EIDI 2 Cheatsheet

github contributors: `https://github.com/zepatrik/eidi2_cheatsheet`

20. Februar 2018

1 Logik

$$\neg(A \vee B) \equiv \neg A \wedge B$$
$$A \vee (B \wedge A) \equiv A \wedge (B \vee A) \equiv A$$
$$A \implies B \equiv \neg A \vee B$$

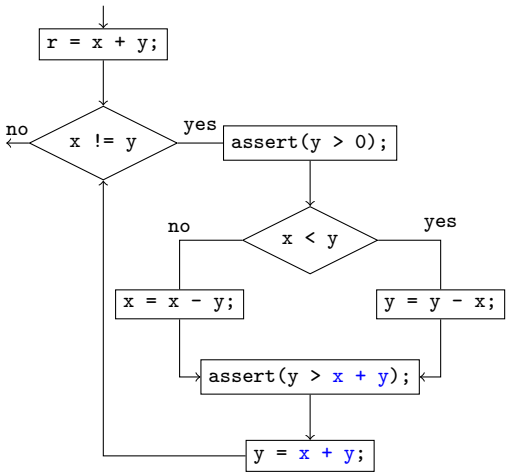
2 Verifikation

2.1 WP

$$\begin{aligned}
\mathsf{WP}[\![\texttt{x = read();}]\!](B) &\equiv \forall x.B \\
I \Rightarrow \mathsf{WP}[\![\texttt{b}]\!](B_0, B_1) &\equiv I \Rightarrow (((\neg b) \Rightarrow B_0) \wedge (b \Rightarrow B_1)) \\
\mathsf{WP}[\![\texttt{b}]\!](B_0, B_1) &\equiv (b \vee B_0) \wedge (\neg b \vee B_1) \\
&\equiv (\neg b \wedge B_0) \vee (b \wedge B_1) \vee (B_0 \wedge B_1) \\
&\equiv (\neg b \wedge B_0) \vee (b \wedge B_1)
\end{aligned}$$

2.2 Terminierung

1. vor jedem Schleifendurchlauf `r > 0`
2. `r` wird bei jedem Durchlauf kleiner



3 ocaml

3.1 Funktoren

```
module type A = sig
  type t
  val f : t -> t
end
module B: A = struct
  type t = int
  let f x = x + 1
end
module Ext(X: A) = struct
  include X
  let g x = f (f x)
end
module C = Ext (B)
```

3.2 Threaded tree Beispiel

```
open Thread open Event
let rec min = function
  | Leaf a -> a
  | Node (a,b) ->
    let c = new_channel () in
    let f t = sync (send c (min t)) in
    let _ = create f a in
    let _ = create f b in
    let x = sync (receive c) in
    let y = sync (receive c) in
    if x < y then x else y
```

3.3 Exceptions

```
type t = exn
try (
  raise Failure "this should fail"
) with Failure s -> print_string s
exeption Custom of int
try (
  raise Custom 0
) with _ -> ()
```

3.4 Lazy List

```
type 'a llist = Cons of 'a * (unit -> a' llist)
let rec lconst n = Cons (n, fun () -> lconst n)
let rec lseq s = Cons (s, fun () -> lseq (s+1))
let lpowers2 () =
  let rec impl n = Cons (n, fun () -> impl (2*n))
  in impl 1
let lhd (Cons (h, _)) = h
let ltl (Cons (_, t)) = t ()
let rec ltake n (Cons (h, t)) =
  if n = 0 then [] else h :: ltake (n-1) (t ())
let rec ldrop n (Cons (h, t)) =
  if n = 0 then Cons (h, t) else ldrop (n-1) (t ())
let rec lfilter f (Cons (h, t)) =
  if f h then Cons (h, fun () -> lfilter f (t ()))
    else lfilter f (t ())
let rec lmap f (Cons (h, t)) =
  Cons (f h, fun () -> lmap f (t ()))
```

3.5 Future

```
module Future = struct
  open Event
  type 'a t = 'a channel
  let create f a =
    let c = new_channel () in
    let rec loop f =
        f ();
        loop f
    in
    let task () =
      let b = f a in
      loop (fun () -> sync (send c b))
    in
    let _ = Thread.create task () in
    c
  let get c = sync (receive c)
end
```

3.6 Invarianten Tipps

1. Wir benötigen eine Aussage über den Wert der Variablen, über die wir etwas beweisen wollen (x) in der Schleifeninvariante. Die Aussage muss dabei mindestens so präzise ($\neq, \geq, \leq, =$) sein, wie die Aussage, die wir beweisen wollen.
2. Variablen, die an der Berechnung von x beteiligt sind **und** Werte von einer Schleifeniteration in die nächste transportieren ("loop-carried"), müssen in die Schleifeninvariante aufgenommen werden.
3. Die Schleife zu verstehen ist unerlässlich. Eine Tabelle für einige Schleifendurchläufe kann helfen die Zusammenhänge der Variablen (insbesondere mit dem Schleifenzähler i) aufzudecken. Oft lassen sich mit einer Tabelle, in der man die einzelnen Berechnungsschritte notiert, diese Zusammenhänge deutlich leichter erkennen, als mit einer Tabelle, die nur konkrete Werte enthält.
4. Die Variablen, die zur Berechnung von x im Beweisziel verwendet werden und diejenigen, die dazu in der Invariante verwendet werden, müssen in Beziehung stehen. Wenn diese Beziehung nicht aus der Verzweigungsbedingung folgt, müssen weitere Aussagen zur Invariante hinzugefügt werden.
5. Bei einer Bedingung mit Ungleichung ($<, \leq, >, \geq$) kann der Schleifenzähler (i) von der anderen Seite ($\geq, \leq$) begrenzt werden, sodass am Ende der Schleife Gleichheit folgt. Natürlich darf das nur dann geschehen, wenn diese Begrenzung im Programm auch tatsächlich gilt.
6. Werden Programmeingaben begrenzt, z.B. durch Ziehen von Beträgen, vorzeitigen Programmabbruch bei unerwünschten Eingaben oder durch gegebene Zusicherungen, so kann die Information über die Werte der Eingabevariablen ($n, m, ...$), die innerhalb der Schleife zulässig sind, in der Invariante sinnvoll sein.
7. Existiert kein klassischer Schleifenzähler, kann ein gedanklicher Zähler verwendet werden um den Zusammenhang zwischen den Variablen herzustellen.
8. Sind in der Schleife Verzweigungen enthalten, so hängt x, neben dem Schleifenzähler, meist auch von einer Variablen ab, die in der Bedingung der Verzweigung vorkommt. In der Invariante muss diese Abhängigkeit dann aufgenommen werden. Lässt sich keine "einfache" Beziehung finden, so kann in der Invariante eine Fallunterscheidung verwendet werden.
9. In einem Terminierungsbeweis wird eine Aussage über r in der Invariante benötigt. Es gilt dabei immer die für r ermittelte Berechnungsvorschrift.
10. Für die Terminierung benötigen wir Aussagen über alle Variablen in der Invarianten, die für die Berechnung von r benötigt werden und über die sich keine starken Beziehungen aus der Schleifenbedingung folgern lassen (wobei "stark" hier mindestens $\leq, \geq$ meint).