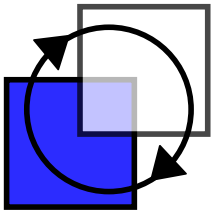
CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Prof. Bernd Finkbeiner, Ph.D.
Jana Hofmann, M.Sc.
Reactive Systems Group



Programmierung 1 (WS 2020/21)
Übungsblatt D

Lesen Sie im Buch Kapitel 3.8 - 4.7.2.

Hinweis: Über Aufgaben, die mit 🤔 markiert sind, müssen Sie eventuell etwas länger nachdenken. Falls Ihnen keine Lösung einfällt - kein Grund zur Sorge. Kommen Sie in die Office Hour, unsere Tutor:innen helfen gerne.

Höherstufige Prozeduren

Aufgabe D.1
Betrachten Sie die Deklaration

```
1 val x = fn x ⇒ (fn x ⇒ (fn y ⇒ let
2     fun x z = (z+y)*z
3     val y = 5
4   in
5     x (y+z)
6   end))
```

(a) Markieren Sie die definierenden Bezeichnerauftreten durch Überstreichen.

(b) Stellen Sie die lexikalischen Bindungen durch Pfeile dar.

(c) Geben Sie alle Bezeichner an, die in dem Ausdruck frei auftreten.

(d) Bereinigen Sie den Ausdruck durch Indizieren der gebundenen Bezeichnerauftreten.

(e) Geben Sie nun die statischen Bezeichnerbindungen an, die durch die semantische Analyse bestimmt werden. Nehmen Sie dabei an, dass die freien Bezeichner typgerecht gebunden sind.

Aufgabe D.2
Schreiben Sie eine Prozedur `reduce : int → int → int * int`, die zu zwei Zahlen $n, p \geq 2$ das eindeutig bestimmte Paar (m, k) liefert, so dass $n = m \cdot p^k$ gilt und m nicht durch p teilbar ist. 🤔

Listen

Aufgabe D.3
Betrachten Sie den Ausdruck `1 :: 2 :: nil @ 3 :: 4 :: nil`.

(a) Geben Sie die Baumdarstellung des Ausdrucks an.

(b) Geben Sie die Baumdarstellung der beschriebenen Liste an.

(c) Geben Sie die beschriebene Liste mit `[...]` an.

Aufgabe D.4
Gegeben ist folgende Deklaration: `val ls = nil::nil::nil`.

Ohne Nachzuschauen: An welchen Wert wird der Bezeichner `ls` gebunden? Was ist der Typ von `ls`?

Aufgabe D.5
Schreiben Sie eine polymorphe Prozedur `member : 'α → 'α list → bool`, die testet, ob ein Wert als Element in einer Liste vorkommt. Lösen Sie dies auf vier Arten:

(a) durch eine regelbasierte Prozedurdeklaration,

(b) mit Hilfe der vordeklarierten Prozedur `List.exists`.

(c) mit Hilfe der vordeklarierten Prozedur `List.filter`.

(d) mit Hilfe der Prozedur `foldl`.

Aufgabe D.6
Das Programm

```
1 fun or (false, false) = false
2 | or       _        = true
```

deklariert eine Prozedur `or` durch zwei überlappende Regeln. Deklarieren Sie eine zu `or` äquivalente Prozedur, die mit disjunkten Regeln formuliert ist. Verwenden Sie dabei kein Konditional.

Schaffen Sie es, die Prozedur mit nur zwei Regeln zu definieren? 🤔

Aufgabe D.7
Schreiben Sie mithilfe von `map` eine Prozedur `increment : int list → int list`, die alle Elemente einer Liste um 1 erhöht.

Aufgabe D.8
Schreiben Sie mit `foldl` eine Prozedur `facSum : int list → int`, welche zu jeder Zahl einer Liste von positiven Zahlen die Fakultät berechnet und alle aufsummiert. `facSum [2,4]` soll zu `26`, also $(2!) + (4!)$ auswerten.

Aufgabe D.9
Schreiben Sie eine Prozedur `stringSize : string → int`, die die Länge eines strings ausgibt. Dabei soll z.B. `stringSize "Hello, world!"` die Zahl 13 ausgeben.

Aufgabe D.10
Deklarieren Sie eine Prozedur `primes : int → int list`, die für $n \geq 0$ die Liste der ersten n Primzahlen in aufsteigender Ordnung liefert.

Aufgabe D.11
Schreiben Sie mit `foldl` eine Prozedur `max : int list → int`, die das größte Element einer Liste liefert. Wenn die Liste leer ist, soll die Ausnahme `Empty` geworfen werden.

