Thillm
Thüringer Institut für Lehrerfortbildung, Lehrplanentwicklung und Medien

Einführung in das Themengebiet "Formale Sprachen und Automaten" am Beispiel der Lernumgebung AtoCC

Christian Wagenknecht, Michael Hielscher

HOCHSCHULE ZITTAU/GÖRLITZ
University of Applied Sciences
Tel.: 035817671251
UM: 032121198650
Prof. Dr. Christian Wagenknecht
Fachbereich Informatik
Fakultät Elektrotechnik und Informatik
Obermarkt 17
02826 Görlitz
www.inf.hs-zigr.de/~wagenkn
c.wagenknecht@hs-zigr.de

Zur Person

1977	Abitur Geschwister-Scholl-Gymnasium-Löbau
1977-1981	**Lehramtsstudium (Mathematik/Physik; Gymnasium), PH Dresden**
1981	Diplomarbeit auf dem Grenzgebiet Mathematik/Informatik
1981-1984	**Tätigkeit als Lehrer**
1984-1987	Aspirantur am Institut für Informatik und ihre Didaktik, PH Dresden
1987	Dissertation (Dr.rer.nat.) in Informatik
1987-1991	**Wissenschaftlicher Mitarbeiter an der PH Dresden, FB Informatik**
1991-1993	**Wissenschaftlicher Mitarbeiter an der PH Ludwigsburg**
seit 1993	Hochschullehrer (Professor für Informatik) Hochschule Zittau/Görlitz

Forschungsaufenthalte in den USA, Schweiz, Kanada, Litauen, Dänemark, China, Japan, Russland, Tschechien, Finnland, Italien, Bulgarien, Großbritannien, Polen.

Tätigkeit an der Wilhelm-Büchner-Hochschule Darmstadt und an der BA Bautzen

Organisatorisches

Zeitplan:

09:00 - 10:30, 10:45 - 12:15

Mittagessen
13:00 - 14:30, 14:45 - 16:00

Abreise

.....

Frau List unterstützt uns freundlicherweise. Vielen Dank!

Teilnehmerzahl

40

Woran liegt diese überdurchschnittliche Zahl?

- Freiwillig???
- Mangelndes Risikobewusstsein???
  Gefährdung des Erholungseffektes der Ferienwoche
- Ursache: wahre BEGEISTERUNG für Theoretische Informatik (TI)??? **ODER?**

Gefühlssituation der Lehrenden

- "TI wollte ich nie machen."
- "TI hat mich nie richtig interessiert."
- "TI war mir immer zu theoretisch und abstrakt."
- "Die TI-Dozenten waren suspekt – TI im (postgradualen) Studium erinnere ich mit Grausen."
- "Die TI-Inhalten helfen mir nicht, wenn das Schulnetzwerk mal wieder zusammenbricht."
- ...

Lehrplan → Ziele und Inhalte

In den meisten Bundesländern sind ausgewählte Inhalte der TI Lehrplaninhalt der Sek. II:

Nr.	Bundesland	Lehrplaninhalt (Lernbereich)	Pflichtbestandteil
1	Baden-Württemberg	Bereich der theoretischen Informatik (Automaten, Berechenbarkeit)	nein
2	Bayern	3. Formale Sprachen (noch Entwurf)	
3	Berlin	4.4 Sprachen und Automaten	ja (auch GK)
4	Brandenburg	4.4 Sprachen und Automaten	Ja (auch GK)
5	Bremen	Grundlagen der Theoretischen Informatik (Automaten, formale Sprachen)	nein
6	Hamburg	Formale Sprachen, endliche Automaten, Keller-automaten, Scanner, Parser, Ableitungsbaum	nein
7	Hessen	Formale Sprachen und Grammatiken Automaten, Fakultativ: Übersetzerbau	**ja (auch GK)**
8	Mecklenburg-Vorpommern	4.4 Sprachen und Automaten	**ja (auch GK)**
9	Niedersachsen	Eigenschaften endlicher Automaten Aspekte formaler Sprachen	nein
10	Nordrhein-Westfalen	Endliche Automaten und formale Sprachen	nein
11	Rheinland-Pfalz	Formale Sprachen und Automaten zur Sprachbeschreibung und Spracherkennung	**ja (nur LK)**
12	Saarland	Automaten und formale Sprachen Fakultativ: Übersetzerbau	**ja (auch GK)**
13	Sachsen	8 A: Formale Sprachen, Kellerautomat, Akzeptor	nein
14	Sachsen-Anhalt	Endliche Automaten und formale Sprachen	nein
15	Schleswig-Holstein	Automaten als mögliches Themengebiet	nein
16	Thüringen	Themenbereich 7.3: Einblick in formale Sprachen	nein

Verortung der TI im Lehrplan Thüringens

Zwei wichtige Quellen:

- Informatik-Lehrplan für Kl. 11,12 (1999)
  (Eingeflossen sind hier: EPA = Einheitliche Prüfungsanforderungen in der Abiturprüfung; Bildungsstandards für Informatik als Minimum)

- Ziele und Inhalte der Qualifizierungsphase der gymnasialen Oberstufe im Fach Informatik
  (Inhalts- und prozessbezogenen Kompetenzen, differenziert nach grundlegendem und erhöhtem Anforderungsniveau)

Themenbereich (Wahlthema in Kl. 12): Einblick in formale Sprachen (ca. 25 Std.)

Die Schüler erwerben wesentliche Kenntnisse allgemein zu Sprachen und speziell zu *regulären* und *kontextfreien Sprachen*. Sie erfahren wichtige Zusammenhänge zu den entsprechenden Automaten (*endliche Automaten*, *Kellerautomaten*).

Die Schüler kennen und nutzen verschiedene *Techniken zur syntaktischen Beschreibung formaler Sprachen*. Sie erlernen ein *Verfahren zur Übersetzung eines Ausdrucks in eine maschinennahe Notation*. Die Schüler können die grundsätzliche Arbeitsweise von *Compilern* und *Interpretern* erklären.

(Zitat aus dem Lehrplan)

Grundfach Informatik: 7.3; *Leistungsfach* Informatik: 10.2

EPA → Anforderungsbereiche: Reproduktion / analoge Rekonstruktion / Konstruktion

Inhaltliche Kompetenzen in der Qualifizierungphase (grundlegendes / erhöhtes Anforderungsniveau)

Formale Sprachen und Automaten Der Schüler kann - natürliche und formale Sprachen vergleichen, - Sätze einer formalen Sprache mithilfe ihrer Grammatik ableiten, - Grammatiken einfacher formaler Sprachen konstruieren, - den Zusammenhang von formalen Sprachen und Automaten erläutern, - am Beispiel einer regulären Sprache die Arbeitsweise eines endlichen Automaten erläutern, - die Arbeitsweisen von Compilern und Interpretern vergleichen.	**Formale Sprachen und Automaten** Der Schüler kann - natürliche und formale Sprachen vergleichen, - Sätze einer formalen Sprache mithilfe ihrer Grammatik ableiten, - Grammatiken einfacher formaler Sprachen konstruieren, - den Zusammenhang von formalen Sprachen und Automaten erläutern, - die Arbeitsweise von Automaten erklären, Automaten konstruieren und implementieren, - die Arbeitsweisen von Compilern und Interpretern vergleichen, - einfache Interpreter entwerfen und implementieren.
Möglichkeiten und Grenzen des Einsatzes von Informatiksystemen Der Schüler kann	**Möglichkeiten und Grenzen des Einsatzes von Informatiksystemen** Der Schüler kann

Inhaltliche Kompetenzen lt. Lehrplan (7.3)

Lernziele/Inhalte	Hinweise
– Anwendung von Sprachen in der Informatik	Computer-Drucker-Kommunikation, Maschinensprache, Programmier-sprache
– formale Sprachen, Grammatiken	Erläutern der Begriffe Terminalsymbol, Nichtterminalsymbol, Startsymbol, Pro-duktionsregel, Wort, Satz, Metasymbol ➔ Themenbereich 3 (Syntax von PASCAL oder OBERON)
– Syntaxdiagramme, erweiterter Backus-Naur-Formalismus (EBNF)	Gegenüberstellen der beiden Beschrei-bungsformen kontextfreier Grammatiken Überführen von Syntaxdiagrammen in den EBNF und umgekehrt
– Synthese und Analyse von Sätzen	➔ Themenbereich 3 (Syntax von PASCAL oder OBERON)
– Konstruieren von Sprachbeschrei-bungen	Steuersprache für Roboter und Turtle (Schildkrötengrafik)
– endliche Automaten als Modelle von realen Automaten	Analysieren und Konstruieren von endli-chen Automaten Angeben des Übergangsgraphen und der Zustandstafel

Inhaltliche Kompetenzen lt. Lehrplan (7.3)

– Zusammenhang zwischen regulären Sprachen und endlichen Automaten	
– Aufbau und Arbeitsweise des Keller-automaten von Dijkstra (1961)	Übersetzung eines Ausdrucks aus der Infix- in die Postfix-Notation ➔ Themenbereich 8 (Stapel)
– Zusammenhang zwischen kontext-freien Sprachen und Kellerautomaten	Palindrome Erläutern, dass PASCAL und OBERON keine kontextfreien Sprachen sind
– Compiler und Interpreter	Vergleichen der Arbeitsweise der bei-den Werkzeuge
– Übersetzen eines Ausdrucks aus der Infix-Notation in die Postfix-Notation und weiter in eine Modell-Assembler-Sprache	Analysieren eines vorgegebenen PASCAL- oder OBERON-Programms, das das Übersetzen eines Ausdrucks realisiert
– Äquivalenz zweier beliebiger Sprach-beschreibungen	Charakterisieren, dass es bei kontext-freien Sprachen keinen Entscheidungs-algorithmus gibt ➔ Themenbereich 6 (theoretischer Aspekt)

TI-Inhalte in der Schulinformatik: Probleme und Chancen

→ Problem mit Zeit-Inhalts-Relation

→ Manche *Lehrende* mögen es nicht. – Motivationsproblem

→ Manche *Lehrende* können es nicht richtig. – Qualifikationsproblem

→ *SchülerInnen/Studierende* fragen gelegentlich: "Wann geht es denn nun endlich richtig los mit der Informatik? Ach so, das ist es schon." ☹ - Vermittlungsproblem

"Ergebnis": Wenn möglich, TI weglassen.

(Dumm, dass techn. Inf. 7.1 oder logikorient. Progr. 7.2 auch nicht so toll sind.)

FALSCH!!!

Chance: **Informatik als Wissenschaft repräsentieren!**

(wie Mathematik und Naturwissenschaften)

Sonst: Studienabbrecher als konkrete Folge!!

