Einführung in Alloy

Nils Asmussen

Institut für SoftwareArchitektur

08.12.2010

Inhaltsverzeichnis

1. Einleitung
2. Grundlegende Sprachelemente
3. Entwicklung eines Modells
4. Dynamik

Motivation

Probleme bei der Softwareentwicklung

- Software entwickeln heißt Abstraktionen finden
- Die Praxis zeigt: Oft gelingt das in der Designphase nicht
- Erst bei der Implementierung werden Probleme offensichtlich

Grund

Man wird in der Designphase nicht zu Präzision gezwungen, sondern erst bei der Implementierung

Mögliche „Lösung“

Einsatz von Alloy während der Designphase um Abstraktionen zu finden und zu analysieren.

Alloy

Was ist Alloy?

- Alloy besteht aus einer Sprache und dem *Alloy Analyzer*
- Grundprinzip: Beschreibung eines Modells, Alloy Analyzer erzeugt konkrete Welten, die laut der Beschreibung gültig sind oder gibt Welten an, die Behauptungen verletzen
- Sprache: Mix aus Prädikatenlogik und relationaler Logik mit objektorientiertem Touch

Besonderheiten

- Es wird nicht die Korrektheit eines Systems bewiesen
- Es werden Welten selbstgewählter Größe überprüft
  Small scope hypothesis: Fehler treten meist in kleinen Welten auf
- OO Sichtweise macht die Entwicklung relativ einfach
- => Leichtgewichtige Modellierungssprache

Relationen

Grundlegendes

- Alles in Alloy ist eine Relation
- Unäre Relation mit einem Element = Skalar
- Unäre Relation = Menge

Beispiele

```
myName = {(N0)}
Name   = {(N0),(N1),(N2)}
names  = {(B0,N0),(B0,N1),(B1,N2)}
addrs  = {(B0,N0,A0),(B0,N0,A1),(B1,N2,A2)}
```

Definition von Mengen: Signaturen

Mengen

```
sig Person {}
```

Disjunkte Teilmengen

```
sig Person {}
sig Man extends Person {}
sig Woman extends Person {}
```

Partitioniert in disjunkte Teilmengen

```
abstract sig Person {}
sig Man extends Person {}
sig Woman extends Person {}
```

Definition von Relationen: Felder

Eine binäre Relation

```
sig Person {
  father: Person
}
```

Eine ternäre Relation

```
sig Name, Address {}
sig Book {
  entries: Name -> Address
}
```

Bedeutung

```
father:  Person -> Person
entries: Book -> Name -> Address
```

Kartesisches Produkt

Beispiel

```
Name      = {(N0),(N1)}
Address   = {(A0),(A1)}
Book      = {(B0)}

Name -> Address = {(N0,A0),(N0,A1),
                   (N1,A0),(N1,A1)}
Book -> Name -> Address = {(B0,N0,A0),
                           (B0,N0,A1),
                           (B0,N1,A0),
                           (B0,N1,A1)}
```

Join

Definition: *dot join*

Sind R und S Relationen des Grades n bzw. m (beide > 0), dann ist

$$R.S = \{(r_1, r_2, \ldots, r_{n-1}, s_2, s_3, \ldots, s_m) | \\ (r_1, \ldots, r_n) \in R \text{ und } (s_1, \ldots, s_m) \in S \text{ mit } r_n = s_1\}$$

Beispiele

```
myName   = {(N0)}
Name     = {(N0),(N1),(N2),(N3)}
Address  = {(A0),(A1)}
addrs    = {(N0,A0),(N1,A0),(N2,A1)}

Name.addrs   = {(A0),(A1)}
myName.addrs = {(A0)}
```

Join

Definition: *box join*

```
S.T[R] = R.(S.T)
```

Beispiel

```
sig Name, Address {}
sig Book {
  entries: Name -> Address
}

Book = {(B0)}
// (Name -> Address) aus B0
B0.entries
// Adresse fuer myName aus B0
B0.entries[myName] = myName.(B0.entries)
```

Transitiver Abschluss

Definition

Eine binäre Relation `R` ist transitiv, wenn aus `(r,s) in R` und `(s,t) in R` stets folgt, dass `(r,t) in R` gilt. Der transitive Abschluss einer binären Relation `R` ist die kleinste Relation, die `R` enthält und transitiv ist.

Beispiel

```
sig Person {
  father: Person
}

Person = {(P0),(P1),(P2)}
father = {(P0,P1),(P1,P2)}
^father = {(P0,P1),(P1,P2),(P0,P2)}
```

Mengenoperationen

Definition

- `+` Vereinigung
- `&` Schnittmenge
- `-` Differenz
- `in` Teilmenge
- `=` Gleichheit

Beispiele

```
{(N0)} & {(N0),(N1)} = {(N0)}
{(N0,A0),(N1,A1)} - {(N0,A0)} = {(N1,A1)}
{(N0),(N1)} in {(N0)} = false
{(N0)} in {(N0),(N1)} = true
```

Quantoren

Definition

- `all x: M | e` Für alle x aus M gilt e
- `some x: M | e` Es existiert ein x aus M, für welches e gilt
- `no x: M | e` Es gibt kein x aus M, ...
- `lone x: M | e` Es gibt höchstens ein x aus M, ...
- `one x: M | e` Es gibt genau ein x aus M, ...

