5. Der Parser-Generator *JavaCup*
   JavaCup ist ein LALR-Parser-Generator für *Java*.
6. Typ-Überprüfung
7. Entwicklung eines einfacher Compiler
8. Register-Zuordnung

Literatur

1. Skript

2. *Introduction to Automata Theory, Languages, and Computation*

   John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman

3. *lex & yacc*

   John R. Levine, Tony Mason, Doug Brown

4. *Compilers — Principles, Techniques and Tools*

   Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, Monica S. Lam

5. *Mastering Regular Expressions*

   Jeffrey E. F. Friedl

Algebra regulärer Ausdrücke

1. $r_1 + r_2 \doteq r_2 + r_1$
2. $(r_1 + r_2) + r_3 \doteq r_1 + (r_2 + r_3)$
3. $(r_1 \cdot r_2) \cdot r_3 \doteq r_1 \cdot (r_2 \cdot r_3)$
4. $\emptyset \cdot r \doteq r \cdot \emptyset \doteq \emptyset$
5. $\varepsilon \cdot r \doteq r \cdot \varepsilon \doteq r$
6. $\emptyset + r \doteq r + \emptyset \doteq r$
7. $(r_1 + r_2) \cdot r_3 \doteq r_1 \cdot r_3 + r_2 \cdot r_3$
8. $r_1 \cdot (r_2 + r_3) \doteq r_1 \cdot r_2 + r_1 \cdot r_3$
9. $r + r \doteq r$
10. $(r^*)^* \doteq r^*$
11. $\emptyset^* \doteq \varepsilon$
12. $\varepsilon^* \doteq \varepsilon$
13. $r^* \doteq \varepsilon + r^* \cdot r$
14. $r^* \doteq (\varepsilon + r)^*$

Schluss-Regel: $\dfrac{r \doteq r \cdot s + t \quad \varepsilon \notin L(s)}{r \doteq t \cdot s^*}$

Eine Grammatik für arithmetische Ausdrücke

ArithExpr	$\rightarrow$	*ArithExpr* “+” *Product*
	|	*ArithExpr* “-” *Product*
	|	*Product*
Product	$\rightarrow$	*Product* “*” *Factor*
	|	*Product* “/” *Factor*
	|	*Factor*
Factor	$\rightarrow$	“(” *ArithExpr* “)”
	|	`NUMBER`

Entfernen von Links-Rekursion

Gegeben: links-rekursive Grammatik für Variable A

$$
\begin{array}{rcl}
A & \rightarrow & A\beta_1 \\
 & | & A\beta_2 \\
 & \vdots & \vdots \\
 & | & A\beta_k \\
 & | & \gamma_1 \\
 & \vdots & \vdots \\
 & | & \gamma_l
\end{array}
$$

Transformation in nicht-links-rekursive Grammatik

$$
\begin{array}{rcl}
A & \rightarrow & \gamma_1\ L \mid \gamma_2\ L \mid \cdots \mid \gamma_l\ L \\
L & \rightarrow & \beta_1\ L \mid \beta_2\ L \mid \cdots \mid \beta_k\ L \mid \varepsilon
\end{array}
$$

Eine Grammatik für arithmetische Ausdrücke ohne Links-Rekursion

$$
\begin{array}{rcl}
\mathit{Expr} & \rightarrow & \mathit{Product}\ \ \mathit{ExprRest} \\
\mathit{ExprRest} & \rightarrow & \text{“+”}\ \mathit{Product}\ \ \mathit{ExprRest} \\
 & | & \text{“-”}\ \mathit{Product}\ \ \mathit{ExprRest} \\
 & | & \varepsilon \\
\mathit{Product} & \rightarrow & \mathit{Factor}\ \ \mathit{ProdRest} \\
\mathit{ProdRest} & \rightarrow & \text{“*”}\ \mathit{Factor}\ \ \mathit{ProdRest} \\
 & | & \text{“/”}\ \mathit{Factor}\ \ \mathit{ProdRest} \\
 & | & \varepsilon \\
\mathit{Factor} & \rightarrow & \text{“(”}\ \mathit{Expr}\ \text{“)”} \\
 & | & \mathit{Number}
\end{array}
$$

Eine Grammatik für reguläre Ausdrücke

RegExp	$\rightarrow$	*RegExp* "|" *RegExp*
	|	*RegExp* *RegExp*
	|	*RegExp* "*"
	|	"(" *RegExp* ")"
	|	"."
	|	Letter

Variablen: *RegExp*

Terminale: "|", "*", ".", "(", ")", Letter.

