Theoretische Grundlagen des Software Engineering

10: Einführung Reduktionssysteme

Stephan Schulz
schulz@eprover.org

Reduktionssysteme

Definition: Reduktionssystem

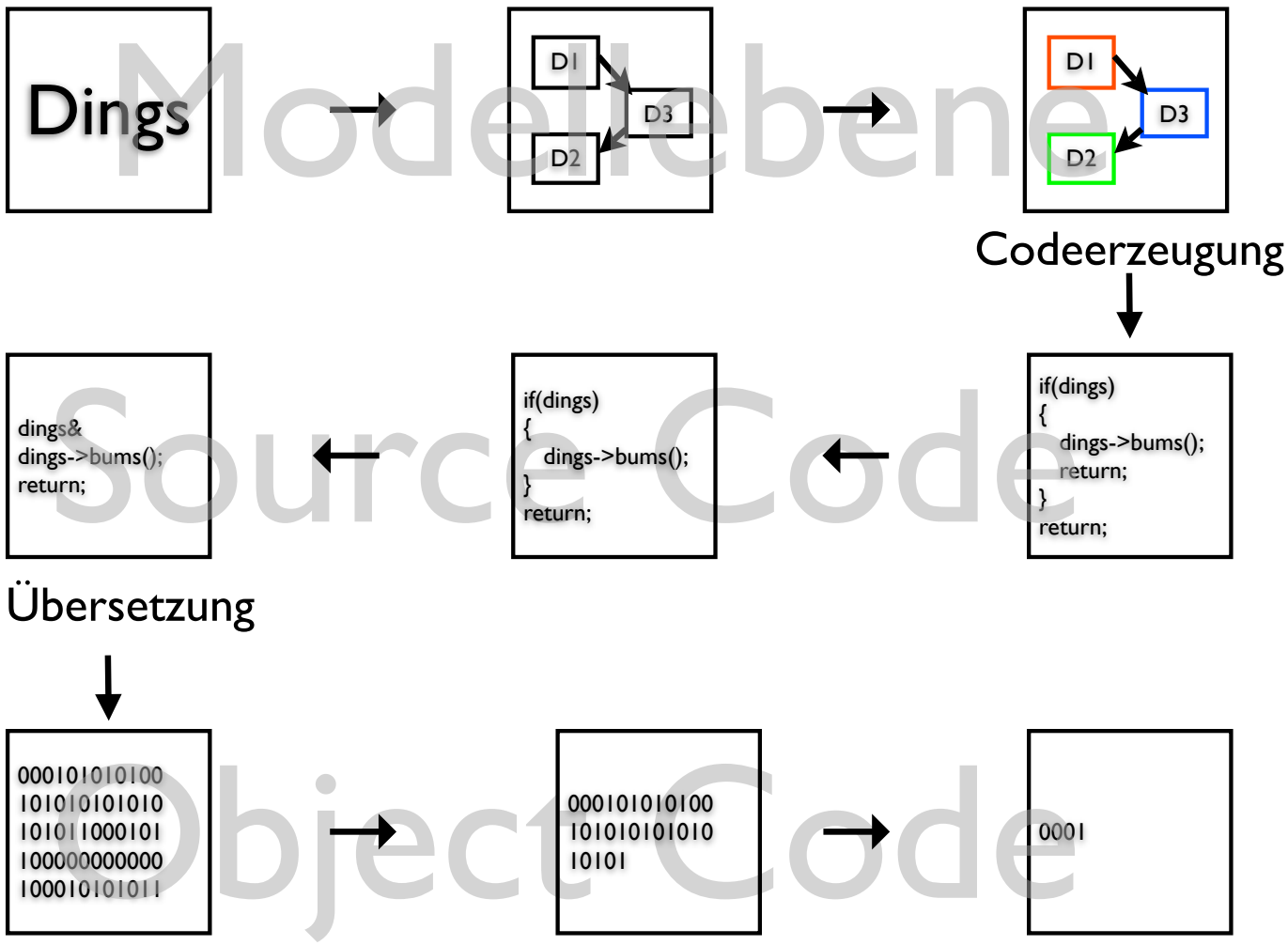
Ein Reduktionssystem ist ein Tupel

$$(A, \rightarrow)$$

Dabei gilt:

