Qualitätssicherung von Software

Prof. Dr. Holger Schlingloff

Humboldt-Universität zu Berlin
und
Fraunhofer FIRST

Kapitel 2. Testverfahren

Lotos

- algebraische Spezifikationssprachen
- Syntax
  - Abstrakter Datentyp
  - Prozessalgebraische Verhaltensbeschreibung
- Beispiel: ADT Stack
  - Problematik partieller Funktionen
- Semantik
  - Termalgebren
  - Gleichungen implizieren Äquivalenzpartitionierung
  - initiale Semantik oder lose Semantik

weitere Möglichkeiten

- bedingte Gleichungen
- Parametrisierte Typen (abstrakte Klassen)
- Überladen von Funktionen (Polymorphie)
  - z.B. Gleichheit
  - **ofsort** zur Kennzeichnung des Typs
- Renaming und Subtypisierung
  - **type** B **is** A **renamedby sortnames** ... **for** ...

Full Lotos

- Erweiterung von ADT's um Verhaltensbeschreibungen
- Basiskomponente: Aktion
  - **interne Aktion:** nach außen nicht sichtbar
  - **beobachtbare Aktion:** Wert erscheint an einer Verbindungsstelle (*gate)*
    - *g!e:* Senden des Ausdrucks *e* über gate *g*
    - *g?x:s[c]:* Empfangen eines neuen Wertes vom Typ *s* für die Variable *x* am gate *g* unter der Bedingung *c*
  - intuitiv: die Verbindungsstellen übertragen Werte der jeweiligen abstrakten Datentypen

Prozesse

- Prozesse entsprechen in etwa den Methoden
  - **process** *P [...]* = ...
- drei Hauptkompositionsmöglichkeiten
  - Sequentialisierung: *(P>>>Q)* oder *(a; P)* (a ist eine Aktion)
  - Alternativen: *([c_1]-> P [] [c_2]-> Q]*
  - Parallelität: *(P /[g_1,...,g_n]/ Q)*

    (P // Q) und *(P /// Q)* als Abkürzungen für Synchronisation über alle oder gar kein gate
- Rekursive Prozessdefinitionen
  - **stop** als reguläres Ende (keine Aktion ausführbar)
  - **exit** als Rückkehr aus einer Prozessdefinition

Kommunikation, Synchronisation, Koordination

- *(P |[g]| Q)* kann
  - entweder eine Aktion von *P* oder *Q* ausführen, die *g* nicht betrifft, oder
  - eine gemeinsame Aktion über gate *g* ausführen, falls diese sowohl für *P* als auch für *Q* ausführbar ist
    - **Kommunikation:** *g!e* und *g?x:s[c]*
      Übertragung des Wertes *e* nach *x*, falls *c* erfüllt ist
    - **Synchronisation:** $g!e_1$ und $g!e_2$
      Falls $e_1 = e_2 = e$, so erscheint *e* an *g*
    - **Koordination:** $g?x_1:s[c_1]$ und $g?x_2:s[c_2]$
      An *g* erscheint irgendein Wert *e*, der c_1 und c_2 erfüllt

Semantik von Prozessen

- falls zwei parallele Prozesse sich nicht synchronisieren können, erfolgt deadlock
  - *g!5 |[g]| g?x:Nat [x>7]*
  - $g_1!5 \;|[g_1,g_2]|\; g_2?x:Nat$
- Ereignis (event) (g,e): Ausführung einer Aktion *g!e* oder *g?x* mit *x=e*
- traces(P): Menge aller Folgen beobachtbarer Ereignisse eines Prozesses
- Trace-, Failure-, Divergence- Semantiken

weitere Sprachkonstrukte

- Parametrisierte Prozesse
  process P[g_1,g_2](p_1:s_1, p_2:s_2) : **exit** = ... **endproc**
- Lokale Variablendefinition
  let *name : sorte = expr* **in** *...*
- Verallgemeinerte Sequenz, Alternative, Parallelität
  $expr_1$ >> **accept** *pardef* **in** *$expr_2$*
  choice *g* **in** *[a1, a2, a3] [ ] B [g]*
  par *g* **in** *[a1, a2, a3] || B [g]*
- Disabling, Hiding, lokale Prozesse, ...
  P [> *Q*, **hide** *g* **in** *P*, **where process** P = ...
- Modulkonzept
  library *importierte Datentypen* **endlib**

Beispiel für Prozesse

```
process Boss [in] : noexit =
    choice item : Nat_Sort [] in!item >> Boss [in] endproc

process ToDo [in, out] (liste: Stack) : noexit =
    (in?item; ToDo[in,out](push(item,liste))
       []
     [not empty(liste)] -> out!peek(liste); ToDo[in,out](pop(liste))
    )
    endproc

process Slave [out] : noexit =
    out? x; i >> Slave [out]
    endproc
```