***Drei Irrtümer** als Reaktion auf die Problemlage*

- Von allem ein bisschen So bitte nicht!!!
  LP: 7.3 **Einblick in** formale Sprachen
  Alternative: Für Kerninhalte zeigen, dass sie Biss haben!
- Zuerst die Theorie, dann die Praxis So bitte nicht!!!
  Anwendung "Compiler" NACHDEM die Theorie vermittelt wurde
  Alternative: FSuA-Theorie mit automat. Compilerbau verzahnen
  → bedarfsorientiertes Lernen
- Sogenannter "Praxisbezug" So bitte nicht!!!
  "Endliche Automaten als Modelle realer Automaten" (LP, S. 26)
  → Automaten mit Ausgabe (Mealy, Moore)
  Alternative: Automaten als adäquate Beschreibungsmittel des Akzeptanzprozesses für Sprachen

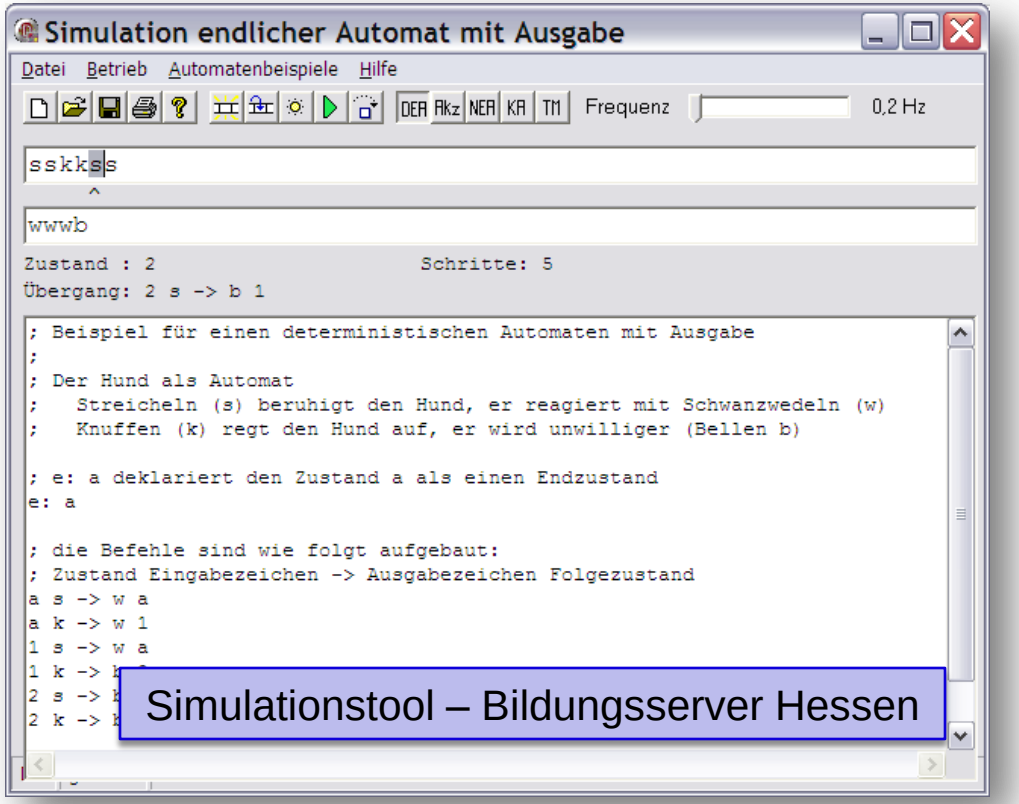
Im Workshop geht es um diese Alternativen!!!

Didaktische Software für TI

- **in Schulen:**
  diverse Simulationstools oder Lernumgebungen, wie Kara; meist von enthusiastischen LehrerInnen entwickelt

- **in Hochschulen:**
  Systeme für die Lehre, wie JFLAP

  LEX und YACC für die Hand des Ingenieurs



Defizite existierender Systeme

- Systeme bzw. separate Module thematisieren Einzelaspekte
- nicht definitionskonform und/oder nicht an Lernprozessen orientiert, sondern die Prozess-Simulation dominiert
- Suggerieren abstrakten Automat als physikalisches Objekt
- Systeme können nur simple Beispiele bearbeiten – zu große Distanz zur Praxis

Lernumgebungen (Quelle: LP Brandenburg)

"Lernumgebungen werden so gestaltet, dass sie das selbst gesteuerte Lernen von Schülerinnen und Schülern fördern. Sie unterstützen durch den Einsatz von Medien sowie zeitgemäßer Kommunikations- und Informationstechnik sowohl die Differenzierung individueller Lernprozesse als auch das kooperative Lernen. Dies trifft sowohl auf die Nutzung von multimedialen und netzbasierten Lernarrangements als auch auf den produktiven Umgang mit Medien zu.

Moderne Lernumgebungen ermöglichen es den Lernenden, eigene Lern- und Arbeitsziele zu formulieren und zu verwirklichen sowie eigene Arbeitsergebnisse auszuwerten und zu nutzen."

AtoCC - Vom abstrakten Automaten zur automatisierten Entwicklung von Sprachübersetzern

- Verzahnung von FSuA-Theorie mit Aspekten des automatisierten Compilerbaus
  (alles in einem modularen System, vereinheitlichte Bedienung, Modulübergänge, ...)

- Wichtige didaktische Entscheidung: Zielsprache des Compilers sollte nicht Maschinencode sein!!

- Herstellung eines lauffähigen(!) Sprachübersetzers durch Anwendung der Kenntnisse aus der TI – erfordert hohe Abstraktion (CC mit VCC)

- Modellierung des Übersetzungsprozesses mit "ausführbaren" T-Diagrammen

Didaktische Gestaltung der Lerneinheit

1. Belastbare Motivation für TI-Inhalte durch *herausfordernde Start-Fragestellung mit Praxisrelevanz* und *Modellierung* eines Zielsystems (Sprachübersetzer) *am Anfang*
2. *Vermittlungs-/Anwendungszyklen* für TI-Wissen *mit Projekt-bezug* (Praxis nicht als "Anhängsel" zur Theorie)
3. *Komplexe Anwendung* von TI-Inhalten auf *sehr hohem Abstraktionsniveau* (automatisierte Compiler-Generierung), *Rückkehr zur und Konkretisierung der Modellierungsebene*

Behauptung: Dabei ist AtoCC ein unverzichtbares Hilfsmittel.

Installation

Installationshinweise: Bitte Reihenfolge einhalten!

Software unter:

www.atocc.de → Workshops → Software

- Installieren Sie Java `jdk-6u18-windows-i586.exe`
- Installieren Sie PostScript `gs860w32.exe`
- Installieren Sie einen pdf-Reader `Foxit Reader.exe`
- Installieren Sie AtoCC `AtoCC Setup.exe`

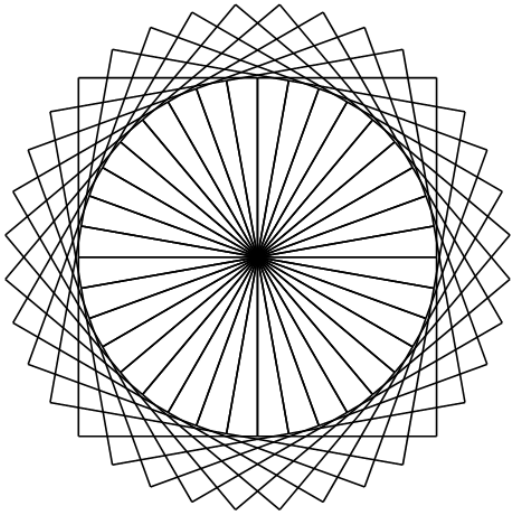
Pfadprobleme durch Neuinstallation von AtoCC (letzter Punkt) beheben!

Beispiel: ZR – eine Sprache für einen Zeichenroboter

Praxisnahe (echte!) Aufgabe mit grafischer (akustischer) Ausgabe:
Entwickeln Sie einen Compiler, der die Sprache ZR (ZeichenRoboter) in PDF übersetzt. (Schülergerecht formulieren!)

Eingabewort (in ZR):

WH 36 [WH 4 [VW 100 RE 90] RE 10]



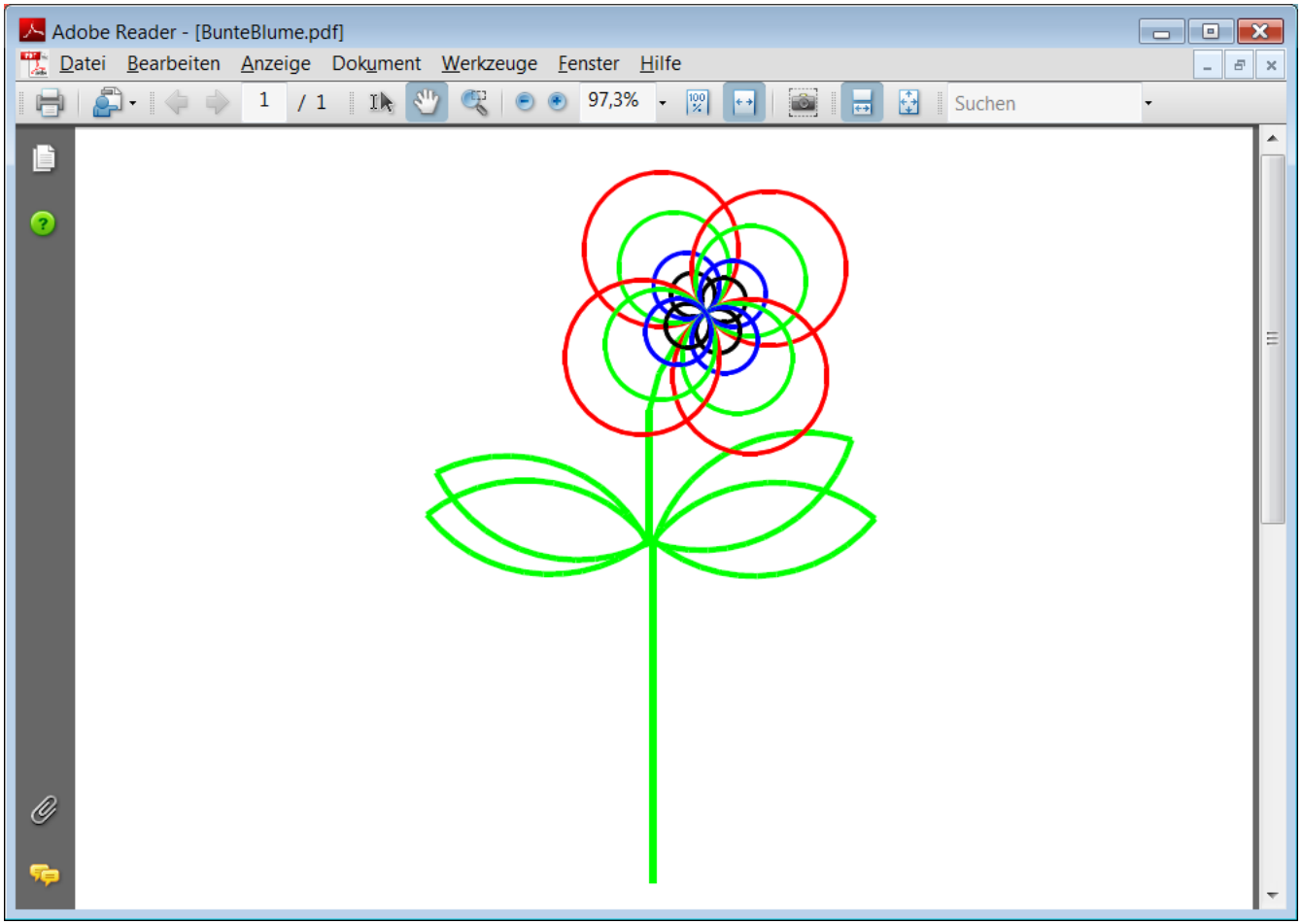
Sprachelemente:

VW *n*	VorWärts *n* Schritte
RE *n*	Rechts um *n* Grad
WH *n* [...]	WiederHole *n*-mal [...]
FARBE *f*	StiftFARRBE *f*
STIFT *n*	Strichstärke *n*

Aufgabe: Verwenden Sie den fertigen Compiler `zr2pdf blume.zr`
(`konsole.bat` aufrufen, `blume.zr` ansehen, später modifizieren)

Beispiel: ZR – eine Sprache für einen Zeichenroboter

Der Zeichenroboter kann auch mehr: BunteBlume.zr



Beispiel: ZR – eine Sprache für einen Zeichenroboter

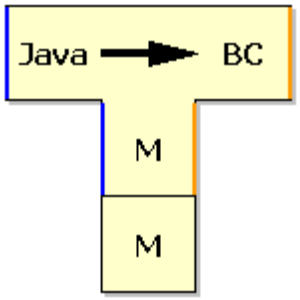
Weiterer Ablauf:

1. Modellierung der Problemlösung mit TDiag
2. Syntax-Definition von ZR: formale Grammatik, Ableitungsbaum mit kfGEdit
3. Parser → Akzeptoren → Automatenmodelle (EA, KA) mit AutoEdit
4. Arbeitsteilung: Scanner, Parser
5. Zielsprachenbezug → automatisierte Compiler-Entwicklung mit VCC
6. Teilsysteme werden in Modellierung eingebracht (TDiag)
7. Ergebnis: lauffähiger (nichttrivialer) Übersetzer, den man benutzen kann!

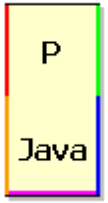
TDiag, kfGEdit, AutoEdit, und VCC sind Bestandteile von **AtoCC**.