- A ist eine beliebige Menge (die Trägermenge)
- $\rightarrow \subseteq A \times A$ ist eine zweistellige Relation über A (die Reduktionsrelation)

Beispiel: SW- Entwickung



Interessante Fragen

Terminiert der Prozess?

- Als PL: Werden die Programmierer fertig?
- Als Programmierer: Wird der Compiler fertig?

Ist das Ergebnis unabhängig von der Reihenfolge?

- Designer: Kann ich verschiedene Entscheidungen unabhängig treffen?
- Compiler: Können Optimierungen in beliebiger Reihenfolge angewendet werden? Kann ich ein “optimales” Ergebnis garantieren?

Sind zwei Artefakte “gleich”?

- Compiler: Können sie substituiert werden?
- Designer: Können sie wiederverwendet werden?
- Rechtlich: Copyright!

Binäre Relationen

Erinnerung

Eine binäre Relation R über einer Menge A ist eine Menge von 2- Tupeln über A, also $R \subseteq A \times A$

- Statt $(a, b) \in R$ schreiben wir auch $R(a, b)$ oder aRb
- Eigenschaften: Rechtstotal, linkseindeutig, reflexiv, symmetrisch, transitiv

Wichtige Klassen von binären Relationen:

- Äquivalenzrelationen
- Ordnungen

Äquivalenzrelation

Erinnerung

Definition: Sei R eine binäre Relation über A. R ist:

reflexiv, falls $\{(a,a)|a \in A\} \subseteq R$
symmetrisch, falls $\forall a,b \in A : aRb \implies bRa$
transitiv, falls $\forall a,b,c : (aRb \wedge bRc) \implies aRc$

