

Grundlagen der Programmierung

Rechnen und rechnen lassen

Ralf Hinze

Fachbereich Informatik
Technische Universität Kaiserslautern

WS 2022/2023

0. Knobelaufgabe #1

Ein Elb hat Ihnen 5 Gegenstände von unbekanntem Wert geschenkt. Die Gegenstände sollen ihrem Wert nach geordnet werden. Sie können dazu einem Orakel jeweils zwei Gegenstände vorlegen und erhalten dann Auskunft, welcher der beiden Gegenstände wertvoller ist.

Wie oft müssen Sie das Orakel *im ungünstigsten Fall* fragen?

Im günstigsten Fall muss das Orakel 4-mal befragt werden. Entwickeln Sie eine Systematik, so dass das Orakel nicht häufiger als nötig beansprucht wird.

Grundlagen der Programmierung



Organisation

Rechnen und
rechnen lassen

Geschichte

Gebiete

Einordnung

Überblick

Teil I

Einführung

1. Das Vorlesungsteam

- ▶ Dozent
 - ▶ Prof. Dr. Ralf Hinze
- ▶ Organisation des Übungsbetriebs
 - ▶ Alexander Dinges
 - ▶ Sebastian Schloßer
- ▶ Tutorinnen und Tutoren
 - ▶ Marvin Ballat
 - ▶ Alexander Gerbig
 - ▶ Anna Klose
 - ▶ Milan Koch
 - ▶ Sarah Langenstein
 - ▶ Jeriel Mbiedou
 - ▶ Martin Memmesheimer
 - ▶ Jakob Meyerhöfer
 - ▶ Sophia Porcher
 - ▶ Charlotte Sailer
 - ▶ Paulina Stähly

Organisation

Rechnen und
rechnen lassen

Geschichte

Gebiete

Einordnung

Überblick

1. Wir helfen Ihnen ...

- ▶ Homepage der Vorlesung:

<https://p1.cs.uni-k1.de/gdp22>

- ▶ Material (Duden: Gesamtheit von Hilfsmitteln, die für eine bestimmte Arbeit benötigt werden):
 - ▶ Vorlesungsvideos
 - ▶ Vorlesungsfolien
 - ▶ Skript
 - ▶ Beispielprogramme aus der Vorlesung
 - ▶ Anleitungen (zum Beispiel zur Installation von Software)
 - ▶ Lösungshinweise zu Übungsblättern
- ▶ Bei Fragen:
 - ▶ vorlesungsbegleitende Übung
 - ▶ Sprechstunden

1. Übungsbetrieb

- ▶ ein Übungstermin pro Woche (2 SWS, verpflichtend)
 - ▶ Teilnahme verpflichtend
 - ▶ erste Übung: ab dem 2. November
- ▶ Anmeldung zur Übung: mit dem ExClaim-System (siehe Homepage)
 - ▶ Anmeldefrist: Freitag 28. Oktober um 12:00 Uhr mittags
 - ▶ Angabe von Präferenzen für Termine und Übungspartner
- ▶ Sprechstunden (2 SWS, freiwillig)
 - ▶ Sprechstunde zur Vorlesung
 - ▶ Sprechstunden zur Übung
 - ▶ Details siehe Homepage/OLAT

1. Übungsblätter

👉 Programmieren lernt man nur durch Üben!

- ▶ Ein Übungsblatt pro Woche
 - ▶ Ausgabe: dienstags
 - ▶ Abgabe: Zwei Tage vor Ihrer Übungsstunde in der darauffolgenden Woche
 - ▶ erstes Übungsblatt: diese Woche (ohne Abgabe)
 - ▶ Programmieraufgaben werden über das Online-System »ExClaim« abgegeben. Andere Aufgaben können auf Papier gelöst und als Foto bzw. Scan über ExClaim abgegeben werden.
- ▶ Bearbeitung voraussichtlich in Zweierteams

1. Klausurzulassung

Organisation

Rechnen und
rechnen lassen

Geschichte

Gebiete

Einordnung

Überblick

- ▶ mindestens 60% der Übungspunkte aus den Übungsblättern (Team-Abgabe)
- ▶ mindestens 60% der Punkte in den OLAT-Tests (Einzelleistung)
- ▶ Anwesenheitspflicht in Übungen (max. 1× entschuldigtes Fehlen)
- ▶ Pflicht zur Präsentation einer Übungsabgabe in den Übungsstunden
- ▶ Abschlussklausur: Termin wird vom zentralen Prüfungsamt festgelegt und rechtzeitig bekanntgegeben.

1. Historische Daten

► Wintersemester 2018/2019

- Anmeldungen: 352
- ≥ 5 Übungsblätter bearbeitet: 253 ($\approx 72\%$)
- Zulassung erreicht: 184 ($\approx 73\%$)
- Abschlussklausur: 187 Teilnehmer (davon 22 mit alter Zulassung)

1,0	1,3	1,7	2,0	2,3	2,7	3,0	3,3	3,7	4,0	5,0
22	18	16	16	11	13	11	10	13	11	46
21%		23%				18%		13%		25%

► Nachklausur: 19 Teilnehmer

1,0	1,3	1,7	2,0	2,3	2,7	3,0	3,3	3,7	4,0	5,0
1					1		1	2	14	
					26%				74%	

Organisation

Rechnen und
rechnen lassen

Geschichte

Gebiete

Einordnung

Überblick

Organisation

Rechnen und
rechnen lassen

Geschichte

Gebiete

Einordnung

Überblick

1. Die erste Woche

Beachten Sie die Hinweise auf der Vorlesungswebseite sowie die Checkliste im OLAT für Woche 1.

