technische universität
dortmund

Einführung in die Programmierung

Wintersemester 2019/20
https://ls11-www.cs.tu-dortmund.de/teaching/ep1920vorlesung

Dr.-Ing. Horst Schirmeier
(mit Material von Prof. Dr. Günter Rudolph)

Arbeitsgruppe Eingebettete Systemsoftware (LS 12)
und Lehrstuhl für Algorithm Engineering (LS11)

Fakultät für Informatik

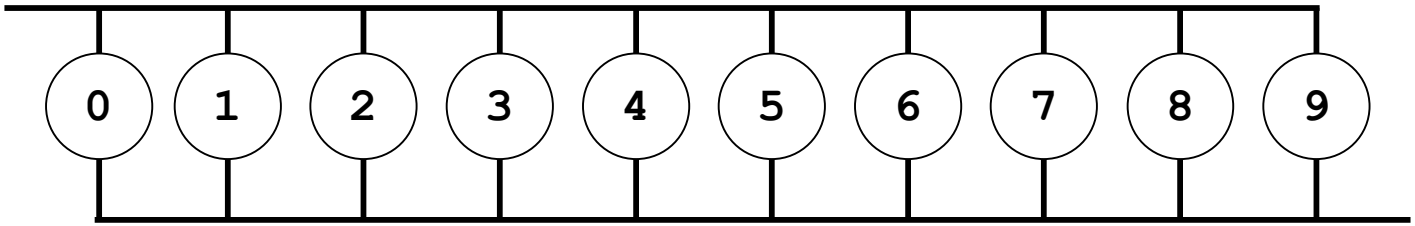
TU Dortmund

Inhalt

- Kontextfreie Grammatiken
- Deterministische Endliche Automaten (DEAs)
- Polymorphie und virtuelle Methoden
- Zeigerarithmetik

Grafische Darstellung

Ziffer :=



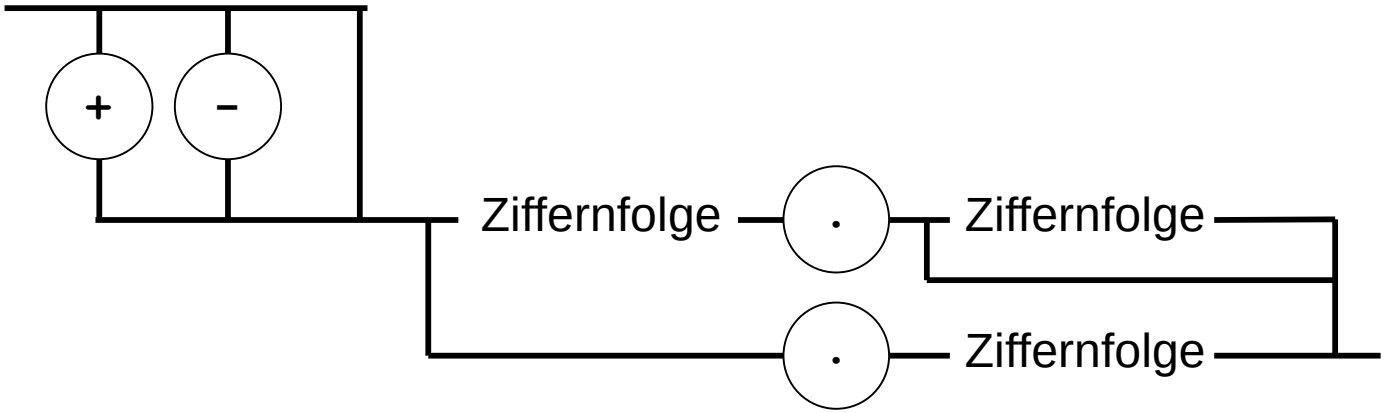
Ohne Pfeile: ***„von links nach rechts, von oben nach unten“***

Ziffernfolge:=

Exkurs: Grammatiken

Ganzzahl mit Vorzeichen :=

Festkommazahlen :=



Definition

Eine ***kontextfreie Grammatik*** G = (N, T, S, P) besteht aus

- einer endlichen Menge von Nichtterminalen N,
- einer endlichen Menge von Terminalen T,
- einem Startsymbol S $\in$ N,
- einer endlichen Menge von Produktionsregeln der Form u $\rightarrow$ v, wobei
  - u $\in$ N und
  - v eine endliche Sequenz von Elementen von N und T ist, sowie
- der Randbedingung N $\cap$ T = $\varnothing$.

