Teil II

Grundlagen

6. Knobelaufgabe #2

In einer dunklen Höhle leben Zwerge, die entweder eine weiße oder eine schwarze Mütze aufhaben. Einmal im Jahr dürfen sie die Höhle verlassen und bekommen eine Aufgabe gestellt. Lösen sie diese, sind sie frei. Misslingt die Lösung, müssen sie zurück in die Finsternis.

In diesem Jahr lautet die Aufgabe: Stellt euch nebeneinander so auf, dass die Zwerge mit einer weißen Mütze auf der linken Seite stehen und die mit schwarzer Mütze auf der rechten. Keiner der Zwerge kann die Farbe seiner eigenen Mütze sehen. Zudem dürfen die Zwerge weder miteinander reden noch sich auf sonstige Weise verständigen oder einander Hinweise geben, etwa mit der Hand oder den Augen. Auch Tricks wie die Verwendung von Spiegeln sind verboten.

Nicht verboten ist den Zwergen aber, ihren Verstand zu nutzen. Und in der Tat bekommen sie es auf Anhieb hin, sich nach der Farbe der Mützen getrennt aufzustellen. Wie haben Sie das bloß angestellt?

6. Motivation

In der Vorlesung werden wir nicht nur lernen

- *mit* der Programmiersprache Mini–F# Probleme zu lösen,

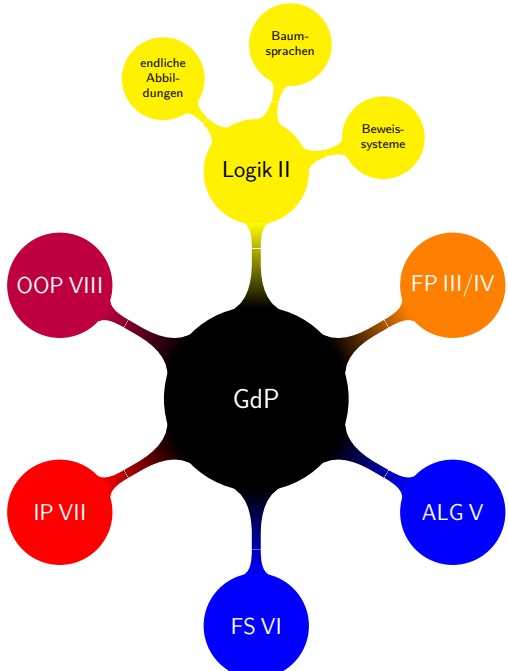
sondern auch

- *über* die Programmiersprache Mini–F# zu reden.

Mini–F# ist sowohl Subjekt als auch Objekt des Studiums.

Dazu benötigen wir ein wenig mathematisches Rüstzeug ...

6. Gliederung

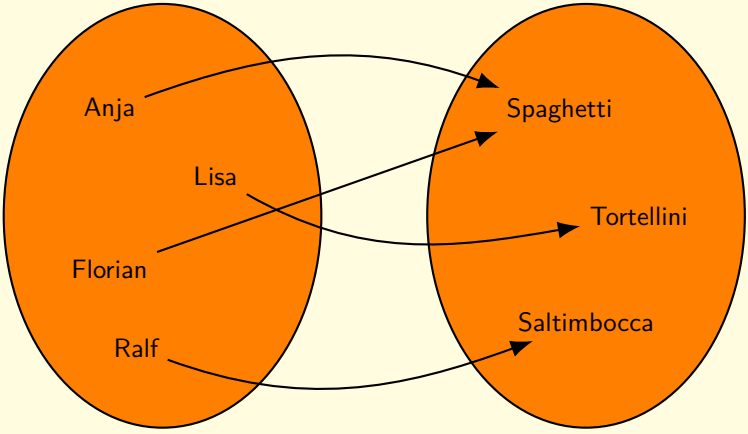

Baum-
sprachen
endliche
Abbil-
dungen
Beweis-
systeme
Logik II
OOP VIII
FP III/IV
GdP
IP VII
ALG V
FS VI

6. Lernziele

Nach Durcharbeitung dieses Kapitels sollten Sie

- mit endlichen Abbildungen und Sequenzen umgehen können,
- die Begriffe Syntax und Semantik definieren können,
- den Unterschied zwischen konkreter und abstrakter Syntax kennen,
- Baumsprachen lesen und selbst definieren können,
- Beweissysteme lesen und selbst definieren können.

