18.10.05

Theorie der Programmierung I

(Mitschrift von Lars Friedrich: email@lars-friedrich-home.de)

Inhalt: Semantik von Programmiersprachen
Genauer: Methoden zur Beschreibung der Semantik

Unter Semantik verstehen wir: Alles, was über die kontextfreie Syntax hinausgeht, d.h.

1) die **Kontextbedingungen** (= „statische Semantik“), z.B.
   - ein Name muss deklariert sein, „bevor“ er benutzt wird
   - eine Funktion (Prozedur, Methode) darf nur mit Parametern vom „richtigen Typ“ aufgerufen werden
2) die Beschreibung des Laufzeitverhaltens (eigentliche Semantik oder „dynamische Semantik“)

Vergleiche die Phasen eines Comilers:

1) gehört noch zur Compile-Zeit („semantische Analyse“)
2) ist die Laufzeit

Vorgehensweise:

Methoden werden an einer „idealisierten“ Programmiersprache eingeübt. Diese Sprache wird Schritt für Schritt eingeführt:

1)	ungetypte funktionale Sprache	TP I
2)	einfaches Typsystem	TP I
3)	polymorphes Typsystem (ML)	TP I
4)	imperative Konstrukte	TP II
5)	objektorientierte Konstrukte	TP II

Über die Semantikbeschreibung hinaus werden **Eigenschaften** der Programmiersprache formuliert und bewiesen, z.B. **Typsicherheit**.
„Wohlgetypte Programme können nicht stecken bleiben“, d.h. wenn ein Programm vom Compiler als wohlgetypt akzeptiert wird, dann können zur Laufzeit keine Typfehler mehr auftreten (also ist zur Laufzeit keine Typüberprüfung mehr nötig).

Eine ungetypte funktionale Sprache

Vorgegeben seien:

- eine unendliche Menge Id von **Namen** (Identifier) id (genaue Definition unwichtig, z.B. Id=Menge aller Wörter über {a-z, 0-9}, die mit einem Buchstaben beginnen
- die Menge Int aller (Darstellung von) **ganzen Zahlen** (Integers) n
- die Menge Bool = {true, false} der **boolschen Werte** b

Darauf aufbauend sei die (kontextfreie) Syntax definiert:

- die Menge Op der **binären Operationen** op ist definiert durch:
  - op ::= + | - | * | / | mod | (arithmetische Operatoren)
  - < | > | ≤ | ≥ | = (Vergleichsoperatoren)
- die Menge Const der **Konstanten** c durch:
  - c ::= () unit-Element
  - | b boolsche Werte
  - | n ganze Zahl
  - | op Operator

- die Menge Exp der **Ausdrücke** (Expressions) e durch:
  - e ::= c — Konstante
  - | id — Name
  - | $e_1 e_2$ — Applikation
  - | λ id.e_1 — λ-Abstraktion
  - | if e_0 then e_1 else e_2 — bedingter Ausdruck
  - | let id = e_1 in e_2 — let-Ausdruck

Diese kontextfreie Grammatik (KFG) ist als Definition einer so genannten „abstrakten Syntax“ aufzufassen, d.h. es werden durch die KFG keine Zeichenreihen definiert, sondern bereits Syntaxbäume. Mit anderen Worten: wir nehmen an, dass die Syntaxanalyse (Parser) bereits stattgefunden hat.

Vorteil: Syntaktische Details wie Klammerung und Prioritäten von Infixoperatoren können in der abstrakten Syntax ignoriert werden.

Für Beispiele: Einige Worte zur konkreten Syntax: Die Applikation hat höchste Priorität (d.h. sie bindet stärker als λ, if_then_else_ oder let_in_) und ist linksassoziativ, d.h. aufeinander folgende Applikationen gelten als „linksgeklammert“: $e_1 e_2 e_3$ steht für $(e_1 e_2) e_3$, + 1 2 steht für (+ 1) 2

Beachte: Wenn eine λ-Abstraktion als linker oder rechter Teil einer Applikation auftritt, muss sie in Klammern stehen, z.B. (λx. + x 1) 2

Die so definierte Sprache (mit Startsymbol e) bezeichnen wir mit $\mathcal{L}_1$. Mitunter betrachten wir eine Teilsprache $\mathcal{L}_0 \subseteq \mathcal{L}_1$, definiert durch:
e ::= id | $e_1 e_2$ | λid. e_1 ($\mathcal{L}_0$ besitzt also keine Konstanten)

$\mathcal{L}_0$ heißt **reiner** (ungetypter) λ-Kalkül. Eine Sprache wie $\mathcal{L}_1$, die aus $\mathcal{L}_0$ durch Hinzunahme einiger Konstanten entsteht, heißt **angewandter** (ungetypter) λ-Kalkül.

Operationale Semantik

Zunächst: small step-Semantik

Idee: Ein Ausdruck wird solange umgeformt, („vereinfacht“) bis man einen Wert erhält.

zu klären: Was versteht man unter einem Wert?
Wie sehen die einzelnen Umformungsschritte aus?

Vorgegeben: Eine Menge Exn von Exeptions exn,
z.B. Exn {Division_by_zero, Overflow, …}

Definition: Ein small step ist eine „Formel“ der Gestalt e→e’ oder e→exn.

Ziel: Genau festlegen, welche small steps erlaubt sind.