Diese Grammatik ist mehrdeutig!

Aufgabe: Transformieren Sie diese Grammatik in eine **eindeutige** Grammatik.

Eine eindeutige Grammatik für reguläre Ausdrücke

RegExp	$\rightarrow$	*RegExp* "|" *Product*
	|	*Product*
Product	$\rightarrow$	*Product Factor*
	|	*Factor*
Factor	$\rightarrow$	*Factor* "*"
	|	"(" *RegExp* ")"
	|	"."
	|	Letter

Diese Grammatik ist linkrekursiv!

Aufgabe: Eleminieren Sie die Links-Rekursion aus dieser Grammatik.

Eine nicht-links-rekursive Grammatik für reguläre Ausdrücke

$$
\begin{array}{rcl}
\mathit{RegExp} & \rightarrow & \mathit{ProductRegExpRest} \\
\mathit{RegExpRest} & \rightarrow & \text{“|”}\ \mathit{Product}\ \ \mathit{RegExpRest} \\
 & | & \varepsilon \\
\mathit{Product} & \rightarrow & \mathit{Factor}\ \ \mathit{ProdRest} \\
\mathit{ProdRest} & \rightarrow & \mathit{Factor}\ \ \mathit{ProdRest} \\
 & | & \varepsilon \\
\mathit{Factor} & \rightarrow & \text{“(”}\ \mathit{RegExp}\ \text{“)”}\ \mathit{Stars} \\
 & | & \text{“.”}\ \mathit{Stars} \\
 & | & \text{Letter}\ \ \mathit{Stars} \\
\mathit{Stars} & \rightarrow & \text{“*”}\ \mathit{Stars} \\
 & | & \varepsilon
\end{array}
$$

Algorithmus des Shift-Reduce-Parser

```
 1    symbols := [];
 2    states  := [ q0 ];
 3    while (not end-of-file) {
 4        q := states.top();
 5        t := peekNextToken();
 6        switch (action(q,t)) {
 7            case ⟨shift, s⟩: {
 8                symbols.push(t);
 9                states .push(s);
10                removeNextToken();
11            }
12            case ⟨reduce, A → x1 ··· xn ⟩: {
13                symbols.pop(n);
14                states .pop(n);
15                symbols.push(A);
16                s = states.top();
17                states.push(goto(s, A));
18            }
19            case accept: {
20                print("Parsen erfolgreich");
21                return;
22            }
23            case error: {
24                error();
25            }
26        }
27    }
```

