13. Knobelaufgabe #4

Ein Ausdruck besteht aus einer Folge von Zeichen; ein Ausdruck vom Typ *String* wertet zu einer Zeichenfolge aus. Schreiben Sie ein Programm, das zu seinem eigenen Programmtext auswertet.

Easy!

```
Mini⟩ "done"
"done"
```

Nicht ganz Harry: das Programm `"done"` besteht aus 6 Zeichen, sein Wert aus 4.

??? … Ach so, die Anführungsstriche!!! Schnell repariert:

```
Mini⟩ show "done"
"\"done\""
```

Ich sehe, Du hast die Funktion *show* entdeckt, die einen Wert in seine externe Darstellung überführt. Jetzt besteht Dein Programm aus 11 Zeichen, der Wert aus 6.
Ein Tipp: wenn Du nach der Auswertung des Ausdrucks *putline it* eintippst, musst Du Zeichen für Zeichen den Programmtext erhalten.

13. Motivation

- Rechnungen sind in der Regel nicht linear, sie enthalten Zwischen- oder Hilfsrechnungen.
- Beispiel: Berechnung des Flächeninhalts eines Quadrats:
  - zunächst: Seitenlänge ausrechnen,
  - dann: Ergebnis der Zwischenrechnung mit sich selbst multiplizieren.
- Um das Ergebnis einer Zwischenrechnung gegebenenfalls mehrfach verwenden zu können, geben wir ihm einen Namen :

let $s = 4711 + 815$

Das Konstrukt ist eine *Wertedefinition*: der Bezeichner links wird an den Wert des Ausdrucks rechts gebunden. *Lies:* sei s gleich $4711 + 815$.

- Ein Bezeichner ist ein Ausdruck und somit auch:

$s * s$

Der obige Ausdruck berechnet den gewünschten Flächeninhalt.

13. Motivation

- Welcher Bezeichner kann wo verwendet werden?
- Wie werden Wertedefinitionen und Ausdrücke verknüpft?
- Der Zusammenhang wird durch einen ***in***-Ausdruck hergestellt.

let $s = 4711 + 815$ ***in*** $s * s$

  - Vor ***in*** steht eine Wertedefinition.
  - Nach ***in*** steht ein Ausdruck.
  - Das gesamte Konstrukt ist wiederum ein Ausdruck.
  - Der Bezeichner *s* ist nur in $s * s$ *sichtbar*.

13. Beispiel: Ausflug in den Zoo

Problem:

Die Klasse 2c macht einen Ausflug in den Zoo. Es fahren 27 Schülerinnen und Schüler und 3 Lehrer mit. Die Fahrtkosten betragen 2€ für Kinder und 3€ für Erwachsene. Kinder zahlen 5€ Eintritt und Erwachsene 10€. Wie teuer ist der Ausflug?

Berechnung der Kosten:

$$
\begin{array}{lll}
\textbf{\textit{let}}\ \mathit{Schüler} & = 27 & \textbf{\textit{in}} \\
\textbf{\textit{let}}\ \mathit{Lehrer} & = 3 & \textbf{\textit{in}} \\
\textbf{\textit{let}}\ \mathit{Fahrtkosten} & = 2 * \mathit{Schüler} + 3 * \mathit{Lehrer} & \textbf{\textit{in}} \\
\textbf{\textit{let}}\ \mathit{Eintritt} & = 5 * \mathit{Schüler} + 10 * \mathit{Lehrer} & \textbf{\textit{in}} \\
\multicolumn{3}{l}{\mathit{Fahrtkosten} + \mathit{Eintritt}}
\end{array}
$$

☞ Benennung von Teilrechnungen.

13. Wahl der Bezeichner

- Der Bezeichner in einer Wertedefinition kann frei gewählt werden:

  let $s = 4711 + 815$ ***in*** $s * s$

  ist gleichwertig zu

  let $size = 4711 + 815$ ***in*** $size * size$

- Für das Ausrechnen spielen Namen keine Rolle, wohl aber für den menschlichen Betrachter eines Programms. *Deshalb:* möglichst aussagekräftige Bezeichner vergeben.
- *Allgemein gilt:* je größer der Sichtbarkeitsbereich eines Namens, desto mehr Sorgfalt sollte man bei der Namenswahl walten lassen.