Aufgabe D.12
Implementieren Sie die folgenden Prozeduren.

(a) Deklarieren Sie `enumerate : int → int → int list` so, dass für $m < n$ gilt:

$$\mathit{enumerate}\ m\ n = [m, m+1, \ldots, n-1, n]$$

(b) Schreiben Sie nun eine Prozedur `tabulateRange : int → int → (int → α) → α list`, sodass:

$$\mathit{tabulateRange}\ m\ n\ f = [f\ m, f\ (m+1), \ldots, f\ (n-1), f\ n] \qquad \text{sofern } m < n$$

(c) Versuchen Sie nun `tabulateRange` auf andere Weise zu implementieren, indem Sie `List.tabulate` und Abstraktionen nutzen. Vergewissern Sie sich anhand von einigen Beispielen, dass beide Implementierungen für gleiche Argumente auch die gleichen Listen als Ergebnis liefern.

Aufgabe D.13
Schreiben Sie eine Prozedur `components : 'α list → 'α list`, die alle Komponenten einer Liste auflistet. D.h., `components` soll Doppelauftreten löschen. Die Reihenfolge der auftretenden Elemente ist dabei egal. Beispielsweise wäre `components [3,6,6,9,3] = [3,9,6]` eine gültige Ausgabe.

Aufgabe D.14
Schreiben Sie eine Prozedur `average : real list → real`, die mithilfe von `foldl` das arithmetische Mittel aller Zahlen einer Liste berechnet wie folgt:

(a) Unter Verwendung einer Hilfsprozedur `lengthReal : α list → real`, die die Länge einer Liste als `real` ausgibt.

(b) Ohne Verwendung von `lengthReal` mit Hilfe von Faltung.

Aufgabe D.15
Gegeben sei eine beliebige Prozedur `p : α * α list → α list`. Gegeben sei außerdem folgende Prozedurdeklaration:

```
1 fun f xr = foldl (fn (a, b) ⇒ foldl p nil b) nil xr
```

Zu welchem Wert wertet eine Prozeduranwendung von f auf eine beliebige Liste immer aus?

Aufgabe D.16
Schreiben Sie eine Prozedur `isPalindrome : string → bool`, die ermittelt, ob ein `string` ein Palindrom ist. Ein String $s_0 s_1 \dots s_n$ ist genau dann ein Palindrom, wenn $s_i = s_{n-i}$ für alle $0 \leq i \leq n$ gilt. Der Einfachheit halber dürfen Sie annehmen, dass die Eingabe nur Kleinbuchstaben und keine Leerzeichen enthält.

Ein Beispiel für ein Palindrom ist `" wasitaratisaw "`.

Wenn Sie es genau wissen wollen: Schreiben Sie zuerst eine Prozedur `prepare : string → string`, die zu einem String die Version des Strings in Kleinbuchstaben und ohne Leerzeichen liefert und bauen Sie diese in `isPalindrome` ein. Ein Umweg über `chr` und `ord` erspart viele Regeln.

Aufgabe D.17 *(Love-Generator)*
Sie wurden bei RTL eingestellt, um für *Der Bachelor* einen neuen *Love-Generator* zu implementierten. Dieser berechnet die Kompatibilität zwischen zwei Menschen anhand ihrer Namen und gibt die Beziehungsaussichten in Prozent an. Ihre Aufgabe ist es nun, folgenden, wissenschaftlich höchst fundierten, Algorithmus zu implementieren: Schreiben Sie eine Prozedur `loveGen : string * string → int`, die für jeden der Strings die Ordinalszahlen aufaddiert, daraus die Summe bildet und davon den Rest der Division mit 100 zurückgibt.

(a) Schreiben Sie zunächst eine Hilfsprozedur `characterStrength : string → int`, die die Summe der Ordinalszahlen aller Buchstaben eines Strings zusammenzählt. Nutzen Sie dafür `foldl`.

(b) Nutzen Sie die Hilfsprozedur `characterStrength`, um `loveGen` wie beschrieben zu implementieren.

Aufgabe D.18 *(Verschlüsselung - rot13)*
Sie sollen eine Verschlüsselungsprozedur `rot13` für *strings* implementieren. `rot13` steht dabei für "rotiere um 13", und basiert auf der Ihnen bereits bekannten Caesar-Verschlüsselung (Blatt B). Die Idee ist es, die Zahlen aus dem Alphabet einem Pendant zuzuordnen, wobei sich deren Ordnungszahlen um genau 13 unterscheiden. Machen Sie sich zunächst klar, warum es genau 13 sein müssen.