Grammatik für arithmetische Ausdrücke

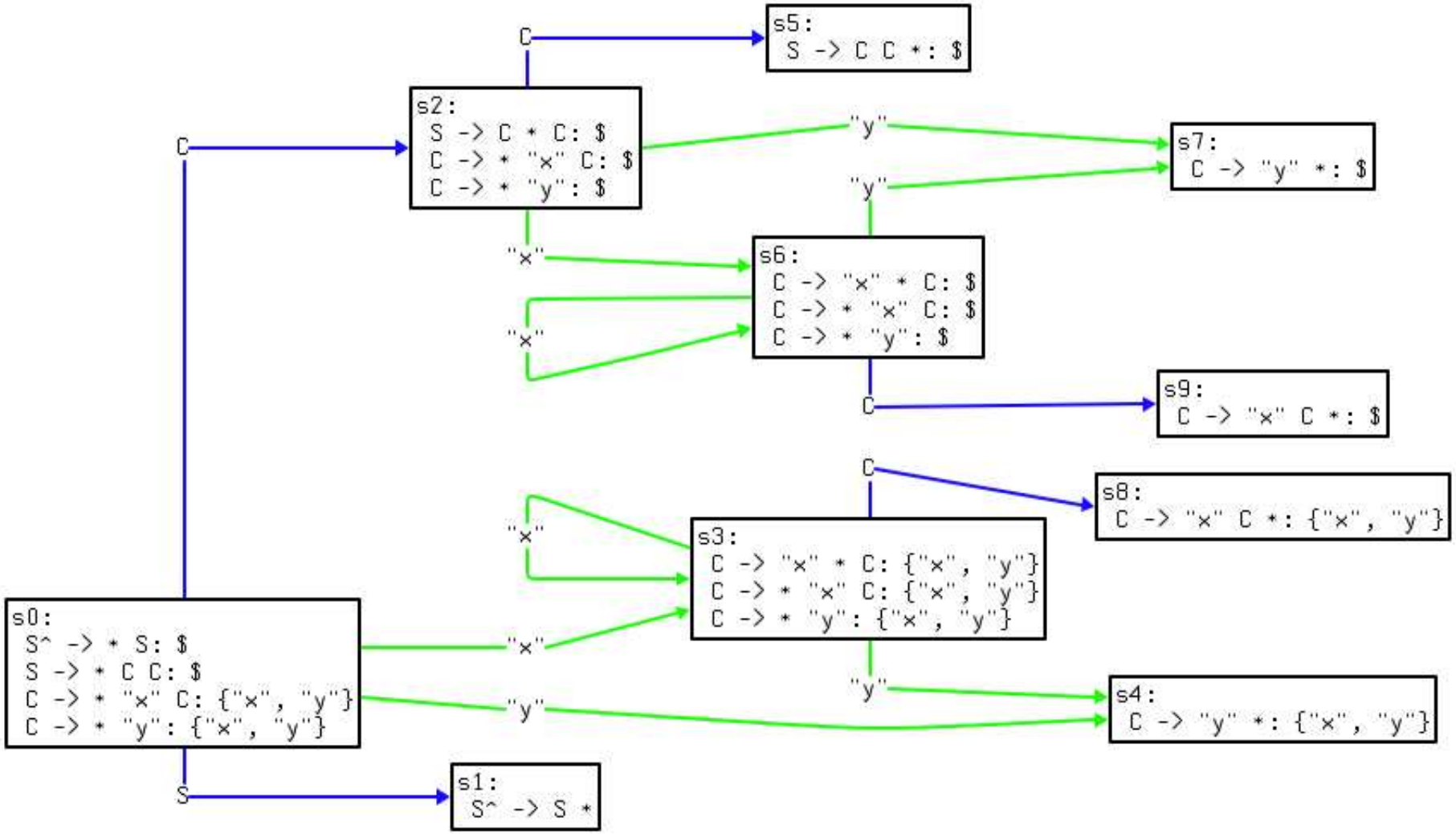
$$
\begin{array}{rcl}
\textit{Expr} & \rightarrow & \textit{Expr}\ \text{“+”}\ \textit{Product} \\
& | & \textit{Expr}\ \text{“-”}\ \textit{Product} \\
& | & \textit{Product} \\
\textit{Product} & \rightarrow & \textit{Product}\ \text{“*”}\ \textit{Factor} \\
& | & \textit{Product}\ \text{“/”}\ \textit{Factor} \\
& | & \textit{Factor} \\
\textit{Factor} & \rightarrow & \text{“(”}\ \textit{Expr}\ \text{“)”} \\
& | & \text{Number}
\end{array}
$$