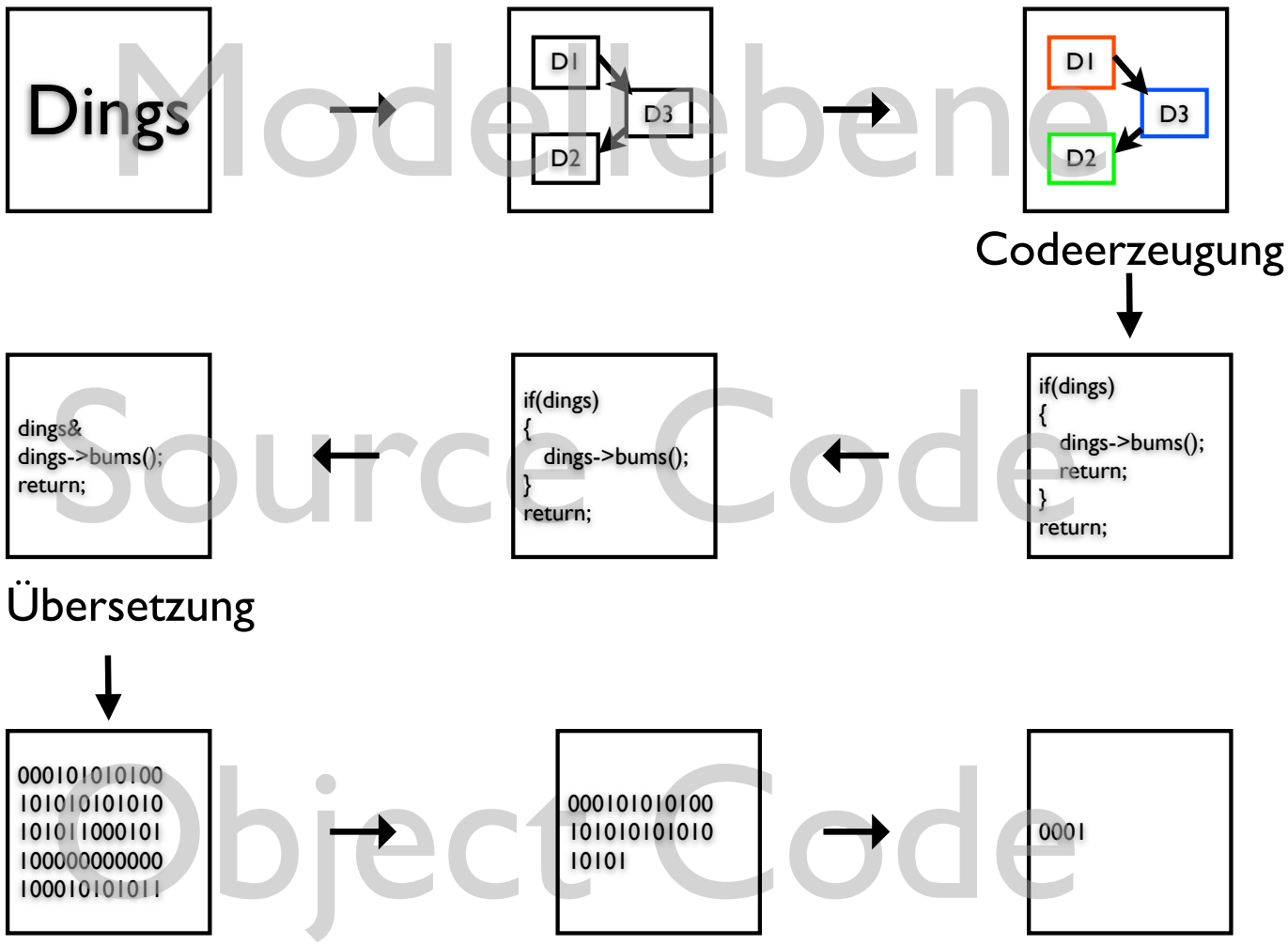
Eine Relation, die reflexiv, symmetrisch, und transitiv ist, heißt Äquivalenzrelation.

Beispiele für Äquivalenzrelationen

- Logische Äquvalenz auf $For0_\Sigma$
- $\equiv = \{(a,a)|a \in A\}$ für beliebiges A
  - $\equiv$ ist die eindeutige kleinste (bzgl. $\subset$) Äquivalenzrelation über A
  - Insbesondere gilt: $\equiv \subseteq R$ für jede Äquivalenzrelation R über A
- Gleichheit auf arithmetischen Ausdrücken:
  $(a+b)^2 = a^2 + 2ab + b^2$, $\frac{6}{4} = \frac{3}{2}$

Generell: Äquivalenzrelationen sind Formen von Gleichheit

Beispiel: SW- Entwickung



Beispiel: Programmentwicklung

Äquivalente Programme:

- A_1 ist die Menge aller Repräsentationen eines Programms
- $\approx_1 \subseteq A_1 \times A_1$ sei definiert als $a \approx_1 b$, wenn a und b die gleiche Funktionalität beschreiben.
- Dann ist $\approx_1$ eine Äquivalenzrelatiom

Äquivalente Funktionen (selbe Idee, konkreter):

- A_2 ist die Menge aller Funktionen in einer Programmiersprache
- $a \approx_2 b$, wenn a und b die selbe Signatur haben und für alle Eingaben das gleiche Ergebnis liefern
- Dann ist $\approx_2$ eine Äquivalenzrelation

Zwei simple Folgen

Fibonacci- Zahlen

$fib(0) = 0$
$fib(1) = 1$
$fib(n) = fib(n-1) + fib(n-2)$ falls $n >= 2$
Folge: $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$