(a) Schreiben Sie zunächst eine Prozedur `rot13char : char → char`, welche Buchstaben aus dem Alphabet ihrem Pendant zuweist, handelt es sich um einen `char`, welcher nicht im Alphabet existiert, soll dieser unverändert ausgegeben werden. Achten Sie dabei auf die Groß- und Kleinschreibung.

(b) Schreiben Sie nun eine Prozedur `rot13 : string → string`.

kNobelpreis

Sie sind an den Lösungen für die letzte kNobelaufgabe interessiert? Der/die Aufgabensteller:in bietet ein kNobeltutorium für alle Interessierte an:

Dienstag, 1.12. um 12:15
Zoom-Link:
https://cs-uni-saarland-de.zoom.us/j/98947936655?pwd=TXdBOXUxSmI2SzRDeWNsellnUEhGQT09

Ihre abgegebenen Lösungen werden bis zum Tutorium korrigiert und mit Dieters bewertet.

Das Erfüllbarkeitsproblem der Aussagenlogik ist eine der wichtigsten Fragestellungen der Informatik. So wichtig, dass man ihm einen eigenen Namen gegeben hat: SAT (aus dem Englischen "satisfiability"). Das Problem fragt, ob eine aussagenlogische Formel erfüllbar ist; also ob man alle Variablen einer Formel entweder mit `wahr` oder `falsch` belegen kann, sodass die Formel insgesamt zu wahr auswertet. Die aussagenlogische Formel $a \to b \vee c$ ist erfüllbar, indem man zum Beispiel a und b mit `wahr` und c mit `falsch` belegt. Die aussagenlogische Formel $a \wedge \neg a$ hingegen ist unerfüllbar.

Das Erfüllbarkeitsproblem der Aussagenlogik spielt, unter anderem, eine große Rolle in Bereichen der künstlichen Intelligenz und der formalen Verifikation. So kann man viele Planungsprobleme der Robotik oder Schaltkreise und deren Eigenschaften in eine aussagenlogische Formel umwandeln. Ist die Formel erfüllbar, kann daraus eine Lösung für das ursprüngliche Problem abgeleitet werden.

In dieser Aufgabe werden Sie Dieter helfen Prozeduren zu schreiben, die das Erfüllbarkeitsproblem der Aussagenlogik lösen.

(a) Dieter Schlau hat im Internet recherchiert und ist auf die Disjunktive Normalform (DNF) für aussagenlogische Formeln gestoßen. Formeln in DNF sind Disjunktionen von Konjunktionen. Beispielsweise ist die folgende Formel in DNF: $(a \wedge b \wedge \neg c) \vee (a) \vee (\neg c \wedge d)$

Dieter Schlau hat gelesen, dass man jede aussagenlogische Formel in eine äquivalente Formel in DNF umwandeln kann. Er ist siegessicher. Helfen Sie ihm. Schreiben Sie eine Prozedur

`solveDNF : string list list → bool`,

welche zu wahr auswertet, wenn eine in DNF gegebene Formel erfüllbar ist, und zu falsch, wenn die Formel unerfüllbar ist. Erlaubte Variablen sind die kleinen Buchstaben des Alphabets $a, b, \ldots$, deren Negation wird mit $!a, !b, \ldots$ dargestellt. Eine Formel in DNF wird als Liste von Listen repräsentiert, z.B., `[["a", "!b"], ["b", "!a"]]` für die Formel $(a \wedge \neg b) \vee (b \wedge \neg a)$. Bevor Sie anfangen, überlegen Sie sich eine praktische Repräsentation für Variablen und deren Negation.

(b) Dieta Schlauer ist skeptisch. Sie glaubt, dass Dieter Schlau es sich zu einfach gemacht hat. Sie recherchiert und denkt nach. Sie findet heraus, dass die Umwandlung einer beliebigen Formel zu einer Formel in DNF algorithmisch sehr aufwändig ist.

Dieta Schlauer stößt bei ihren Recherchen auf die Konjuntive Normalform (CNF). Aussagenlogische Formeln können sehr viel einfacher in diese Form überführt werden. CNF ist daher die Standardform beim Arbeiten mit aussagenlogischen Formeln. Formeln in CNF bestehen aus Konjunktionen von Disjunktionen. Beispielsweise ist die folgende Formel in CNF: $(a \vee b \vee \neg c) \wedge (a) \wedge (\neg c \vee d)$. Schreiben Sie eine Prozedur

`solveCNF : string list list → bool`,

welche zu wahr auswertet, wenn eine in CNF gegebene Formel erfüllbar ist, und zu falsch, wenn sie unerfüllbar ist.