Prof. Dr. Helmut Seidl, Ralf Vogler, Stefan Schulze Frielinghaus

OCaml-Hausaufgaben die nicht kompilieren oder nicht in annehmbarer Zeit terminieren werden nicht gewertet! Solltet Ihr einmal in die Situtation kommen, dass sich ein Fehler in einen Teil Eurer Abgabe eingeschlichen hat, der zur Folge hat, dass sich die gesamte Datei nicht mehr kompilieren lässt, dann kommentiert diesen Teil aus und ersetzt ihn durch den Standardwert aus der Angabe (z.B. let f x = todo ()). Andernfalls wird die gesamte OCaml-Hausaufgabe mit Null Punkten gewertet.

Aufgabe 10.1 OCaml-Hausaufgabe: Boolesche Algebra

Im folgenden sei ϕ ein beliebiger boolescher Ausdruck sowie x und y boolesche Variablen. Verwenden Sie folgenden OCaml-Typen um boolesche Ausdrücke abzubilden:

```
type var = int
type t = True | False | Var of var | Neg of t |
         Conj of t Set.t | Disj of t Set.t
```

♦ Schreiben Sie eine Funktion `eval : (var -> bool) -> t -> bool`, die als ersten Parameter eine Abbildung von Variablen nach `true` bzw. `false` sowie als zweites Argument einen booleschen Ausdruck vom Typen `t` erwartet. Das Ergebnis der Funktion soll die Auswertung des booleschen Ausdruckes unter der gegebenen Variablenbelegung sein.

♦ Schreiben Sie eine weitere Funktion `push_neg : t -> t`, die jedes vorkommende `Neg` in einem Ausdruck so weit wie möglich nach "unten" propagiert. Zwei aufeinanderfolgende `Neg`'s müssen entfernt werden sowie ein `Neg True` muss zu einem `False` umgewandelt werden. Analog ein `Neg False` zu `True`.

$$\frac{\neg\neg\phi}{\phi} \qquad \frac{\neg\text{True}}{\text{False}} \qquad \frac{\neg\text{False}}{\text{True}} \qquad \frac{\neg\bigwedge_{i=1}^{n}\phi_i}{\bigvee_{i=1}^{n}(\neg\phi_i)} \qquad \frac{\neg\bigvee_{i=1}^{n}\phi_i}{\bigwedge_{i=1}^{n}(\neg\phi_i)}$$

Zum Beispiel

```
push_neg (Neg (Conj (Set.from_list [Var 1; True; Var 2; Neg (Var 3)])))
```

liefert als Ergebnis den Wert

```
Disj (Set.from_list [Neg (Var 1); False; Neg (Var 2); Var 3])
```

♦ Schreiben Sie eine Funktion `cleanup : t -> t`, die einen Ausdruck nach folgenden Regeln "aufräumt":

a) Der Konstruktor einer einelementigen Konjunktion muss verworfen werden.

$$\frac{\bigwedge\{\,\phi\,\}}{\phi}$$

Das heißt `cleanup (Conj (Set.from_list [e]))` liefert `e` für einen beliebigen Ausdruck `e : t`. Analog dazu muss der Konstruktor einer einelementigen Disjunktion verworfen werden.

b) Eine Konjunktion, die eine Konjunktion enthält muss zu einer Konjunktion zusammengefasst werden.

$$\frac{\bigwedge\{\,\phi_1,\ldots,\phi_n,\bigwedge\{\,\phi'_1,\ldots,\phi'_m\,\}\,\}}{\bigwedge\{\,\phi_1,\ldots,\phi_n,\phi'_1,\ldots,\phi'_m\,\}}$$

Zum Beispiel

```
cleanup (Conj (Set.from_list [e1; Conj (Set.from_list [e2; e3])))
```

liefert als Ergebnis einen Wert der äquivalent zu folgendem ist

```
Conj (Set.from_list [e1; e2; e3])
```

für beliebige Ausdrücke `e1:t, e2:t, e3:t`. Das gleiche muss auch für eine Disjunktion anstelle einer Konjunktion gelten.

Beachten Sie, dass die Regeln a) und b) so oft wie möglich angewendet werden müssen.