Systemspezifikation: Boss |[in]| ToDo |[out]| Slave

Systemspezifikation

specification S [a, b, c, d] : **noexit**
library verwendete (vordefinierte) Datentypen **endlib**

type BeispielTyp **is**
  sorts BeispielSorten
  opns BeispielOperationen: BeispielSorten - > BeispielSorten
  endtype

behaviour
  (P [a, b, c] |[b]| Q [b, d])
where
  process P[a, b, c] ... **endproc**
  process Q[b, d] ... **endproc**
endspec

ein größeres Beispiel (1)

```
1  specification Example1 : exit
2
3  library Boolean, OctetString, NaturalNumber
4
5  type Message is
6    Octet, NaturalNumber, Boolean
7    sorts
8      Message
9    opns
10     ε :  -> Message
11     _ . _ :  Octet, Message-> Message
12     Pack :  Message, Message -> Message
13     Size :  Message -> Nat
14   eqns
15     forall m1, m2: Message, o1: Octet
16     ofsort Message
17       Pack(ε, m1) = m1;
18       Pack(b.m1, m2) = b.Pack(m1, m2);
19     ofsort Nat
20       Size(ε) = 0;
21       Size(o1.m1) = Succ(Size(m1));
22   endtype
```

Quelle:

M.-C. Gaudel, P. R. James.
Testing Algebraic Data Types and Processes: A Unifying Theory.
Formal Aspects of Computing, 10(5-6), (1999) Seite 436-451

ein größeres Beispiel (2)

```
24 where
25    process Compact[inGate, outGate, control](Max: Nat) : exit :=
26        control ? newMax:Nat[ newMax  > 0];
27         ( Compact[inGate, outGate, control](newMax)
28     [] control ? newMax:Nat[newMax = 0]; exit
29     [] inGate ? x:Message;
30          ( inGate ? y:Message[Size(x) + Size(y)>Max];
31            outGate ! x ! y; Compact[inGate, outGate, control](Max)
32          [] inGate ? y:Message[Size(x) + Size(y) <= Max];
33            outGate ! Pack(x, y);
34            Compact[inGate,outGate, control](Max)
35          )
36   endproc
37 endspec
```

Gegeben eine Implementierung *Imp* für diese Spezifikation *Spec*.
Wie ist diese zu testen?

Testerzeugung aus abstrakten Datentypen

- Gegeben ADT *Spec= (Σ, Eq)*

  (*Σ* = Signatur, *Eq* = Gleichungen oder Axiome)

  - Implementierung *Imp* ist korrekt bzgl. *Spec* falls alle Gleichungen für alle Terme erfüllt sind
- **Testfall:** Einsetzung von Termen für Variablen
  - Problem: Welche Terme auswählen?
- **Testauswertung:** Überprüfung der Gleichheit
  - Problem: Nichtprimitive Datentypen?

G. Bernot, M.-C. Gaudel, B. Marre: Software testing based on formal specifications: a theory and a tool. Software Engineering Journal Volume 6.6, pp. 387 - 405 (Nov. 1991)

vollständige Testsuiten

- **Testsuite T:** Menge von Grundformeln
  - Annahme: Jedes Objekt ist termerzeugt
  - Bsp.: s(s(s(z)))=p(s(z),s(s(z)))
- **Testorakel O** $\subseteq$ T (bzw. O: T→{true,false})
  - sagt zu jedem Testfall, ob er erfüllt ist oder nicht (aus den Axiomen folgt oder nicht)
  - z.B. s(s(s(z)))=p(s(z),s(s(z))) → true
  - im Allgemeinen unentscheidbares Problem! (partielle Fkt.)
- **vollständige Testsuite (exhaustive test set):** Menge von Testfällen, so dass gilt: Falls alle Testergebnisse positiv sind, so ist die Implementierung korrekt
  - im Allgemeinen unendlich groß
  - wie kommt man zu einer Approximation?

Testhypothesen

- **Testkontext TC:** (T, O, H)
  - Testsuite T (Menge von Grundformeln)
  - Testorakel O $\subseteq$ T (bzw. O: T→{true,false})
  - Testhypothese H über die Implementierung
- H $\wedge$ O=T → Correct(Imp,Spec)
  TC ist relativ vollständig („valid“): Falls die Testhypothese erfüllt ist und alle Testergebnisse positiv ausfallen, so ist die Implementierung korrekt
- *Minimale Testhypothese:* „leere Annahme“
  - Menge aller ableitbaren Grundformeln bzw. Menge aller Grundinstanzen von Gleichungen ist vollständige Testsuite
- *Maximale Testhypothese:* „Imp ist korrekt“
  - leere Menge ist vollständige Testsuite

Testsuiteverfeinerungen

- TC_2 verfeinert TC_1 ($TC_2 < TC_1$), falls
  - TC_2 macht stärkere Annahmen als TC_1
    $H_2 \rightarrow H_1$
  - TC_2 entdeckt mindestens so viele Fehler wie TC_1
    $failed(T_1,Imp) \rightarrow failed(T_2,Imp)$
  - TC_2 hat mehr erfolgreiche Tests als TC_1
    $success(T_2,Imp) \rightarrow success(T_1,Imp)$
- Menge aller Grundterme mit leerer Annahme ist größte Testsuite in dieser Halbordnung
- Testentwicklung = Hinzufügen von Annahmen