Fakultät

$fak(0) = 1$
$fak(n) = n * fak(n-1)$ falls $n >= 1$
Folge: $1, 1, 2, 6, 24, 120, \ldots$

Fibonacci- Zahlen (1)

```
def fib1(a):
    if a==0:
        return 0
    elif a==1:
        return 1
    else:
        return fib1(a-1)+fib1(a-2)
```

Fibonacci- Zahlen (2)

```
def fib2(a):
    if a<=1:
        return a
    else:
        return fib2(a-1)+fib2(a-2)
```

Fibonacci- Zahlen (3)

```
def fib3help(a, cache):
    if a in cache:
        return cache[a]
    elif a<=1:
        return a
    else:
        res = fib3help(a-1, cache)+fib3help(a-2, cache)
        cache[a] = res
        return res

def fib3(a):
    cache = {}
    return fib3help(a, cache)
```

Fibonacci- Zahlen (4)

```
def fib4(a):
    if a<=1:
        return a
    cache = {}
    cache[0] = 0
    cache[1] = 1
    i=2
    while i<=a:
        cache[i] = cache[i-1]+cache[i-2]
        i=i+1
    return cache[a]
```

Fibonacci- Zahlen (5)

```
def fib5(a):
    if a<=1:
        return a

    n1 = 1
    n2 = 0
    i=2
    while i<=a:
        res = n1+n2
        n2  = n1
        n1  = res
        i=i+1
    return n1
```

Fakultät (1)

```
def fak1(a):
    if a==0:
        return 1
    else:
        return  a*fak1(a-1)

def fak2(a):
    if a==0:
        return 1
    return  a*fak2(a-1)
```

Fakultät (2)

```
def fak3(a):
    res = 1
    while a>0:
        res = res*a
        a=a-1
    return res

def fak4(a):
    return reduce(lambda x,y:x*y, xrange(1,a+1), 1)
```

Übersicht über die Funktionen

Funktion f	**len(f)**	**rt(f)**
fib1	120	123.2470
fib2	88	116.3020
fib3	269	0.0029
fib4	193	0.0015
fib5	173	0.0007
fak1	82	0.0013
fak2	67	0.0013
fak3	89	0.0012
fak4	65	0.0057

len(f): Länge des Quellcodes von fin Zeichen
rt(f): Laufzeit (s) für 100 Ausführungen von **f(30)**

Äquivalenzklassen

Definition

Sei A eine Menge und $\approx$ eine Äquivalenzrelation. Sei $a \in A$.

- Die $\approx$- Äquivalenzklasse von a ist $[a]_\approx = \{b \in A | a \approx b\}$
- Anmerkung: Wenn die Äquivalenzrelation klar ist, witd $\approx$ oft weggelassen

Äquivalenzklassen (Beispiel)

Beispiel

Sei $A = \{fib1, fib2, fib3, fib3, fib4, fib5, fak1, fak2, fak3, fak4\}$ die Menge der oben definierten Funktionen

Sei $\approx_{eq} = \{(f, g) | f, g \in A, \forall x \in \mathbb{N} : f(x) = g(x)\}$

Es gibt 2 $\approx_{eq}$- Äquivalenzklassen:

- $[fak_1]_{\approx_{eq}} = \{fak1, fak2, fak3, fak4\}$
- $[fib_1]_{\approx_{eq}} = \{fib1, fib2, fib3, fib4, fib5\}$

Beachte: Wir können verschiedene Repräsentanten einer Äquivalenzklasse wählen:

- $[fak1] = [fak2] = [fak3] = [fak4]$

Komposition von Relationen

Definition

Seien $R \subseteq A \times B$ and $S \subseteq B \times C$ zwei Relationen.

