Hauptseminar math. Logik
Sommersemester 2014
Übung Nr.2

U. Matzner, J. Speer, J. Jentgens

21. Mai 2014

Listenoperationen und Ocaml Äquivalenzrelationen

(i) Schreibe eine Funktion, die als Input eine Liste erhält und diese in umgekehrter Reihenfolge zurückgibt.

(ii) Schreibe eine Funktion, die eine Liste l mit Ganzzahlen erhält und drei Listen $l_1 = \{x \in l : x < 10\}$, $l_2 = l \cap \{10, \ldots, 20\}$ und $l_3 = \{x \in l : x > 20\}$ zurückgibt.

(iii) Erzeuge eine Äquivalenzrelation auf einer Domäne, die mindestens aus drei Äquivalenzklassen besteht. Fasse zwei dieser Äquivalenzklassen zu einer Äquivalenzklasse zusammen und speichere die entstehende Äquivalenzrelation.

Lösung:

(i)
```
let rec myappend l l' =
match l with
[] -> l'
| h::t -> h::(append t l');;


let rec myturn list =
match list with
[] ->[]
|[x]->[x]
|x::oth -> append (turn(oth)) [x];;
```

(ii)
```
let mysplit list =
let l_1 = filter (fun x -> (x < 10)) list in
```

```
3 let l_2 = filter (fun x -> (x >= 10 & x <= 20) list in
4 let l_3 = filter (fun x -> (x > 20) list in
5 l_1 l_2 l_3;;
```

(iii)
```
let eq = equate (0,1) unequal;;
let eq = equate (2,3) eq;;
let eq = equate (4,5) eq;;
equivalent eq 0 3;;
let eq = equate (1,2) eq;;
equivalent eq 0 3;;
```

Davis Putnam und DPLL

(i) Gib eine möglichst kleine Formel und ein Literal an, für die *resolve_on* die Klauselmenge mindestens verdoppelt.

(ii) Verändere die Formel aus (i) so, dass auch *resolution_rule* die Kardinalität der Klauselmenge mindesten verdoppelt.

(iii) Implementiere die zweite Regel des Davis Putnam Algorithmus! Schreibe also eine Funktion, die für Aussagen als Listen von Listen (interpretiert als CNF) alle Klauseln löscht, die ein Literal enthalten, das nur positiv bzw. nur negativ vorkommt.

(iv) Manipuliere den Code von *dpli* so, dass der Trail in jeder Iteration ausgegeben wird.

(i)
```
resolve_on <<p>> [[<<p>> ; <<q>>] ; [<<p>> ; <<~q>>] ;
[<<p>> ; <<r>>] ; [<<p>> ; <<~r>>] ; [<<~p>> ; <<q>>] ;
[<<~p>> ; <<~q>>] ; [<<~p>> ; <<r>>] ; [<<~p>> ; <<~r>>]] ;;
```

```
resolution_rule [[<<p>> ; <<q>> ; <<r>>] ;
[<<p>> ; <<q>> ; <<~r>>] ; [<<p>> ; <<~q>> ; <<r>>] ;
[<<p>> ; <<~q>> ; <<~r>>] ; [<<~p>> ; <<q>> ; <<r>>] ;
[<<~p>> ; <<q>> ; <<~r>>] ; [<<~p>> ; <<~q>> ; <<r>>] ;
[<<~p>> ; <<~q>> ; <<~r>>]] ;;
```

(ii) Siehe Harrisons Code...

(iv)

Stalmarck

(i) Experimentiere mit der Funktion *triggers*!

(ii) Schreibe eine Funktion, die überprüft, ob zwei Elemente einer Domäne in der gleichen Äquivalenzklasse bezüglich einer gegebenen Äquivalenzrelation liegen und verwende dabei die funktion *canonize*. Teste die Funktion an deinen Ergebnissen aus der ersten Aufgabe!

(iii) Schau Dir die Implementierung der Toplevelfunktion *stalmarck*[1] an und finde eine Formel, für die die Methode kein Ergebnis liefert. Überprüfe Dein Ergebnis!

```
let stalmarck fm =
let include_trig (e,cqs) f =
(e |-> union cqs (tryapplyl f e)) f in
let fm' = psimplify(Not fm) in
if fm' = False then true else if fm' =
   True then false else
let p,triplets = triplicate fm' in
let trigfn = itlist (itlist include_trig ** trigger)
   triplets undefined and vars =
map (fun p -> Atom p) (unions(map atoms triplets)) in
saturate_upto vars 0 2 (graph trigfn) [p,True];;
```

(iv) Für welche Typen von Formeln ist Stalmarcks Algorithmus geeignet?

Lösung:

(ii)

```
let myequal eq (x, y) =
(canonize eq x == canonize eq y);;
```

(iii)

```
stalmarck <<(p/\q) ==> (r/\s)>>;;
```

(iv) Für welche Typen von Formeln ist Stalmarcks Algorithmus geeignet?

