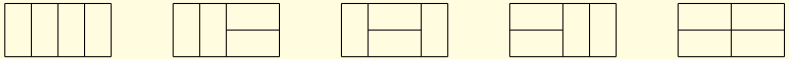
Teil III

Werte

10. Knobelaufgabe #3

Bob der Baumeister will eine 100 m breite und 2 m hohe Mauer bauen. Als Baumaterial hat er 2 m breite und 1 m hohe Quader zur Verfügung.

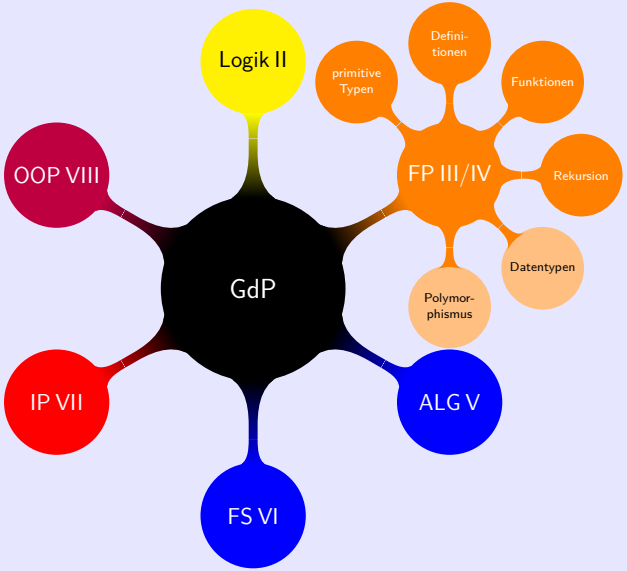
Wieviele Möglichkeiten gibt es die Mauer zu konstruieren?



Eine 4 m breite Mauer lässt sich zum Beispiel auf 5 Arten zusammensetzen.

Wie erhöht sich die Zahl der Möglichkeiten, wenn die Mauer statt 2 m sogar 3 m hoch werden soll?

10. Gliederung



10. Lernziele

Nach Durcharbeitung dieses Kapitels sollten Sie

- den Unterschied zwischen statischer und dynamischer Semantik kennen,
- Sinn und Zweck von Typsystemen verstanden haben,
- das Konzept von Bindungen verstanden haben,
- Funktionen und Rekursion verstanden haben,
- Entwurfsmuster kennen und anwenden können,
- Mini–F# Programme lesen und selbst schreiben können.

10. Semantik

Semantics is a strange kind of applied mathematics; it seeks profound definitions rather than difficult theorems. The mathematical concepts which are relevant are immediately relevant. Without any long chains of reasoning, the application of such concepts directly reveals regularity in linguistic behaviour, and strengthens and objectifies our intuitions of simplicity and uniformity.

— J.C. Reynolds (1980)

10. Growing a language ...

I think you know what a man is. A woman is more or less like a man, but not of the same sex. (This may seem like a strange thing for me to start with, but soon you will see why.)

Next, I shall say that a person is a woman or a man (young or old).

To keep things short, when I say "he" I mean "he or she", and when I say "his" I mean "his or her."

A machine is a thing that can do a task with no help, or not much help, from a person.

— Growing a Language, Guy L. Steele Jr.

10. Struktur der folgenden Abschnitte

- Motivation
- Abstrakte Syntax

$e \in \mathsf{Expr} ::= \cdots$ *Ausdrücke*

- Statische Semantik

$t \in \mathsf{Type} ::= \cdots$ *Typen*

- Dynamische Semantik

$\nu \in \mathsf{Val} ::= \cdots$ *Werte*

- Vertiefung
- Blick über den Tellerrand

10. Statische Semantik — Sinn und Zweck

- Ein Programm in Mini–F# ist ein *Ausdruck*:
  - $4711 + 815$
  - `"Hello, world!"`
- Ausdrücke sind beliebig kombinierbar.
- Nicht alle Kombinationen machen jedoch Sinn:
  - `"Hello, world!"` $* 4711$
- Die *statische Semantik* fängt diese sinnlosen Ausdrücke ab.
- Zu diesem Zweck werden Ausdrücke mit Hilfe von *Typen* in Schubladen eingeteilt:
  - `"Hello, world!"` hat den Typ *String*
  - 4711 hat den Typ *Nat*
- Die Multiplikation arbeitet auf den natürlichen Zahlen: `"Hello, world!"` $* 4711$ ist nicht wohlgetypt.