13. Abstrakte Syntax — Definitionen

Wir führen eine neue syntaktische Kategorie ein: *Deklarationen*.

$x \in \mathsf{Id}$	*Bezeichner*
$d \in \mathsf{Decl} ::=$	*Deklarationen:*
$\quad \mid \ \textbf{\textit{let}}\ x = e$	Wertedefinition

☞ Der Bereich der Bezeichner Id wird in Teil VI genau festgelegt.

Für's erste: ein Bezeichner fängt mit einem Buchstaben an. Danach können weitere Buchstaben, Ziffern, und Sonderzeichen wie ein Unterstrich oder ein Apostroph folgen.

13. Abstrakte Syntax — Ausdrücke

Ein ***in***-Ausdruck verknüpft eine Definition mit einem Ausdruck.

$e ::= \cdots$	*lokale Definitionen:*
$\mid\ x$	Bezeichner
$\mid\ d$ ***in*** e	lokale Definition

Der Teilausdruck e heißt Rumpf des ***in***-Ausdrucks.

13. Statische Semantik

- Wie werden ***in***-Ausdrücke typisiert?

  let $s = 4711 + 815$ ***in*** $s * s$

  Wenn wir den Teilausdruck $s * s$ typisieren, woher kennen wir den Typ von s?
- Ein Bezeichner kann in unterschiedlichen Kontexten einen unterschiedlichen Typ besitzen.

  $(\textbf{\textit{let}}\ s = \mathit{true}\ \textbf{\textit{in}}\ s)\ \&\&\ (\textbf{\textit{let}}\ s = 815\ \textbf{\textit{in}}\ s * s > 4711)$

13. Statische Semantik — Signatur

- Wir merken uns die Typen von Bezeichnern mit Hilfe einer sogenannten Signatur.

$$\Sigma \in \mathsf{Sig} = \mathsf{Id} \to_{\mathrm{fin}} \mathsf{Type} \qquad \textit{Signatur}$$

Eine Signatur ist eine endliche Abbildung von Bezeichnern auf Typen.

- Wir erweitern die Typregeln um Signaturen. Die Regeln spezifizieren die nunmehr *dreistellige* Relation

$$\Sigma \vdash e : t$$

zwischen Signaturen, Ausdrücken und Typen. *Lies:* „bezüglich der Signatur Σ hat e den Typ t“.

13. Statische Semantik — Definitionen

Die statische Semantik ordnet

- einem Ausdruck einen Typ und
- einer Definition eine Signatur

zu.

Typregel:

$$\frac{\Sigma \vdash e : t}{\Sigma \vdash (\textbf{\textit{let}}\ x = e) : \{x \mapsto t\}}$$

☞ Eine Signatur repräsentiert — ähnlich wie ein Typ — das, was wir über eine Definition statisch wissen.

13. Statische Semantik — Ausdrücke

Typregeln:

$$\frac{}{\Sigma \vdash x : \Sigma(x)} \qquad x \in dom\ \Sigma$$

☞ Zu jedem Bezeichner muss eine definierende Bindung existieren.

$$\frac{\Sigma \vdash d : \Sigma' \qquad \Sigma, \Sigma' \vdash e : t}{\Sigma \vdash (d\ \textbf{in}\ e) : t}$$

☞ Der Teilausdruck e wird bezüglich der Signatur Σ, Σ' typisiert. Der Kommaoperator erweitert Σ um Σ' und regelt Überschneidungen.

13. Statische Semantik — Beispiel

Beispiel für einen wohlgetypten Ausdruck:

$$\dfrac{\dfrac{\overline{\varnothing \vdash 47 : Nat}}{\varnothing \vdash (\textbf{\textit{let}}\ s = 47) : \{s \mapsto Nat\}} \quad \dfrac{\overline{\{s \mapsto Nat\} \vdash s : Nat} \quad \overline{\{s \mapsto Nat\} \vdash s : Nat}}{\{s \mapsto Nat\} \vdash s * s : Nat}}{\varnothing \vdash (\textbf{\textit{let}}\ s = 47\ \textbf{\textit{in}}\ s * s) : Nat}$$