Beispiel

T = { +, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

N = { Z, A, D }

S = { Z }

Z → +A

Z → -A

Z → A

A → D

A → AD

D → 0

D → 1

…

D → 9

= P

Kompaktere Notation:

Z → +A|-A|A

A → D|AD

D → 0|1|2|3|4|5|6|7|8|9

Beispiel

T = { +, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

N = { Z, A, D }

S = { Z }

Z → +A|-A|A

A → D|AD

D → 0|1|2|3|4|5|6|7|8|9

- Nichtterminale sind Platzhalter.
- Man kann dort eine Produktionsregel anwenden.
- Der Ersetzungsprozess endet,
  wenn **alle Nichtterminale durch Terminale ersetzt** worden sind.

Beispiel

T = { +, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

N = { Z, A, D }

S = { Z }

Z → +A|-A|A

A → D|AD

D → 0|1|2|3|4|5|6|7|8|9

Können wir mit dieser Grammatik +911 erzeugen?

Start mit Z → +A, wende Produktionsregel A → AD auf A an, ergibt Z → +AD

Wende A → AD auf A an, ergibt Z → +ADD

Wende A → D auf A an, ergibt Z → +DDD,

Wende D → 9 auf das erste D, D → 1 auf die übrigen D an, ergibt Z → +911.

Beispiele zum Üben

- Erstellen einer kontextfreien Grammatik G = (N, T, S, P) für die **Sprache aller Palindrome** aus den Buchstaben a, b, c
  - Beispielwörter: abcba, caabaac, abba
  - Gegenbeispiele: abcb, ab, acbacb
- Gegeben: kontextfreie Grammatik G = (N, T, S, P)
  N := { S, A }
  T := { a, b }
  S := { S }
  P := { S → bSbb | A, A → aA | ε }

  ε: „leeres Wort“
  - Welche der folgenden Wörter sind in dieser Sprache? Welche Ableitungsregeln müssen angewendet werden, um sie zu bilden?
    bbb, **aaabbb**, **aaa**, **bbaabbbb**, **bbbb**, **bbbabbb**

engl. FSM: finite state machine

Der DEA ist **zentrales Modellierungswerkzeug** in der Informatik.

Definition

Ein **deterministischer endlicher Automat** ist ein 5-Tupel $(S, \Sigma, \delta, F, s_0)$, wobei

- S eine endliche Menge von Zuständen,
- Σ das endliche Eingabealphabet,
- δ: $S \times \Sigma \rightarrow S$ die Übergangsfunktion,
- F eine Menge von Finalzuständen mit $F \subseteq S$ und
- s_0 der Startzustand. ■

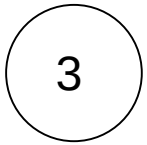
Er startet immer im Zustand s_0, verarbeitet Eingaben und wechselt dabei seinen Zustand. Er terminiert ordnungsgemäß, wenn Eingabe leer **und** ein Endzustand aus F erreicht.

$\Rightarrow$ Beschreibung eines Programms!

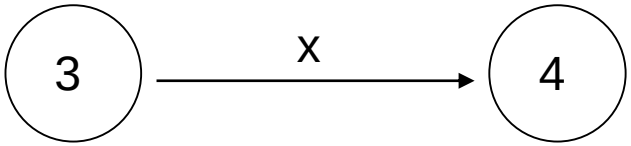
Grafische Darstellung

Zustände als Kreise

im Kreis der Bezeichner des Zustands (häufig durchnummeriert)



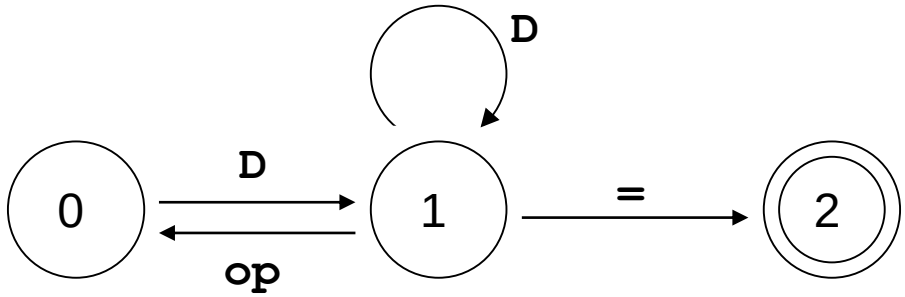
Übergänge von einem Zustand zum anderen sind **abhängig von der Eingabe**.
Mögliche Übergänge sind durch Pfeile zwischen den Zuständen dargestellt;
über / unter dem Pfeil steht das **Eingabesymbol**, das den Übergang auslöst.



Endzustände werden durch „Doppelkreise“ dargestellt.

Beispiel:

Entwerfe DEA, der arithmetische Ausdrücke ohne Klammern für nichtnegative Ganzzahlen auf Korrektheit prüft.