1. Helfen Sie uns ...

- ▶ Fehler in den Folien (mehr als tausend Folien; jede Einzelne bietet die Gelegenheit Fehler zu machen)
- ▶ Fehler in den Übungsaufgaben
- ▶ Schicken Sie uns originelle oder schöne Lösungen für Aufgaben und/oder Knobelaufgaben:

support@harry-hacker.org

1. Helfen Sie sich und anderen ...

- ▶ Nutzen Sie die unterschiedlichen Angebote ...
 - ▶ Vorlesung und Übung *ergänzen* sich; sie ersetzen sich nicht.
- ▶ Seien Sie aktiv ...
 - ▶ Stellen Sie Fragen!
 - ▶ Üben Sie die Formalismen.
 - ▶ Schreiben Sie Programme.
 - ▶ Nehmen Sie die Angebote wahr: Sprechstunden.
 - ▶ Nachbearbeitungszeit für die Vorlesung: durchschnittlich 3-4 Stunden pro Woche
(Gesamtaufwand: 12-14 Stunden)
- ▶ Organisieren Sie sich ...
 - ▶ Zu zweit oder zu dritt lernt es sich besser.

1. Hörsaal Etikette

Organisation

Rechnen und
rechnen lassen

Geschichte

Gebiete

Einordnung

Überblick

- ▶ Schalten Sie bitte Ihr Handy *aus*.
- ▶ Schalten Sie bitte alle anderen elektronischen Geräte in den *Flugmodus*.
- ▶ Kommen Sie bitte frühzeitig.
- ▶ Verlassen Sie die Veranstaltungen *bitte* nicht vorzeitig.
- ▶ Kurzum: nehmen Sie auf Ihre Kommilitonen/Kommilitoninnen Rücksicht.

1. Kurzum ...

Organisation

Rechnen und
rechnen lassen

Geschichte

Gebiete

Einordnung

Überblick

- ▶ Die »Grundlagen der Programmierung« ist *Ihre* Vorlesung.
- ▶ Studieren Sie mit Spaß und Begeisterung.
- ▶ Helfen Sie mit, dass die Vorlesung für Sie und für uns zu einem Erfolg wird.

2. Informatik

Organisation

Rechnen und
rechnen lassen

Geschichte

Gebiete

Einordnung

Überblick



Informatik

Informatik ist die Wissenschaft vom maschinellen Rechnen.

2. Die Natur des Rechnens

- ▶ Rechnen?
- ▶ Umgang mit Zahlen, Arithmetik!?
- ▶ Rechnen ist etwas Mechanisches; es folgt festen Regeln.
- ▶ Die Regeln für Addition und Multiplikation lernt man in der Grundschule.
Schriftliche Addition:

$$\begin{array}{r} & 8 & 1 & 5 \\ + & 4 & 7 & 1 & 1 \\ \hline = & 5 & 5 & 2 & 6 \end{array}$$

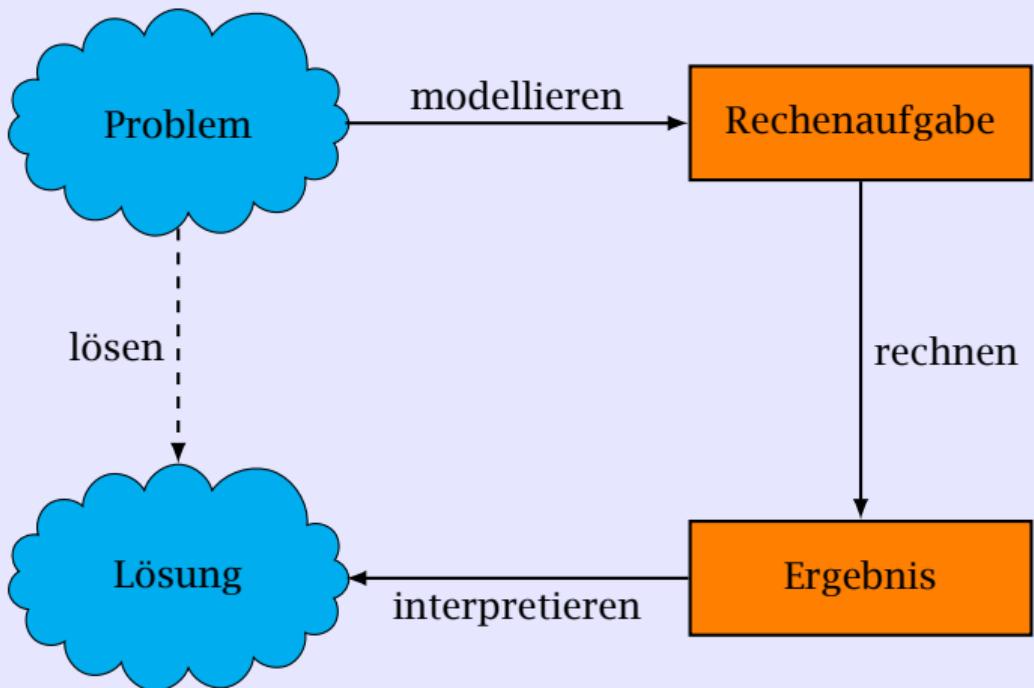
- ▶ Kreative gedankliche Leistung: arabisches Stellenwertsystem.
- ▶ Zum Vergleich: das römische System.

$$\begin{array}{r} \text{DCCCXV} \\ + \text{ MMMMDCCXI} \\ \hline = ? \end{array}$$

2. Rechnen lassen

- ▶ Idee: das Rechnen Maschinen übertragen.
- ▶ Mit der Existenz von Rechenmaschinen wird es interessant, Aufgaben in Rechenaufgaben zu verwandeln, die es von Natur aus nicht sind.
- ▶ Es gilt, eine Problemstellung durch Formeln und Rechengesetze so weitgehend zu erfassen, dass wir dem Ergebnis der Rechnung eine Lösung des Problems entnehmen können.
- ▶ Informatiker/-innen bilden abstrakte Modelle der konkreten Wirklichkeit.
- ▶ Die Wirklichkeitstreue dieser Modelle macht den Reiz und die Schwierigkeit dieser Aufgabe, und die Verantwortung der Informatiker/-innen bei der Entwicklung von Software aus.