Beispiel: ZR – eine Sprache für einen Zeichenroboter

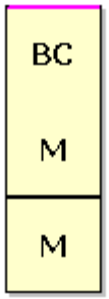
- Übersetzungsprozess analysieren und modellieren
- Verwendung von T-Diagrammen:
  - T-Diagramme bestehen aus 4 Bausteintypen.
  - Compilerbaustein, Programmbaustein, Interpreterbaustein und Ein/Ausgabe-Baustein



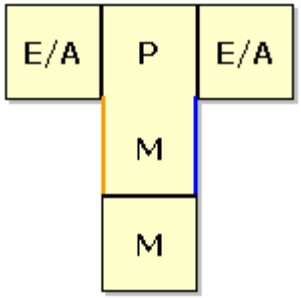
Compiler



Programm



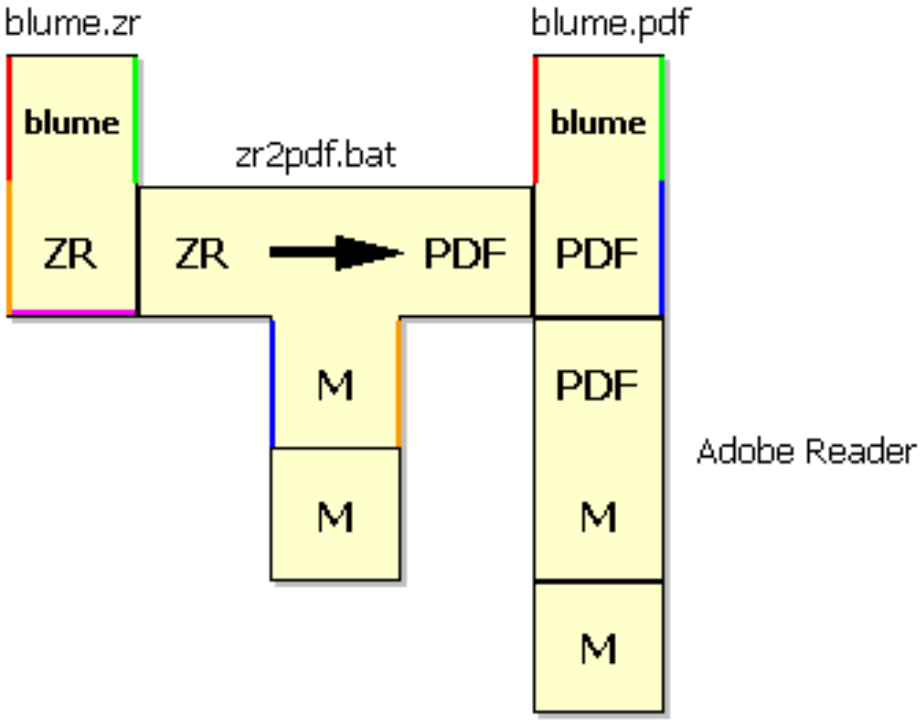
Interpreter



Ein/Ausgabe an Programmbaustein

Beispiel: ZR – eine Sprache für einen Zeichenroboter

T-Diagramm: 1. Entwurf



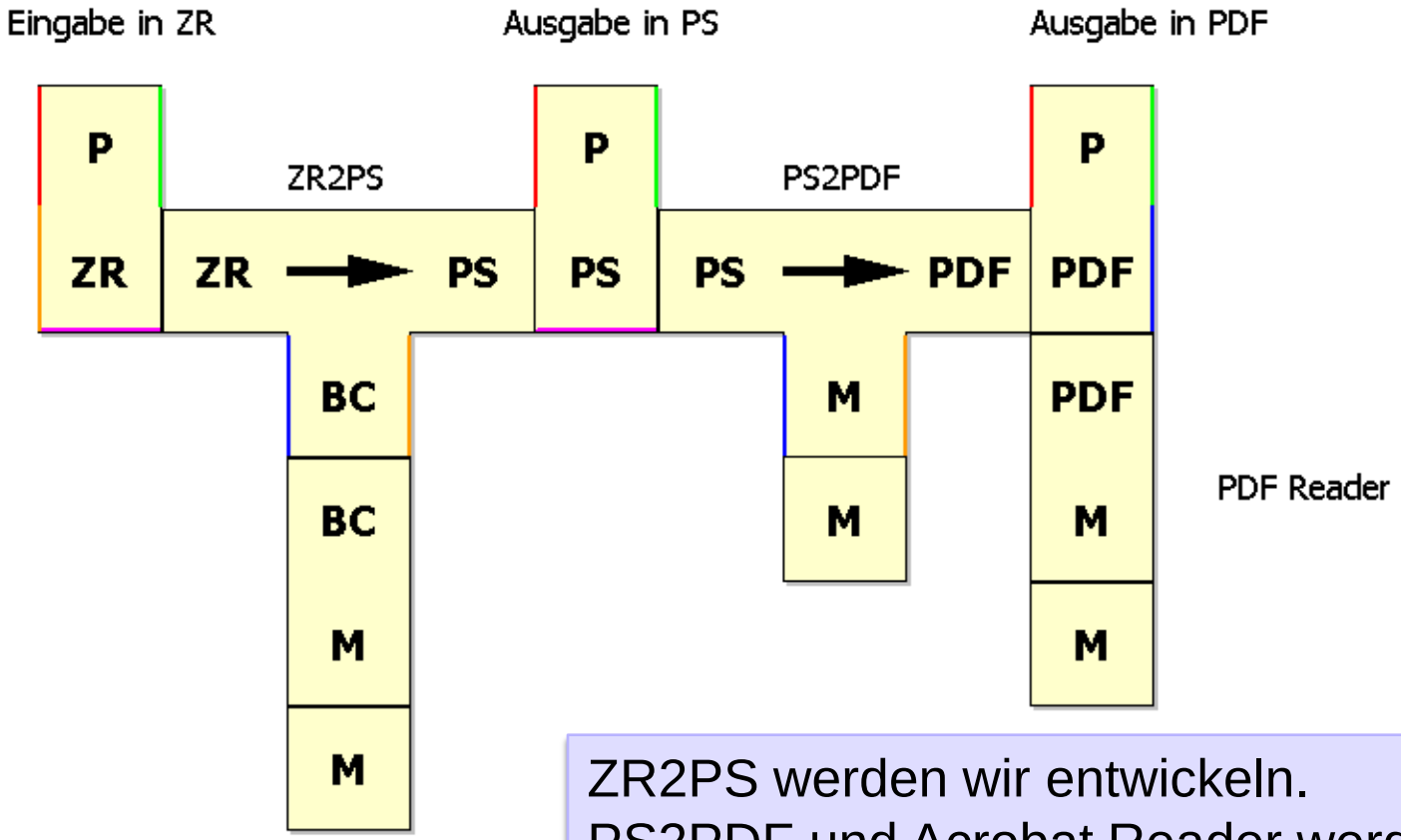

```
%PDF-1.4
%Çì¦¢
5 0 obj
<</Length 6 0 R/Filter /FlateDecode>>
stream
x¦í¦M´å¸
¦÷u¦w¦¦¦Iý¦Àk¦¦à¦7®¦Ã _ß¦¦
¦¦\£¦TLDwëS¦È¦$HJä¿?9¦ÖÖÿìï¦|¦ýå¦íóÏÿ|¦üùï×yëóÇ_¿jÎùÓô¦
¦1QÕå|jë«««¦¦ª-5ååÓÖÜR1¦1QõEz*àüëÔ¦"¦õ¦XI±iÏöØ¦1Qí¥¦|zML
½&æÕ¦"@Ì+;D¦¦¦¦¦¦f>â`¦¦ùè¦¦aö
#ï¦~s\\[Z¤6ó¶¦µØýÈ´Ç-S]¦ý=z¦¦ÆhÙÔûR¦®"vÆ»¦cØ"¾ÎÇ¦w`¦»QßÌ
¦c¦ªíAuùw>¦J¦Ði±¦¦^ÿ<N¦y½ÛØ:îG¦±¦P¦¦j9Ot¬Ð¦¦u¦¦6¦ü²Cg¦3
Ê¥¦H¦ÚÏõVÝ»Ï
ÄÀíÛÖÒÀn7¦ªu¦ußS¦ªÈ´¾ò´V!ýõù¦Øë¦À$õÀU]eO_¦¦í¦[4¦½_²$¦zo
É14Ç¦c¦¦÷ãÀx[´;¦_«¦béØ¦ÕvS-´o¦kh´¦Ö¦¦·¦¦¦ÍX¦àï'$¦YÐéJ
&1ä~köN¦~¦Ì½¦TûMe¦Çö"úy¦<%Õþ¦q; ÷_¦¦l¦ªÍUãá¦¦lÏÐ\ù¦ úÅ¦
À¦¦¦S¦þ~C¦Éfs"Fr2ÏJ³ù¦³2¦g¸Ù\¦¦¦ÌŸÖlÞG·LæÎk¶¦@çMæ.¦ª~S¦
3P¦¦Uq¦=ÚÞ¹CsÆ¼ßÍÞß±ß¦y¦¦} À¦Læñlö]¦ã9¦ç¦Ù7
ÌÍd¦çfß:0Í¦¹g¦};AÏLæþkö¦¦ý7¦{9¨ÛMeÑ¦-¦ó¦ ½'æõ¦"AÌk?D¦Ø5¦
ªí¦2Ú¦¦¦]Õ¦ú¦Ñ¬E¦A¦z¦iÿa¸¦Á¦+ä¦ÙùÍ×¦ç¦ÜùÕèÜãD¿0{}Þd´ö j
³×_¦Ì]ÍöôÑq¦®{Çy¦fç¦¦{#ÓJ¦g p¦¦Üëw¦9¦Ù¦¦0/¦SÕy|¢õë¦O$®Xd
¦ï¦¼3æu|2³sWèÎ›ö´¦~}¦g¦õ¦3ÓŸªqN
£Ç¦N[@&¦©j¦ßÅ¬2Ó½¶q¦¦¦ÅiÕy3»î7E¿¦öTp=3è5T¦³×çMfä ²¦yfp¦
õÞ¦Db¦V6¦¦±¦É½¦k¦¦ÁûÜ¦Ä¬éûãT¦So¦¦ÉÀz¤ÍTÀÔg¦µ]oµýr¦¦1 -¦
3=o4ÚÆ<3SÕø¦è¦fß/O¦^¤›¨ª«ièP¦Ì ÷Pc Î^¦<¦¦¼J¦¦ç¦¦¿17¦y¦Ãî
endobj
6 0 obj
2327
endobj
4 0 obj
<</Type/Page/MediaBox [0 0 612 792]
/Parent 3 0 R
/Resources<</ProcSet[/PDF]
/ExtGState 8 0 R
>>
```

ZR2PDF möchte niemand schreiben!!!

Beispiel: ZR – eine Sprache für einen Zeichenroboter

T-Diagramm: 2. Entwurf

TDiag



ZR2PS werden wir entwickeln.
PS2PDF und Acrobat Reader werden vom System bereitgestellt.