Zustände S = { 0, 1, 2 }

Startzustand s_0 = 0

Endzustände F = { 2 }

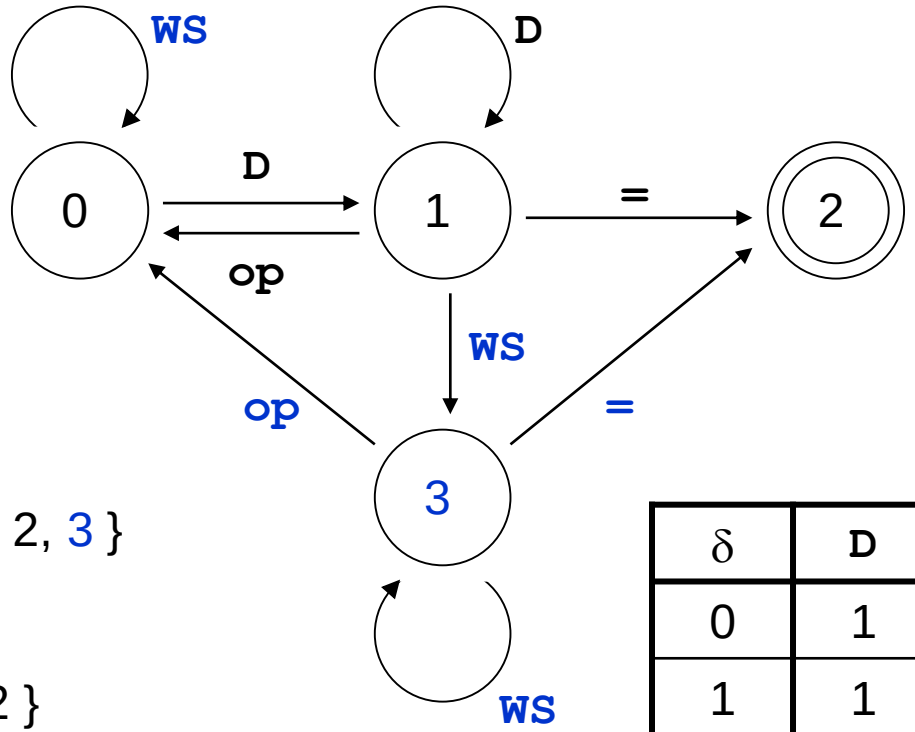
Eingabealphabet Σ = { D, op, = }

δ	D	op	=
0	1	-1	-1
1	1	0	2
2	-	-	-

-1: Fehlerzustand

Beispiel:

Erweiterung: Akzeptiere auch „white space“ zwischen Operanden und Operatoren



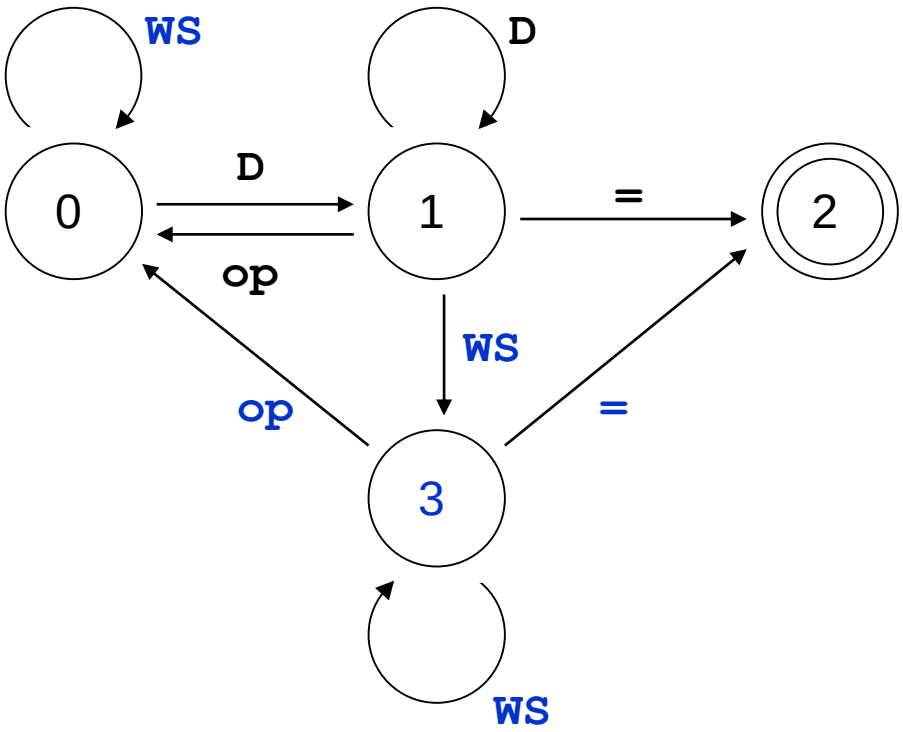
Zustände S = { 0, 1, 2, 3 }

Startzustand s_0 = 0

Endzustände F = { 2 }

Eingabealphabet Σ = { D, op, =, WS }

δ	D	op	=	WS
0	1	-1	-1	0
1	1	0	2	3
2	-	-	-	-
3	-1	0	2	3



Eingabe:

```
3+ 4 - 5=
```

durchlaufene Zustände?