10. Statische Semantik — Typregeln

Die *statische Semantik* legt fest, dass die Multiplikation zwei natürliche Zahlen nimmt und eine natürliche Zahl zum Ergebnis hat.

$$\frac{e_1 : \mathit{Nat} \qquad e_2 : \mathit{Nat}}{e_1 * e_2 : \mathit{Nat}}$$

Für jedes neu eingeführte Konstrukt wird eine solche *Typregel* angegeben. Diese Beweisregeln spezifizieren die zweistellige Relation

$$e : t$$

zwischen Ausdrücken und Typen. *Lies:* „e hat den Typ t“.

10. Statische Semantik — Begriffe

- Die statische Semantik hat eine ordnende Funktion: fast alle Abschnitte führen *einen* neuen Typ ein, zusammen mit Sprachkonstrukten, die auf diesem Typ arbeiten.
- Ein Ausdruck e heißt *wohlgetypt*, wenn es einen Typ t gibt, so dass sich $e : t$ mit den Regeln der statischen Semantik ableiten lässt.
- Wohlgetypte Ausdrücke können ausgerechnet werden.

10. Dynamische Semantik — Auswertungsregeln

Wie ein Ausdruck ausgerechnet wird, legt die *dynamische Semantik* fest. Für die Multiplikation: beide Argumente werden ausgerechnet, das Ergebnis ist das Produkt der Teilergebnisse.

$$\frac{e_1 \Downarrow n_1 \qquad e_2 \Downarrow n_2}{e_1 * e_2 \Downarrow n_1 n_2}$$

Für jedes Konstrukt wird mindestens eine *Auswertungsregel* angegeben. Diese Regeln spezifizieren die zweistellige Relation

$$e \Downarrow \nu$$

zwischen Ausdrücken und Werten. *Lies:* „e wertet zu ν aus“.

10. Dynamische Semantik — Werte

- Was ist ein Wert?
- Ein Wert ist das Ergebnis eines Programms; der Wert eines arithmetischen Ausdrucks ist zum Beispiel eine natürliche Zahl.
- Mit jedem neu eingeführten Typ werden wir auch den Bereich der Werte erweitern.
- Ein Typ ist im Prinzip die Menge aller zugehörigen Werte.

10. Dynamische Semantik — Beweisbäume

Ein Programm wird ausgerechnet, indem die Auswertungsregeln für die Teilausdrücke zu einem Beweisbaum kombiniert werden.

$$\dfrac{\dfrac{\overline{4711 \Downarrow 4711} \qquad \overline{2 \Downarrow 2}}{4711 * 2 \Downarrow 9422} \qquad \overline{815 \Downarrow 815}}{4711 * 2 + 815 \Downarrow 10237}$$

Konstanten wie 4711 oder 815 werten zu sich selbst aus — Konstanten *sind* Werte.

11. Motivation

Nicht-triviale Programme treffen viele Entscheidungen. Im einfachsten Fall wird geprüft, ob ein bestimmter Sachverhalt wahr oder falsch ist:

- Ist das Konto überzogen?
- Ist Florian größer als Lisa?

Das Ergebnis einer solchen Überprüfung repräsentieren wir durch einen Wahrheitswert: *true* oder *false*.

11. Motivation

In Abhängigkeit von einem Wahrheitswert kann die Rechnung dann einen bestimmten Verlauf nehmen.

if e_1 ***then*** e_2 ***else*** e_3

Wertet der Ausdruck e_1, die Bedingung, zu *true* aus, dann wird mit der Auswertung von e_2 fortgefahren; wertet e_1 zu *false* aus, dann wird e_3 ausgewertet.

11. Abstrakte Syntax

$e \in \text{Expr} ::=$		*Boolesche Ausdrücke:*
	false	falsch
|	*true*	wahr
|	***if*** e_1 ***then*** e_2 ***else*** e_3	Alternative

Der Teilausdruck e_1 der Alternative heißt *Bedingung*; die Teilausdrücke e_2 und e_3 heißen *Zweige* der Alternative.