Beispiel für einen *nicht* wohlgetypten Ausdruck:

$$\dfrac{\dfrac{\overline{\varnothing \vdash 47 : Nat}}{\varnothing \vdash (\textbf{\textit{let}}\ s = 47) : \{s \mapsto Nat\}} \quad \dfrac{\overline{\{s \mapsto Nat\} \vdash s : Nat} \quad \{s \mapsto Nat\} \vdash a : Nat?}{\{s \mapsto Nat\} \vdash s * a : Nat}}{\varnothing \vdash (\textbf{\textit{let}}\ s = 47\ \textbf{\textit{in}}\ s * a) : Nat}$$

☞ Die Formel $\{s \mapsto Nat\} \vdash a : Nat$ lässt sich nicht ableiten.

13. Dynamische Semantik

- Wie werden ***in***-Ausdrücke ausgerechnet?

  let $s = 4711 + 815$ ***in*** $s * s$

  Wenn wir den Teilausdruck $s * s$ ausrechnen, woher kennen wir den Wert von s?
- Ein Bezeichner kann in unterschiedlichen Kontexten einen unterschiedlichen Wert besitzen.

  (***let*** $s = \mathit{true}$ ***in*** s) && (***let*** $s = 815$ ***in*** $s * s > 4711$)

13. Dynamische Semantik — Umgebung

- Wir merken uns die Werte von Bezeichnern mit Hilfe einer sogenannten Umgebung.

$$\delta \in \mathsf{Env} = \mathsf{Id} \to_{\mathrm{fin}} \mathsf{Val} \qquad \textit{Umgebung}$$

Wie eine Signatur ist eine Umgebung eine endliche Abbildung; im Unterschied zur Signatur bildet sie Bezeichner auf *Werte* ab.

- Wir erweitern die Auswertungsregeln um Umgebungen. Die Regeln spezifizieren die nunmehr *dreistellige* Relation

$$\delta \vdash e \Downarrow \nu$$

zwischen Umgebungen, Ausdrücken und Werten. *Lies:* „bezüglich der Umgebung δ wertet e zu ν aus“.

13. Dynamische Semantik — Definitionen

Die dynamische Semantik ordnet

- einem Ausdruck einen Wert und
- einer Definition eine Umgebung

zu.

Auswertungsregel:

$$\frac{\delta \vdash e \Downarrow \nu}{\delta \vdash (\textbf{\textit{let}}\ x = e) \Downarrow \{x \mapsto \nu\}}$$

13. Dynamische Semantik — Ausdrücke

Auswertungsregeln:

$$\frac{}{\delta \vdash x \Downarrow \delta(x)} \qquad x \in \mathit{dom}\ \delta$$

☞ Zu jedem Bezeichner muss eine definierende Bindung existieren.

$$\frac{\delta \vdash d \Downarrow \delta' \qquad \delta, \delta' \vdash e \Downarrow \nu}{\delta \vdash (d\ \textbf{\textit{in}}\ e) \Downarrow \nu}$$

☞ Der Teilausdruck e wird bezüglich der Umgebung δ, δ' ausgewertet. Der Kommaoperator erweitert δ um δ' und regelt Überschneidungen.

13. Dynamische Semantik — Beispiel

$$\dfrac{\dfrac{}{\varnothing \vdash 4711 + 815 \Downarrow 5526} \quad \dfrac{\dfrac{}{\delta \vdash s \Downarrow 5526} \quad \dfrac{}{\delta \vdash s \Downarrow 5526}}{\delta \vdash s * s \Downarrow 30536676}}{\varnothing \vdash \textbf{let } s = 4711 + 815 \Downarrow \delta \quad \varnothing \vdash \textbf{let } s = 4711 + 815 \textbf{ in } s * s \Downarrow 30536676}$$

wobei $\delta = \{s \mapsto 5526\}$

☞ Der Rumpf $s * s$ wird bezüglich der Umgebung $\{s \mapsto 5526\}$ ausgerechnet.

☞ Aus Gründen der Übersichtlichkeit kürzen wir einfache Teilrechnungen ab.

