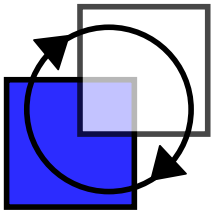
CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

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



Programmierung 1 (WS 2020/21)
Übungsblatt I

Lesen Sie im Buch Kapitel 10.3 - 11.6.

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.

Induktive Korrektheitsbeweise

Aufgabe I.1
Geben Sie für die Prozedur *rev* die Rekursionsfunktion, die Rekursionsrelation, sowie eine natürliche Terminierungsfunktion an.

Aufgabe I.2
Sei X eine Menge. Beweisen Sie mit struktureller Induktion, dass für alle Listen $xs, ys \in \mathscr{L}(X)$ gilt:

(a) $|xs \mathbin{@} ys| = |xs| + |ys|$

(b) $|rev\ xs| = |xs|$

Aufgabe I.3
Sei *filter* wie folgt definiert:

$$\begin{aligned} &filter : (X \to \mathbb{B}) \times \mathscr{L}(X) \to \mathscr{L}(X) \\ &filter(p, nil) = nil \\ &filter(p, x :: xr) = \text{if } p\ x \text{ then } x :: filter(p, xr) \text{ else } filter(p, xr) \end{aligned}$$

Beweisen Sie die Aussage $\forall xs \in \mathscr{L}(X)\, \forall p \in X \to \mathbb{B} : |filter(p, xs)| \leq |xs|$.

Aufgabe I.4
Sei X eine Menge. Beweisen Sie mit struktureller Induktion, dass für alle Listen $xs, ys \in \mathscr{L}(X)$ gilt:

(a) $xs \mathbin{@} nil = xs$

(b) $rev\ (xs \mathbin{@} ys) = rev\ ys \mathbin{@} rev\ xs$

(c) $rev\ (rev\ xs) = xs$ 🤔

Aufgabe I.5
In Kapitel 4.4 haben Sie gelernt, dass Listen mit `foldl` reversiert werden können. Jetzt können Sie die Korrektheit dieses Vorgehens beweisen.

Sei X eine Menge und sei f die Funktion $\lambda(x, xs) \in X \times \mathscr{L}(X)\,.\, x :: xs$. Für die Korrektheit der Reversion mit `foldl` muss die Gültigkeit der Aussage $\forall xs \in \mathscr{L}(X) : rev\ xs = foldl\ (f, nil, xs)$ gezeigt werden.

Suchen Sie eine geeignete Verstärkung dieser Korrektheitsaussage und beweisen Sie die Gültigkeit der Verstärkung durch strukturelle Induktion über xs. Orientieren Sie sich an der Funktion *revi* aus Aufgabe I.6.

Aufgabe I.6
Listen lassen sich mit einer endrekursiven Prozedur reversieren, die die folgende Funktion berechnet:

$revi \in \mathscr{L}(X) \times \mathscr{L}(X) \to \mathscr{L}(X)$
$revi\ (xs, ys) = (rev\ ys) \mathbin{@} xs$

Konstruieren Sie eine endrekursive Prozedur $revi'$, die die Funktion $revi$ berechnet und beweisen Sie die Korrektheit Ihrer Prozedur.

Hinweis: Überlegen Sie sich bei dieser Aufgabe zuerst, ob Sie Induktion brauchen, oder ob der Korrektheitssatz genügt. Sie benötigen die Assoziativität der Konkatenation (Proposition 10.5).

