Prof. Dr. Peter Koepke, Regula Krapf
A. Pick, A. Ribelles, T. Weiß

# Syntax und Semantik der Aussagenlogik

## Aufgaben

Mit dem Befehl `ocaml` lässt sich der Interpreter starten. Anschließend sollte man über den Befehl (man beachte, dass das Zeichen # hier tatsächlich eingegeben werden muss)

```
#use "init.ml";;
```

alles Nötige einbinden. Um aussagenlogische Formeln parsen zu können, muss der standardmäßig auf *First Order Logic* eingestellte Parser zunächst umgestellt werden:

```
# let default_parser = parse_prop_formula;;
```

**Aufgabe 1.** Betrachte die Funktionen `odd` und `even` auf Seite 610:

```
# let rec even n = if n = 0 then true else odd (n-1)
  and     odd n = if n = 1 then true else even (n-1);;
```

Teste damit, ob die Zahl 3 gerade oder ungerade ist. Wie sind die Beobachtungen zu erklären?

**Aufgabe 2.** Schreibe eine rekursive Funktion, die `fib n`, die als Eingabe eine natürliche Zahl bekommt und die $n$-te Fibonacci-Zahl ausgibt.

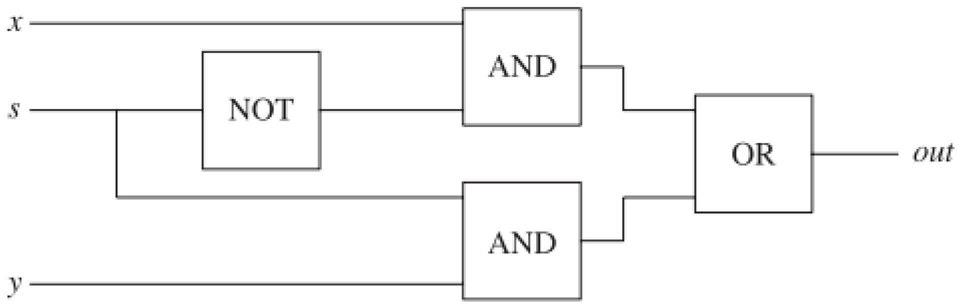
**Aufgabe 3.** Schreibe einen Destruktor `dest_imp` für Aussagen der Form $\varphi \rightarrow \psi$, der das **Paar** $(\varphi, \psi)$ zurückgibt, wobei $\varphi$ und $\psi$ aussagenlogische Formeln sind. Schreibe anschließend jeweils einen Destruktor `antecedent` bzw. `consequent` für die Implikation, der zu einer Implikation das Antezedenz bzw. das Sukzedens ausgibt.
Schreibe nun einen Destruktor `conjuncts`, der zu einer Aussage der Form $\varphi_1 \wedge \varphi_2 \wedge \dots \varphi_n$ alle Konjunkte in einer Liste zurückgibt. Ist der Input konjunktionsfrei, so soll er direkt ausgegeben werden.
*Hinweis: Nutze pattern-matching für diese Aufgaben. Die Funktion, die die Konjunkte liefert, sollte rekursiv implementiert werden.*

**Aufgabe 4.** Experimentiere mit den Funktionen `print_truthtable`, `dnf` und `cnf`. Alle bekommen eine Formel als Input. Warum könnte die Formel

$$p \wedge (q \vee r) \vee q \wedge r$$

auch als *Majorität* oder *Mehrheitsfunktion* bezeichnet werden?

Abbildung 1: Schaltkreis zu Aufgabe 5. S. 64 im Buch.




**Aufgabe 5.** Übertrage den Schaltkreis auf Seite 64 (s. Abb. 1) in eine logische Formel in interner Darstellung (`Or(p,q)`, `And(p,q)`, usw.), also ohne die Symboldarstellung (mit $\vee, \wedge, \ldots$) zu benutzen. Gib die Wahrheitstabelle aus. Welche Rolle spielt $s$?

**Aufgabe 6.** Betrachte die Funktion `tautology`:

```
# let tautology fm = onallvaluations (eval fm) (fun s -> false) (
    atoms fm);;
```

Diese Funktion überprüft, ob die übergebene Formel eine Tautologie ist. Schreibe nun Funktionen, die für eine gegebene Formel überprüfen, ob diese erfüllbar bzw. unerfüllbar ist.
*Hinweis: Es ist einfacher, zunächst auf Unerfüllbarkeit zu testen und dafür den schon bekannten Code zu modifizieren.*

**Aufgabe 7.** Experimentiere mit der Funktion `psubst`. Was geschieht, wenn man Formel substitutieren möchte, die nicht atomar sind?

```
# let psubst subfn = onatoms (fun p -> tryapplyd subfn p (Atom p));;
```

**Aufgabe 8.** Sei `fm` eine aussagenlogische Formel. Die zu `fm` *duale* Formel entsteht, indem man die Wahrheitswerte aller Atome umdreht und jedes $\wedge$ durch ein $\vee$ und umgekehrt simultan ersetzt. Implementiere eine Funktion, die zu einer gegebenen Formel die duale Formel ausgibt.
*Hinweis: pattern-matching könnte hilfreich sein.*