7. Endliche Abbildungen



7. Endliche Abbildungen

$$\begin{aligned}\{&\mathit{Anja} &&\mapsto \mathit{Spaghetti},\\ &\mathit{Lisa} &&\mapsto \mathit{Tortellini},\\ &\mathit{Florian} &&\mapsto \mathit{Spaghetti},\\ &\mathit{Ralf} &&\mapsto \mathit{Saltimbocca}\}\end{aligned}$$

7. Notation endlicher Abbildungen

- Sind X und Y Mengen, dann bezeichnet $X \to_{\text{fin}} Y$ die Menge aller *endlichen Abbildungen* von X nach Y.
- Ist $\varphi \in X \to_{\text{fin}} Y$, dann bezeichnet $\mathit{dom}\ \varphi \subseteq X$ die Menge aller Elemente aus X, auf denen φ definiert ist, den *Definitionsbereich* von φ. Der Definitionsbereich $\mathit{dom}\ \varphi$ muss endlich sein.
- Die Anwendung einer endlichen Abbildung φ auf ein Element x notieren wir mit $\varphi(x)$.

Beispiel:

$$\varphi = \{\mathit{Anja} \mapsto \mathit{Spaghetti}, \mathit{Lisa} \mapsto \mathit{Tortellini}, \mathit{Florian} \mapsto \mathit{Spaghetti}, \mathit{Ralf} \mapsto \mathit{Saltimbocca}\}$$

$$\mathit{dom}\ \varphi = \{\mathit{Anja}, \mathit{Lisa}, \mathit{Florian}, \mathit{Ralf}\}$$

$$\varphi(\mathit{Anja}) = \mathit{Spaghetti}$$
$$\varphi(\mathit{Lisa}) = \mathit{Tortellini}$$

7. Konstruktion und Manipulation endlicher Abbildungen

Um endliche Abbildungen zu konstruieren bzw. zu manipulieren, verwenden wir die folgenden Operationen:

- die *leere Abbildung* $\varnothing$
  - $\mathit{dom}\ \varnothing = \emptyset$
- die *einelementige Abbildung* (auch: *Bindung*) $\{x \mapsto y\}$
  - $\mathit{dom}\ \{x \mapsto y\} = \{x\}$
  - $\{x \mapsto y\}(x) = y$
- die *Erweiterung* von φ_1 um φ_2 notiert φ_1, φ_2 (*Kommaoperator*)
  - $\mathit{dom}\ (\varphi_1, \varphi_2) = \mathit{dom}\ \varphi_1 \cup \mathit{dom}\ \varphi_2$
  - $(\varphi_1, \varphi_2)(x) = \begin{cases} \varphi_2(x) & \text{falls } x \in \mathit{dom}\ \varphi_2 \\ \varphi_1(x) & \text{sonst} \end{cases}$
- die *Einschränkung* von φ auf $\mathit{dom}\ \varphi \setminus A$ notiert $\varphi \setminus A$
  - $\mathit{dom}\ (\varphi \setminus A) = \mathit{dom}\ \varphi \setminus A$
  - $(\varphi \setminus A)(x) = \varphi(x)$

7. Eigenschaften endlicher Abbildungen

- Der Kommaoperator ist assoziativ:

$$(\varphi_1, \varphi_2), \varphi_3 = \varphi_1, (\varphi_2, \varphi_3)$$

Statt $(\varphi_1, \varphi_2), \varphi_3$ schreiben wir kurz $\varphi_1, \varphi_2, \varphi_3$. Zusätzlich verwenden wir $\{x_1 \mapsto y_1, \ldots, x_n \mapsto y_n\}$ als Abkürzung für $\{x_1 \mapsto y_1\}, \ldots, \{x_n \mapsto y_n\}$ (wiederholte Anwendung des Kommaoperators).

- Der Kommaoperator ist aber *nicht* kommutativ: φ_1, φ_2 ist in der Regel verschieden von φ_2, φ_1.

$$\{\mathit{Ralf} \mapsto \mathit{Pizza}\}, \{\mathit{Ralf} \mapsto \mathit{Saltimbocca}\} = \{\mathit{Ralf} \mapsto \mathit{Saltimbocca}\}$$
$$\{\mathit{Ralf} \mapsto \mathit{Saltimbocca}\}, \{\mathit{Ralf} \mapsto \mathit{Pizza}\} = \{\mathit{Ralf} \mapsto \mathit{Pizza}\}$$

☞ das zweite Argument des Kommaoperators hat Vorrang.