11. Statische Semantik

$t \in \text{Type} ::=$	*Typen:*
$\mid$ *Bool*	Typ der Booleschen Werte

Typregeln:

$$\frac{}{\mathit{false} : \mathit{Bool}} \qquad \frac{}{\mathit{true} : \mathit{Bool}}$$

$$\frac{e_1 : \mathit{Bool} \qquad e_2 : t \qquad e_3 : t}{\textbf{\textit{if}}\ e_1\ \textbf{\textit{then}}\ e_2\ \textbf{\textit{else}}\ e_3 : t}$$

☞ Die Bedingung muss vom Typ *Bool* sein; die Zweige der Alternative können einen beliebigen Typ besitzen; dieser ist auch der Typ des gesamten Ausdrucks.

11. Dynamische Semantik

$\nu \in \text{Val} ::=$		*Boolesche Werte:*
	$\mid$ *false*	falsch
	$\mid$ *true*	wahr

Auswertungsregeln:

$$\frac{}{\mathit{false} \Downarrow \mathit{false}} \qquad \frac{}{\mathit{true} \Downarrow \mathit{true}}$$

$$\frac{e_1 \Downarrow \mathit{true} \qquad e_2 \Downarrow \nu}{\textbf{\textit{if}}\ e_1\ \textbf{\textit{then}}\ e_2\ \textbf{\textit{else}}\ e_3 \Downarrow \nu} \qquad \frac{e_1 \Downarrow \mathit{false} \qquad e_3 \Downarrow \nu}{\textbf{\textit{if}}\ e_1\ \textbf{\textit{then}}\ e_2\ \textbf{\textit{else}}\ e_3 \Downarrow \nu}$$

11. Beispielrechnung

$$\dfrac{\dfrac{\dfrac{\dfrac{}{4711 \Downarrow 4711} \quad \dfrac{}{815 \Downarrow 815}}{4711 < 815 \Downarrow \mathit{false}} \quad \dfrac{}{\mathit{true} \Downarrow \mathit{true}}}{\textbf{\textit{if}}\ 4711 < 815\ \textbf{\textit{then}}\ \mathit{false}\ \textbf{\textit{else}}\ \mathit{true} \Downarrow \mathit{true}} \quad \dfrac{}{\texttt{"yes"} \Downarrow \texttt{"yes"}}}{\textbf{\textit{if}}\ (\textbf{\textit{if}}\ 4711 < 815\ \textbf{\textit{then}}\ \mathit{false}\ \textbf{\textit{else}}\ \mathit{true})\ \textbf{\textit{then}}\ \texttt{"yes"}\ \textbf{\textit{else}}\ \texttt{"no"} \Downarrow \texttt{"yes"}}$$

☞ Alternativen dürfen beliebig geschachtelt werden: die Bedingung der äußeren Alternative ist wiederum eine Alternative.

11. Demo

```
Mini⟩ false
false
Mini⟩ true
true
Mini⟩ if false then "yes" else "no"
"no"
Mini⟩ if true then "yes" else "no"
"yes"
Mini⟩ if 4711 < 815 then "yes" else "no"
"no"
Mini⟩ if (if 4711 < 815 then false else true) then "yes" else "no"
"yes"
Mini⟩ if 4711 < 815 then true else 7 > 1
true
Mini⟩ if 4711 < 815 then 7 > 1 else false
false
```

11. Vertiefung

Ein Boolescher Ausdruck modelliert einen Sachverhalt oder eine Aussage.

Wir sind gewohnt, einfache Aussagen zu komplexen Aussagen zusammenzusetzen:

- *Negation*: Der Kunde ist *nicht* kreditwürdig.
- *Konjunktion*: Das Konto ist überzogen *und* der Kunde ist nicht kreditwürdig.
- *Disjunktion*: Das Netzteil ist defekt *oder* die Leitung ist unterbrochen.

11. Vertiefung

- *Negation* von e:

  if e ***then*** *false* ***else*** *true*

- *Konjunktion* von e_1 und e_2:

  if e_1 ***then*** e_2 ***else*** *false*

- *Disjunktion* von e_1 und e_2:

  if e_1 ***then*** *true* ***else*** e_2

☞ Wir kürzen die Negation von e mit *not* e, die Konjunktion von e_1 und e_2 mit e_1 && e_2 und die Disjunktion mit e_1 || e_2 ab.

11. George Boole (1815–1864)

Der englische Mathematiker George Boole entwickelte in seiner Schrift „The Mathematical Analysis of Logic“ von 1847 den ersten algebraischen Logikkalkül und begründete damit die moderne mathematische Logik.

Boole stellte die Wahrheitswerte durch die Zahlen 0 und 1 dar und drückte die logischen Operationen entsprechend durch arithmetische Operationen aus.