- $R \circ S = \{(x, z) \in A \times C | \exists y \in B : (x, y) \in A \text{ and } (y, z) \in S\}$

Spezialfall: $A = B = C$

Beispiel:

Abgeleitete Relationen

Definition

Sei $\rightarrow \subseteq A \times A$ eine binäre Relation. Wir schreiben:

- $\rightarrow^{-1} = \{(b,a)|(a,b) \in \rightarrow\}$ (Inverse Relation)
- $\leftarrow = \rightarrow^{-1}$ (Inverse Relation, Alternative)
- $\overset{0}{\rightarrow} = \{(a,a)|a \in A\}$ (Identität)
- $\overset{i+1}{\rightarrow} = \overset{i}{\rightarrow} \circ \rightarrow$ (i+1- fache Iteration)
- $\overset{+}{\rightarrow} = \cup_{i \in \mathbb{N}_+} \overset{i}{\rightarrow}$ (transitiver Abschluß)
- $\overset{=}{\rightarrow} = \rightarrow \cup \overset{0}{\rightarrow}$ (reflexiver Abschluß)
- $\overset{*}{\rightarrow} = \overset{0}{\rightarrow} \cup \overset{+}{\rightarrow}$ (relexiver transitiver Abschluß)
- $\leftrightarrow = \rightarrow \cup \leftarrow$ (symmetrischer Abschluß)
- $\overset{+}{\leftrightarrow} = (\leftrightarrow)^+$ (transitiver symmetrischer Abschluß)
- $\overset{*}{\leftrightarrow} = (\leftrightarrow)^*$ (reflexiver, transitiver symmetrischer Abschluß)

Beispiel

Sei $A = \{a, b\}^*$ und $\rightarrow = \{(ubav, uabv) | u, v \in A\}$

- $ababba \rightarrow aabbba \rightarrow aabbab \rightarrow aababb \rightarrow aaabbb$
- $ababba \xrightarrow{0} ababba$
- $ababba \xrightarrow{1} aabbba$
- $ababba \xrightarrow{2} aabbab$
- $ababba \xrightarrow{3} aababb$
- $ababba \xrightarrow{4} aaabbb$
- $ababba \xrightarrow{+} aaabbb$
- $ababba \xrightarrow{*} aaabbb$

Generell: $w \xrightarrow{*} a^{|w|_a} b^{|w|_b}$ ($\rightarrow$ sortiert Worte)

Beispiel (fortgesetzt)

Sei $A = \{a, b\}^*$ und $\rightarrow = \{(ubav, uabv) | u, v \in A\}$

- $\leftarrow$ $(= \rightarrow^{-1})$ sortiert Worte umgekehrt:
  $ababba \leftarrow baabba \leftarrow bababa \leftarrow bbaaba \leftarrow bbabaa \leftarrow bbbaaa$
- $u \overset{*}{\leftrightarrow} v$ wenn u, v gleichviele a und b enthalten
- $\{[a^i b^j]) \overset{*}{\leftrightarrow} | i, j \in \mathbb{N}_0\}$ sind die Äquivalenzklassen von $\overset{*}{\leftrightarrow}$

Antisymmetrie, Antireflexivität

Definition

Sei $\rightarrow$ eine binäre Relation über A

$\rightarrow$ heißt antisymmetrisch, falls
$\forall a, b \in A : a \rightarrow b$ und $b \rightarrow a \implies a = b$

- Also: Verschiedene Elemente stehen höchstens in einer Richtung in Relation!

$\rightarrow$ heißt antireflexiv, falls $\forall a \in A : (a, a) \notin \rightarrow$

- Also: Es gibt kein Element, das zu sich selbst in Relation steht

Partialordnungen

Definition: Partialordnung

Eine binäre Relation $\geq$ über A ist eine Partialordnung, falls gilt:

- $\geq$ ist reflexiv
- $\geq$ ist antisymmetrisch
- $\geq$ ist transitiv

Der strikte Teil von $\geq$ ist $>=\geq \setminus \equiv$

- $>$ ist antireflexiv
- $>$ ist antisymmetrisch
- $>$ ist transitiv

Gelegentlich wird $>$ als (strikte) Partialordnung definiert und $\geq$ davon abgeleitet

