Überblick über die formale Spezifikation

Dennis Walter
Deutsches Forschungszentrum für Künstliche Intelligenz, Bremen

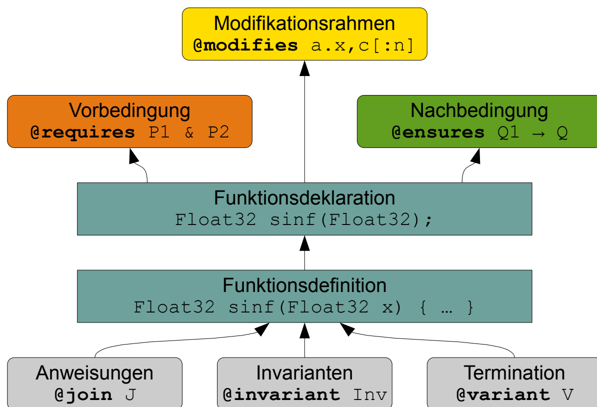
Universität Bremen, Cartesium, 13.10.09

Übersicht

SAMS

- Verifikationspraxis: Wie wird spezifiziert?
  - Relevante Sprachkonstrukte
  - Integration von Programmdaten und Domänenkonzepten
  - Programmierrichtlinien für beweisbaren Code
- Konkretes Beispiel aus dem SAMS-Code
  - Berechnung der Approximationspunkte für das Schutzfeld

Spezifikationen – Strukturell



Beispiel 1: Transformation

Deklaration

```
int transformiere(Punkt * v,
                  Punkt * v_res, int len,
                  Matrix * m);
```

Informelle Spezifikation

- Punkte aus *v* werden mittels Matrix *m* transformiert
- Ergebnis wird in *v_res* geschrieben
- Rückgabe: Anzahl transformierter Punkte

Beispiel 1: Spezifikation

```
/*@
  @requires \separated(v, len, v_res, len)
    && $!istSKT(m)
  @modifies v_res[:len]
  @ensures 0 <= \result <= len &&
    ${ ^PSet{v_res, \result} =
       ^SKT{m} ` ^PSet{v, \result} }
 @*/

int transformiere(Punkt * v, Punkt * v_res,
                  int len, Matrix * m);
```

Beispiel 2: Komposition

SAMS

Deklaration

```
void verkette_transformationen(
        const StarrkoerperTransformation * a2b,
        const StarrkoerperTransformation * b2c,
        StarrkoerperTransformation * a2c );
```

Informelle Spezifikation

- Eingabetransformationen *a2b* und *b2c* werden komponiert (Matrixmultiplikation)
- Ergebnistransformation wird in *a2c* abgelegt
- *a2b* und *b2c* bleiben unverändert

Beispiel 2: Spezifikation

SAMS

```
/*@
  @requires $!ist_SKT(a2b) && $!ist_SKT(b2c) &&
    \valid(a2b) && \valid(b2c) && \valid(a2c) &&
   a2b !=a2c && b2c != a2c
  @ensures $!ist_SKT(a2c) &&
    ${ ^SKT{*a2c} = ^SKT{*b2c} o ^SKT{*a2b} }
  @assigns *a2c
  @*/
void verkette_transformationen(
    const StarrkoerperTransformation * a2b,
    const StarrkoerperTransformation * b2c,
    StarrkoerperTransformation * a2c );
```

Programmdaten $\Longrightarrow$ Domänenelemente

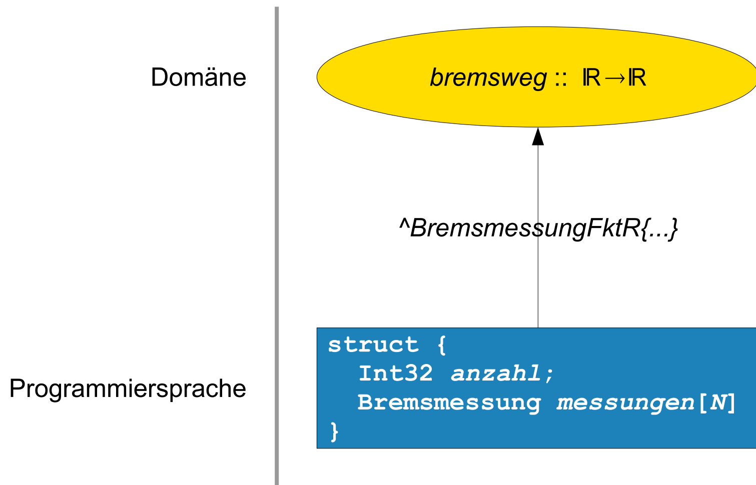
SAMS

- Spezifikation beinhaltet Programmdaten *und* Domänenkonzepte
- Interpretation der Programmdaten (Zustandsmodell) in Domäne notwendig
- Über Abstraktionsfunktionen realisiert

Abstraktionsfunktionen

- In Isabelle definierte Funktionen
- Eingabe: Programmdaten (Arrays, Zahlwerte, Strukturen)
- Resultatwert: Datentyp (Konzept) der Domäne
- Beispiel: `Vektor2D[N]` abstrahiert zu $\mathbb{R}^2$ *set*

Programmdaten ⟹ Domänenelemente



Abstraktionsfunktionen

SAMS

Definition (^function)

```
^function {
  _Any Vektor2DMenge(Vektor2D* vs, int n);
}
```

Verwendung

```
@ensures ${
    ^Vektor2DMenge{ausgabe, laenge} \<subset>
      ^Vektor2DMenge{eingabe, k}
 }
```

Definition häufig verwendeter Prädikate

SAMS

Definition (::abbreviation)

```
::abbreviation {
  _Bool einstellungen_OK() =
    konfiguration.einstellungen.t_r >= 0 &&
    konfiguration.einstellungen.e_scanner >= 0 &&
    konfiguration.einstellungen.e_radius >= 0;
}
```

Verwendung

```
@requires ::einstellungen_OK() && ...
```

Technische Spezifikationen

SAMS

Speicherlayout

- **separated**(a, i, b, j)
- **array**(v, len), **valid**(p)
- \unrelated(p, q)

Schleifen

- Invarianten (etwa für **for** (i = 0; i < n; ++i))
  - 0 <= i <= n && (**forall int** j; j < i --> P(j))
- Termination durch Verringerung einer Variante
  - len − i

Technische Spezifikationen (Forts.)