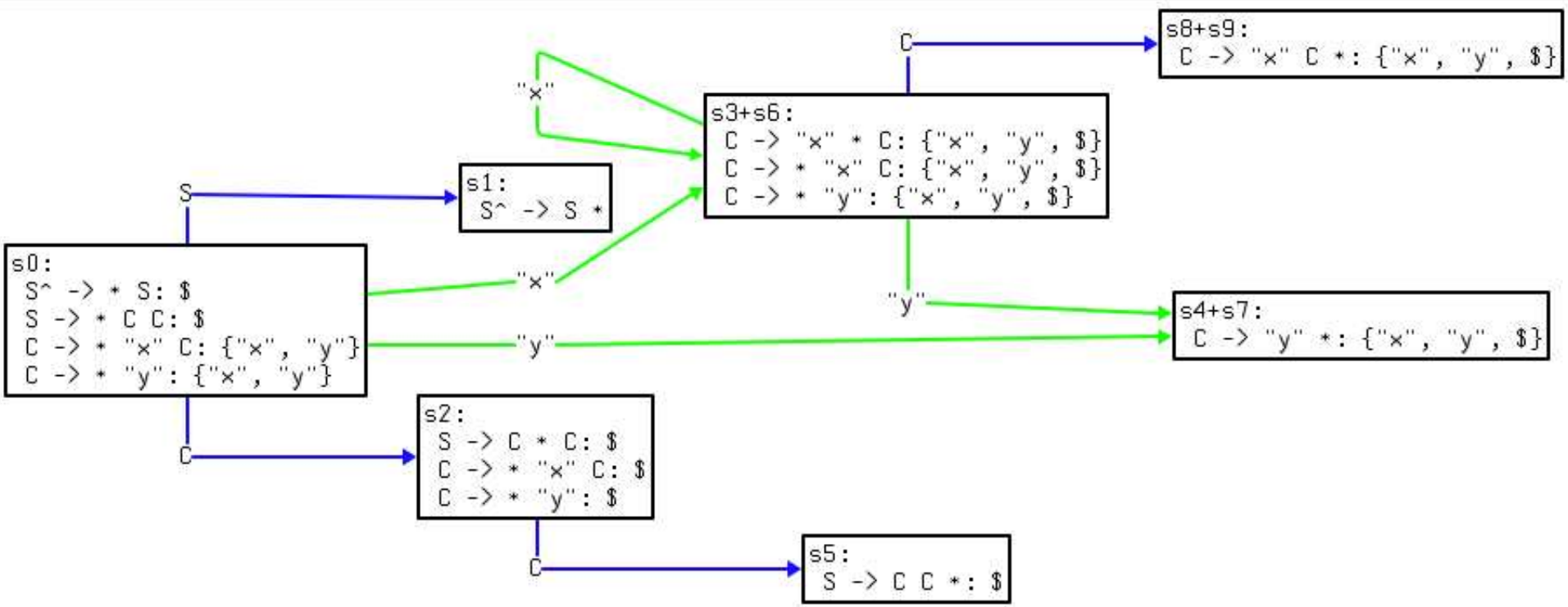
Grammatik für konjunktive Normalform

$$
\begin{array}{rcl}
\textit{Conjunction} & \rightarrow & \textit{Conjunction}\ \text{“\&”}\ \textit{Disjunction} \\
& | & \textit{Disjunction} \\
\textit{Disjunction} & \rightarrow & \textit{Disjunction}\ \text{“|”}\ \textit{Literal} \\
& | & \textit{Literal} \\
\textit{Literal} & \rightarrow & \text{“!”}\ \text{Identifier} \\
& | & \text{Identifier}
\end{array}
$$

Zustände des SLR-Parsers für arithmetische Ausdrücke

$$
\begin{array}{lcll}
s_0 & := & \{ & S \to \star E, \\
 & & & E \to \star E \text{ “+” } P,\ E \to \star E \text{ “-” } P,\ E \to \star P, \\
 & & & P \to \star P \text{ “*” } F,\ P \to \star P \text{ “/” } F,\ P \to \star F, \\
 & & & F \to \star \text{“(” } E \text{ “)” },\ F \to \star N \quad \} \\
s_1 & := & \{ & P \to F \star \} \\
s_2 & := & \{ & F \to N \star \} \\
s_3 & := & \{ & P \to P \star \text{ “*” } F,\ P \to P \star \text{ “/” } F \} \\
s_4 & := & \{ & S \to E \star,\ E \to E \star \text{ “+” } P,\ E \to E \star \text{ “-” } P \} \\
s_5 & := & \{ & F \to \text{ “(” } \star E \text{ “)”} \\
 & & & E \to \star E \text{ “+” } P,\ E \to \star E \text{ “-” } P,\ E \to \star P, \\
 & & & P \to \star P \text{ “*” } F,\ P \to \star P \text{ “/” } F,\ P \to \star F, \\
 & & & F \to \star \text{“(” } E \text{ “)” },\ F \to \star N \quad \} \\
s_6 & := & \{ & F \to \text{ “(” } E \star \text{ “)” },\ E \to E \star \text{ “+” } P, \\
 & & & E \to E \star \text{ “-” } P \}.
\end{array}
$$