2. Modellbildung in der Informatik



2. Beispiel: Vom Problem zur Rechenaufgabe

Problem:

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

In der Grundschule lernt man, Textaufgaben in Rechenaufgaben zu verwandeln:

$$27 * (2 + 5) + 3 * (3 + 10)$$

2. Beispiel: Von der Rechenaufgabe zum Ergebnis

Das Ergebnis der Rechenaufgabe erhalten wir durch Ausrechnen:

$$\begin{aligned} 27 * (2 + 5) + 3 * (3 + 10) &= 27 * 7 + 3 * (3 + 10) \\ &= 189 + 3 * (3 + 10) \\ &= 189 + 3 * 13 \\ &= 189 + 39 \\ &= 228 \end{aligned}$$

2. Beispiel: Vom Ergebnis zur Lösung

$$27 * (2 + 5) + 3 * (3 + 10) = 228$$

Interpretieren wir das Ergebnis der Rechenaufgabe, erhalten wir die Lösung:

Lösung:

Der Ausflug kostet 228€.

3. Eine kleine Geschichte des Rechnens

Organisation

Rechnen und
rechnen lassen

Geschichte

Gebiete

Einordnung

Überblick

- ▶ Historie des Rechnens
- ▶ Rechner und Rechnerräume

Organisation

Rechnen und
rechnen lassen

Geschichte

Gebiete

Einordnung

Überblick

3. Geschichte — Euklid



~400-300 v.Chr.: Der griechische Mathematiker Euklid von Alexandria erfindet den ersten nicht-trivialen Algorithmus: Bestimmung des größten gemeinsamen Teilers zweier Zahlen.

```
let rec ggt (m:Nat, n:Nat) : Nat =  
  if m = 0 then n  
  elif n = 0 then m  
  elif m ≥ n then ggt (m % n, n)  
  else ggt (m, n % m)
```

3. Geschichte – Abū ‘Abdallāh Muḥammad ibn Mūsā al-Khwārizmī



9. Jahrhundert: der Persische Mathematiker Abū ‘Abdallāh Muḥammad ibn Mūsā al-Khwārizmī macht das Dezimalsystem bekannt.
Von seinem Namen leitet sich der Begriff *Algorithmus* ab.

3. Geschichte — Joseph Jacquard



1801: Der Franzose Joseph Jacquard erfindet einen mechanischen Webstuhl, der mit Hilfe von Lochstreifen »programmiert« werden kann.

Organisation

Rechnen und
rechnen lassen

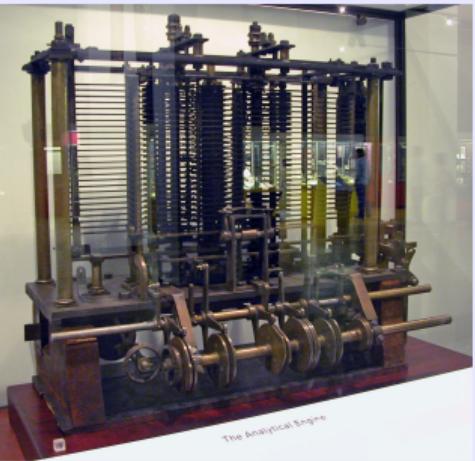
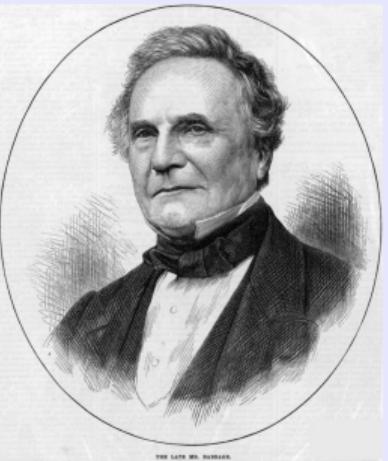
Geschichte

Gebiete

Einordnung

Überblick

3. Geschichte — Charles Babbage



1833: Der englische Mathematiker Charles Babbage konzipiert die »analytical engine«, den ersten Universalrechner. Babbage war seiner Zeit weit voraus: obwohl er keine funktionstüchtige Maschine fertigstellen konnte, gilt sein Konzept als Vorläufer des digitalen Rechners.

3. Geschichte — Ada Lovelace

1843: Die englische Mathematikerin Ada Lovelace schreibt das erste Computerprogramm — das Programm berechnet die Bernoullizahlen auf der »analytical engine«.



(Die Bernoullizahlen B_n ergeben sich als Lösungen der Gleichungen:

$$B_0 = (1 + B)^0 = 1$$

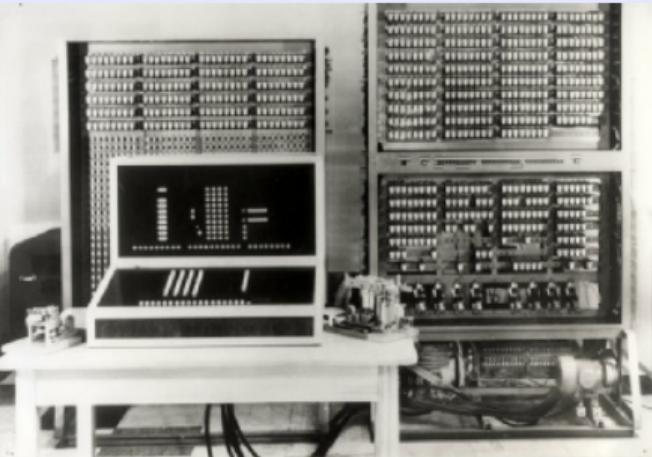
$$B_2 = (1 + B)^2 = 1 \cdot B_0 + 2 \cdot B_1 + 1 \cdot B_2$$

$$B_3 = (1 + B)^3 = 1 \cdot B_0 + 3 \cdot B_1 + 3 \cdot B_2 + 1 \cdot B_3$$

$$B_4 = (1 + B)^4 = 1 \cdot B_0 + 4 \cdot B_1 + 6 \cdot B_2 + 4 \cdot B_3 + 1 \cdot B_4 \dots$$

☞ Notationeller Trick: $B^n = B_n$.)

3. Geschichte — Konrad Zuse



1941: Der deutsche Ingenieur Konrad Zuse stellt die Z3 fertig, den ersten funktionsfähigen, frei programmierbaren, digitalen Rechner der Welt.