7. Sequenzen

Sequenzen, endliche Folgen von Elementen, lassen sich mit Hilfe endlicher Abbildungen modellieren.

Anja Florian Lisa Ralf

Die Elemente werden von links nach rechts beginnend mit 0 durchnummeriert.

$\{0 \mapsto \textit{Anja}, 1 \mapsto \textit{Florian}, 2 \mapsto \textit{Lisa}, 3 \mapsto \textit{Ralf}\}$

7. Notation von Sequenzen

- Eine Sequenz s von Elementen aus A ist eine endliche Abbildung des Typs $\mathbb{N} \to_{\text{fin}} A$ mit $dom\ s = \mathbb{N}_n$.
- $\mathbb{N}_n = \{0, \ldots, n-1\}$ ist ein Anfangsabschnitt der natürlichen Zahlen.
- Die Menge aller Sequenzen über A notieren wir mit A^*.
- Ist $dom\ s = \mathbb{N}_n$, dann heißt n die Länge von s, notiert $len\ s = n$.

7. Konstruktion und Manipulation von Sequenzen

Um Sequenzen zu konstruieren bzw. zu manipulieren, verwenden wir die folgenden Operationen:

- die *leere Sequenz* ϵ mit
  - $\mathit{dom}\ \epsilon = \mathbb{N}_0$;
- ist $a \in A$, dann verwenden wir oft das Element a selbst als Abkürzung für die *einelementige Sequenz* $\{0 \mapsto a\}$;
- die *Konkatenation* von s_1 und s_2 notiert $s_1 \cdot s_2$ mit
  - $\mathit{dom}\ (s_1 \cdot s_2) = \mathit{dom}\ s_1 \cup \{\, i + \mathit{len}\ s_1 \mid i \in \mathit{dom}\ s_2 \,\}$ und
  - $(s_1 \cdot s_2)(i) = \begin{cases} s_1(i) & \text{falls } i \in \mathit{dom}\ s_1 \\ s_2(i - \mathit{len}\ s_1) & \text{sonst} \end{cases}$
- die *n-fache Wiederholung* von s notiert s^n mit $s^0 = \epsilon$ und $s^{n+1} = s \cdot s^n$.

8. Syntax und Semantik

Wenn wir eine Programmiersprache präzise beschreiben wollen, müssen wir zwei Dinge festlegen:

- die äußere Form von Programmen, die *Syntax*,
- und deren Bedeutung, die *Semantik*.

8. Lexikalische Syntax

Betrachten wir ein einfaches Beispielprogramm:

```
4711* (a11 (* speed *) + 815 )
```

Mikroskopisch gesehen besteht das Programm aus einer Folge von Zeichen:

- der Ziffer `4`,
- gefolgt von der Ziffer `7`,
- gefolgt von der Ziffer `1`,
- gefolgt von der Ziffer `1`,
- gefolgt von einem Asteriskus `*`,
- gefolgt von einem Leerzeichen usw.

8. Lexikalische Syntax: Lexeme

Als menschlicher Leser sind wir gewohnt — bzw. durch jahrelanges Training geschult — mehrere Zeichen zu einer Einheit zusammenzufassen.

4711	*	(	a11	+	815	)

☞ Nicht alle Zeichen sind für den Rechner gedacht: `(* speed *)` ist ein Kommentar, der sich an den menschlichen Leser richtet.

In der *lexikalischen Syntax* einer Programmiersprache wird festgelegt, wie Zeichen zu größeren Einheiten, sogenannten *Lexemen* (engl. tokens), zusammengefasst werden.

8. Kontextfreie Syntax

Nicht alle Folgen von Lexemen stellen ein gültiges Programm dar:

```
) * 4711 815 + ( a11
```

umfasst die gleichen Lexeme, ist aber *kein* Mini–F# Programm.

In der *kontextfreien Syntax* einer Programmiersprache wird festgelegt, welche Folgen von Lexemen gültige Programme sind und welche nicht.

8. Konkrete Syntax

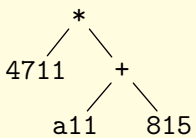
Die lexikalische und die kontextfreie Syntax bilden zusammen die *konkrete Syntax* einer Programmiersprache.