Aufgabe I.7
Seien W und X beliebige Mengen. Sei $f \in W \times X \to X$ eine beliebige Funktion. Die Prozeduren *foldl* und *foldr* sind wie folgt definiert:

$$\begin{aligned} &foldl : (W \times X \to X) \times X \times \mathscr{L}(W) \to X \\ &foldl(f, s, nil) = s \\ &foldl(f, s, x :: xr) = foldl(f, f(x, s), xr) \end{aligned}$$

$$\begin{aligned} &foldr : (W \times X \to X) \times X \times \mathscr{L}(W) \to X \\ &foldr(f, s, nil) = s \\ &foldr(f, s, x :: xr) = f(x, foldr(f, s, xr)) \end{aligned}$$

Wir wollen folgende Aussage beweisen: $\forall xs \in \mathscr{L}(W)\, \forall s \in X : foldr(f, s, xs) = foldl(f, s, rev\ xs)$

Gehen Sie dabei wie folgt vor:

(a) Beweisen Sie $\forall w \in W\ \forall xs \in \mathscr{L}(W)\, \forall s \in X : foldr(f, s, xs \mathbin{@} [w]) = foldr(f, f(w, s), xs)$

(b) Beweisen Sie $\forall xs \in \mathscr{L}(W)\, \forall s \in X : foldr(f, s, rev\ xs) = foldl(f, s, xs)$

(c) Zeigen Sie nun $\forall xs \in \mathscr{L}(W)\, \forall s \in X : foldr(f, s, xs) = foldl(f, s, rev\ xs)$.

Sie dürfen die folgende Aussage verwenden: $\forall xs \in \mathscr{L}(W) : rev(rev\ xs) = xs$

Aufgabe I.8
Schreiben Sie eine binärrekursive Prozedur $depth : \mathscr{T} \to \mathbb{N}$, die die Tiefe eines Baums liefert. Beweisen Sie die Korrektheit Ihrer Prozedur.

Aufgabe I.9
Betrachten Sie die Prozedur:

$$\begin{aligned} &p : \mathbb{N} \to \mathbb{N} \\ &p\ 0 = 0 \\ &p\ 1 = 1 \\ &p\ n = p\ (n-1) + p\ (p\ (n-1) - p\ (n-2)) && \text{für } n > 1 \end{aligned}$$

Wegen der geschachtelten Rekursion lässt sich die Terminierung der Prozedur nicht ohne Information über die Ergebnisfunktion der Prozedur zeigen. Beweisen Sie durch natürliche Induktion, dass die Prozedur die Identitätsfunktion $\lambda n \in \mathbb{N}.\ n$ berechnet.

Aufgabe I.10
Sei die folgende Prozedur gegeben:

$$\begin{aligned} &p : \mathbb{Z}^2 \to \mathbb{Z} \\ &p\ (x, y) = 0 && \text{für } x > y \\ &p\ (x, y) = x + p\ (x+1, y) && \text{für } x \leq y \end{aligned}$$

(a) Geben Sie eine natürliche Terminierungsfunktion für p an.

(b) Zeigen Sie durch Induktion: $\forall (x, y) \in \mathbb{Z}^2 : x \leq y \to p\ (x, y) = p\ (x, y-1) + y$.

Laufzeit rekursiver Prozeduren

Aufgabe I.11
Beweisen Sie die folgenden Gleichungen. Begründen Sie jeden Umformungsschritt mit den Propositionen im Buch.

(a) $\mathcal{O}\left(\frac{n^2}{7} + 789n \log n + 45n + 77\right) = \mathcal{O}\left(n^2\right)$

(b) $\mathcal{O}\left(\frac{2^n}{23} + 12n^2 + 789n + 77\right) = \mathcal{O}\left(2^n\right)$

Aufgabe I.12
Sei eine Prozedur $p : \mathbb{N} \to \mathbb{N}$ wie folgt gegeben:

$$p\ 0 = 7$$
$$p\ n = p\ (n - 1)$$

Bestimmen Sie die rekursive Darstellung der Laufzeitfunktion von p bezüglich folgender Größenfunktionen:

(a) $\lambda n.\ n$

(b) $\lambda n.\ n + 5$

(c) $\lambda n.\ 5n$

(d) $\lambda n.\ 2^n$ 🤔

Aufgabe I.13
Geben Sie eine baumrekursive Prozedur $\mathbb{N} \to \mathbb{N}$ an, deren Komplexität konstant ist.