👉 Schauen wir uns weitere Rechner an ...

Organisation

Rechnen und
rechnen lassen

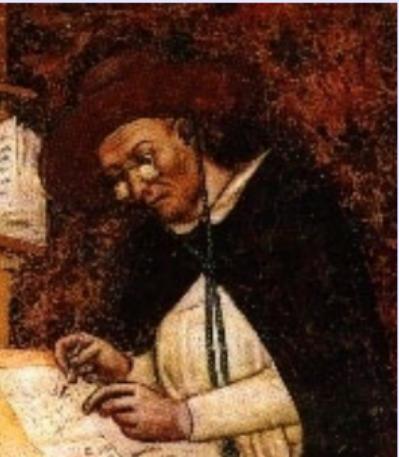
Geschichte

Gebiete

Einordnung

Überblick

3. Rechner — Hugo von Saint-Cher



- ▶ Hugo von Saint-Cher (~1200-1263) war ein französischer Mönch.
- ▶ Mit Hilfe der Mönche seines Ordens (~500 Mönche), erstellte er die erste *Konkordanz* der Bibel.
- ▶ Eine Bibelkonkordanz ist ein alphabetisches Verzeichnis aller Wörter im Text der Bibel, mit Angabe der jeweiligen Fundstellen.
- ▶ (Parallelverarbeitung mit 500 Prozessoren!)

Organisation

Rechnen und
rechnen lassen

Geschichte

Gebiete

Einordnung

Überblick

3. Rechner — Henrietta Swan Leavitt



- ▶ Henrietta Swan Leavitt (July 4, 1868–December 12, 1921) war eine US-amerikanische Astronomin.
- ▶ Sie arbeitete am Harvard College Observatory als *Computer*.
- ▶ Leavitt beobachtete und katalogisierte veränderliche Sterne.

Organisation

Rechnen und
rechnen lassen

Geschichte

Gebiete

Einordnung

Überblick

3. Menschliche Computer



- ▶ ... zeigt den »Computer Room« der NACA (Vorgänger der NASA).
- ▶ Der Begriff »Computer« ist seit dem 17. Jahrhundert im engl. Sprachraum in Gebrauch und bezeichnet »eine Person, die rechnet«.
- ▶ Bis ~1950 war Computer ein Beruf wie etwa Kaufmann oder -frau.
- ▶ Gruppen von menschlichen Computern führten langwierige Berechnungen nach festgelegten Regeln durch; die Arbeit wurde aufgeteilt, so dass Berechnungen parallel erfolgen konnten
- ▶ Sehenswert: »Hidden Figures — Unerkannte Heldinnen«.

4. Die Aufgabengebiete der Informatik

Angewandte Informatik

Technische Informatik

Praktische Informatik

Theoretische Informatik

Organisation

Rechnen und
rechnen lassen

Geschichte

Gebiete

Einordnung

Überblick

4. Technische Informatik

Organisation

Rechnen und
rechnen lassen

Geschichte

Gebiete

Einordnung

Überblick

Die *Technische Informatik* kümmert sich um die Rechenmaschine selbst:

- ▶ Rechnerarchitektur und Rechnerentwurf,
- ▶ Speichermedien,
- ▶ Übertragungskanäle,
- ▶ Sensoren.

Ziel: immer schnellere und / oder kleinere Rechner bei fallenden Preisen.

4. Theoretische Informatik

Die *Theoretische Informatik* kümmert sich um die prinzipiellen Möglichkeiten und Grenzen des Rechnens:

- ▶ Was lässt sich berechnen? Was nicht?
- ▶ Nicht berechenbare Probleme, formal unentscheidbare Probleme!
- ▶ Wie schnell lassen sich bestimmte Aufgaben lösen?
- ▶ Laufzeit von Rechenverfahren.
- ▶ Komplexität von Problemen.

4. Praktische Informatik

Zwischen den prinzipiellen Möglichkeiten des Rechnens und dem Rechner mit seinen primitiven Operationen klafft eine riesige Lücke.

Die Aufgabe der *Praktischen Informatik* ist es, den Rechner für Menschen effektiv nutzbar zu machen:

- ▶ Schließen der »semantischen Lücke« durch Programmiersprachen auf immer höherer Abstraktionsstufe;
- ▶ Benutzung der Rechner: Betriebssysteme, Benutzeroberflächen;
- ▶ Organisation großer Datenmengen: Datenbanken, Informationssysteme;
- ▶ Kommunikation der Rechner untereinander: Rechnernetze, Verteilte Systeme.

4. Angewandte Informatik

In der *Angewandten Informatik* kommt endlich der Zweck der ganzen Veranstaltung zum Zuge:

- ▶ Sie wendet
 - ▶ die schnellen Rechner,
 - ▶ die effizienten Algorithmen,
 - ▶ die höheren Programmiersprachen,
 - ▶ die Datenbanken,
 - ▶ die Rechnernetze usw.

an, um Aufgaben jeglicher Art immer weiter und vollkommener zu lösen.

- ▶ Die Informatik dringt immer weiter in verschiedenste Anwendungsbereiche vor.
- ▶ Obwohl die Informatik eine vergleichsweise junge Wissenschaft ist, haben sich interdisziplinäre Anwendungen als eigenständige Disziplinen von der »Kerninformatik« abgespalten.
 - ▶ Wirtschaftsinformatik
 - ▶ Bioinformatik

Organisation

Rechnen und
rechnen lassen

Geschichte

Gebiete

Einordnung

Überblick

Organisation

Rechnen und
rechnen lassen

Geschichte

Gebiete

Einordnung

Überblick

5. Einordnung der Informatik in die Wissenschaftsfamilie

- ▶ Grundlagen der Informatik: aus der Mathematik;
- ▶ der reale Ausgangspunkt ihrer Entwicklung: die Konstruktion der ersten Rechner.