Will man eine Programmiersprache selbst zum Objekt des Studiums machen, dann ist die konkrete Syntax als Ausgangspunkt ungeeignet:

- sie ist technischen Einschränkungen unterworfen,
- sie ist das Endprodukt vieler Kompromisse und
- sie spiegelt den persönlichen Geschmack der Sprachdesigner wider.

8. Abstrakte Syntax

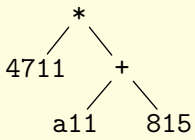
Für das Studium von Programmiersprachen ist die hierarchische Struktur eines Programms relevant.



☞ Die hierarchische Struktur verdeutlicht, dass sich der Ausdruck aus mehreren Teilausdrücken zusammensetzt. Man spricht auch von einem *Rechenbaum* oder allgemein von einem *abstrakten Syntaxbaum*.

8. Konkrete versus abstrakte Syntax

Abstrakte Syntax:



Konkrete Syntax in

- Mini–F#: `4711 * (a11 + 815)`,
- Scheme: `(* 4711 (+ a11 815))`,
- PostScript: `4711 a11 815 + *`.

8. Semantik

- Die Semantik legt die Bedeutung von Programmen fest.
- Die Bedeutung eines arithmetischen Ausdrucks ist eine Zahl.
- Die Semantik lässt sich mit Auswertungsregeln beschreiben:
  - wenn `a11` zu 1 auswertet, dann wertet `a11 + 815` zu 816 aus;
  - wenn `a11 + 815` zu 816 auswertet, dann wertet `4711 * (a11 + 815)` zu 3844176 aus.
- Die Auswertungsregeln orientieren sich eng an der Struktur eines Programms — die Bedeutung eines Ausdrucks wird auf die Bedeutung der Teilausdrücke zurückgeführt (kompositional).

9. Baumsprachen

Den hierarchischen Aufbau von Programmen, die abstrakte Syntax, beschreiben wir mit *Baumsprachen*.

Beispiel: eine einfache Form von arithmetischen Ausdrücken.

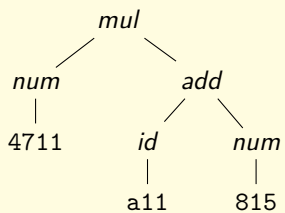
$$
\begin{aligned}
e \in \mathrm{Expr} ::= \; & \mathit{num}\ (\mathbb{N}) \\
\mid \; & \mathit{id}\ \ \ (\mathrm{Id}) \\
\mid \; & \mathit{add}\ (\mathrm{Expr}, \mathrm{Expr}) \\
\mid \; & \mathit{mul}\ (\mathrm{Expr}, \mathrm{Expr})
\end{aligned}
$$

9. Baumsprachen: Beispiel

Beispiel für einen arithmetischen Ausdruck:

$$mul\ (num\ (\texttt{4711}), add\ (id\ (\texttt{a11}), num\ (\texttt{815})))$$

Dargestellt als Baum:



9. Baumsprachen: Aufbau

$$
\begin{aligned}
e \in \text{Expr} ::= \; & \mathit{num}\ (\mathbb{N}) \\
\mid \; & \mathit{id}\ \ (\text{Id}) \\
\mid \; & \mathit{add}\ (\text{Expr}, \text{Expr}) \\
\mid \; & \mathit{mul}\ (\text{Expr}, \text{Expr})
\end{aligned}
$$

Die Definition führt drei verschiedene Dinge ein:

- einen Namen für die Baumsprache: Expr,
- Namen für die unterschiedlichen Knotenarten: *num*, *id*, *add* und *mul*, und
- eine *Metavariable* für Elemente der Baumsprache: e.

Die griechische Vorsilbe meta (μετά) bedeutet unter anderem „über“. Eine Metavariable ist Bestandteil der Metasprache, mit der wir *über* die Programmiersprache Mini–F# reden. Metavariablen sind von Variablen zu unterscheiden, die Bestandteil der Programmiersprache Mini–F# sind (und erst etwas später eingeführt werden).

9. Baumsprachen: umgangssprachliche Lesart

Umgangssprachlich lässt sich die Definition wie folgt lesen: ein Element $e \in \text{Expr}$ ist entweder

- von der Form $\mathit{num}\ (n)$ mit $n \in \mathbb{N}$, oder
- von der Form $\mathit{id}\ (x)$ mit $x \in \text{Id}$, oder
- von der Form $\mathit{add}\ (e_1, e_2)$ mit $e_1, e_2 \in \text{Expr}$ oder
- von der Form $\mathit{mul}\ (e_1, e_2)$ mit $e_1, e_2 \in \text{Expr}$.

