Tutorium 09: Prolog

Paul Brinkmeier

10. Januar 2023

Tutorium Programmierparadigmen am KIT

Rückblick

Rückblick

Bisherige Themen:

- Haskell: Funktionale Programmierung
- Lambda-Kalkül: Beta-Reduktion, Church-Encodings
- Typisierung: Lambda-Typen, Let, Typinferenz
- Prolog: Logische Programmierung

Ab nächster Woche:

- Parallelprogrammierung: MPI, Java
- Design by Contract: OpenJML
- Compiler: Parser + Java ByteCode

Klausuranmeldung

- Klausur: 31.03.2023 um 11:30
- Anmeldung ab Montag bis 24.02.2023 im CAS
- Dauer: 2 Stunden
- Erlaubtes Hilfsmaterial: „Alles aus Papier“

Übungsblätter

Prolog

Maximum-Funktion

Schreibt ein Prädikat `max/3`, das das Maximum bestimmt:

```
?- max(15, 27, X).
X = 27.
```

- Schreibt die Funktion zuerst in Haskell
  - Verwendet die Guard-Notation
  - Wie viele Vergleiche benötigt man?
- Lässt sich die Haskell-Version 1:1 übersetzen?

Einteilen in Gruppen

Schreibt ein Prädikat `classify/2`, das Zahlen in folgende Gruppen einteilt:

- `negativ`: Zahlen kleiner als 0
- `interessant`: Zahlen von 0 bis 42 (inklusiv)
- `zugross`: Zahlen über 42

```
?- klasse(12, K).
K = interessant.
?- klasse(1000, K).
K = zugross.
```

Cut!

```
klasse(X, negativ)      :- X < 0.
klasse(X, interessant) :- X >= 0, X =< 42.
klasse(X, zugross)      :- X > 42.
```

- Wenn die Zahl `negativ` ist, können `X >= 0` und `X > 42` nicht stimmen
- $\leadsto$ Prolog sollte diese eigentlich überspringen
- Prolog macht aber keine Arithmetik für uns
- Aushilfe: Der Cut!

Cut!

```
klasse(X, negativ)      :- X < 0, !.
klasse(X, interessant) :- X >= 0, X =< 42, !.
klasse(X, zugross)      :- X > 42.
```

- Vermeidung redundanter Berechnungen: Cut, geschrieben `!`
- Cut kann immer einmal erfüllt werden
- Reerfüllung lässt aber die ganze Regel fehlschlagen
- Verschiedene Erklärungen:
  - Cut „schneidet Reerfüllungsbaum ab“
  - Cut entfernt Choice Points
  - Cut „verpflichtet“ Prolog zur Erfüllung einer bestimmten Regel

Cut!

```
klasse(X, negativ)      :- X < 0, !.
klasse(X, interessant) :- X >= 0, X =< 42, !.
klasse(X, zugross)      :- X > 42.
```

- Schließen sich die Tests einzelner Fälle gegenseitig aus, kann man nach ihnen gefahrlos Cuts einfügen
- „Grüner“ Cut: Ändert nicht Verhalten, nur Laufzeit des Programms
- Hier:
  - `X < 0` $\rightsquigarrow$ `X >= 0` und `X > 42` unmöglich
  - `X =< 42` $\rightsquigarrow$ `X > 42` unmöglich
  - `!` $\Leftrightarrow$ „Ab hier kann kein anderer Fall mehr eintreten, fertig“

Cut!

```
klasse(X, negativ)     :- X < 0, !.
klasse(X, interessant) :- X =< 42, !.
klasse(X, zugross).
```

- Einige Tests lassen sich *durch Cuts ersetzen*
- „Roter“ Cut: Ändert Verhalten *und* Laufzeit des Programms
- Fehleranfällig! Wenn möglich vermeiden.
- Programme ohne rote Cuts sind
  - einfacher zu verstehen.
  - etwa genauso performant.
- Faustregel: Reihenfolge der Regeln sollte austauschbar sein

Summe Vorwärts und Rückwärts

Schreibt ein Prädikat `sum/3`, sodass `sum(A, B, C)` die Gleichung

$$\texttt{A} + \texttt{B} = \texttt{C}$$

löst, wenn *zwei oder mehr Variablen belegt sind*.

- 3 Fälle.
- Verwendet `number`, um zu Prüfen ob eine Variable mit einer Zahl belegt ist.

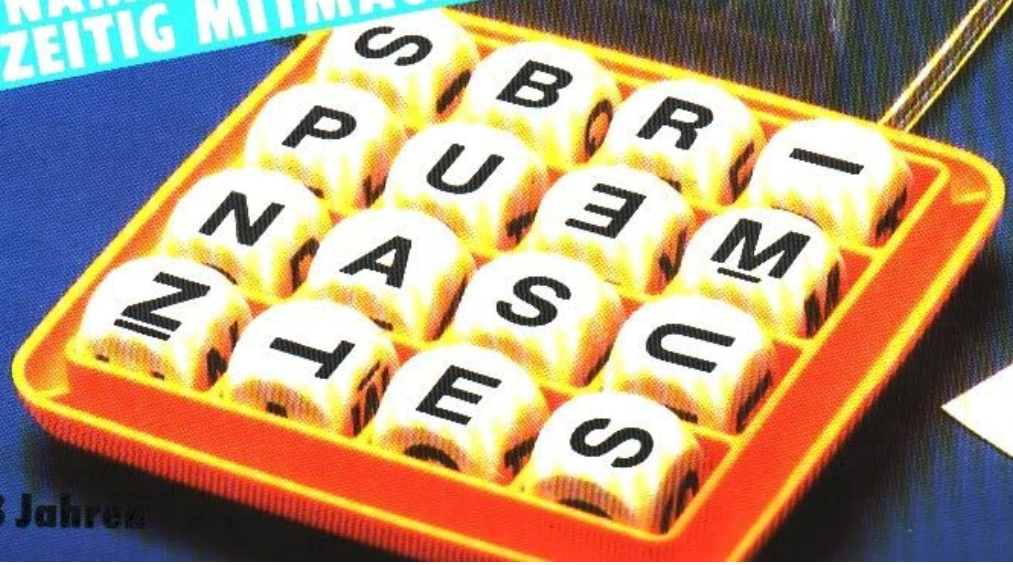
Klausur SS15: Haus vom Nikolaus

Boggle

PARKER

Boggle®

DAS DYNAMISCHE WORTSPIEL, WO ALLE SPIELER GLEICHZEITIG MITMACHEN



1-8 Spieler
(oder mehr...) ab 8 Jahren

CE

Boggle

- 16 Buchstabenwürfel werden gleichzeitig gewürfelt
- Jedes gefundene Wort gibt Punkte
  - Wort: *Mindestens drei zusammenhängende Würfel*
  - Regeln ähnlich wie Scrabble: Wörterbuch, keine Abkürzungen/Eigennamen, etc.

Boggle

- Schreibt ein Programm in Prolog, das ein Boggle löst
- Vorlage: `demos/boggle.pro`
  - Implementiert: `matchCharAt`, `neighbor` und `search`
  - Das vorgegeben Prädikat `word` definiert unsere Wortliste