Beispiel: ZR – eine Sprache für einen Zeichenroboter

Zwischenstand:

- Schüler kennen die Übersetzungsaufgabe und Verwendungsform des zukünftigen Compilers
- nun Compiler-Entwurf beginnen: Quell-/Zielsprache

Quellsprache:

- Mit Sprache auseinandersetzen: Beispielwörter bilden; Grammatik definieren, d.h. Terminale bestimmten, Produktionsregeln angeben und dabei Nichtterminale festlegen = induktives Vorgehen
- Ableitungsbäume erzeugen

Beispiel: ZR – eine Sprache für einen Zeichenroboter

ZR-Beispielwörter → Aufbau allg. beschreiben:

- **VW 50**
- **RE 270**
- **RE 45 WH 2 [VW 100]**
- **WH 4 [VW 100 RE 100]**
- **WH 36 [WH 4 [VW 100 RE 90] RE 10]**

→ Magnetkarten an der Tafel

Beispiel: ZR – eine Sprache für einen Zeichenroboter

- "Baustein" Zahl:
  - **0** soll in ZR keine Zahl sein, da **`VW 0`** oder **`RE 0`** keine Veränderung herbeiführen.
  - Vorangestellte Nullen, wie bei **`0815`**, sind nicht erlaubt.
- Regeln der formalen Grammatik:

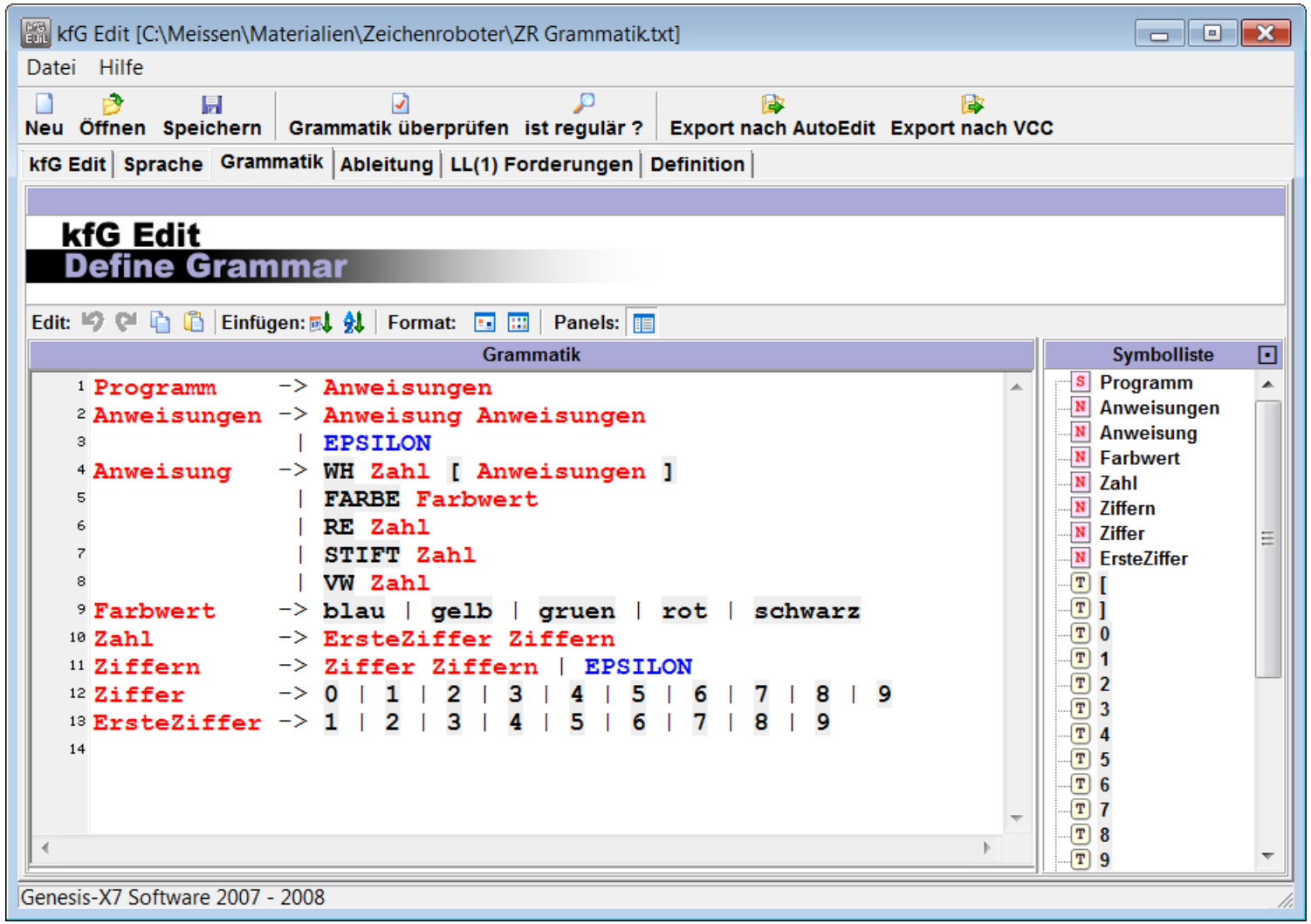
Zahl → **ErsteZiffer Ziffern**

Ziffern → **Ziffer Ziffern | ε**

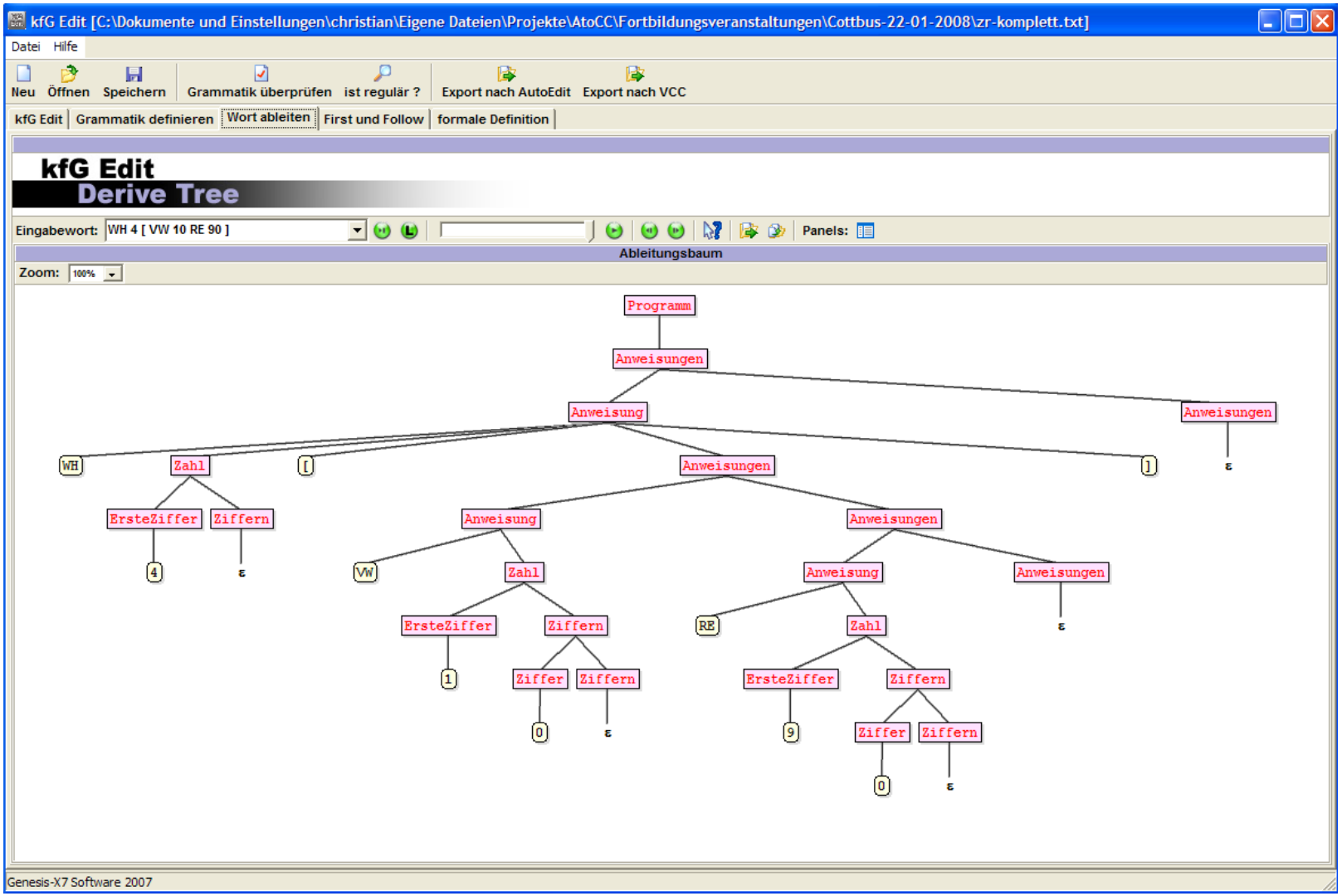
Ziffer → **0 | 1 | ... | 9**

ErsteZiffer → **1 | 2 | ... | 9**

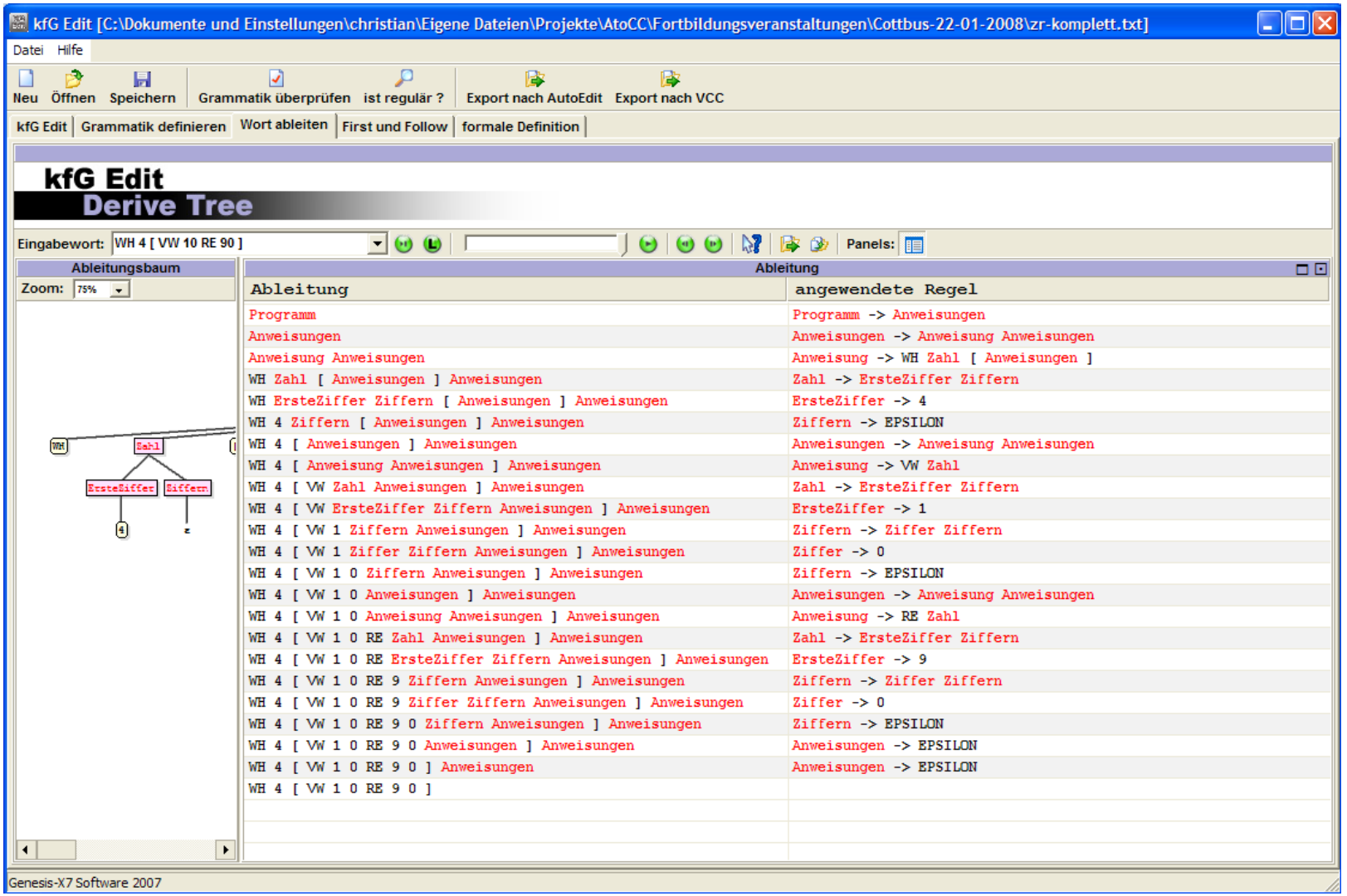
Beispiel: ZR – eine Sprache für einen Zeichenroboter



Beispiel: ZR – eine Sprache für einen Zeichenroboter



Beispiel: ZR – eine Sprache für einen Zeichenroboter



Beispiel: ZR – eine Sprache für einen Zeichenroboter

Programm → **Anweisungen**

Anweisungen → **Anweisung Anweisungen | EPSILON**

Anweisung → **VW Zahl**
| **RE Zahl**
| **WH Zahl [Anweisungen]**
| **FARBE Farbwert**
| **STIFT Zahl**

Farbwert → **rot | blau | gruen | gelb | schwarz**

Zahl → **ErsteZiffer Ziffern**

Ziffern → **Ziffer Ziffern | EPSILON**

Ziffer → **0 | 1 | ... | 9**

ErsteZiffer → **1 | 2 | ... | 9**

Beispiel: ZR – eine Sprache für einen Zeichenroboter

Ableitung ist generativ, regelbasiert.

