SS 2005

Einführung in die Informatik IV

Ernst W. Mayr

Fakultät für Informatik
TU München

http://www14.in.tum.de/lehre/2005SS/info4/index.html.de

3. Juni 2005

6. Übersicht Chomsky-Hierarchie

6.1 Die Chomsky-Hierarchie

Typ 3	reguläre Grammatik DFA NFA regulärer Ausdruck
DCFL	$LR(k)$-Grammatik deterministischer Kellerautomat
Typ 2	kontextfreie Grammatik (nichtdeterministischer) Kellerautomat
Typ 1	kontextsensitive Grammatik (nichtdet.) linear beschränkter Automat (LBA)
Typ 0	Chomsky-Grammatik, Phrasenstrukturgrammatik det./nichtdet. Turingmaschine

6.2 Wortproblem

Typ 3, gegeben als DFA	lineare Laufzeit
DCFL, gegeben als DPDA	lineare Laufzeit
Typ 2, CNF-Grammatik	CYK-Algorithmus, Laufzeit $O(n^3)$
Typ 1	exponentiell
Typ 0	—

6.3 Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Typ 3	ja	ja	ja	ja	ja
DCFL	nein	nein	ja	nein	nein
Typ 2	nein	ja	nein	ja	ja
Typ 1	ja	ja	ja	ja	ja
Typ 0	ja	ja	nein	ja	ja

6.4 Entscheidbarkeit

	Wortproblem	Leerheit	Äquivalenz	Schnittproblem
Typ 3	ja	ja	ja	ja
DCFL	ja	ja	ja	nein
Typ 2	ja	ja	nein	nein
Typ 1	ja	nein	nein	nein
Typ 0	nein	nein	nein	nein

Kapitel II Berechenbarkeit, Entscheidbarkeit

1. Der Begriff der Berechenbarkeit

Unsere Vorstellung ist:
$f : \mathbb{N}_0^k \to \mathbb{N}_0$ ist berechenbar, wenn es einen Algorithmus gibt, der f berechnet, und genauer, der bei Eingabe $(n_1, \ldots, n_k) \in \mathbb{N}_0^k$ nach endlich vielen Schritten mit dem Ergebnis $f(n_1, \ldots, n_k) \in \mathbb{N}_0$ hält.

Was bedeutet „Algorithmus“ an dieser Stelle?

AWK, B, C, Euler, Fortran, Id, JAVA, Lisp, Modula, Oberon, Pascal, Simula, . . .-Programme?
Gibt es einen Unterschied, wenn man sich auf eine bestimmte Programmiersprache beschränkt?

Kapitel II Berechenbarkeit, Entscheidbarkeit

1. Der Begriff der Berechenbarkeit

Unsere Vorstellung ist:
$f : \mathbb{N}_0^k \to \mathbb{N}_0$ ist berechenbar, wenn es einen Algorithmus gibt, der f berechnet, und genauer, der bei Eingabe $(n_1, \ldots, n_k) \in \mathbb{N}_0^k$ nach endlich vielen Schritten mit dem Ergebnis $f(n_1, \ldots, n_k) \in \mathbb{N}_0$ hält.

Was bedeutet „Algorithmus“ an dieser Stelle?

AWK, B, C, Euler, Fortran, Id, JAVA, Lisp, Modula, Oberon, Pascal, Simula, . . .-Programme?
Gibt es einen Unterschied, wenn man sich auf eine bestimmte Programmiersprache beschränkt?

Kapitel II Berechenbarkeit, Entscheidbarkeit

1. Der Begriff der Berechenbarkeit

Unsere Vorstellung ist:
$f : \mathbb{N}_0^k \to \mathbb{N}_0$ ist berechenbar, wenn es einen Algorithmus gibt, der f berechnet, und genauer, der bei Eingabe $(n_1, \ldots, n_k) \in \mathbb{N}_0^k$ nach endlich vielen Schritten mit dem Ergebnis $f(n_1, \ldots, n_k) \in \mathbb{N}_0$ hält.

Was bedeutet „Algorithmus“ an dieser Stelle?

AWK, B, C, Euler, Fortran, Id, JAVA, Lisp, Modula, Oberon, Pascal, Simula, . . .-Programme?
Gibt es einen Unterschied, wenn man sich auf eine bestimmte Programmiersprache beschränkt?