Regularitätsannahme

- Gegeben: Komplexitätsmaß auf Formeln
- **Regularitätsannahme:** Wenn eine Aussage *A* für alle Formeln bis zu einer bestimmten Größe δ gilt, so gilt *A* für alle Formeln
- erlaubt, die Testfälle auf solche kleiner als δ einzuschränken
- z.B. p(x,y)=p(y,x) für |x|<3, |y|<3

Uniformitätsannahme

- Gegeben: Eigenschaft von Ausdrücken
- **Uniformitätsannahme:** Wenn eine Aussage für alle Formeln mit Ausdrücken dieser Eigenschaft gilt, so gilt sie für alle Formeln
- Verallgemeinerung des vorherigen
- Erlaubt, Testfälle auf bestimmte Variablenmuster einzuschränken

- Anwendung: Partitionierung von Wertebereichen
- Extremfall: Kollabieren eines ganzen Teilbereichs durch Auswahl eines einzelnen Repräsentanten

Beobachtbarkeit

- Gleichheit primitiver Daten (bool, integer,…) ist *beobachtbar*
- Problem, die Gleichheit nichtprimitiver Datentypen zu entscheiden
  - spezielle Gleichheitsfunktion in *Imp* einbauen? ➔ verlagert das Problem nur
  - komponentenweiser Vergleich: Ersetze x=y durch $C_1(x)=C_1(y)$, $C_2(x)=C_2(y)$, …
- **Beobachtbarer Kontext:** Abbildung von nichtprimitivem in primitiven Datentyp
- **Leibnitz‘sches Extensionalitätsprinzip:** zwei Werte sind gleich, wenn sie sich in jedem beobachtbaren Kontext gleich verhalten

Beobachtungsäquivalenzannahmen

- Problem: „sehr viele“ mögliche Kontexte
- Lege Satz von Beobachtungskontexten für jeden nichtprimitiven Datentyp fest
- **Beobachtungsäquivalenzannahme:** Wenn eine Aussage für alle derart beobachtbaren Kontexte gilt, so gilt sie für alle Formeln

- spezielle Uniformitätsannahme
- Erlaubt Reduktion auf primitive Vergleiche

- Beispiel: oberstes und zweites Stackelement, Hash o.ä.

Methodik der Verfeinerung

- Ausgehend von der größten Testsuite
  - Regularitätshypothesen für die definierten Typen
  - Uniformitätshypothesen für die importierten Typen
  - Beobachtungsfunktionen und -äquivalenzhypothesen für die zusammengesetzten Typen
- Solange bis Testsuite endlich und Orakel vollständig definiert
- Toolunterstützung möglich

im Beispiel

$$
\begin{aligned}
exhaust_{ax1} &= \{\mathrm{Pack}(\varepsilon,\ \mathrm{m}) = \mathrm{m} \mid \mathrm{m} \in \mathrm{T}_{Message}\} \\
exhaust_{ax2} &= \{\mathrm{Pack}(\mathrm{o}_1.\mathrm{m}_1,\ \mathrm{m}) = \mathrm{o}_1.\mathrm{Pack}(\mathrm{m}_1,\ \mathrm{m}) \mid \mathrm{o}_1 \in \mathrm{T}_{Octet},\ \mathrm{m}_1, \\
&\qquad \mathrm{m} \in \mathrm{T}_{Message}\} \\
exhaust_{ax3} &= \{\mathrm{Size}(\varepsilon) = 0\} \\
exhaust_{ax4} &= \{\mathrm{Size}(\mathrm{o}_1.\mathrm{m}) = \mathrm{Succ}(\mathrm{Size}(\mathrm{m})) \mid \mathrm{o}_1 \in \mathrm{T}_{Octet},\ \mathrm{m} \in \mathrm{T}_{Message}\}
\end{aligned}
$$

- Uniformitätsannahme „alle Variablen gleich“ ergibt 4 Testfälle
- „Entfaltung“ von *Pack* ergibt neue Testfälle

$$
\begin{aligned}
\mathrm{Pack}(\mathrm{o}_1.\varepsilon,\ \mathrm{m}) &= \mathrm{o}_1.\mathrm{m} \\
\mathrm{Pack}(\mathrm{o}_1.\mathrm{o}_1'.\mathrm{m}_1',\ \mathrm{m}) &= \mathrm{o}_1.\mathrm{o}_1'.\mathrm{Pack}(\mathrm{m}_1',\ \mathrm{m})
\end{aligned}
$$

Pause?


HOW SANTA REALLY FINDS OUT WHICH REINDEER CAN FLY.
DANG! LET'S TRY ANOTHER ONE.
www.funny-city.com