Zustand 0, lese D →
Zustand 1, lese op →
Zustand 0, lese WS →
Zustand 0, lese D →
Zustand 1, lese WS →
Zustand 3, lese op →
Zustand 0, lese WS →
Zustand 0, lese D →
Zustand 1, lese = →
Zustand 2 (Endzustand)

Beispiele zum Üben

- Zeichnen Sie den DEA mit S := {0, 1, 2, 3}, s_0 := 0, F := {1, 3}, Σ := {a, b} und der in der Tabelle dargestellten Übergangsfunktion δ.
  - Welche der folgenden Wörter wird von diesem Automaten akzeptiert, und welche Zustände werden dabei durchlaufen?
  - **abbbabb, a, bbb, aa, abaaabb, bababab, bbbb**

δ	a	b
0	3	2
1	3	2
2	3	1
3	2	3

- Entwerfen Sie einen DEA, der Wörter über dem Eingabealphabet { a, b } akzeptiert, in denen die **Anzahl der vorkommenden „a“ durch 3 teilbar ist**.

Virtuelle Methoden

```
class Polygon {
protected:
    int width, height;
public:
    void set_values(int a, int b) { width = a; height = b; }
    int area() { return 0; }
};
class Rectangle : public Polygon {
public:
    int area() {
        return width * height;
    }
};
class Triangle : public Polygon {
public:
    int area() {
        return width * height / 2;
    }
};
```

Virtuelle Methoden

```
int sum_areas(int const n, Polygon **p) {
    int sum = 0;
    for (int i = 0; i < n; ++i) {
        sum += p[i]->area();
    }
    return sum;
}

int main()
{
    Rectangle r;  r.set_values(3, 3);
    Triangle t;   t.set_values(4, 2);
    Polygon *array[2] = { &r, &t };
    std::cout << sum_areas(2, array) << std::endl;
}
```

Ausgabe: 0

Merke:

- Zuweisungen sind **entlang der Vererbungshierarchie** möglich

  → Objekt kann einem Objekt seiner Oberklasse zugewiesen werden

- Methoden sind (hier) **statisch** an Objekt gebunden

  → zur Übersetzungszeit bekannte Methode wird ausgeführt

  → Zuweisung eines Objekts einer abgeleiteten Klasse führt **nicht** zur Übernahme der überschriebenen Methoden der Unterklasse

⇓

Wenn man das haben möchte, dann müssten die Methoden der Unterklasse **zur Laufzeit** (bei der Zuweisung) an das Objekt **gebunden** werden.

→ **dynamische Bindung**

Virtuelle Methoden

```
class Polygon {
protected:
    int width, height;
public:
    void set_values(int a, int b) { width = a; height = b; }
    virtual int area() { return 0; }
};
class Rectangle : public Polygon {
public:
    int area() {
        return width * height;
    }
};
class Triangle : public Polygon {
public:
    int area() {
        return width * height / 2;
    }
};
```

Virtuelle Methoden

```
int sum_areas(int const n, Polygon **p) {
    int sum = 0;
    for (int i = 0; i < n; ++i) {
        sum += p[i]->area();
    }
    return sum;
}

int main()
{
    Rectangle r;  r.set_values(3, 3);
    Triangle t;   t.set_values(4, 2);
    Polygon *array[2] = { &r, &t };
    std::cout << sum_areas(2, array) << std::endl;
}
```

Ausgabe: 13

Virtuelle Methoden

```
class Polygon {
protected:
    int width, height;
public:
    void set_values(int a, int b) { width = a; height = b; }
    virtual int area() = 0;
};
class Rectangle : public Polygon {
public:
    int area() {
        return width * height;
    }
};
class Triangle : public Polygon {
public:
    int area() {
        return width * height / 2;
    }
};
```

Zeigerarithmetik

```
int main() {
// Nehmen Sie an, dass ein short int eine Größe von 16 Bit / 2 Bytes hat.
    short x = 42;
    short a[] = { 300, 301, 302 };
    short *p1 = &a[2], *p2 = new short, *p3 = &x;
    short **pp = &p1;
    *p2 = 8;
    // Zeitpunkt 1

    ++x;
    p2 = p1;
    a[0] = a[1] / 2;
    // Zeitpunkt 2

    *p2 = 0;
    *(p2 - 2) = 23;
    // Zeitpunkt 3

    p1++;
    // Zeitpunkt 4

    *(*pp - 2) = 5;
    **pp = 6;
    // Zeitpunkt 5
}
```

	x	a[0]	a[1]	a[2]	p1	*p1	p2	*p2	p3	*p3	pp	*pp	**pp
1					996		2000		998		984		
2	…												