Kapitel II Berechenbarkeit, Entscheidbarkeit

1. Der Begriff der Berechenbarkeit

Unsere Vorstellung ist:
$f : \mathbb{N}_0^k \to \mathbb{N}_0$ ist berechenbar, wenn es einen Algorithmus gibt, der f berechnet, und genauer, der bei Eingabe $(n_1, \ldots, n_k) \in \mathbb{N}_0^k$ nach endlich vielen Schritten mit dem Ergebnis $f(n_1, \ldots, n_k) \in \mathbb{N}_0$ hält.

Was bedeutet „Algorithmus“ an dieser Stelle?

AWK, B, C, Euler, Fortran, Id, JAVA, Lisp, Modula, Oberon, Pascal, Simula, ...-Programme?
Gibt es einen Unterschied, wenn man sich auf eine bestimmte Programmiersprache beschränkt?

Analog für partielle Funktionen

$$f : \mathbb{N}_0^k \supseteq D \to \mathbb{N}_0$$

bedeutet berechenbar Folgendes:

1. Algorithmus soll mit dem richtigen Ergebnis stoppen, wenn $(n_1, \ldots, n_k) \in D$
2. und nicht stoppen, wenn $(n_1, \ldots, n_k) \notin D$.

Analog für partielle Funktionen

$$f : \mathbb{N}_0^k \supseteq D \to \mathbb{N}_0$$

bedeutet berechenbar Folgendes:

1. Algorithmus soll mit dem richtigen Ergebnis stoppen, wenn $(n_1, \ldots, n_k) \in D$
2. und nicht stoppen, wenn $(n_1, \ldots, n_k) \notin D$.

Analog für partielle Funktionen

$$f : \mathbb{N}_0^k \supseteq D \to \mathbb{N}_0$$

bedeutet berechenbar Folgendes:

1. Algorithmus soll mit dem richtigen Ergebnis stoppen, wenn $(n_1, \ldots, n_k) \in D$
2. und nicht stoppen, wenn $(n_1, \ldots, n_k) \notin D$.

Beispiel 108

Wir definieren folgende Funktionen:

$$f_1(n) = \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge Anfangsstück von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

$$f_2(n) = \begin{cases} 1 & \text{falls } n \text{ interpretiert als Ziffernfolge in } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

$$f_3(n) = \begin{cases} 1 & \text{falls mindestens } n \text{ aufeinanderfolgende Ziffern} \\ & \text{in } \pi \text{ gleich 1 sind} \\ 0 & \text{sonst} \end{cases}$$

Beispiel 108

Wir definieren folgende Funktionen:

$$f_1(n) = \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge Anfangsstück von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

$$f_2(n) = \begin{cases} 1 & \text{falls } n \text{ interpretiert als Ziffernfolge in } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

$$f_3(n) = \begin{cases} 1 & \text{falls mindestens } n \text{ aufeinanderfolgende Ziffern} \\ & \text{in } \pi \text{ gleich 1 sind} \\ 0 & \text{sonst} \end{cases}$$

$\pi = 3,14159265358979323846264338327950288419716939937\ldots$

Einige Beispiele sind damit:

$$f_1(314) = 1, f_1(415) = 0, f_2(415) = 1\ .$$

Beispiel 108

$$f_1(n) = \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge Anfangsstück von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

Wie man leicht einsieht, ist f_1 berechenbar, denn um festzustellen, ob eine Ziffernfolge ein Anfangsstück von π ist, muss π nur auf entsprechend viele Dezimalstellen berechnet werden.

Beispiel 108

$$f_2(n) = \begin{cases} 1 & \text{falls } n \text{ interpretiert als Ziffernfolge in } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

Für f_2 wissen wir nicht, ob es berechenbar ist. Um festzustellen, dass die Ziffernfolge in π vorkommt, müsste man π schrittweise immer genauer approximieren. Der Algorithmus würde stoppen, wenn die Ziffernfolge gefunden wird. Aber was ist, wenn die Ziffernfolge in π nicht vorkommt?

Beispiel 108

$$f_2(n) = \begin{cases} 1 & \text{falls } n \text{ interpretiert als Ziffernfolge in } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

Für f_2 wissen wir nicht, ob es berechenbar ist. Um festzustellen, dass die Ziffernfolge in π vorkommt, müsste man π schrittweise immer genauer approximieren. Der Algorithmus würde stoppen, wenn die Ziffernfolge gefunden wird. Aber was ist, wenn die Ziffernfolge in π nicht vorkommt?
Vielleicht gibt es aber einen (noch zu findenden) mathematischen Satz, der genaue Aussagen über die in π vorkommenden Ziffernfolgen macht.