s0:
S^ -> * S: $
S -> * C C: $
C -> * "x" C: {"x", "y"}
C -> * "y": {"x", "y"}
s1:
S^ -> S *
s2:
S -> C * C: $
C -> * "x" C: $
C -> * "y": $
s3:
C -> "x" * C: {"x", "y"}
C -> * "x" C: {"x", "y"}
C -> * "y": {"x", "y"}
s4:
C -> "y" *: {"x", "y"}
s5:
S -> C C *: $
s6:
C -> "x" * C: $
C -> * "x" C: $
C -> * "y": $
s7:
C -> "y" *: $
s8:
C -> "x" C *: {"x", "y"}
s9:
C -> "x" C *: $
S
C
"x"
"y"
C
"x"
"y"
C
"x"
"y"
C
"x"
"y"


s0:
S^ -> * S: $
S -> * C C: $
C -> * "x" C: {"x", "y"}
C -> * "y": {"x", "y"}
s1:
S^ -> S *
s2:
S -> C * C: $
C -> * "x" C: $
C -> * "y": $
s3+s6:
C -> "x" * C: {"x", "y", $}
C -> * "x" C: {"x", "y", $}
C -> * "y": {"x", "y", $}
s4+s7:
C -> "y" *: {"x", "y", $}
s5:
S -> C C *: $
s8+s9:
C -> "x" C *: {"x", "y", $}
S
C
C
"x"
"y"
"x"
"y"
C

$$
\begin{array}{rcl}
\textit{program} & \rightarrow & \textit{typeDefList}\ \ \textit{signatures}\ \ \textit{typedTerm} \\
\textit{typeDefList} & \rightarrow & \textit{typeDefList}\ \ \textit{typeDef} \\
& | & \textit{typeDef} \\
\textit{typeDef} & \rightarrow & \text{“}\texttt{type}\text{”}\ \text{Function}\ \text{“}\texttt{:=}\text{”}\ \textit{typeSum}\ \text{“}\texttt{;}\text{”} \\
& | & \text{“}\texttt{type}\text{”}\ \text{Function}\ \text{“(”}\ \textit{varList}\ \text{“)”}\ \ \text{“}\texttt{:} \\
\textit{type} & \rightarrow & \text{Function}\ \text{“(”}\ \textit{typeList}\ \text{“)”} \\
& | & \text{Function} \\
& | & \text{Variable} \\
\textit{typeList} & \rightarrow & \textit{typeList}\ \text{“,”}\ \textit{type} \\
& | & \textit{type} \\
\textit{typeSum} & \rightarrow & \textit{typeSum}\ \text{“}\texttt{+}\text{”}\ \textit{type} \\
& | & \textit{type} \\
\textit{signature} & \rightarrow & \text{“}\texttt{signature}\text{”}\ \text{Function}\ \text{“}\texttt{:}\text{”}\ \textit{argTypes} \\
& | & \text{“}\texttt{signature}\text{”}\ \text{Function}\ \text{“}\texttt{:}\text{”}\ \textit{type}\ \text{“}\texttt{;}\text{”} \\
\textit{signatures} & \rightarrow & \textit{signatures}\ \ \textit{signature} \\
& | & \textit{signature} \\
\textit{argTypes} & \rightarrow & \textit{argTypes}\ \text{“}\texttt{*}\text{”}\ \textit{type} \\
& | & \textit{type} \\
\textit{varList} & \rightarrow & \textit{varList}\ \text{“,”}\ \text{Variable} \\
& | & \text{Variable}
\end{array}
$$

$$
\begin{array}{rcl}
\textit{term} & \rightarrow & \text{Function } \text{“(”} \ \textit{termList} \ \text{“)”} \\
& | & \text{Function} \\
\textit{termList} & \rightarrow & \textit{termList} \ \text{“,”} \ \textit{term} \\
& | & \textit{term} \\
\textit{typedTerm} & \rightarrow & \textit{term} \ \text{“:”} \ \textit{type} \ \text{“;”} \\
\textit{typedTerms} & \rightarrow & \textit{typedTerms} \ \ \textit{typedTerm} \\
& | & \textit{typedTerm}
\end{array}
$$