5. Einordnung der Informatik in die Wissenschaftsfamilie

- ▶ *Naturwissenschaften*: untersuchen Phänomene, die ihr ohne eigenes Zutun gegeben sind. Die Gesetze der Natur müssen entdeckt und erklärt werden.
- ▶ *Ingenieurwissenschaften*: wenden dieses Wissen an und konstruieren damit neue Gegenstände, die selbst wieder auf ihre Eigenschaften untersucht werden müssen. Daraus ergeben sich neue Verbesserungen, neue Eigenschaften, und so weiter.

5. Einordnung der Informatik in die Wissenschaftsfamilie

Organisation

Rechnen und
rechnen lassen

Geschichte

Gebiete

Einordnung

Überblick

 Die Informatik steht eher den Ingenieurwissenschaften nahe.

- ▶ Auch sie konstruiert Systeme, die — abgesehen von denen der Technischen Informatik — allerdings immateriell sind.
- ▶ Wie die Ingenieurwissenschaften untersucht sie die Eigenschaften ihrer eigenen Konstruktionen, um sie weiter zu verbessern.
- ▶ Wie bei den Ingenieurwissenschaften spielen bei der Informatik die Aspekte der Zuverlässigkeit und Lebensdauer ihrer Produkte eine wesentliche Rolle.

6. Überblick über die Vorlesung

In der Vorlesung beschäftigt uns das »Rechnen lassen«:

- ▶ Wie können wir Aufgaben in Rechenaufgaben verwandeln?
- ▶ Wie können wir einen Rechner programmieren, so dass er diese Aufgaben ohne weiteres Zutun erledigt?

 Thematisch gehört die Vorlesung somit zur »Praktischen Informatik«.

Organisation

Rechnen und
rechnen lassen

Geschichte

Gebiete

Einordnung

Überblick

6. Überblick über die Vorlesung — Mini-F#

Im Laufe der Vorlesung werden wir schrittweise unsere eigene Programmiersprache entwickeln:

Mini-F#

eine Sprache, in der sich Rechenregeln bequem und vor allem problemnah formulieren lassen.

6. Überblick über die Vorlesung

Die Einführung einer »eigenen« Programmiersprache erlaubt es uns, viele typische Fragestellung der Praktischen Informatik zu motivieren und zu untersuchen:

- ▶ Wie lässt sich die äußere Form von Programmen festlegen?
- ▶ Wie kann man die Bedeutung eines Programms präzise definieren?
- ▶ Wie entwickelt man systematisch ein Programm?
- ▶ Wann ist ein Programm korrekt?

 Auf diese Weise wird die Programmiersprache selbst zum Objekt des Studiums. Wie gesagt, die Informatik steht den Ingenieurwissenschaften nahe.

6. Überblick über die Vorlesung — F#

- ▶ Mini-F# ist — wie der Name andeutet — stark an die Programmiersprache F# angelehnt.
 - ▶ F# ist Mitglied der .NET Sprachfamilie (Visual Basic, C#).
 - ▶ F# ist eine Multiparadigmensprache: sie unterstützt funktionale, imperative und objektorientierte Programmierung.

No one's perfect.

- ▶ Unerreichtes Ideal: Mini-F# ist eine Teilmenge von F#.
- ▶ ... gelegentlich vereinfachen wir etwas, sehr selten mogeln wir.
- ▶ Das kennen Sie aus der Schule: auch im Deutschunterricht wird nicht sofort die vollständige Grammatik der deutschen Sprache eingeführt.

6. Gliederung

- I. Einführung\Rechnen und rechnen lassen
- II. Grundlagen\Vor dem Rechnen
- III. Werte\Elementares Rechnen
- IV. Datentypen\Rechnen mit Daten
- V. Algorithmik\Rechnen mit System
- VI. Grammatiken\Konkrete Syntax
- VII. Effekte\Effektvolles Rechnen
- VIII. Objekte\Rechnen mit Objekten

Organisation

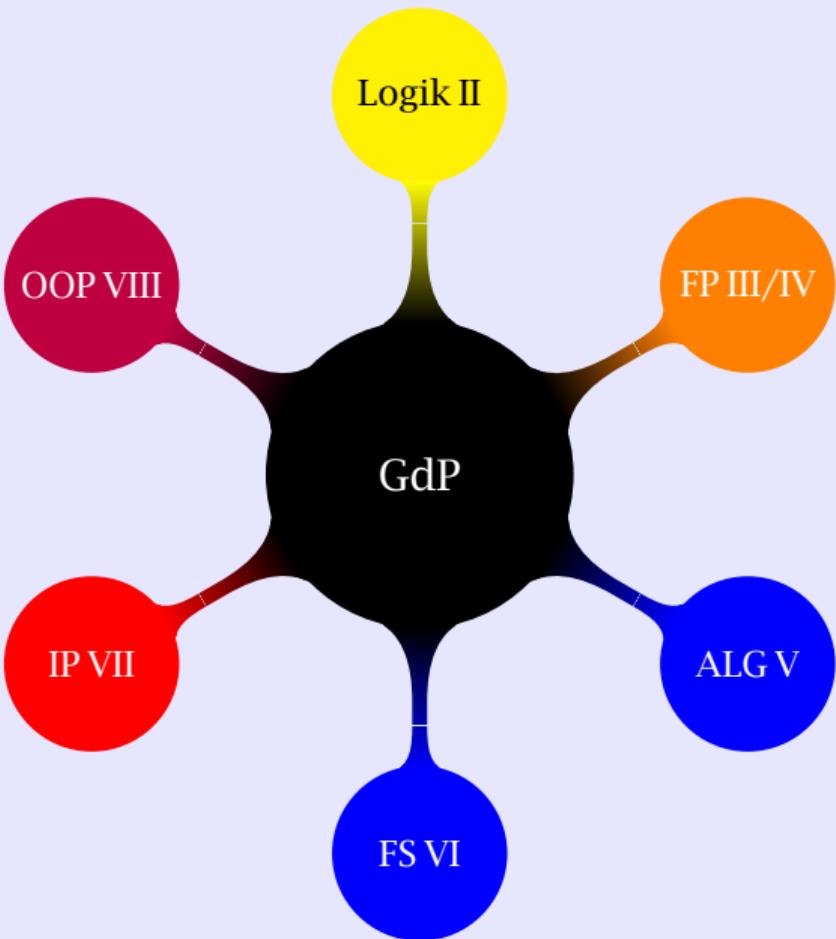
Rechnen und
rechnen lassen

Geschichte

Gebiete

Einordnung

Überblick



6. Gastrollen ...



Lisa Lista ist die ideale Studentin: Sie ist unvoreingenommen und aufgeweckt, greift neue Gedanken auf und denkt sie in viele Richtungen weiter. Kein Schulunterricht in Informatik hat sie verdorben. Lisa Lista ist 12 Jahre alt.