[1] „Handbook of Practical Logic and Automated Reasoning“, John Harrison - Seite 98

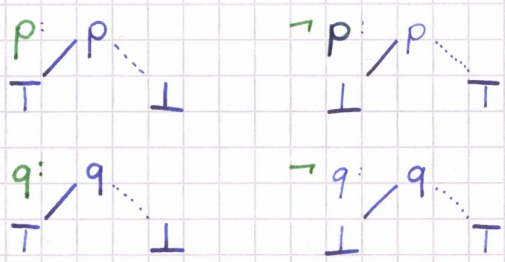
BDD

Erstelle auf dem Papier, schrittweise dem Algorithmus folgend, ein BDD zu der Formel $(p \vee q) \rightarrow (s \wedge q)$.

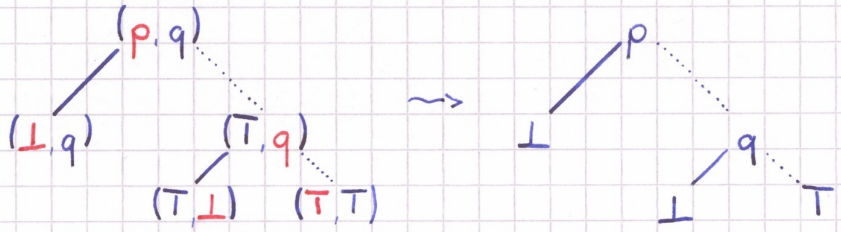
BDD: $(p \vee q) \rightarrow (s \wedge q)$

(1) $p \vee q \rightsquigarrow \neg(\neg p \wedge \neg q)$

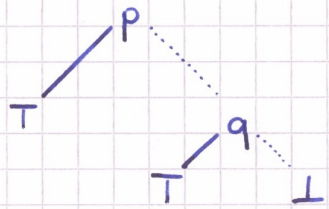
(1.1) p: $\neg p$: q: $\neg q$:



(1.2) $\neg p \wedge \neg q$:

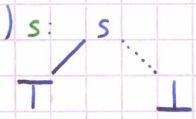


(1.3) $\neg(\neg p \wedge \neg q)$

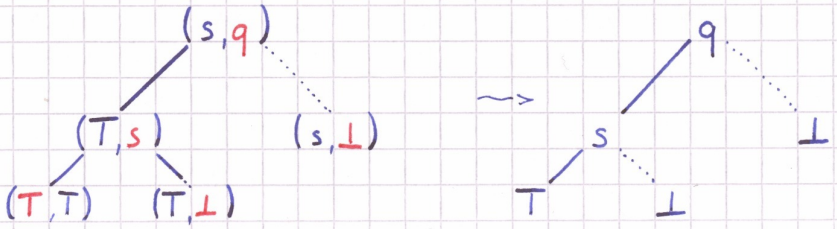


(2) $s \wedge q$

(2.1) s:

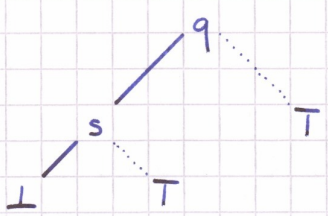


(2.2) $s \wedge q$

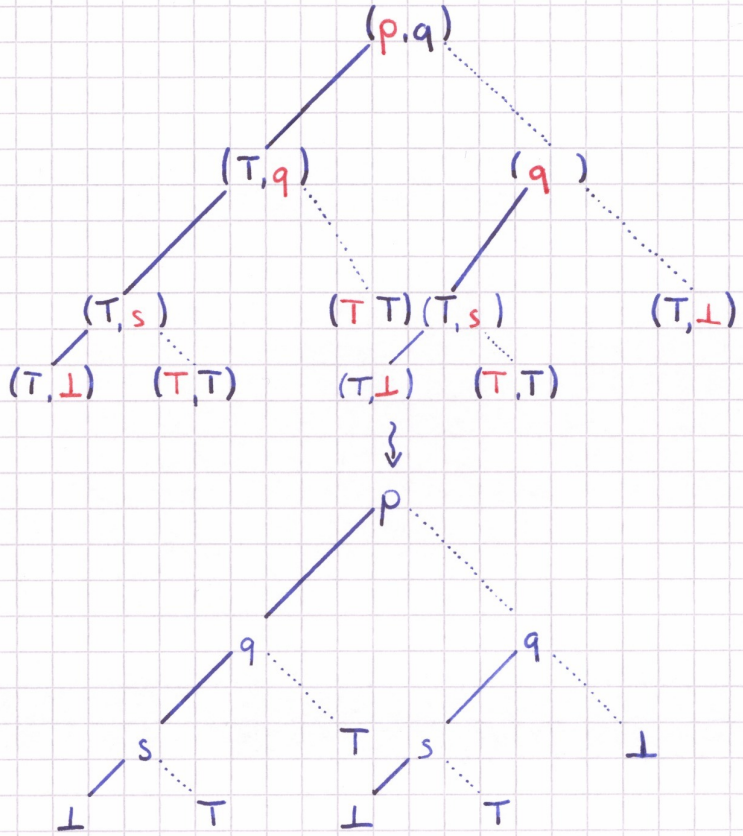


(3) $(p \vee q) \rightarrow (s \wedge q) \rightsquigarrow \neg((p \vee q) \wedge \neg(s \wedge q))$

(3.1) $\neg(s \wedge q)$



(3.2) $(p \vee q) \wedge \neg(s \wedge q)$



(3.3) $\neg((p \vee q) \wedge \neg(s \wedge q))$