Automaten als **Akzeptoren** für Sprachen

- Akzeptor prüft, ob ein Wort zur Sprache gehört oder nicht. (Keine Ausgabe → Wort akzeptiert)
  (Thema: Programmiersprachen und Syntaxfehler)
- Wir nehmen zwei Ausschnitte aus den Produktionen:

Zahl	→ **ErsteZiffer Ziffern**
Ziffern	→ **Ziffer Ziffern | EPSILON**
Ziffer	→ **0 | 1 | ... | 9**
ErsteZiffer	→ **1 | 2 | ... | 9**
Anweisungen	→ **Anweisung Anweisungen | EPSILON**
Anweisung	→ **VW Zahl | WH Zahl [Anweisungen]**

Beispiel: ZR – eine Sprache für einen Zeichenroboter

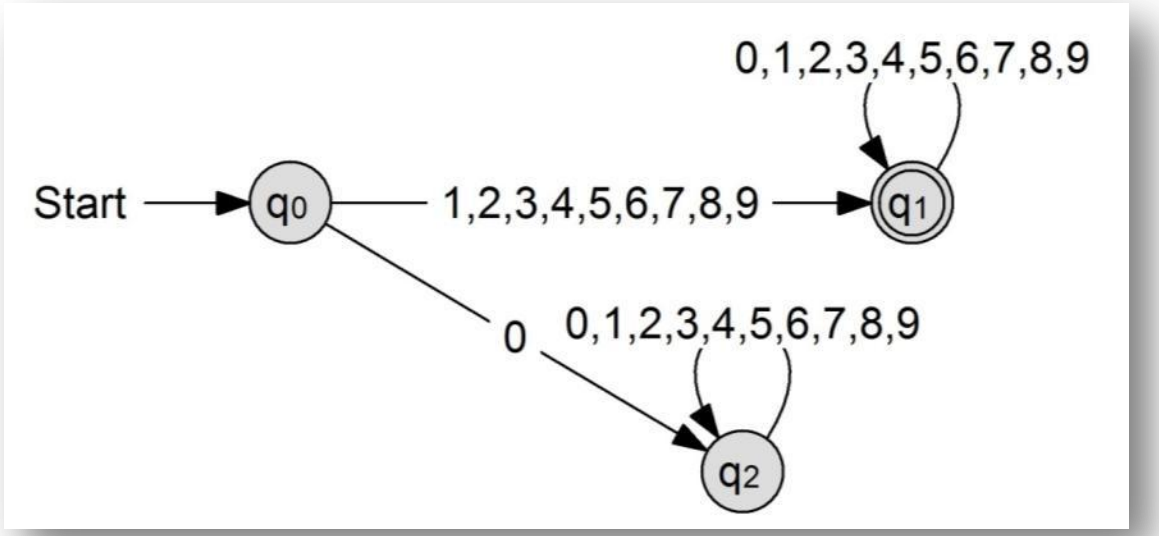
Kleiner Sprachausschnitt:

Zahl → **ErsteZiffer Ziffern**

Ziffern → **Ziffer Ziffern | EPSILON**

Ziffer → **0 | 1 | ... | 9**

ErsteZiffer → **1 | 2 | ... | 9**



Beispiel: ZR – eine Sprache für einen Zeichenroboter

Ein ***DEA*** ist ein Quintupel:

$M = (Q, \Sigma, \delta, q_0, E)$, mit

Q ... endliche Menge von Zuständen,

Σ ... Eingabealphabet,

δ ... totale Überführungsfunktion, $Q \times \Sigma \rightarrow Q$,

q_0 ... Anfangszustand ($q_0 \in Q$),

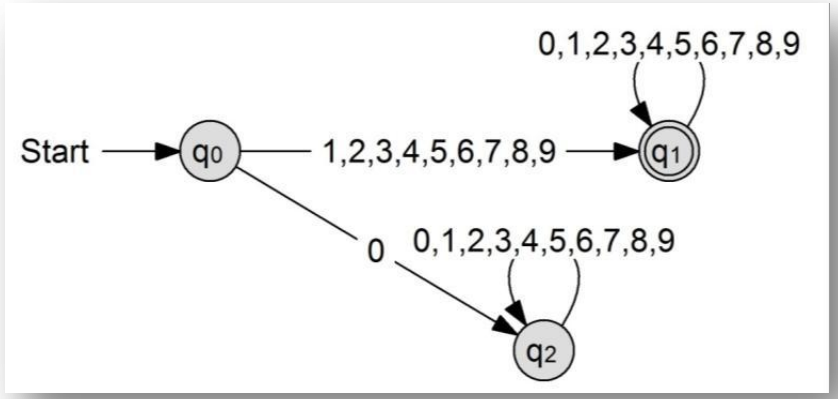
E ... endliche Menge von Endzuständen ($E \subseteq Q$)

Die Sprache L, die durch den Automaten beschrieben wird:

$L(M) = \{w \mid w \in \Sigma^* \text{ und } (q_0, w) \vdash^* (q_e, \varepsilon) \text{ und } q_e \in E\}$

Beispiel: ZR – eine Sprache für einen Zeichenroboter

Für EA-Sprachen können auch **reguläre Ausdrücke** *(Schablonen-Vorstellung)* verwendet werden:
Beispiel Zahl (nicht 0, ohne Vornullen): [1-9][0-9]*



Grammatik – Ableitung – konstruierend/generierend

Automat – Akzeptor – analysierend/verarbeitend

Reg. Ausdruck – Muster – vergleichend/abgleichend

Beispiel: ZR – eine Sprache für einen Zeichenroboter

Kleiner Sprachausschnitt:

Anweisungen → **Anweisung Anweisungen | EPSILON**

Anweisung → **VW Zahl | WH Zahl [Anweisungen]**

Versuch einer DEA-Konstruktion scheitert!

Beispiel: ZR – eine Sprache für einen Zeichenroboter

DKA sind äquivalente Beschreibungen für LR(k) Sprachen.

Ein ***DKA*** ist ein 7-Tupel,

$M = (Q, \Sigma, \Gamma, \delta, q_0, k_0, E)$, mit

Q ... endliche Menge von Zuständen,

Σ ... Eingabealphabet,

Γ ... Stackalphabet,

δ ... partielle Überführungsfunkt., $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*$,

q_0 ... Anfangszustand ($q_0 \in Q$),

k_0 ... Stackvorbelegungszeichen ($k_0 \in \Gamma$ und $k_0 \notin \Sigma$),

E ... endliche Menge von Endzuständen ($E \subseteq Q$)

Die Sprache L die durch den Automaten beschrieben wird:

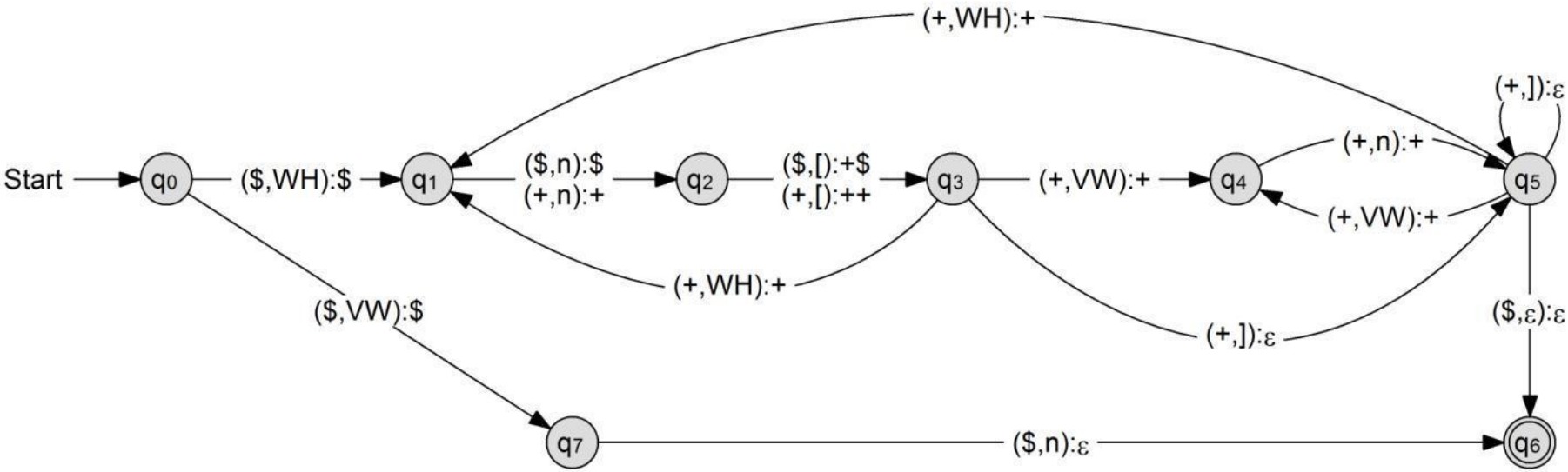
$L(M) = \{w \mid w \in \Sigma^* \text{ und } (q_0,w,k_0) \vdash^* (q_e,\varepsilon,K) \text{ und } q_e \in E \text{ und } K \in \Gamma^* \}$

Beispiel: ZR – eine Sprache für einen Zeichenroboter

Anweisungen → Anweisung Anweisungen | EPSILON

Anweisung → VW n

| WH n [Anweisungen]



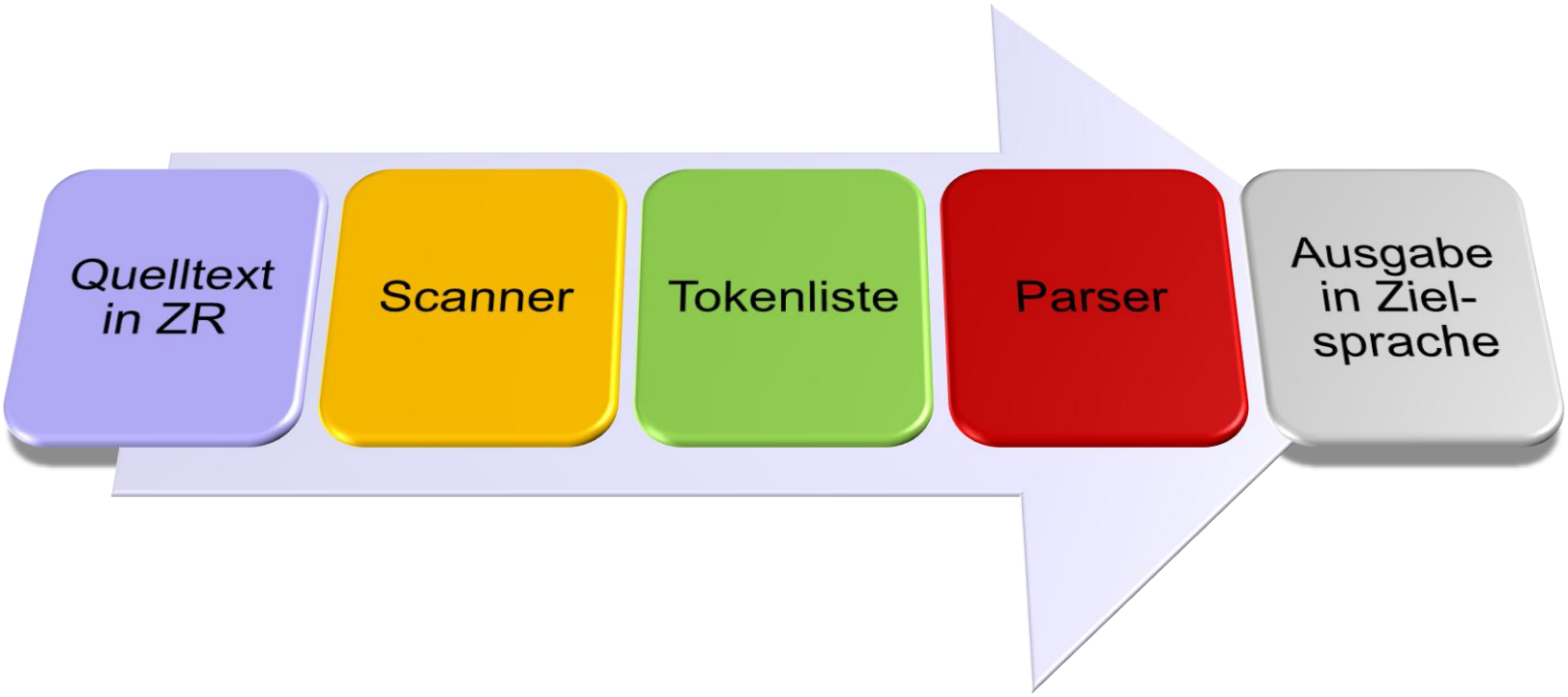
DKA für obigen Grammatik-Ausschnitt

Beispiel: ZR – eine Sprache für einen Zeichenroboter

- Compiler ist Kombination eines (sehr komplexen) *Kellerautomaten* und kleinen *endlichen Automaten*

- Kellerautomat ist sehr komplex und kann praktisch nicht vom leeren Blatt aus per Hand entwickelt werden. → VCC (AtoCC-Modul)
  Deshalb:
  - Einfache ÜA für KA in der Lehre (z.B. Palindrome)
  - Sprachklassen, die *automatisierte Erzeugung effizienter Compiler* ermöglichen (LL, LR)
    Formal ist kfG → NKA möglich, praktisch nutzlos!