☞ Wir verwenden die Metavariable e, um arithmetische Ausdrücke zu benennen (gegebenenfalls mit einem Index versehen).

9. Baumsprachen: formale Lesart

Formal ist die Menge Expr durch eine *induktive Definition* gegeben.

Die Menge Expr ist die *kleinste* Menge mit den folgenden Eigenschaften:

1. ist $n \in \mathbb{N}$, dann ist $num\ (n) \in \text{Expr}$;
2. ist $x \in \text{Id}$, dann ist $id\ (x) \in \text{Expr}$;
3. sind $e_1 \in \text{Expr}$ und $e_2 \in \text{Expr}$, dann ist auch $add\ (e_1, e_2) \in \text{Expr}$;
4. sind $e_1 \in \text{Expr}$ und $e_2 \in \text{Expr}$, dann ist auch $mul\ (e_1, e_2) \in \text{Expr}$.

☞ Wichtig ist, dass Expr die *kleinste* Menge mit diesen Eigenschaften ist; nur Elemente, die sich auf eine der vier Arten bilden lassen, sind in Expr enthalten.

9. Notationelle Freiheiten

$$
\begin{aligned}
&n \in \mathbb{N} \\
&x \in \mathsf{Id} \\
&e \in \mathsf{Expr} ::= n \\
&\qquad\qquad \mid\ x \\
&\qquad\qquad \mid\ e_1 + e_2 \\
&\qquad\qquad \mid\ e_1 * e_2
\end{aligned}
$$

☞ Der Unterschied zur ursprünglichen Definition ist nur ein äußerlicher: statt *mul* (*num* (4711), *add* (*id* (a11), *num* (815))) schreiben wir kurz $4711 * (\mathtt{a11} + 815)$, gemeint ist aber stets der gleiche abstrakte Syntax*baum*.

9. Beispiel: Klötzchenwelt — Baumsprachen

Ein weiteres Beispiel für eine Baumsprache: Folgen von Klötzchen.

$$
\begin{aligned}
w \in \text{Wall} ::= \; & \textit{yellow-brick} \\
\mid \; & \textit{red-brick} \\
\mid \; & \textit{sequ}\,(\text{Wall}, \text{Wall})
\end{aligned}
$$

Die gleiche Definition mit notationellen Freiheiten:

$w \in \text{Wall} ::=$
$\mid$ 
$\mid \; w_1 \; w_2$

10. Beweissysteme

Bei der umgangssprachlichen Beschreibung der Bedeutung arithmetischer Ausdrücke haben wir wenn-dann Aussagen formuliert:

- *wenn* `a11` zu 1 auswertet, *dann* wertet `a11 + 815` zu 816 aus;
- *wenn* `a11 + 815` zu 816 auswertet, *dann* wertet `4711 * (a11 + 815)` zu 3844176 aus.

Wenn-dann Aussagen lassen sich mit *Beweisregeln* formalisieren.

$$\frac{\mathit{id}\ (\texttt{a11}) \Downarrow 1}{\mathit{add}\ (\mathit{id}\ (\texttt{a11}), \mathit{num}\ (815)) \Downarrow 816}$$

$$\frac{\mathit{add}\ (\mathit{id}\ (\texttt{a11}), \mathit{num}\ (815)) \Downarrow 816}{\mathit{mul}\ (\mathit{num}\ (4711), \mathit{add}\ (\mathit{id}\ (\texttt{a11}), \mathit{num}\ (815))) \Downarrow 3844176}$$

Die Formulierung „wertet aus zu“ symbolisieren wir durch '$\Downarrow$'. Über dem Strich der Regeln stehen die Voraussetzungen (der „wenn“ Teil); unter dem Strich die Schlussfolgerung (der „dann“ Teil).