56 OF HYPOTHETICALS.

1st. Disjunctive Syllogism.

Either X is true, or Y is true (exclusive),	$x + y - 2xy = 1$
But X is true,	$x = 1$
Therefore Y is not true,	$\therefore y = 0$
Either X is true, or Y is true (not exclusive),	$x + y - xy = 1$
But X is not true,	$x = 0$
Therefore Y is true,	$\therefore y = 1$

12. Motivation

Die ganze Zahl schuf der liebe Gott,
alles übrige ist Menschenwerk.
— Leopold Kronecker (1823–1891)

- Den Begriff Rechnen werden die meisten mit Zahlen in Verbindung bringen.
- In diesem Abschnitt führen wir einige elementare Konstrukte zum Rechnen mit *natürlichen* Zahlen ein.

12. Abstrakte Syntax

$n \in \mathbb{N} ::= 0 \mid 1 \mid 2 \mid \cdots$	*natürliche Zahlen*
$e ::= \cdots$	*Arithmetische Ausdrücke:*
$\mid n$	natürliche Zahl
$\mid e_1 + e_2$	Addition
$\mid e_1 \dot{-} e_2$	natürliche Subtraktion („monus“)
$\mid e_1 * e_2$	Multiplikation
$\mid e_1 \div e_2$	natürliche Division
$\mid e_1 \,\%\, e_2$	Divisionsrest
$\mid e_1 < e_2$	kleiner
$\mid e_1 \leq e_2$	kleiner gleich
$\mid e_1 = e_2$	gleich
$\mid e_1 <> e_2$	ungleich
$\mid e_1 \geqslant e_2$	größer gleich
$\mid e_1 > e_2$	größer

☞ Die Notation $e ::= \cdots$ soll andeuten, dass wir die Kategorie der Ausdrücke um arithmetische Operatoren und Vergleichsoperatoren *erweitern*.

12. Statische Semantik

$t ::= \cdots$	*Typen:*
$\mid$ *Nat*	Typ der natürlichen Zahlen

Typregeln:

$$\frac{}{n : \mathit{Nat}}$$

$$\frac{e_1 : \mathit{Nat} \qquad e_2 : \mathit{Nat}}{e_1 + e_2 : \mathit{Nat}} \quad \text{usw.}$$

$$\frac{e_1 : \mathit{Nat} \qquad e_2 : \mathit{Nat}}{e_1 < e_2 : \mathit{Bool}} \quad \text{usw.}$$

12. Dynamische Semantik

$\nu ::= \cdots$	*Werte:*
$\mid n$	natürliche Zahlen

Auswertungsregeln:

$$\frac{}{n \Downarrow n}$$

$$\frac{e_1 \Downarrow n_1 \qquad e_2 \Downarrow n_2}{e_1 + e_2 \Downarrow n_1 + n_2}$$

$$\frac{e_1 \Downarrow n_1 \qquad e_2 \Downarrow n_2}{e_1 * e_2 \Downarrow n_1 n_2}$$

12. Dynamische Semantik

Ist doch alles ziemlich offensichtlich, oder? Ich meine, es ist doch klar, dass '$*$' beide Argumente ausrechnet und die Ergebnisse dann multipliziert.

Was ist denn, wenn eins der Argumente 0 ist?

Na ja, dann ist das Ergebnis halt auch 0. Ich sehe nicht, worauf Du hinaus willst.

Wenn das erste Argument schon 0 ist, müssen wir das zweite Argument ja gar nicht mehr ausrechnen!

$$\frac{e_1 \Downarrow 0}{e_1 * e_2 \Downarrow 0}$$

Ok, ist vielleicht schneller, aber 0 kommt immer noch raus.

12. Dynamische Semantik — Subtraktion

Die Differenz zweier Ausdrücke ist 0, wenn das zweite Argument größer ist als das erste.

$$\frac{e_1 \Downarrow n + k \qquad e_2 \Downarrow n}{e_1 \dot{-} e_2 \Downarrow k} \qquad \frac{e_1 \Downarrow n \qquad e_2 \Downarrow n + k + 1}{e_1 \dot{-} e_2 \Downarrow 0}$$

☞ Mini–F# arbeitet auf den natürlichen und nicht auf den ganzen Zahlen. (F# kennt von Haus aus keine natürlichen Zahlen, dafür aber ganze Zahlen und Fließkommazahlen. In Kürze mehr dazu.)