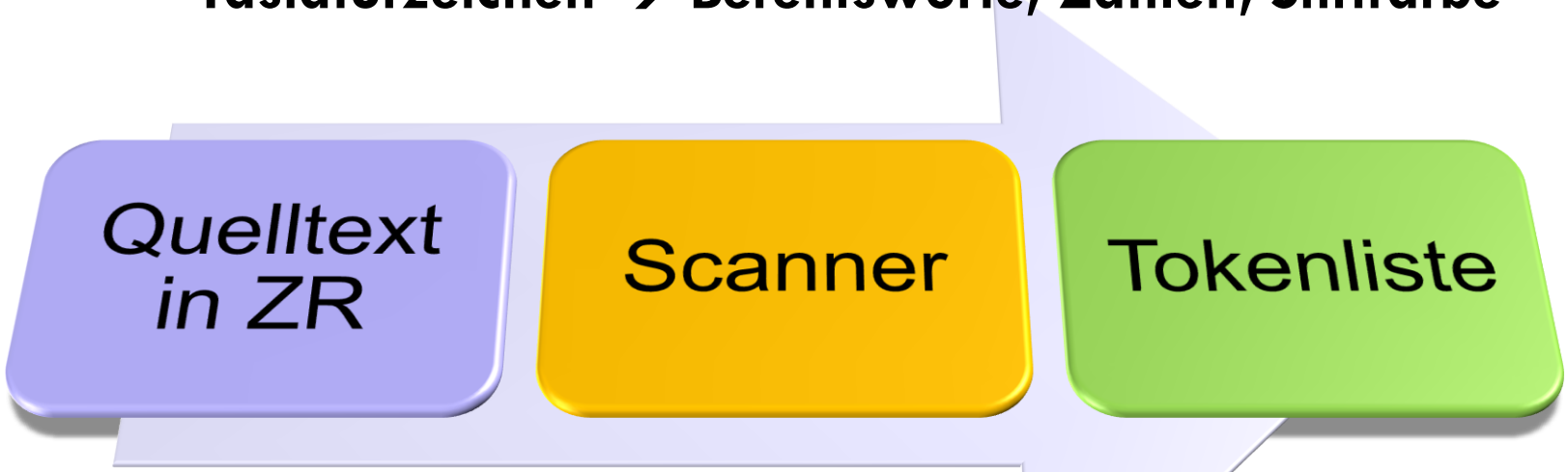
Arbeitsweise des Compilers



Arbeitsweise eines Scanners

➔Terminale der Grammatik:

Tastaturzeichen → Befehlsworte, Zahlen, Stiftfarbe



"Quelltext besteht aus Zeichen und der Rechner weiß noch nicht wie diese zusammengehören."

"Viele kleine endliche Automaten entscheiden welche Teilwörter im Quelltext stehen."

Token als Paare
[Tokenname, Lexem]
z.B.:
[Wiederhole, "WH"]
[Zahl, "12"]
[KlammerAuf, "["]

Beispiel: ZR – eine Sprache für einen Zeichenroboter

Programm → **Anweisungen**

Anweisungen → **Anweisung Anweisungen | EPSILON**

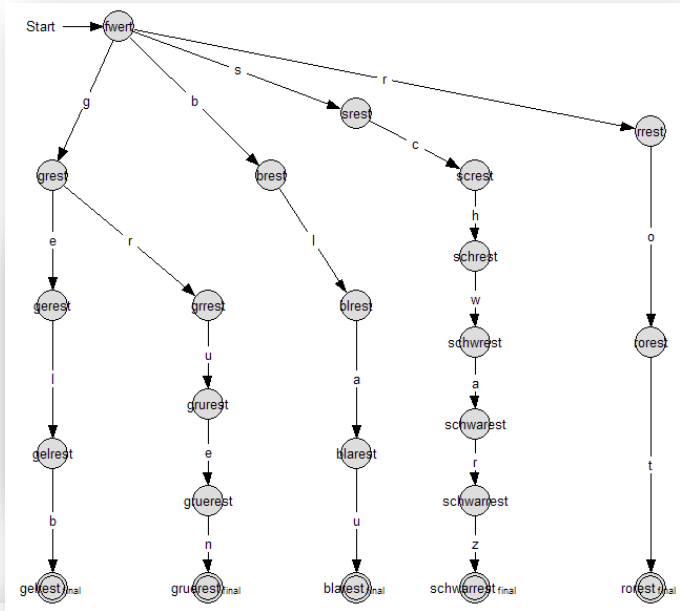
Anweisung → **VW Zahl**

| **RE Zahl**

| **WH Zahl [Anweisungen]**

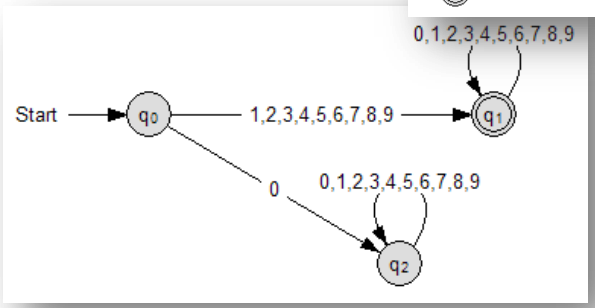
| **FARBE Farbwert**

| **STIFT Zahl**



Farbwert : **rot|blau|gruen|gelb|schwarz**

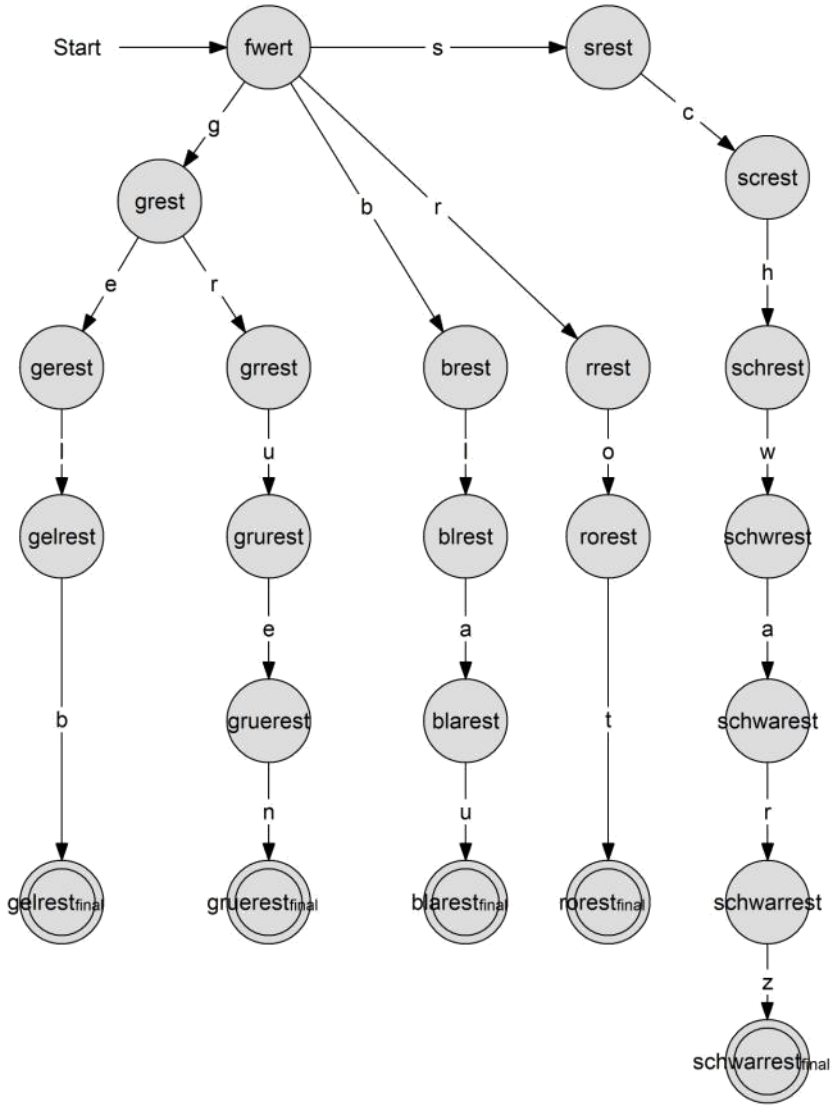
Zahl : **[1-9][0-9]***



Reguläre Grammatik → NEA → DEA


```
fwert -> b brest
brest -> l blrest
blrest -> a blarest
blarest -> u

fwert -> g grest
grest -> e gerest | r grrest
gerest -> l gelrest
gelrest -> b

fwert -> g grest
grrest -> u grurest
grurest -> e gruerest
gruerest -> n

fwert -> s srest
srest -> c screst
screst -> h schrest
schrest -> w schwrest
schwrest -> a schwarest
schwarest -> r schwarrest
schwarrest -> z

fwert -> r rrest
rrest -> o rorest
rorest -> t
```

farbwert.txt



Beispiel: ZR – eine Sprache für einen Zeichenroboter

Endliche Automaten (RegExp) für alle Terminale der ZR-Grammatik:

KlammerAuf : \[

KlammerZu : \]