10. Beweisbäume

Beweisregeln:

$$\dfrac{\dfrac{}{\mathit{id}\ (\mathtt{a11}) \Downarrow 1}}{\mathit{add}\ (\mathit{id}\ (\mathtt{a11}), \mathit{num}\ (815)) \Downarrow 816}$$

$$\dfrac{\mathit{add}\ (\mathit{id}\ (\mathtt{a11}), \mathit{num}\ (815)) \Downarrow 816}{\mathit{mul}\ (\mathit{num}\ (4711), \mathit{add}\ (\mathit{id}\ (\mathtt{a11}), \mathit{num}\ (815))) \Downarrow 3844176}$$

Beweisregeln können zu *Beweisbäumen* zusammengesetzt werden.

$$\dfrac{\dfrac{\dfrac{\vdots}{\mathit{id}\ (\mathtt{a11}) \Downarrow 1}}{\mathit{add}\ (\mathit{id}\ (\mathtt{a11}), \mathit{num}\ (815)) \Downarrow 816}}{\mathit{mul}\ (\mathit{num}\ (4711), \mathit{add}\ (\mathit{id}\ (\mathtt{a11}), \mathit{num}\ (815))) \Downarrow 3844176}$$

10. Beweisbäume

Ohne Auslassungen:

$$\dfrac{\dfrac{}{\mathit{num}\ (4711) \Downarrow 4711} \qquad \dfrac{\dfrac{\vdots}{\mathit{id}\ (\mathtt{a11}) \Downarrow 1} \qquad \dfrac{}{\mathit{num}\ (815) \Downarrow 815}}{\mathit{add}\ (\mathit{id}\ (\mathtt{a11}), \mathit{num}\ (815)) \Downarrow 816}}{\mathit{mul}\ (\mathit{num}\ (4711), \mathit{add}\ (\mathit{id}\ (\mathtt{a11}), \mathit{num}\ (815))) \Downarrow 3844176}$$

10. Notationelle Freiheiten — da capo

Erlauben wir notationelle Freiheiten, gerät der Beweisbaum weniger mächtig.

$$\dfrac{\dfrac{}{4711 \Downarrow 4711} \qquad \dfrac{\dfrac{\vdots}{a11 \Downarrow 1} \qquad \dfrac{}{815 \Downarrow 815}}{a11 + 815 \Downarrow 816}}{4711 * (a11 + 815) \Downarrow 3844176}$$

10. Beweissysteme

Ist eine Menge von Formeln gegeben, etwa durch eine Baumsprache $\phi \in \Phi ::= \cdots$, dann hat eine *Beweisregel* die allgemeine Form

$$\frac{\phi_1 \qquad \cdots \qquad \phi_n}{\phi}$$

Über dem Strich der Regeln stehen die *Voraussetzungen* (n Formeln); unter dem Strich steht die *Schlussfolgerung* (eine einzige Formel). Ist $n = 0$, so spricht man auch von einem *Axiom*.

Ein *Beweissystem* ist eine Menge von Beweisregeln.

10. Beweisbäume

Die Menge aller *Beweisbäume* ist induktiv definiert: Sind $\mathcal{P}_1$, ..., $\mathcal{P}_n$ Beweisbäume mit den Wurzeln ϕ_1, ..., ϕ_n und ist

$$\frac{\phi_1 \quad \cdots \quad \phi_n}{\phi}$$

eine Beweisregel, dann ist

$$\frac{\mathcal{P}_1 \quad \cdots \quad \mathcal{P}_n}{\phi}$$

ein Beweisbaum mit der Wurzel ϕ.

Ist $\mathcal{P}$ ein Beweisbaum mit der Wurzel ϕ, dann sagt man auch $\mathcal{P}$ zeigt oder beweist ϕ.

10. Regelschemata

Die obigen Beweisregeln für die Auswertung arithmetischer Ausdrücke sind sehr speziell; allgemeinere Regeln lassen sich mit Hilfe von Metavariablen formulieren.

$$\frac{}{num\ (n)\ \Downarrow\ n}$$

$$\frac{e_1\ \Downarrow\ n_1 \qquad e_2\ \Downarrow\ n_2}{add\ (e_1, e_2)\ \Downarrow\ n_1 + n_2} \qquad \frac{e_1\ \Downarrow\ n_1 \qquad e_2\ \Downarrow\ n_2}{mul\ (e_1, e_2)\ \Downarrow\ n_1 n_2}$$

Die erste Regel — ein Axiom — legt fest, dass Konstanten zu sich selbst auswerten. Die beiden anderen Regeln formalisieren, dass zunächst beide Teilausdrücke ausgerechnet werden und dass das Ergebnis die Summe bzw. das Produkt der Teilergebnisse ist.

Regeln, die Metavariablen, enthalten, heißen *Regelschemata*.

10. Regelinstanzen

Bevor Regelschemata zu Beweisbäumen zusammengesetzt werden können, müssen die Metavariablen erst durch konkrete Konstanten bzw. Ausdrücke ersetzt werden.

