Qualitätssicherung von Software

Prof. Dr. Holger Schlingloff

Humboldt-Universität zu Berlin
und
Fraunhofer FIRST

Test des Prozessteils

- Für ADT
  - Korrektheit von Imp bzgl. Spec = alle Gleichungen / Formeln von Spec sind durch Imp erfüllt
- Für PA?
  - Korrektheit definiert durch Gleichheit des beobachtbaren Verhaltens
  - Simulation bzw. Enthaltensein des Verhaltens
  - mehrere mögliche Definitionen

Literatur: J. Tretmans, Conformance Testing with Labelled Transition Systems: Implementation Relations and Test Generation. Report, Univ. Twente, Nov. 1999

trace-Verfeinerung

- Traces(P) = Menge der Folgen σ von beobachtbaren Aktionen eines Prozesses P
- $Imp \leq_{\mathbf{T}} Spec$ gdw. Traces(Imp) $\subseteq$ Traces($Spec$)

- Prä- oder Halbordnung (transitiv, reflexiv)
- Minimalelement: {stop}
- vergleichbar: Sprachenthaltung bei Automaten

trace-Verfeinerungstest

- **Testfall** = Trace
- **Testsuite** = Menge von Traces
- **Testdurchführung** eines Tests σ für *Imp* und *Spec*:
  - $\sigma \notin$ Traces(*Imp*) ➔ pass
  - $\sigma \in$ Traces(*Imp*) $\cap$ Traces(*Spec*) ➔ pass
  - fail, sonst
- Verdikt einer Testsuite als Konjunktion der Einzelergebnisse

- **vollständige Testsuite:** Menge aller Traces (unendlich)
- *Imp* $\leq_{\mathbf{T}}$ *Spec gdw.* vollständige Testsuite erfolgreich
- nicht praktikabel, daher zusätzliche Testhypothesen üblich (Länge der Traces, Anzahl bestimmter Aktionen usw.)

failures

- Failure = (σ, A), wobei σ Folge von beobachtbaren Aktionen, A Menge von Aktionen
- Failures(P) = Menge von Failures (σ, A), so dass es eine Ausführung von P gibt, bei der σ beobachtbar ist, so dass danach keine Aktion aus A aktiviert ist
  (P **after** σ **refuses** A)
- Bei Automaten: „nicht-Übergänge"

- $Imp \leq_F Spec$ gdw. Failures(Imp) $\subseteq$ Failures($Spec$)

- ebenfalls eine Halbordnung
- feiner als Trace-Refinement:

$$Imp \leq_F Spec \rightarrow Imp \leq_T Spec$$

Failure-Refinement

- *Imp* $\leq_F$ *Spec* gdw. Failures(*Imp*) $\subseteq$ Failures(*Spec*)
- *Imp* $\leq_F$ *Spec* gdw. (*Imp* **after** σ **refuses** A) impliziert (*Spec* **after** σ **refuses** A)
  - *Imp* darf nur solche Aktionen verweigern, die auch *Spec* verweigert
  - *Imp* kann nur Aktionsfolgen ausführen, die *Spec* erlaubt
  - *Imp* enthält „weniger deadlocks" als *Spec*
- Verfeinerung bezüglich dieser Relation
  - transformationelle Entwicklung
  - Korrektheitsbeweise

Test von Failure-Refinement

- Testsuite T = Menge von Failures (σ, A)
- vollständige Testsuite = Menge aller Failures für eine Menge beobachtbarer Ereignisse
- Verdikt einer Tests (σ, A) bezüglich *Imp* und *Spec*
  - $\sigma \notin$ Traces(Imp) ➔ pass
  - (σ,a) $\in$ Traces(Imp) für ein a $\in$ A ➔ pass
  - (σ,A) $\in$ Failures(Imp) $\cap$ Failures(Spec) ➔ pass
  - fail, sonst
- Verdikt einer Testsuite: alle Testfälle passieren
- Test einer Implementierung Imp
  Imp $\leq_{\mathbf{F}}$ *Spec* gdw. vollständige Testsuite ergibt pass
  - gilt nur unter bestimmten Voraussetzungen

Konformanz

- *Imp* **conf** *Spec* gdw. für alle σ in Traces(*Spec*): (*Imp* **after** σ **refuses** A) → (*Spec* **after** σ **refuses** A)
  - für Aktionsfolgen der Spezifikation wie $\leq_F$
  - *Imp* darf „zusätzliche Funktionalität" implementieren
  - schwächer als $\leq_F$ (d.h. *Imp* $\leq_F$ *Spec* → *Imp* **conf** *Spec*)
- Konformanztest ähnlich wie Failure-Refinement-Test definiert und durchgeführt
- als Korrektheitskriterium weithin gebräuchlich
- Konformanzanalyse als Forschungsthema

IOCO

- Berücksichtigung von Ein- und Ausgaben

  out(P **after** σ)={a! | P kann σ ausführen und danach ein a! ausgeben}
- *Imp* **ioco** *Spec* gdw. für alle σ in Traces(*Spec)*: (out(*Imp* **after** σ) $\subseteq$ out(*Spec* **after** σ)
- Idee
  - *Imp* ist bezüglich spezifizierter Eingaben deterministischer als *Spec*
  - *Imp* darf für nichtspezifizierte Eingaben zusätzliche Funktionalität implementieren

Testgenerierung für Full Lotos

- je eine Testsuite für Daten- und Prozessteil
- Verwendung der Datentyp-Eigenschaften im Test der Prozesse
  - Beispiel: Aufspaltung von <= in < und =
  - Beispiel: Size(ε)=0, Size(o.m)=… ergibt vier Testfälle für den Prozessteil [Size(x)+Size(y)<Max] ->…
  - Berechnung von Grenzwerten aus Gleichungen
  - Forschungsbedarf!

Einige Testfälle für das Beispiel

Teste *Compact(7)*:

- control!4; ...
- control!0; ...
- inGate!H.E.L.L.O.ε ; inGate!W.O.R.L.D.ε ; outGate!H.E.L.L.O.ε!W.O.R.L.D.ε ; ...
- inGate!H.E.L.ε ; inGate!W.O.ε ; outGate!H.E.L.W.O.ε ; ...

Uniformitätsannahmen und Repräsentanten
für Max, newMax,
sowie Size(x)+Size(y)>Max

systematische Ableitung durch Algorithmus?

Nichtimplementierbare Spezifikation