SAMS

C-Präprozessor & Konstanten

- Symbolische Behandlung von Konstanten in Beweisen
- @define **int** STRAHLENANZAHL = 529;
  #define STRAHLENANZAHL 529

Wertebereiche

- Spezifikation formalisiert Einschränkungen / Anforderungen aus Detailspezifikation:
  - ```
    ::abbreviation { _Bool zustand_OK() =
      konfiguration.v_max = 95 &&
      konfiguration.anz_scanner >= 1 &&
      warnlicht = \if schleichfahrt \then ON \else OFF;
    }
    ```
    ```

(Mikro-)Modulare Programmierung

SAMS

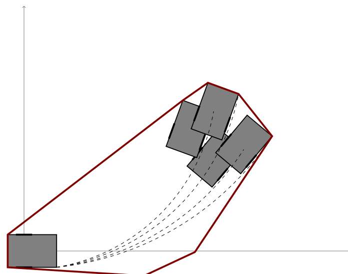
- Motto: *Kein Stück Code ist zu klein, um ausgelagert zu werden*
- Beweisaufwand als Code-Metrik!
- Beispiel: `for`-Schleife. Zu zeigen:
  - Termination: $i < len \Rightarrow len - (i+1) < len - i$
  - Erhalt der Invariante: $(\forall j < i.P(j)) \land P(i) \Rightarrow (\forall j \leq i.P(j))$
  - Invariante $\land \neg$ Schleifenbedingung $\Rightarrow$ Spezifikation
- Bei Auslagerung in Funktion:
  - Alle Anforderungen in Vorbedingung gekapselt
  - Spezifikation direkt als Nachbedingung verfügbar

▸ Beispiel

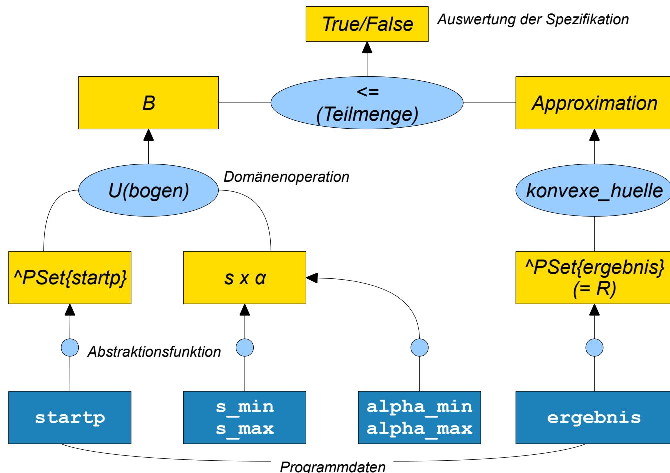
Beispiel aus dem SAMS-Code

```
SAMSStatus punktmenge_berechnen(
            Laenge s_min, ...
            WinkelRad alpha_min, ...
            Vektor2D * startp, ...
            Vektor2D * ergebnis, ... )
```

- "Bremsbögen" der Konturpunkte (in *startp*) sollen für Kombinationen aus Bremskonfigurationen (s, α) in der konvexen Hülle der Resultatpunkte (in *ergebnis*) liegen



Datenflusssicht einer Spezifikation



Konkrete Spezifikation

SAMS

```
/*@
  @requires s_min <= s_max && alpha_min <= alpha_max
    && \separated(startpunkte_daten, startpunkte_laenge,
                  ergebnis_daten, ergebnis_laenge_max)
    && \unrelated(ergebnis_daten, &sams_andere)
    && \unrelated(startpunkte_daten, &sams_andere)
    && 4 < l && (4 * l + 5) * startpunkte_laenge <= ergebnis_laenge_max

  @modifies sams_andere, ergebnis_daten[:(4 * l + 5) * startpunkte_laenge], *ergebnis_laenge

  @ensures \result == sams_sicher -->
    *ergebnis_laenge == (4 * l + 5) * startpunkte_laenge &&
    ${ let S = ^Vektor2DMenge{startpunkte_daten, startpunkte_laenge};
           B = \<Union> x \<in> S. \<Union> s \<in> {'s_min, 's_max}.
               \<Union> a \<in> {'alpha_min, 'alpha_max}. bogen s a x;
           R = ^Vektor2DMenge{ergebnis_daten, *ergebnis_laenge}
       in
         B \<subseteq> konvexe_huelle R
     }
  @*/

SAMSStatus punktmenge_berechnen( Laenge s_min, Laenge s_max,
                                 WinkelRad alpha_min, WinkelRad alpha_max,
                                 Int32 l,
                                 const Vektor2D * startpunkte_daten,
                                 Int32 startpunkte_laenge,
                                 Vektor2D * ergebnis_daten,
                                 Int32 ergebnis_laenge_max,
                                 Int32 * ergebnis_laenge );
```

Vielen Dank!