Wiederhole : WH

Rechts : RE

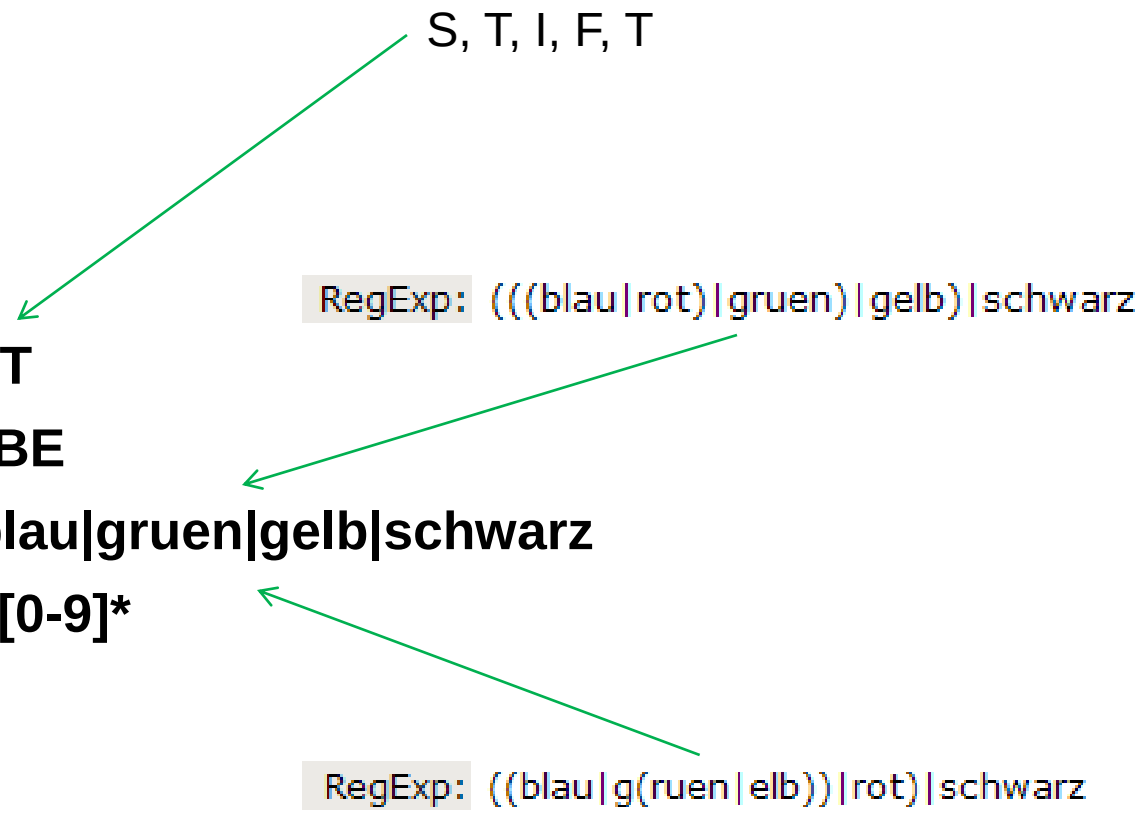
Vor : VW

Stift : STIFT

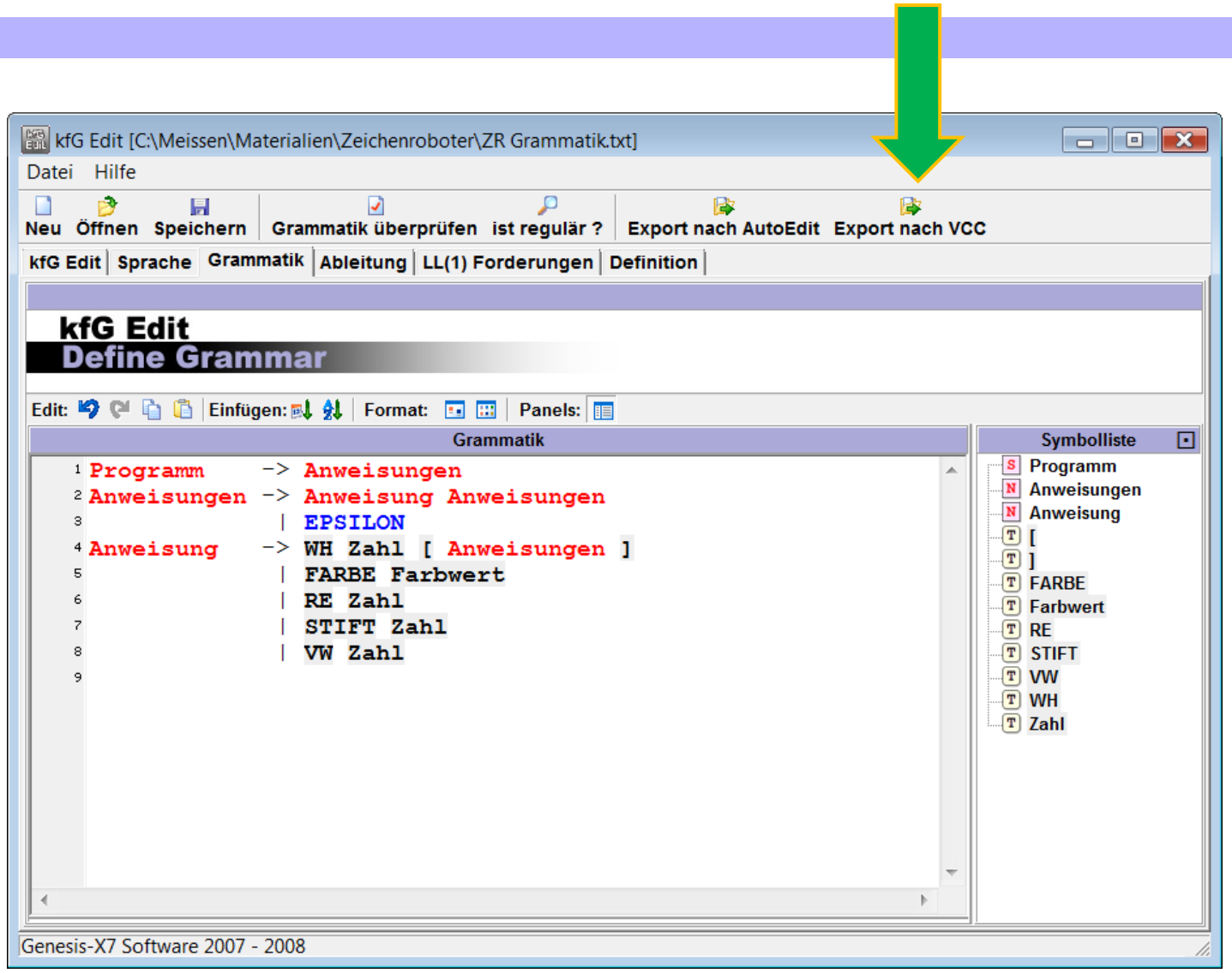
Farbe : FARBE

Farbwert : rot|blau|gruen|gelb|schwarz

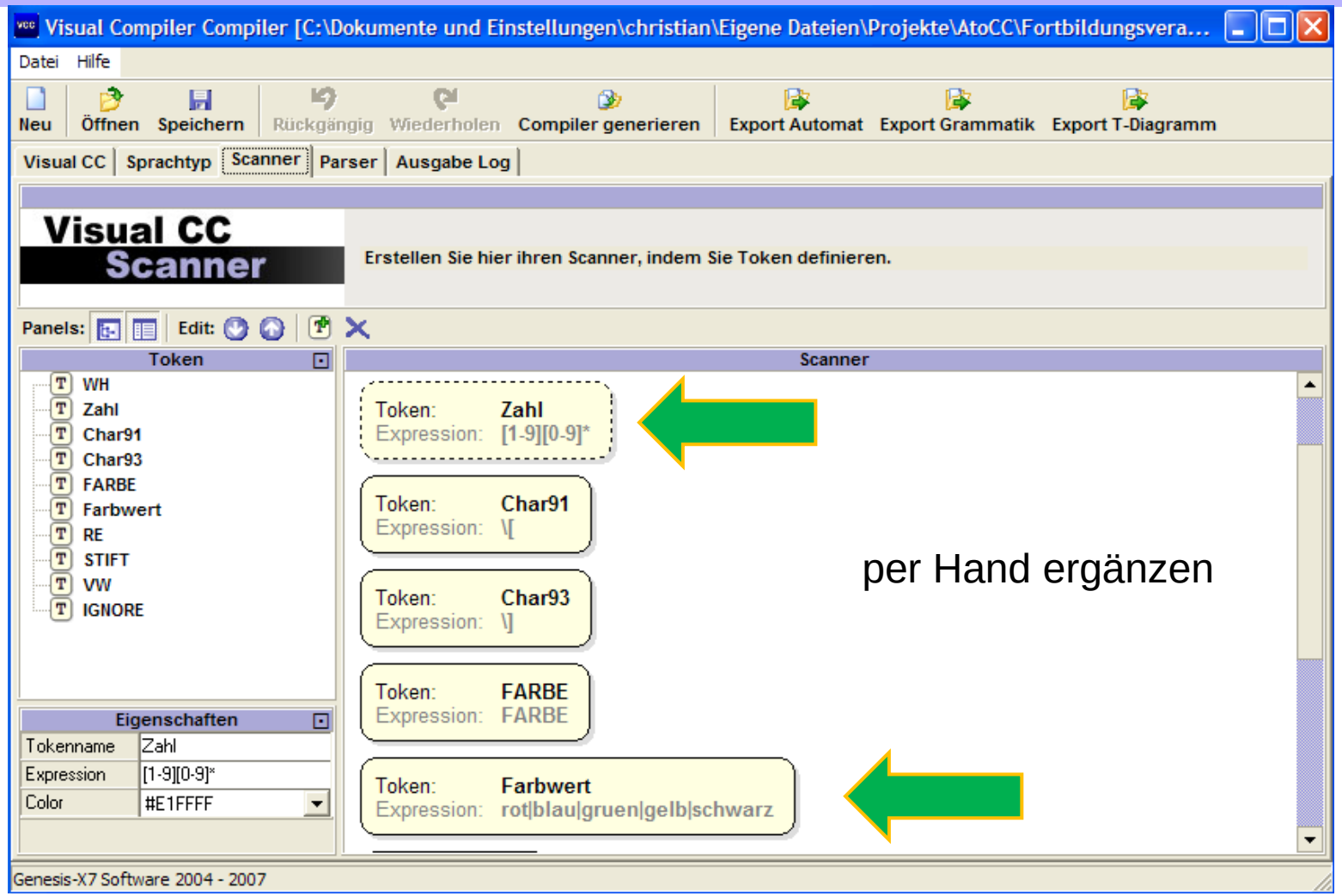
Zahl : [1-9][0-9]*



Beispiel: ZR – eine Sprache für einen Zeichenroboter



Beispiel: ZR – eine Sprache für einen Zeichenroboter



Arbeitsweise des Parsers

➔ **Reduzierte Grammatik durch Scannereinsatz**
Bestimmte Nichtterminale werden zu Terminalen.

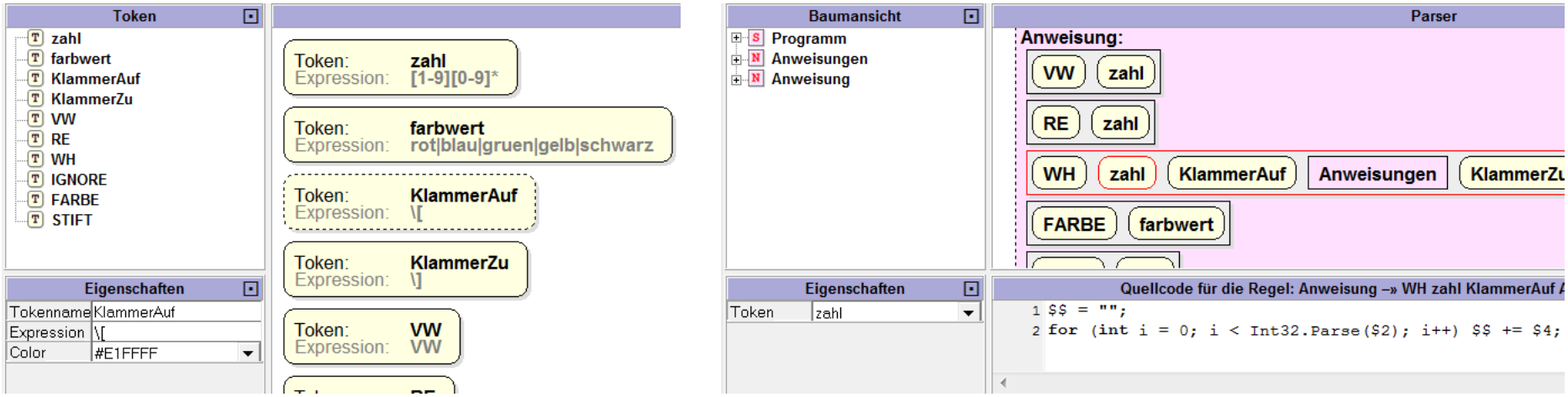
"Beinhaltet die aufgetretenen Terminale der Grammatik des Parsers"

"Grammatik von ZR in Form eines Kellerautomaten → Prüft, ob Wort zur Sprache gehört."