Beispiel 108

$$f_2(n) = \begin{cases} 1 & \text{falls } n \text{ interpretiert als Ziffernfolge in } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

Für f_2 wissen wir nicht, ob es berechenbar ist. Um festzustellen, dass die Ziffernfolge in π vorkommt, müsste man π schrittweise immer genauer approximieren. Der Algorithmus würde stoppen, wenn die Ziffernfolge gefunden wird. Aber was ist, wenn die Ziffernfolge in π nicht vorkommt?
Vielleicht gibt es aber einen (noch zu findenden) mathematischen Satz, der genaue Aussagen über die in π vorkommenden Ziffernfolgen macht.
Wäre π vollkommen zufällig, was es aber nicht ist, dann würde jedes n als Ziffernfolge irgendwann vorkommen.

Beispiel 108

$$f_3(n) = \begin{cases} 1 & \text{falls mindestens } n \text{ aufeinanderfolgende Ziffern} \\ & \text{in } \pi \text{ gleich 1 sind} \\ 0 & \text{sonst} \end{cases}$$

f_3 ist berechenbar, denn $f_3 \equiv f_4$, mit

$$f_4(n) = \begin{cases} 1 & n \leq n_0 \\ 0 & \text{sonst} \end{cases}$$

wobei n_0 die maximale Anzahl von aufeinanderfolgenden 1en in π ist. Hierbei ist es nicht von Bedeutung, wie die Zahl n_0 berechnet werden kann - wichtig ist nur, dass eine solche Zahl $n_0 \in \mathbb{N}$ existiert.

Weitere Vorschläge, den Begriff der Berechenbarkeit zu präzisieren und zu formalisieren:

1. Turing-Berechenbarkeit
2. Markov-Algorithmen
3. λ-Kalkül
4. μ-rekursive Funktionen
5. Registermaschinen
6. AWK, B, C, Euler, Fortran, Id, JAVA, Lisp, Modula, Oberon, Pascal, Simula, ...-Programme
7. while-Programme
8. goto-Programme
9. DNA-Algorithmen
10. Quantenalgorithmen
11. u.v.a.m.

Weitere Vorschläge, den Begriff der Berechenbarkeit zu präzisieren und zu formalisieren:

1. Turing-Berechenbarkeit
2. Markov-Algorithmen
3. λ-Kalkül
4. μ-rekursive Funktionen
5. Registermaschinen
6. AWK, B, C, Euler, Fortran, Id, JAVA, Lisp, Modula, Oberon, Pascal, Simula, ...-Programme
7. while-Programme
8. goto-Programme
9. DNA-Algorithmen
10. Quantenalgorithmen
11. u.v.a.m.

Weitere Vorschläge, den Begriff der Berechenbarkeit zu präzisieren und zu formalisieren:

1. Turing-Berechenbarkeit
2. Markov-Algorithmen
3. λ-Kalkül
4. μ-rekursive Funktionen
5. Registermaschinen
6. AWK, B, C, Euler, Fortran, Id, JAVA, Lisp, Modula, Oberon, Pascal, Simula, ...-Programme
7. while-Programme
8. goto-Programme
9. DNA-Algorithmen
10. Quantenalgorithmen
11. u.v.a.m.

Weitere Vorschläge, den Begriff der Berechenbarkeit zu präzisieren und zu formalisieren:

1. Turing-Berechenbarkeit
2. Markov-Algorithmen
3. λ-Kalkül
4. μ-rekursive Funktionen
5. Registermaschinen
6. AWK, B, C, Euler, Fortran, Id, JAVA, Lisp, Modula, Oberon, Pascal, Simula, . . .-Programme
7. while-Programme
8. goto-Programme
9. DNA-Algorithmen
10. Quantenalgorithmen
11. u.v.a.m.

Weitere Vorschläge, den Begriff der Berechenbarkeit zu präzisieren und zu formalisieren:

1. Turing-Berechenbarkeit
2. Markov-Algorithmen
3. λ-Kalkül
4. μ-rekursive Funktionen
5. Registermaschinen
6. AWK, B, C, Euler, Fortran, Id, JAVA, Lisp, Modula, Oberon, Pascal, Simula, . . .-Programme
7. while-Programme
8. goto-Programme
9. DNA-Algorithmen
10. Quantenalgorithmen
11. u.v.a.m.