Totalität

Definition

- Eine Partialordnung $\geq$ auf A heißt total, falls für alle $a, b \in A$ $a \geq b$ oder $b \geq a$ gilt
  - Alle Elemente sind vergleichbar
- Eine strikte Partialordnung $>$ auf A heißt total, falls $\overset{=}{>}$ total ist
  - Also: Der reflexive Abschluß von $>$ ist total
  - Alle Paare von verschiedenen Elementen sind bzgl $>$ vergleichbar

Partialordnungen

Beispiele

$\subseteq$ auf beliebiger Menge M ist eine Partialordnung

- $\subseteq$ ist nicht total: $\{1,2\} \not\subseteq \{2,3\}$ und $\{2,3\} \not\subseteq \{1,2\}$
- $\emptyset$ ist eindeutiges kleinstes Element bzg. $\subseteq$

Die lexikographische Ordnung auf Worten ist eine (totale) Partialordnung

- Vergleiche zeichenweise nach Vorordnung auf Zeichen
- $baa > aabb > aab > ab > a$

Die längenlexikographische Ordnung auf Worten ist eine (totale) Partialordnung

- Vergleiche Länge, bei gleicher Länge lexikographisch
- $aabb > baa > aab > ab > a$

Ordnungen auf Programmen

Beispiele

$A = \{fib1, fib2, fib3, fib3, fib4, fib5, fak1, fak2, fak3, fak4\}$
$\approx_{eq}$ wie oben.

- Definiere $s >_l t$ gdw. s ist länger als t
  $>_1 = \{(s,t) | len(s) > len(t)\}$
- Definiere $s >_r t$ gdw. s ist langsamer als t
  $>_r = \{(s,t) | rt(s) > rt(t)\}$
- Definiere $s >_{le} t$ gdw. s ist äquivalent, aber länger als t
  $>_{le} = \approx_{eq} \cap >_l$
- Definiere $s >_{re} t$ gdw. s ist äquivalent, aber langsamer als t
  $>_{re} = \approx_{eq} \cap >_r$

Erinerung: Eigenschaften der Funktionen

Funktion f	**len(f)**	**rt(f)**
fib1	120	123.2470
fib2	88	116.3020
fib3	269	0.0029
fib4	193	0.0015
fib5	173	0.0007
fak1	82	0.0013
fak2	67	0.0013
fak3	89	0.0012
fak4	65	0.0057

Falls Zeit: Bestimme eine/mehrere der Relationen an der Tafel!

Ordnungen auf Programmen (2)

$>_l$, $>_r$, $>_{le}$, $>_{re}$ sind strikte Partialordnungen auf A

$>_l$ ist total, $>_r$ ist nicht total

$>_{le}$ und $>_{re}$ sind nicht total

Anmerkung: Die Nicht- strikten Varianten der definierten Ordnungen auf A sind i.A. nur Quasi- Ordnungen auf beliebigen Mengen von Programmen!

Zusammenfassung

Definition Reduktionssystem

Eigenschaften von Relationen

Abschlüsse (reflexiv/transitiv/symmetrisch)

Äquivalenzrelationen

Partialordnungen

Aufgaben

Geben Sie eine Relation an, die weder symmetrisch noch antisymmetrisch ist

Geben Sie eine Relation an, die weder reflexiv noch antireflexiv ist

Gibt es Relationen, die symmetrisch und transitiv, aber nicht reflexiv sind?

Sei $\rightarrow$ eine Relation auf A. Zeigen oder widerlegen Sie:

- Reflexiver und transitiver Abschluß kommutieren: $(\overset{+}{\rightarrow})^{=} = (\overset{=}{\rightarrow})+)$
- Transitiver und symmetrischer Abschluß kommutieren: $\overset{+}{\leftrightarrow} = (\overset{+}{\rightarrow} \cup \overset{+}{\leftarrow})$