#false erfolgt meist durch Ausgabe von „Syntax Error“

Beispiel: ZR – eine Sprache für einen Zeichenroboter

- Entwicklung des ZR2PS Compilers in VCC
  - Übertragen der EA in die Scannerdefinition
  - Übertragen der vereinfachten Grammatik in die Parserdef.
  - Entwickeln sogenannter S-Attribute (synthetisierter Attribute) durch Angabe von S-Ausdrücken in der Impl.-Sprache des Compilers - für jede Regel zur (syntaxgesteuerten) Zielcodegenerierung



Beispiel: Postscript als Zielsprache des ZR-Compilers

ZR → PS → PDF

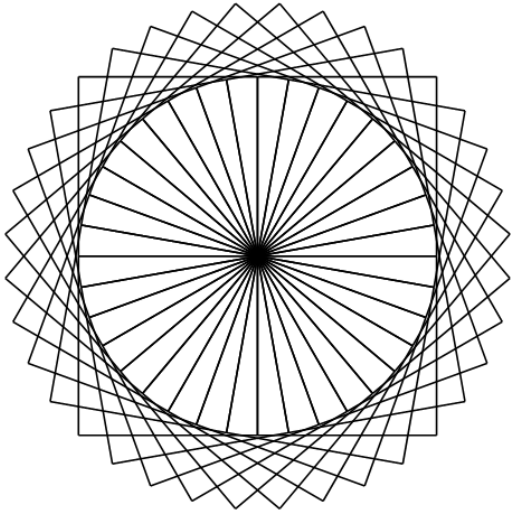
Eingabewort (in ZR):

WH 36 [WH 4 [VW 100 RE 90] RE 10]

Ausgabewort (in PS):

(PS - Arbeitsblatt)

```
%!PS-Adobe-2.0
/orient 0 def /xpos 0 def /ypos 0 def
0 0 0 setrgbcolor
/goto { /ypos exch def /xpos exch def xpos ypos moveto} def
/turn { /orient exch orient add def} def
/draw { /len exch def newpath xpos ypos moveto
   /xpos xpos orient sin len mul add def
   /ypos ypos orient cos len mul add def
   xpos ypos lineto stroke
} def
300 400 goto
10 draw 10 turn ... 10 draw 10 turn
```



Beispiel: ZR – eine Sprache für einen Zeichenroboter

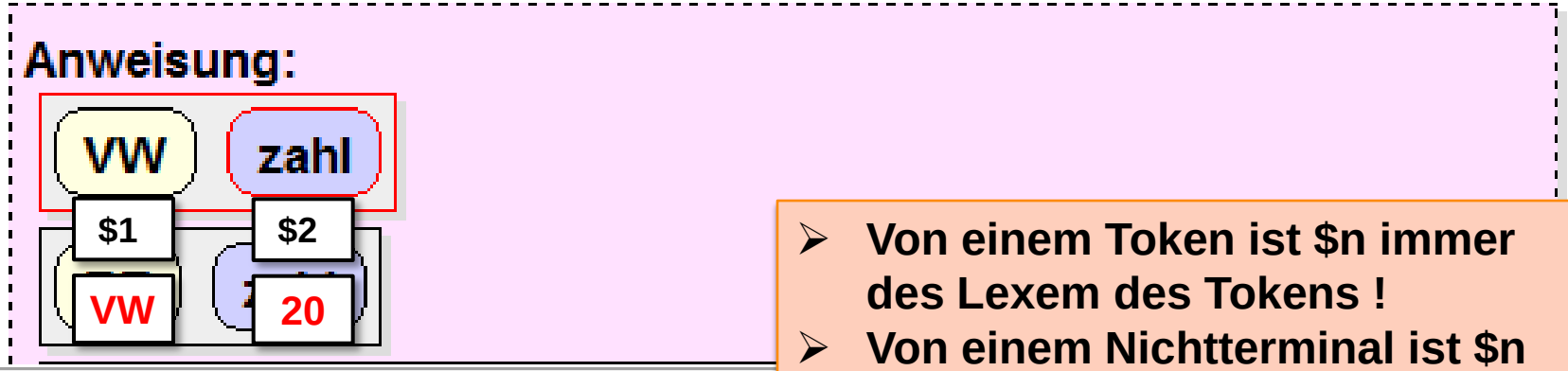
- Syntaxgesteuerte Zielcodegenerierung:
  - Der Compiler soll PostScript erzeugen, nicht nur #true und #false ausgeben.
  - Definition von S-Attributen (mit S-Ausdrücken)
  - S-Ausdrücke beschreiben die Berechnung von Zielcodefragmenten, die für jede rechte Regelseite definiert werden können.
  - Bei jeder Regelanwendung (Knoten im Parsebaum) wird der entsprechende S-Attribut-Wert ermittelt und weitergereicht. (eindeutige Auswertungsreihenfolge)

Beispiel: ZR – eine Sprache für einen Zeichenroboter

Die Platzhalter $1 bis $n:

- In S-Attributen verwenden wir Platzhalter für die Ergebnisse der einzelnen Regelbausteine.

Eingabewort sei: **VW 20 RE 10**



- **Von einem Token ist $n immer des Lexem des Tokens !**
- **Von einem Nichtterminal ist $n immer das Ergebnis $$ des Nichtterminals !**

Quellcode für d

```
1 $$ = $2+" draw ";
```

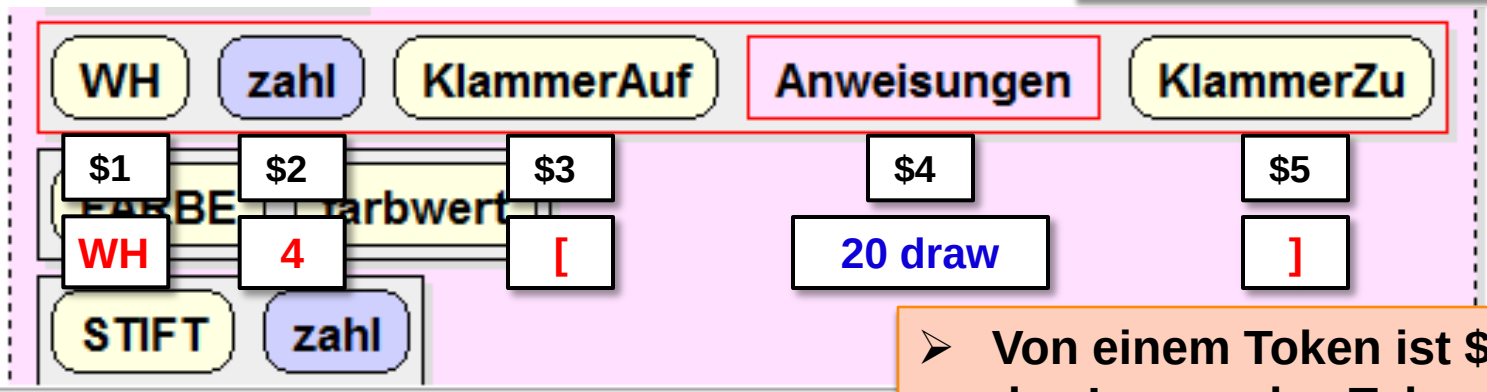
$$ = "20 draw "

Beispiel: ZR – eine Sprache für einen Zeichenroboter

Die Platzhalter $1 bis $n:

- In S-Attributen verwenden wir Platzhalter für die Ergebnisse der einzelnen Regelbausteine.

Eingabewort sei: **WH 4 [VW 20]**



Quellcode für die Regel: Anweisung –» WH z

```
1 $$ = "";
2 for (int i = 0; i < Int32.Parse(
```

$$ = "20 draw 20 draw 20 draw 20 draw "

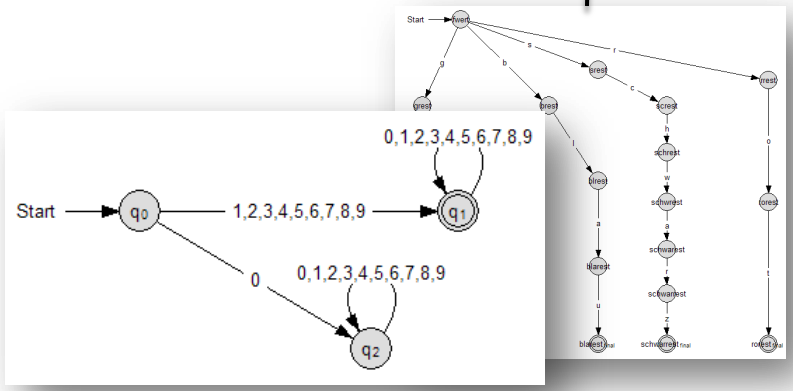
- **Von einem Token ist $n immer des Lexem des Tokens !**
- **Von einem Nichtterminal ist $n immer das Ergebnis $$ des Nichtterminals !**
- **Alle $n und $$ sind vom Datentyp String !!!**

Arbeitsweise des Compilers

WH 36 [WH 4 [VW 100 RE 90] RE 10]

[Wiederhole, "WH"]
[Zahl, "12"]
[KlammerAuf" "["]
…

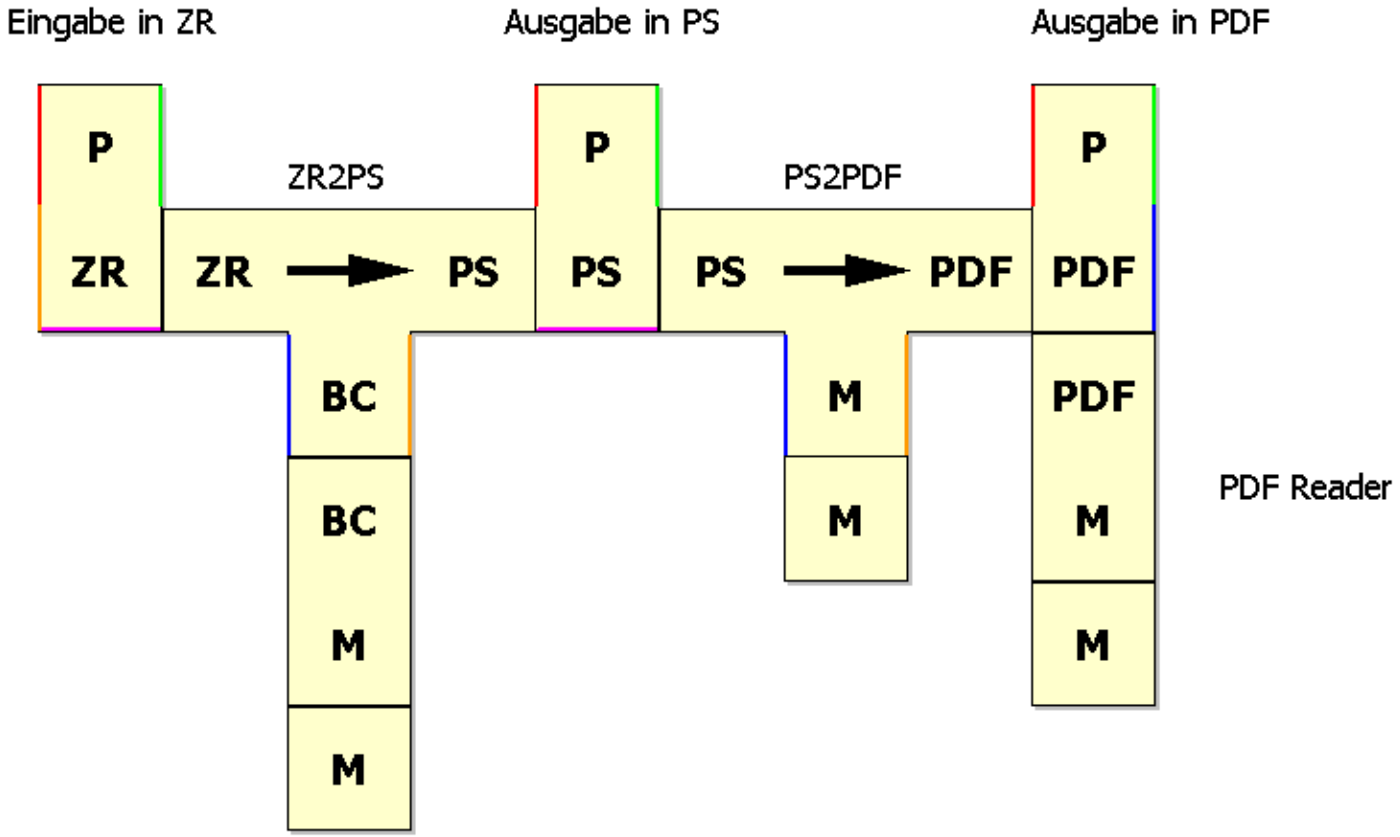
%!PS-Adobe-2.0
/orient 0 def /xpos 0 def /ypos 0 def
0 0 0 setrgbcolor
/goto { /ypos exch def /xpos exch
def xpos ypos moveto} def
…



Programm → **Anweisungen**
Anweisungen → **Anweisung Anweisungen | ε**
Anweisung → **VW Zahl**
| RE Zahl
| WH Zahl [Anweisungen]
| FARBE Farbwert
| STIFT Zahl

Beispiel: ZR – eine Sprache für einen Zeichenroboter

- Anwenden des Compilers auf der Modellierungsebene der T-Diagramme in TDiag.



Antworten auf weitere Fragen ...

Autoren: Wagenknecht, Christian / Hielscher, Michael

Formale Sprachen, abstrakte Automaten und Compiler

Lehr- und Arbeitsbuch für Grundstudium und Fortbildung

2009. XII, 243 S. Mit 95 Abb. Br.
ISBN: 978-3-8348-0624-6

Lehrbuch

Theoretische Informatik mit echten praktischen Anwendungen

29,90 €

Lieferbar, versandfertig in 1-2 Werktagen

Bestellen

http://www.inf.hs-zigr.de/~wagenkn/
http://www.atocc.de