13. Demo

Mini⟩ ***let*** $s = 4711 + 815$ ***in*** $s * s$
30536676
Mini⟩ ***let*** *size* $= 4711 + 815$ ***in*** *size* $*$ *size*
30536676
Mini⟩ (***let*** $s = 4711 + 815$ ***in*** $s * s$) $+ 1$
30536677
Mini⟩ (***let*** $s = 4711$ ***in*** $s * s$) $+$ (***let*** $s = 815$ ***in*** $s * s$)
22857746
Mini⟩ ***let*** $s = 4711$ ***in let*** $a = s * s$ ***in*** $a + s$
22198232

13. Freie Bezeichner

Ein Bezeichner, der nicht im Geltungsbereich einer Definition liegt, heißt *frei*.

Ausdruck	freie Bezeichner
$s * s$	$\{s\}$
$s * a$	$\{a, s\}$
let $s = 4711$ ***in*** $s * s$	$\emptyset$
let $s = 4711$ ***in*** $s * a$	$\{a\}$

☞ In dem Ausdruck *d* ***in*** *e* werden die in *d* definierten Bezeichner von den freien Bezeichner in *e* abgezogen.

Ein Ausdruck, der keine freien Bezeichner enthält, heißt *geschlossen*.

13. Invarianten der Semantik

Invariante der statischen Semantik:

Die statische Semantik typisiert nur Ausdrücke, deren freie Bezeichner in der Signatur aufgeführt werden. — Bevor ein Teilausdruck, der freie Bezeichner enthält, typisiert wird, wird zunächst die Signatur um die Typen der freien Bezeichner erweitert.

Invariante der dynamischen Semantik:

Die dynamische Semantik legt nur die Bedeutung von Ausdrücken fest, deren freie Bezeichner in der Umgebung aufgeführt werden. — Bevor ein Teilausdruck, der freie Bezeichner enthält, ausgewertet wird, wird zunächst die Umgebung um die Werte der freien Bezeichner erweitert.

13. Redefinition

Was passiert, wenn man den gleichen Bezeichner mehrfach definiert?

$$\dfrac{\dfrac{\dfrac{}{\varnothing \vdash 4 \Downarrow 4}}{\varnothing \vdash \textbf{let } s = 4 \Downarrow \{s \mapsto 4\}} \quad \dfrac{\dfrac{\dfrac{}{\{s \mapsto 4\} \vdash 7 \Downarrow 7}}{\{s \mapsto 4\} \vdash \textbf{let } s = 7 \Downarrow \{s \mapsto 7\}} \quad \dfrac{\dfrac{}{\{s \mapsto 7\} \vdash s \Downarrow 7} \quad \star\{s \mapsto 7\} \vdash s * s \Downarrow 49}{\vdots}}{\{s \mapsto 4\} \vdash \textbf{let } s = 7 \textbf{ in } s * s \Downarrow 49}}{\varnothing \vdash \textbf{let } s = 4 \textbf{ in let } s = 7 \textbf{ in } s * s \Downarrow 49}$$

$\star$ Der Kommaoperator räumt der "neuen" Definition Vorrang ein:
$\{s \mapsto 4\}, \{s \mapsto 7\} = \{s \mapsto 7\}$.

13. Redefinition

Sollte man den überhaupt Bezeichner redefinieren? Ich meine, ist das nicht schlechter Programmierstil?

Man soll in der Tat darauf achten, Bezeichner nicht zu redefinieren. *Allerdings:* neue Namen zu erfinden ist schwer.

13. Umbenennungen

Die Festlegung, “neuen” Definitionen Vorrang einzuräumen, ist sinnvoll. Sie ist insbesondere kompatibel zum Umbenennen von Bezeichnern.

Der Bezeichner *s* in

let $s = 815$ ***in*** $s * s$

kann zu *size* umbenannt werden,

let $size = 815$ ***in*** $size * size$

ohne die Bedeutung des Programms zu ändern.

13. Umbenennungen

Die Bedeutung bleibt ebenso unverändert, wenn der Ausdruck *Teil* eines größeren Programms ist.

Der Bezeichner *s* im rechten ***in***-Ausdruck

let $s = 4711$ ***in let*** $s = 815$ ***in*** $s * s$

kann zu *size* umbenannt werden,

let $s = 4711$ ***in let*** $size = 815$ ***in*** $size * size$

ohne die Bedeutung des Programms zu ändern.