♦ Schreiben Sie eine weitere Funktion `simplify : t -> t`, die einen booleschen Ausdruck entgegen nimmt und als Wert einen *maximal* vereinfachten Ausdruck liefert. Ein Ausdruck ist *maximal* vereinfacht sofern folgende Regeln nicht mehr anwendbar sind:

$$\frac{\phi \land \text{False}}{\text{False}} \qquad \frac{\phi \lor \text{True}}{\text{True}}$$

$$\frac{\phi \land \text{True}}{\phi} \qquad \frac{\phi \lor \text{False}}{\phi}$$

$$\frac{x \land \neg x}{\text{False}} \qquad \frac{x \lor \neg x}{\text{True}}$$

$$\frac{\bigwedge \emptyset}{\text{True}} \qquad \frac{\bigvee \emptyset}{\text{False}}$$

$$\frac{x \land (x \lor y)}{x} \qquad \frac{x \lor (x \land y)}{x}$$

Bonus-Aufgaben für die Funktion `simplify`:
Erweitern Sie die Funktion `simplify` um das Extremalgesetz angewendet auf beliebige Ausdrücke, die *syntaktisch* gleich sind bis auf ein vorangestelltes Nicht-Zeichen:

$$\frac{\phi \land \neg\phi}{\text{False}} \qquad \frac{\phi \lor \neg\phi}{\text{True}}$$

Beobachtung: Um das Extremalgesetz in voller Gänze umzusetzen müsste nicht nur auf *syntaktische* Gleichheit von Ausdrücken geachtet werden, sondern auch auf *semantische* Gleichheit was ungleich schwerer ist. Zum Beispiel:

$$(x \lor y) \land (\neg x \land \neg y) \qquad \Longleftrightarrow \qquad (x \lor y) \land (\neg(x \lor y))$$

Hier ist der zweite Konjunkt $\neg x \land \neg y$ gleich dem ersten Konjunkt $x \lor y$ nur negiert, was nicht syntaktisch aus dem Ausdruck hervorgeht.

Erweitern Sie die Funktion `simplify` um das Absorbtionsgesetz angewendet auf beliebige Ausdrücke, die *syntaktisch* gleich sind:

$$\frac{\phi \land (\phi \lor \phi')}{\phi} \qquad \frac{\phi \lor (\phi \land \phi')}{\phi}$$

♦ Schreiben Sie eine Funktion `is_simplified : t -> bool`, die überprüft ob ein gegebener Ausdruck *maximal* vereinfacht ist oder nicht.

♦ Schreiben Sie eine Funktion `to_nnf : t -> t`, die einen Ausdruck in Negationsnormalform (NNF) überführt. Verwenden Sie dafür die Funktionen `push_neg`, `cleanup` und `simplify`. Zum Beispiel

```
to_nnf (Neg (Conj (Set.from_list [Var 1; True; Var 2])))
```

liefert als Ergebnis den Wert

```
Disj (Set.from_list [Neg (Var 1); Neg (Var 2)])
```

Beachten Sie, dass ein Ausdruck ϕ mit `True` $\neq \phi \neq$ `False` in NNF kein `True` oder `False` als Unterausdruck enthalten darf! Ein Ausdruck in NNF muss maximal aufgeräumt und vereinfacht sein.

♦ Schreiben Sie nun zwei Funktionen `to_dnf : t -> t` bzw. `to_cnf : t -> t` die einen beliebigen booleschen Ausdruck in einen Ausdruck in disjunktiver bzw. konjunktiver Normalform überführt. Bringen Sie dafür den gegeben Ausdruck in NNF und wenden so oft wie möglich das Distributivgesetz $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$ bzw. $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$ an. Danach muss der Ausdruck noch maximal aufgeräumt und vereinfacht werden.

Eine Datei `ha6.ml` mit den Angaben finden Sie wie gewohnt auf Moodle. Zur Abgabe melden Sie sich mit Ihrer TUM-Kennung auf `https://vmnipkow3.in.tum.de` an und laden ihre `ha6.ml`-Datei (nicht umbenennen!) hoch. Die Ergebnisse der Tests sind sichtbar sobald die Abgabe abgearbeitet wurde. Die endgültige Bewertung erfolgt aber erst nach der Frist.

Die Datei `batteries.ml` enthält die bisher implementierten Funktionen und ist auch in der Testumgebung verfügbar (muss nicht hochgeladen werden).

Aufgabe 10.2 OCaml-Tutoraufgabe: Module und Funktoren

1. Was ist ein Modul und was können Module beinhalten?
2. Welchen Effekt hat die Angabe einer Signatur?
3. Was ist ein Funktor? Welches Problem lässt sich damit lösen?
4. Diskutieren Sie die folgenden (gekürzten) Versionen unseres Map-Moduls (einfaches Modul, Funktor mit ein, Funktor zwei Parametern) und überlegen Sie wie sich die Definitionen und Signaturen unterscheiden würden.
5. ```
   module type S = sig
     type t
     val show : t -> string
   end
   module Map0 = struct
     type ('k,'v) t = ..
     let empty = ..
     let show sk sv m = ..
   end
   module Map1 (K: S) = struct
     type k = K.t
   ```
   ```

```
    type 'v t = ..
    let empty = ..
    let show sv m = ..
  end
  module Map2 (K: S) (V: S) = struct
    type k = K.t
    type v = V.t
    type t = ..
    let empty = ..
    let show m = ..
  end
```

6. Was ist der Unterschied zwischen `open` und `include`?