Für unser laufendes Beispiel benötigen wir:

$$\frac{}{\mathit{num}\ (4711) \Downarrow 4711} \qquad \frac{}{\mathit{num}\ (815) \Downarrow 815}$$

$$\frac{\mathit{id}\ (\texttt{a11}) \Downarrow 1 \qquad \mathit{num}\ (815) \Downarrow 815}{\mathit{add}\ (\mathit{id}\ (\texttt{a11}), \mathit{num}\ (815)) \Downarrow 816}$$

$$\frac{\mathit{num}\ (4711) \Downarrow 4711 \qquad \mathit{add}\ (\mathit{id}\ (\texttt{a11}), \mathit{num}\ (815)) \Downarrow 816}{\mathit{mul}\ (\mathit{num}\ (4711), \mathit{add}\ (\mathit{id}\ (\texttt{a11}), \mathit{num}\ (815))) \Downarrow 3844176}$$

Das Ergebnis einer solchen Ersetzung nennt man auch *Regelinstanz* oder kurz *Instanz*.

Mit diesen Instanzen können wir wieder den Beweisbaum zusammensetzen.

10. Notationelle Freiheiten — da capo

Wir nehmen uns bei der Definition der abstrakten Syntax einige Freiheiten heraus.

Dies birgt die Gefahr von Verwechslungen.

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

Das Pluszeichen tritt zweimal auf, meint aber zwei grundsätzlich verschiedene Dinge:

- links ist '+' ein syntaktischer Bestandteil unserer Programmiersprache,
- rechts bezeichnet '$+$' das mathematische Konzept der Addition zweier natürlicher Zahlen.

☞ '+' ist Syntax und '$+$' ist Semantik.

10. Beispiel: Klötzchenwelt — Beweisregeln

Formeln: $n \sim w$. *Lies:* w ist eine schöne Mauer der Ordnung n.

Beweisregeln:

$$\frac{}{0 \sim \textit{yellow-brick}} \qquad \frac{}{1 \sim \textit{red-brick}}$$

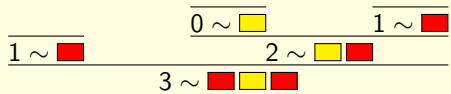
$$\frac{n \sim w_1 \qquad n+1 \sim w_2}{n+2 \sim \textit{sequ}\ (w_1, w_2)}$$

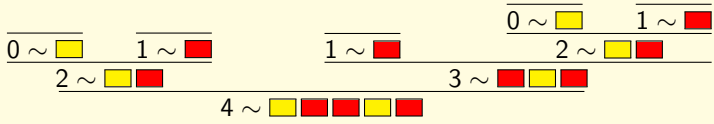
Beweisregeln mit notationellen Freiheiten:

$$\frac{}{0 \sim \square} \qquad \frac{}{1 \sim \blacksquare}$$

$$\frac{n \sim w_1 \qquad n+1 \sim w_2}{n+2 \sim w_1\, w_2}$$

10. Beispiel: Klötzchenwelt — Beweisbäume





10. Beispiel: Klötzchenwelt — ableitbare Formeln

Ableitbare oder beweisbare Formeln:

- 0 ∼ 
- 1 ∼ 
- 2 ∼ 
- 3 ∼ 
- 4 ∼ 
- 5 ∼ 
- 6 ∼ 
- 7 ∼ 
- ...

10. Beispiel: Klötzchenwelt—nicht ableitbare Formeln

Ableitbare oder beweisbare Formeln:

- $0 \sim$ 
- $1 \sim$ 
- $2 \sim$ 
- $3 \sim$ 
- $4 \sim$ 
- ...

Eigenschaften:

- Für $n > 0$ enden die Reihen in einem roten Klötzchen.
- Es folgen nie zwei gelbe Klötzchen aufeinander.
- Es folgen nie mehr als zwei rote Klötzchen direkt hintereinander.

☞ Die Formeln $n \sim$  und $n \sim$  sind *nicht* beweisbar.

10. Zusammenfassung

Wir haben

- Notation eingeführt, um endliche Abbildungen und Sequenzen zu konstruieren und zu manipulieren,
- den Unterschied zwischen konkreter und abstrakter Syntax kennengelernt,
- Baumsprachen zur Definition abstrakter Syntax eingeführt und
- Beweissysteme zur Definition der Semantik.

Wann geht's denn endlich mit dem Programmieren los?

Nächste Woche, Harry.