13. Demo

Mini⟩ $(\textbf{let } s = 4711 \textbf{ in } s * s) + (\textbf{let } s = 815 \textbf{ in } s * s)$
22857746
Mini⟩ $\textbf{let } s = 4711 \textbf{ in let } s = 815 \textbf{ in } s * s$
664225
Mini⟩ $\textbf{let } s_1 = 4711 \textbf{ in let } s_2 = 815 \textbf{ in } s_2 * s_2$
664225
Mini⟩ $\textbf{let } s = 4711 \textbf{ in let } a = s * s \textbf{ in } a + a$
44387042
Mini⟩ $\textbf{let } a = \textbf{let } s = 4711 \textbf{ in } s * s \textbf{ in } a + a$
44387042

13. Über den Tellerrand: F#

Wollen wir mehrere Definitionen in einem Ausdruck verwenden, müssen wir ***in***-Ausdrücke aneinanderreihen.

let $a = 4711$
$\quad\quad + 815$
in let $b = a * a$
$\quad$ ***in*** $a + b$

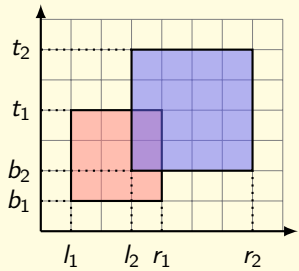
F# erlaubt alternativ das Ende einer lokalen Definition und den Anfang des Rumpfes mit Hilfe des *Layouts* festzulegen.

let $a = 4711$
$\quad\quad + 815$
let $b = a * a$
$a + b$

Abseitsregel: die Einrückung bestimmt die Zugehörigkeit von Teilausdrücken. Einrückung: der „aktuelle“ Ausdruck wird weitergeführt; keine Einrückung: ein neue Definition oder der Rumpfausdruck wird begonnen.

13. Beispiel: Berechnung der Wohnfläche

Die Fläche einer Wohnung mit dem unten skizzierten Grundriss soll berechnet werden.



(Das ist eine Vereinfachung, das Ergebnis des schon angesprochenen Abstraktionsprozesses, mit dem wir Aufgaben in Rechenaufgaben verwandeln.)

13. Beispiel: Berechnung der Wohnfläche

Erfassung der Daten:

let $l_1 = 1$
let $r_1 = 4$
let $b_1 = 1$
let $t_1 = 4$

let $l_2 = 3$
let $r_2 = 7$
let $b_2 = 2$
let $t_2 = 6$

13. Beispiel: Berechnung der Wohnfläche

Gesamtfläche: Summe der beiden Quadratflächen minus der Schnittfläche.

let $s_1 = r_1 \dot{-} l_1$
let $s_2 = r_2 \dot{-} l_2$
let $w = r_1 \dot{-} l_2$
let $h = t_1 \dot{-} b_2$
$s_1 * s_1 + s_2 * s_2 \dot{-} w * h$

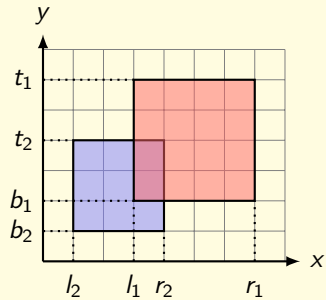
☞ Vermeidung von Mehrfachrechnungen: $s_1 * s_1$ statt $(r_1 \dot{-} l_1) * (r_1 \dot{-} l_1)$.

```
Mini⟩ ...
Mini⟩ s1 * s1 + s2 * s2 ∸ w * h
23
```

Die Wohnfläche beträgt somit $23\,m^2$.