Harry Hacker ist ein alter Programmierfuchs, nach dem Motto: Eine Woche Programmieren und Testen kann einen ganzen Nachmittag Nachdenken ersetzen. Er ist einfallsreich und ein gewitzter Beobachter; allerdings liegt er dabei oft haarscharf neben der Wahrheit. Zu Programmiersprachen hat er eine dezidiert subjektive Haltung: Er findet die Sprache am besten, die ihm am besten vertraut ist.



Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme

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 verstündigen oder einander Hinweise geben, etwa mit der Hand oder den Augen. Auch Tricks wie die Verwendung von Spiegeln sind verboten.

Nicht verboten ist den Zergen 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 ...

Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme

6. Gliederung

7 Endliche Abbildungen

8 Syntax und Semantik

9 Baumsprachen

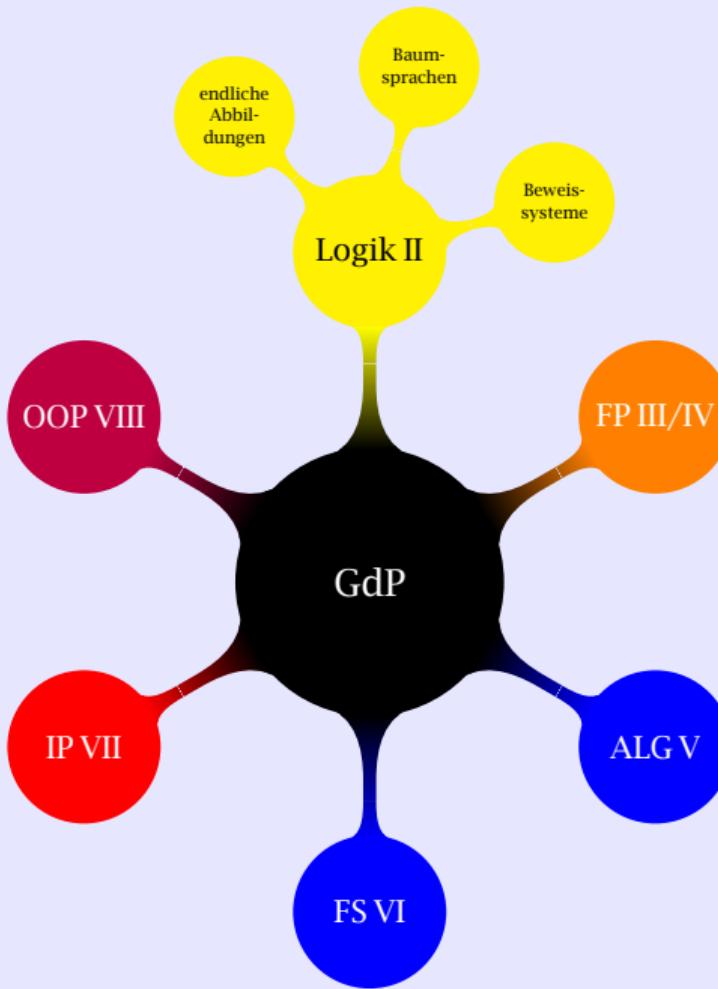
10 Beweissysteme

Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme

Endliche
AbbildungenSyntax und
Semantik

Baumsprachen

Beweissysteme

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.

Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme

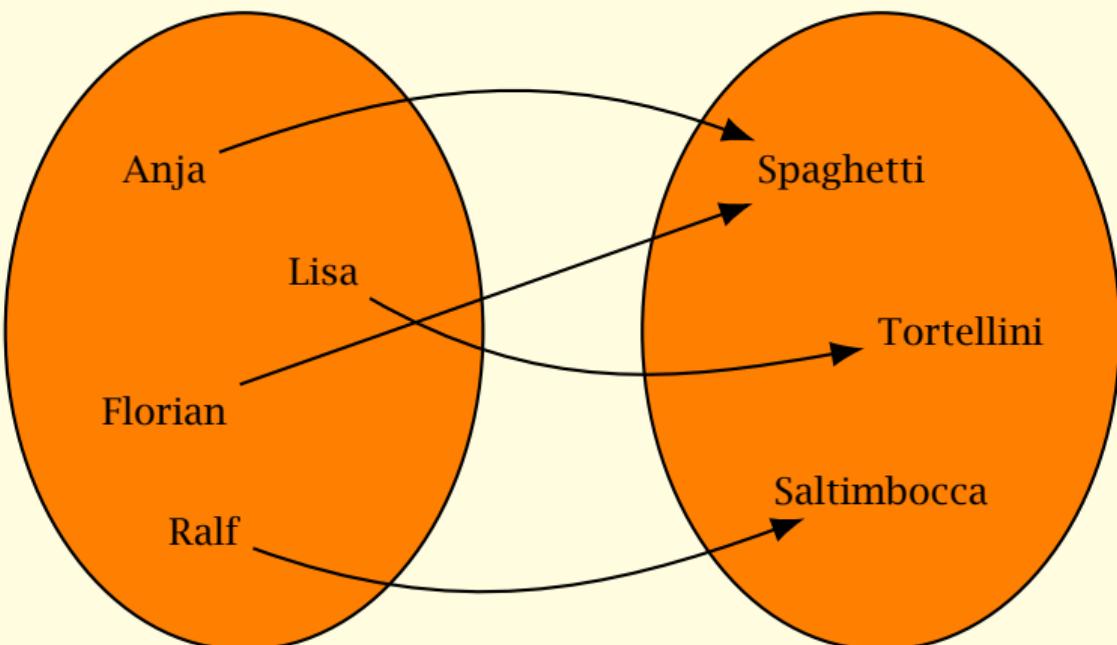
7. Endliche Abbildungen

Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme



7. Endliche Abbildungen

$\{ \begin{array}{l} Anja \mapsto Spaghetti, \\ Lisa \mapsto Tortellini, \\ Florian \mapsto Spaghetti, \\ Ralf \mapsto Saltimbocca \end{array} \}$

Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme

7. Notation endlicher Abbildungen

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

Beispiel:

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

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

$$\varphi(\text{Anja}) = \text{Spaghetti}$$

$$\varphi(\text{Lisa}) = \text{Tortellini}$$

Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme

7. Konstruktion und Manipulation endlicher Abbildungen

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

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

Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme

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 $\{x_1 \mapsto y_1, \dots, x_n \mapsto y_n\}$ als Abkürzung für $\{x_1 \mapsto y_1\}, \dots, \{x_n \mapsto y_n\}$ (wiederholte Anwendung des Kommaoperators).

- Der Kommaoperator ist aber *nicht* kommutativ: φ_1, φ_2 ist in der Regel nicht das Gleiche wie φ_2, φ_1 .

$$\begin{aligned}\{Ralf \mapsto Pizza\}, \{Ralf \mapsto Saltimbocca\} &= \{Ralf \mapsto Saltimbocca\} \\ \{Ralf \mapsto Saltimbocca\}, \{Ralf \mapsto Pizza\} &= \{Ralf \mapsto Pizza\}\end{aligned}$$

👉 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 durchnummieriert.

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

Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme

7. Notation von Sequenzen

Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme

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

7. Konstruktion und Manipulation von Sequenzen

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

- ▶ die **leere Sequenz** ϵ mit
 - ▶ $\text{dom } \epsilon := \mathbb{N}_0$;
- ▶ 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
 - ▶ $\text{dom } (s_1 \cdot s_2) := \text{dom } s_1 \cup \{i + \text{len } s_1 \mid i \in \text{dom } s_2\}$ und
 - ▶ $(s_1 \cdot s_2)(i) := \begin{cases} s_1(i) & \text{falls } i \in \text{dom } s_1 \\ s_2(i - \text{len } s_1) & \text{sonst} \end{cases}$
- ▶ die **n -fache Wiederholung** von s notiert s^n mit $s^0 := \epsilon$ und $s \cdot s^n =: s^{n+1} := s^n \cdot s$.

Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme

8. Syntax und Semantik

Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme

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

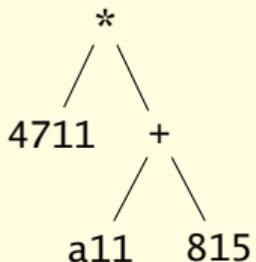
Die lexikalische und die kontextfreie Syntax bilden zusammen die ***konkrete Syntax*** einer Programmiersprache.

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

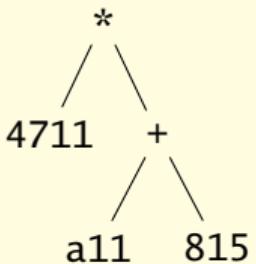
Für das Studium von Programmiersprachen ist die hierarchische Struktur eines Programms relevant.



☞ 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 (kompositionale).

9. Baumsprachen

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

Beispiel: eine einfache Form von arithmetischen Ausdrücken.

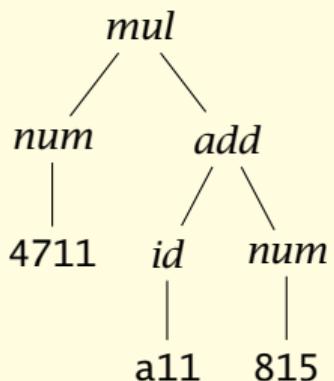
```
 $e \in \text{Expr} ::= \text{num } (\mathbb{N})$ 
  |  $\text{ident } (\text{Ident})$ 
  |  $\text{add } (\text{Expr}, \text{Expr})$ 
  |  $\text{mul } (\text{Expr}, \text{Expr})$ 
```

9. Baumsprachen: Beispiel

Beispiel für einen arithmetischen Ausdruck:

mul (num (4711), add (ident (a11), num (815)))

Dargestellt als Baum:



Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme

9. Baumsprachen: Aufbau

```
e ∈ Expr ::= num (ℕ)
          | ident (Ident)
          | add  (Expr, Expr)
          | mul  (Expr, Expr)
```

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* ($\mu\varepsilon\tau$) 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).

Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme

9. Baumsprachen: umgangssprachliche Lesart

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

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

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

Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme

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 $\text{num } (n) \in \text{Expr}$;
2. ist $x \in \text{Ident}$, dann ist $\text{ident } (x) \in \text{Expr}$;
3. sind $e_1 \in \text{Expr}$ und $e_2 \in \text{Expr}$, dann ist auch $\text{add } (e_1, e_2) \in \text{Expr}$;
4. sind $e_1 \in \text{Expr}$ und $e_2 \in \text{Expr}$, dann ist auch $\text{mul } (e_1, e_2) \in \text{Expr}$.