Hinweis: Formulieren Sie die Prozedur so, dass sie ab einer bestimmten Argumentgröße nicht mehr rekurriert. Das Beispiel zeigt, dass man pathologische Prozeduren konstanter Komplexität konstruieren kann, die für alle praktisch relevanten Argumente sehr hohe Laufzeiten haben.

Aufgabe I.14
Geben Sie eine Prozedur $p : \mathscr{L}\left(\mathbb{N}\right) \to \{0\}$ an, die die folgenden Eigenschaften besitzt:

1. $\lambda xs.\ |xs|$ ist eine zulässige Größenfunktion für p.
2. Bezüglich dieser Größenfunktion hat p die Laufzeitfunktion $\lambda n.\ n + 1$.
3. Für alle $n \in \mathbb{N}$ gibt es eine Liste xs der Länge n, für die p die Laufzeit 1 hat.

Aufgabe I.15
Nachdem er über die schlechten Laufzeiten seiner Prozeduren so enttäuscht war, hat Dieter Schlau sich vorgenommen, Laufzeiten ganz genau zu verstehen und hat sich überlegt, dass man die Dominanzrelation zwischen zwei Funktionen aus OF auch so definieren könnte: 🤔

- Seien $f, g \in OF$. Dann:

$$g \preceq' f :\iff \exists n_0 \in \mathbb{N} : \exists c \in \mathbb{N} : \forall n > n_0 : g\ n < c(f\ n)$$

- Oder auch so: Seien $f, g \in OF$. Dann

$$g \preceq'' f :\iff \exists n_0 \in \mathbb{N} : \exists c \in \mathbb{N} : \forall n > n_0 : g\ n \leq (f\ n) + c$$

Beweisen oder widerlegen Sie, dass Dieter Recht hat!

kNobelpreis

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

Mittwoch, 13.1. um 12:15
Zoom-Link:
https://cs-uni-saarland-de.zoom.us/j/98947936655?pwd=TXdBOXUxSmI2SzRDeWNsellnUEhGQT09

Ihr Freund Dieter Schlau steht am Rande der Verzweiflung.

Fast jeden Tag findet er eine neues Easter-Egg in seinem Lieblings-SML-Interpreter *SOSML*[1], welches er einfach nicht versteht. Nach unzähligen schlaflosen Nächten, in denen er sich über den tieferen Sinn von Fehlermeldungen wie *„RAINBOW!“*, *„Being an interpreter is suffering.“* oder *„Pipiru piru piru pipiru pii!“* den Kopf zebrochen hat, hat er Sie um Hilfe gebeten, um nun ein für alle Mal Abhilfe zu schaffen.

In einer, wie Sie finden, zu gut passenden Wendung haben Sie und Ihr Freund Dieter eine dubiöse Liste von einer noch dubiöseren Gestalt „Inkubator“ zugespielt bekommen, auf welcher angeblich zu jeglichen allgemein bekannten Redewendungen und Zitaten deren Ursprünge verzeichnet sind. Schnell stellen Sie fest, dass Ihre Zweifel nicht ganz grundlos waren: Während manche der Redewendungen tatsächlich wie auf der Liste in SOSML als Fehler auftauchen, gibt der Interpreter manche Redewendungen scheinbar auch leicht abgeändert wieder. Nach einigem Grübeln und Beraten mit Dieter, haben Sie sich daher folgendes überlegt:

Zu zwei Strings `A` und `B` sei deren Ähnlichkeit $L(\texttt{A},\texttt{B})$ dadurch bestimmt, wie viele Zeichen man im String `A` mindestens ändern, hinzufügen oder entfernen muss, um zum String `B` zu gelangen. So gilt beispielsweise

$$L(\texttt{NERV},\texttt{SEELE}) = 4,$$

da aus dem String `NERV` die Zeichen `N`, `R` und `V` geändert und ein zusätzliches `E` angefügt werden müssen, um zum String `SEELE` zu gelangen.
Um nun den Ursprung eines Easter-Eggs `E` herauszufinden, möchte Dieter einen Eintrag `F` in der dubiösen Liste von „Inkubator“ finden, dessen Ähnlichkeit zu `E` höchstens der Hälfte der Länge von `E` entspricht. (Dieter gibt sich mit einem beliebigen Eintrag zufrieden, sollte sich ein Easter-Egg nicht eindeutig einem Ursprung zuordnen lassen.)

Getrieben von immer neuen Easter-Eggs hat sich Dieter schon eine Prozedur

$$\texttt{eggsource} : (\texttt{string} * \texttt{string})\ \texttt{list} \to \texttt{string} \to (\texttt{string} * \texttt{string})\ \texttt{option}$$

überlegt, die, gegeben die Liste von „Inkubator“ und ein Easter-Egg, dieses Problem für ihn löst:

```
1  fun comp t v = if t = v then 0 else 1
2
3  fun L [] v = length v
4  | L t [] = length t
5  | L (t::ts) (v::vs)
6  = Int.min (L ts (v::vs) + 1,            (* Zeichen in T entfernen *)
7  (Int.min (L (t::ts) vs + 1,       (* Zeichen in V entfernen *)
8  (L ts vs + (comp v t)))))   (* Zeichen anpassen/beibehalten *)
9
10 fun eggsource [] _ = NONE
11 | eggsource ((a,b)::xs) e = if
12 L (explode a) (explode e) <= String.size e div 2
13 then SOME (a,b)                      (* Ursprung des Easter Eggs zurückgeben
   ↳  *)
14 else eggsource xs e                  (* Nächsten Listeneintrag probieren *)
```

Leider ist diese Lösung sehr langsam und daher für Dieter unbrauchbar. In einem letzten Verzweiflungsakt wendet er sich daher an Sie, ihm mit einer schnelleren Lösung zu helfen. Versuchen Sie, für möglichst lange Listen mit möglichst langen Redewendungen Lösungen zu finden. Können Sie für Listen von 100 Redewendungen mit je 100 Zeichen pro Redewendung in weniger als 5 Sekunden in SOSML eine Lösung berechnen? Für 1000? 10000? Falls nein, warum nicht?

[1] https://sosml.org

Nachdem Sie Dieter mit Ihrer neuen Prozedur schon deutlich beruhigen konnten, bittet er Sie nun nur noch um eine Kleinigkeit, wie Dieter meint: Erweitern Sie Ihre Prozedur `eggsource` so, dass sie zu einer ausgegebenen Redewendung auch ausgibt, wie man von der Fehlermeldung in SOSML zur Redewendung gelangen kann. Zum Beispiel sollte bei der Eingabe `NERV` und `SEELE` die Folge `c1 c3 c4 a5` ausgegeben werden; bei Eingabe `CCAABBB` und `AAABBBC` sollte `d1 c2 a7` ausgegeben werden.

Sollten Sie zusätzlich eine besonders umfangreiche Referenz an im Interpreter vorkommenden Redewendungen und deren Ursprünge zusammenstellen, wird Dieter dies natürlich belohnen. (Schließlich ist die Liste von „Inkubator“ doch sehr dubiös.) Noch mehr freut sich Dieter natürlich, wenn sie Beispielprogramme mitliefern, die entsprechende Easter-Eggs hervorrufen.

Achtung. Da Dieter schon verzweifelt genug ist, möchte er sich nicht noch am Verstehen einer Lösung die Zähne ausbeißen, welche Nebeneffekte verwendet. Vermeiden Sie diese in Ihrer Lösung daher. Vermeiden sie insbesondere Konstrukte, die explizit oder implizit `ref`*,* `!`*,* `:=` *oder* `array` *verwenden.*