13. Beispiel: Berechnung der Wohnfläche

Ändern wir die Daten, indem wir die Koordinaten der beiden Quadrate “vertauschen”,



let $l_1 = 3$
let $r_1 = 7$
let $b_1 = 2$
let $t_1 = 6$
let $l_2 = 1$
let $r_2 = 4$
let $b_2 = 1$
let $t_2 = 4$

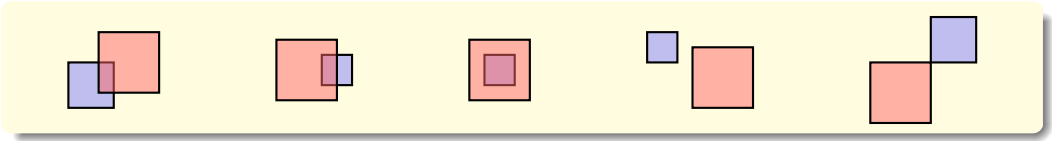
erleben wir eine unangenehme Überraschung:

Mini⟩ ***let*** $l_1 = 3$
Mini⟩ ...
Mini⟩ $s_1 * s_1 + s_2 * s_2 \doteq w * h$
0

13. Beispiel: Berechnung der Wohnfläche

☞ Das obige Programm macht unausgesprochene Annahmen über die relative Lage der beiden Quadrate. (Welche?)

Wollen wir das Programm “robuster” machen, müssen wir die Lage der Quadrate berücksichtigen. Viele unterschiedliche Konstellationen sind denkbar:

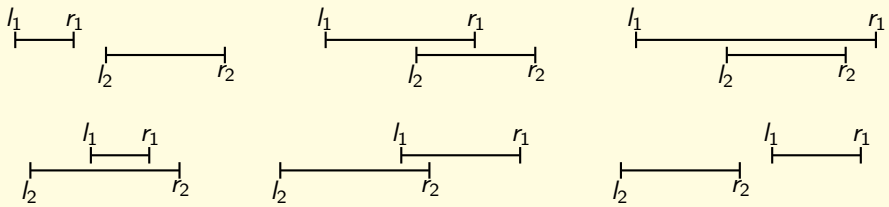


Wie werden wir Herr oder Frau der Lage?

☞ *Entscheidende Einsicht:* Überschneidungen können *getrennt* für jede der beiden Koordinatenachsen ausgerechnet werden. Auf diese Weise wird ein 2-dimensionales auf ein 1-dimensionales Problem zurückgeführt.

13. Beispiel: Berechnung der Wohnfläche

Konzentrieren wir uns auf die x-Achse. Wenn wir annehmen, dass $l_1 < r_1$ und $l_2 < r_2$ gilt, dann ergeben sich sechs mögliche Konstellationen:



Länge der Überschneidung: $\min\ r_1\ r_2 \dot{-} \max\ l_1\ l_2$.

13. Intermezzo: Minimum und Maximum

- *Minimum* von e_1 und e_2:

  if $e_1 \leqslant e_2$ ***then*** e_1 ***else*** e_2

- *Maximum* von e_1 und e_2:

  if $e_1 \leqslant e_2$ ***then*** e_2 ***else*** e_1

☞ Wir kürzen das Minimum von e_1 und e_2 mit *min* e_1 e_2 und das Maximum mit *max* e_1 e_2 ab.

13. Intermezzo: Minimum und Maximum

☞ Ist die Berechnung von e_1 oder e_2 aufwändig, formuliert man besser:

- *Minimum* von e_1 und e_2:

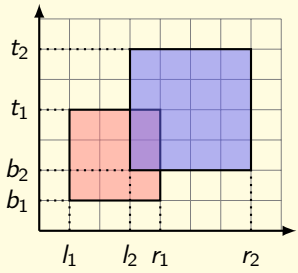
  let $a_1 = e_1$
  let $a_2 = e_2$
  if $a_1 \leqslant a_2$ ***then*** a_1 ***else*** a_2

- *Maximum* von e_1 und e_2:

  let $a_1 = e_1$
  let $a_2 = e_2$
  if $a_1 \leqslant a_2$ ***then*** a_2 ***else*** a_1

Warum?

13. Beispiel: Berechnung der Wohnfläche



let $l_1 = 1$
let $r_1 = 4$
let $b_1 = 1$
let $t_1 = 4$
let $l_2 = 3$
let $r_2 = 7$
let $b_2 = 2$
let $t_2 = 6$

Lösung:

let $s_1 = r_1 \dot{-} l_1$
let $s_2 = r_2 \dot{-} l_2$
let $w = min\ r_1\ r_2 \dot{-} max\ l_1\ \ l_2$
let $h\ = min\ t_1\ t_2 \dot{-} max\ b_1\ b_2$
$s_1 * s_1 + s_2 * s_2 \dot{-} w * h$