Der Operator $\dot{-}$ hört auch auf den Namen „monus“.

12. Dynamische Semantik — Division

Die Operatoren '$\div$' und '%' implementieren die Division mit Rest: $a \div b$ ist der Quotient von a und b und $a \,\%\, b$ ist der Divisionsrest.

$$\frac{e_1 \Downarrow n_1 \qquad e_2 \Downarrow n_2}{e_1 \div e_2 \Downarrow q} \qquad n_1 = qn_2 + r \text{ und } r < n_2$$

$$\frac{e_1 \Downarrow n_1 \qquad e_2 \Downarrow n_2}{e_1 \,\%\, e_2 \Downarrow r} \qquad n_1 = qn_2 + r \text{ und } r < n_2$$

☞ Die Nebenbedingungen legen die Metavariablen $q, r \in \mathbb{N}$ eindeutig fest.

☞ Für $b > 0$ gilt stets

$$a = (a \div b) * b + (a \,\%\, b) \quad \text{und} \quad 0 \leq a \,\%\, b < b$$

☞ Jede Zahl a lässt sich eindeutig in einen Quotienten und in einen Rest zerlegen für ein festes $b > 0$.

12. Dynamische Semantik — Division

- Die Ausdrücke $e \div 0$ und $e \,\%\, 0$ sind undefiniert; die dynamische Semantik ordnet ihnen keinen Wert zu: es gibt kein r mit $r < 0$.
- Das ist unbefriedigend — die statische Semantik sollte ja gerade derartige Programme herausfiltern.
- Eine Lösung für dieses Problem stellen wir erst *sehr* viel später vor (in Teil VII).

12. Dynamische Semantik

Ich habe noch mal über die Multiplikation nachgedacht. Wenn wir die Regel

$$\frac{e_1 \Downarrow 0}{e_1 * e_2 \Downarrow 0}$$

hinzunehmen, dann können wir Programme auswerten, die sonst keinen Wert haben.

Ähem, wie meinst Du das? Die statische Semantik, oder wie das heißt, lässt sowas doch gar nicht zu.

Hast Du nicht aufgepasst ;-)? Der Ausdruck $0 \div 0$ hat zum Beispiel keinen Wert.

Ja ...

Na, dann hat $0 * (0 \div 0)$ auch keinen Wert. Aber *mit* meiner Regel können wir den Ausdruck zu 0 auswerten.

12. Dynamische Semantik — Vergleichsoperatoren

Für jeden Vergleichsoperator gibt es zwei Auswertungsregeln.

$$\frac{e_1 \Downarrow n + k \qquad e_2 \Downarrow n}{e_1 < e_2 \Downarrow \mathit{false}} \qquad \frac{e_1 \Downarrow n \qquad e_2 \Downarrow n + 1 + k}{e_1 < e_2 \Downarrow \mathit{true}}$$

usw.

12. Demo

Mini⟩ $4711 * 2 + 815$
10237
Mini⟩ $11 * 11$
121
Mini⟩ $111 * 111$
12321
Mini⟩ $111111111 * 111111111$
12345678987654321

☞ Die Ergebnisse sind stets exakt. (Die Genauigkeit ist *nicht* auf die native Genauigkeit von Rechnern, 32 oder 64 Bit, eingeschränkt.)

12. Über den Tellerrand: F#

F# kennt eine Reihe verschiedener Zahlentypen (*Nat* ist leider nicht vordefiniert):

- *byte*: natürliche Zahlen von 0 bis $2^8 - 1 = 255$;
- *sbyte*: ganze Zahlen von $-2^7 = -128$ bis $2^7 - 1 = 127$ (signed byte);
- *int16*: ganze Zahlen von $-2^{15} = -32768$ bis $2^{15} - 1 = 32767$;
- *uint16*: natürliche Zahlen von 0 bis $2^{16} - 1 = 65535$ (unsigned int16);
- *int* und *uint32*: ditto mit 32 Bit;
- *int64* und *uint64*: ditto mit 64 Bit;
- *float32*: 32-bit Fließkommazahlen;
- *float*: 64-bit Fließkommazahlen;
- *bigint*: ganze Zahlen.

☞ Rechnen mit beschränkter Genauigkeit hat seine Tücken: $255uy + 1uy = 0uy$ und $2147483647 + 1 = -2147483648$.