 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

$n \in \mathbb{N}$

$x \in \text{Ident}$

$e \in \text{Expr} ::= n$

| x

| $e_1 + e_2$

| $e_1 * e_2$

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

Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme

9. Beispiel: Klötzchenwelt — Baumsprachen

Ein weiteres Beispiel für eine Baumsprache: Folgen von Klötzchen.

```
w ∈ Wall ::= yellow-brick
  | red-brick
  | sequ (Wall, Wall)
```

Die gleiche Definition mit notationellen Freiheiten:

```
w ∈ Wall ::= 
  | 
  | w1 w2
```

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.

$$\begin{array}{c}
 \frac{\text{ident } (\text{a11}) \Downarrow 1}{\text{add } (\text{ident } (\text{a11}), \text{num } (815)) \Downarrow 816} \\
 \frac{\text{add } (\text{ident } (\text{a11}), \text{num } (815)) \Downarrow 816}{\text{mul } (\text{num } (4711), \text{add } (\text{ident } (\text{a11}), \text{num } (815))) \Downarrow 3844176}
 \end{array}$$

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:

$$\frac{\frac{ident(a11) \Downarrow 1}{add(ident(a11), num(815)) \Downarrow 816} \quad add(ident(a11), num(815)) \Downarrow 816}{mul(num(4711), add(ident(a11), num(815))) \Downarrow 3844176}$$

Beweisregeln können zu **Beweisbäumen** zusammengesetzt werden.

$$\frac{\vdots}{\frac{ident(a11) \Downarrow 1}{add(ident(a11), num(815)) \Downarrow 816} \quad add(ident(a11), num(815)) \Downarrow 816}{mul(num(4711), add(ident(a11), num(815))) \Downarrow 3844176}$$

10. Beweisbäume

Ohne Auslassungen:

$$\frac{\vdots}{\frac{\overline{num(4711) \Downarrow 4711} \quad \overline{num(815) \Downarrow 815}}{\overline{add(ident(a11), num(815)) \Downarrow 816}} \quad add(ident(a11), \overline{add(ident(a11), num(815))) \Downarrow 3844176}}}$$

Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme

10. Notationelle Freiheiten — da capo

Erlauben wir notationelle Freiheiten, gerät der Beweisbaum weniger mächtig.

$$\begin{array}{c} \vdots \\ \overline{a11 \Downarrow 1} \qquad \overline{815 \Downarrow 815} \\ \overline{4711 \Downarrow 4711} \qquad \overline{a11 + 815 \Downarrow 816} \\ 4711 * (a11 + 815) \Downarrow 3844176 \end{array}$$

10. Beweissysteme

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

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

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 $\mathcal{P}_1, \dots, \mathcal{P}_n$ Beweisbäume mit den Wurzeln ϕ_1, \dots, 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.

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

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

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:

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

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

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

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.

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

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:

$$\frac{\overline{0 \sim \text{yellow-brick}} \quad \overline{1 \sim \text{red-brick}}}{\frac{n \sim w_1 \quad n + 1 \sim w_2}{n + 2 \sim \text{sequ}(w_1, w_2)}}$$

Beweisregeln mit notationellen Freiheiten:

$$\frac{\overline{0 \sim \blacksquare} \quad \overline{1 \sim \blacksquare}}{\frac{n \sim w_1 \quad n + 1 \sim w_2}{n + 2 \sim w_1 \ w_2}}$$

10. Beispiel: Klötzchenwelt – Beweisbäume

$$\frac{\overline{1 \sim \boxed{\textcolor{red}{\square}}}}{} \qquad \frac{\overline{0 \sim \boxed{\textcolor{yellow}{\square}}} \qquad \overline{1 \sim \boxed{\textcolor{red}{\square}}}}{2 \sim \boxed{\textcolor{yellow}{\square}} \boxed{\textcolor{red}{\square}}} \\ \underline{3 \sim \boxed{\textcolor{red}{\square}} \boxed{\textcolor{yellow}{\square}} \boxed{\textcolor{red}{\square}}}$$

$$\frac{\overline{0 \sim \boxed{\textcolor{yellow}{\square}}} \qquad \overline{1 \sim \boxed{\textcolor{red}{\square}}}}{2 \sim \boxed{\textcolor{yellow}{\square}} \boxed{\textcolor{red}{\square}}} \qquad \frac{\overline{1 \sim \boxed{\textcolor{red}{\square}}}}{} \qquad \frac{\overline{0 \sim \boxed{\textcolor{yellow}{\square}}} \qquad \overline{1 \sim \boxed{\textcolor{red}{\square}}}}{2 \sim \boxed{\textcolor{yellow}{\square}} \boxed{\textcolor{red}{\square}}} \\ \underline{3 \sim \boxed{\textcolor{red}{\square}} \boxed{\textcolor{yellow}{\square}} \boxed{\textcolor{red}{\square}}} \qquad \underline{4 \sim \boxed{\textcolor{yellow}{\square}} \boxed{\textcolor{red}{\square}} \boxed{\textcolor{red}{\square}} \boxed{\textcolor{yellow}{\square}} \boxed{\textcolor{red}{\square}}}$$

10. Beispiel: Klötzchenwelt – ableitbare Formeln

Ableitbare oder beweisbare Formeln:

- ▶ $0 \sim \text{[Yellow]}$
- ▶ $1 \sim \text{[Red]}$
- ▶ $2 \sim \text{[Yellow][Red]}$
- ▶ $3 \sim \text{[Red][Yellow][Red]}$
- ▶ $4 \sim \text{[Yellow][Red][Red][Yellow][Red]}$
- ▶ $5 \sim \text{[Red][Yellow][Red][Yellow][Red][Yellow][Red]}$
- ▶ $6 \sim \text{[Yellow][Red][Red][Yellow][Red][Yellow][Red][Yellow][Red][Yellow]}$
- ▶ $7 \sim \text{[Red][Yellow][Red][Yellow][Red][Yellow][Red][Yellow][Red][Yellow][Red][Yellow]}$
- ▶ ...

Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme

10. Beispiel: Klötzchenwelt—nicht ableitbare Formeln

Ableitbare oder beweisbare Formeln:

- ▶ $0 \sim \text{[Yellow]}$
- ▶ $1 \sim \text{[Red]}$
- ▶ $2 \sim \text{[Yellow][Red]}$
- ▶ $3 \sim \text{[Red][Yellow][Red]}$
- ▶ $4 \sim \text{[Yellow][Red][Red][Yellow][Red]}$
- ▶ ...

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 $n \sim \text{[Red][Yellow]}$ und $n \sim \text{[Red][Yellow][Yellow][Red]}$ sind *nicht* beweisbar.

Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme

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.

Endliche
Abbildungen

Syntax und
Semantik

Baumsprachen

Beweissysteme



Wann geht's denn endlich mit dem Programmieren los?

Nächste Woche, Harry.