Beispiele

```
all m: Man | m in Person
some m: Man | m.father = m
```

Kardinalitäten und logische Operatoren

Kardinalitäten

- `#r` Die Zahl der Tupel in r
- `0,1,...` Literale für ganze Zahlen
- `+,-` Addition und Substraktion
- `=,<,>,<=,>=` Vergleichsoperatoren

Logische Operatoren

- `not` Negation
- `and,or` Logisches UND/ODER
- `implies` Implikation
- `else` Alternative
- `iff` Äquivalenz

Kardinalitäten und logische Operatoren

Beispiel

```
abstract sig Person {}
sig Man extends Person {
  wife: lone Woman
}
sig Woman extends Person {
  husband: lone Man
}

all m: Man | #m.wife > 0
all m: Man, w: Woman |
  m.wife = w iff w.husband = m
```

Funktionen und Prädikate

Definition

- Funktionen sind parametrisierte Ausdrücke, die Relationen liefern
- Prädikate sind parametrisierte Formeln, die einen Wahrheitswert haben

Beispiele

```
fun lookup [b: Book, n: Name] : Address {
  b.entries[n]
}

pred contains [b: Book, n: Name, a: Address] {
  (n -> a) in b.entries
}
```

Fakten

Definition

Fakten sind Bedingungen, die in jeder Welt gelten müssen, die der Alloy Analyzer erzeugen soll

Beispiel

```
fact {
  all n: Name, a: Address |
    (some b: Book | contains[b,n,a])
}
```

Ausdrucksmöglichkeiten

Relational

```
pred contains [b: Book, n: Name, a: Address] {
  (b -> n -> a) in entries
}
```

Prädikatenlogik

```
pred contains [b: Book, n: Name, a: Address] {
  some a': b.entries[n] | a' = a
}
```

Dynamik

Problem

Die erzeugten Welten sind grundsätzlich unveränderlich => Dynamik lässt sich in Alloy nicht direkt ausdrücken

Ansätze

Operationsfokussiert Einführung einer Zeit-Komponente in die veränderlichen Relationen und Analyse der Operationen

Ablauffokussiert Arbeit mit verschiedenen Ereignissen (anstelle der Operationen) und Analyse von gültigen Reihenfolgen

Operationsfokussiert: Das Modell

Beispielprojekt

- Modellierung eines Flughafens
- Operationen: Buchen und Stornieren von Flügen

Die Relationen

```
sig Passenger {}
sig Flight {}
sig Time {
  passengers: Flight -> Passenger
}
```

Operationsfokussiert: Operationen

Die Operation „Buchen“

```
pred bookFlight[f: Flight, p: Passenger, t,t’: Time] {
  p not in t.passengers[f]
  t’.passengers = t.passengers + (f -> p)
}
```

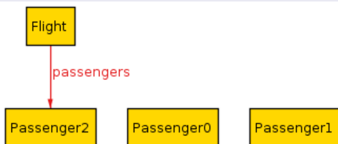
Fragestellungen

- Bucht *bookFlight* so wie vorgesehen? Gibt es Seiteneffekte?
- Kombinierung mit Stornierung: Macht eine Stornierung eine Buchung rückgängig?
- ...

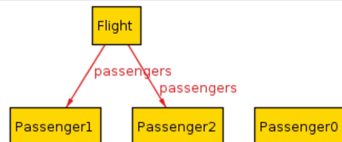
Operationsfokussiert: Visualisierung

Time0

Time1



Time2



Time3

Ablauffokussiert: Das Modell

Die Relationen

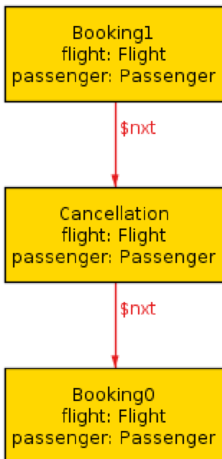
```
sig Passenger {}
sig Flight {}
abstract sig Event {
  flight: Flight
  passenger: Passenger
}
sig Booking, Cancellation extends Event {}
```

Fragestellungen

- Kann ein Flug von einem Passagier häufiger storniert als gebucht worden sein?
- Gibt es zu jeder Stornierung eine vorherige Buchung?
- ...

Ablauffokussiert: Visualisierung

\$nxt: 2



Ende

Fragen?