Weitere Vorschläge, den Begriff der Berechenbarkeit zu präzisieren und zu formalisieren:

1. Turing-Berechenbarkeit
2. Markov-Algorithmen
3. λ-Kalkül
4. μ-rekursive Funktionen
5. Registermaschinen
6. AWK, B, C, Euler, Fortran, Id, JAVA, Lisp, Modula, Oberon, Pascal, Simula, . . .-Programme
7. while-Programme
8. goto-Programme
9. DNA-Algorithmen
10. Quantenalgorithmen
11. u.v.a.m.

Weitere Vorschläge, den Begriff der Berechenbarkeit zu präzisieren und zu formalisieren:

1. Turing-Berechenbarkeit
2. Markov-Algorithmen
3. λ-Kalkül
4. μ-rekursive Funktionen
5. Registermaschinen
6. AWK, B, C, Euler, Fortran, Id, JAVA, Lisp, Modula, Oberon, Pascal, Simula, . . .-Programme
7. while-Programme
8. goto-Programme
9. DNA-Algorithmen
10. Quantenalgorithmen
11. u.v.a.m.

Weitere Vorschläge, den Begriff der Berechenbarkeit zu präzisieren und zu formalisieren:

1. Turing-Berechenbarkeit
2. Markov-Algorithmen
3. λ-Kalkül
4. μ-rekursive Funktionen
5. Registermaschinen
6. AWK, B, C, Euler, Fortran, Id, JAVA, Lisp, Modula, Oberon, Pascal, Simula, . . .-Programme
7. while-Programme
8. goto-Programme
9. DNA-Algorithmen
10. Quantenalgorithmen
11. u.v.a.m.

Weitere Vorschläge, den Begriff der Berechenbarkeit zu präzisieren und zu formalisieren:

1. Turing-Berechenbarkeit
2. Markov-Algorithmen
3. λ-Kalkül
4. μ-rekursive Funktionen
5. Registermaschinen
6. AWK, B, C, Euler, Fortran, Id, JAVA, Lisp, Modula, Oberon, Pascal, Simula, . . .-Programme
7. while-Programme
8. goto-Programme
9. DNA-Algorithmen
10. Quantenalgorithmen
11. u.v.a.m.

Weitere Vorschläge, den Begriff der Berechenbarkeit zu präzisieren und zu formalisieren:

1. Turing-Berechenbarkeit
2. Markov-Algorithmen
3. λ-Kalkül
4. μ-rekursive Funktionen
5. Registermaschinen
6. AWK, B, C, Euler, Fortran, Id, JAVA, Lisp, Modula, Oberon, Pascal, Simula, . . .-Programme
7. while-Programme
8. goto-Programme
9. DNA-Algorithmen
10. Quantenalgorithmen
11. u.v.a.m.

Weitere Vorschläge, den Begriff der Berechenbarkeit zu präzisieren und zu formalisieren:

1. Turing-Berechenbarkeit
2. Markov-Algorithmen
3. λ-Kalkül
4. μ-rekursive Funktionen
5. Registermaschinen
6. AWK, B, C, Euler, Fortran, Id, JAVA, Lisp, Modula, Oberon, Pascal, Simula, . . .-Programme
7. while-Programme
8. goto-Programme
9. DNA-Algorithmen
10. Quantenalgorithmen
11. u.v.a.m.

Weitere Vorschläge, den Begriff der Berechenbarkeit zu präzisieren und zu formalisieren:

1. Turing-Berechenbarkeit
2. Markov-Algorithmen
3. λ-Kalkül
4. μ-rekursive Funktionen
5. Registermaschinen
6. AWK, B, C, Euler, Fortran, Id, JAVA, Lisp, Modula, Oberon, Pascal, Simula, . . .-Programme
7. while-Programme
8. goto-Programme
9. DNA-Algorithmen
10. Quantenalgorithmen
11. u.v.a.m.

Es wurde bewiesen: Alle diese Beschreibungsmethoden sind in ihrer Mächtigkeit äquivalent.

Church'sche These

Dieser formale Begriff der Berechenbarkeit stimmt mit dem intuitiven überein.

Es wurde bewiesen: Alle diese Beschreibungsmethoden sind in ihrer Mächtigkeit äquivalent.

Church'sche These
Dieser formale Begriff der Berechenbarkeit stimmt mit dem intuitiven überein.