Aantekeningen PA

Rob Tiemens

Intro

slide 1: Intro

slide 2: Overzicht

slide 3: Korte uitleg. B technology is compleet. B method is een formeel software process

slide 4: Door wie gemaakt

slide 5: Door wie onerhouden

slide 6: Vorgangers

slide 7: Vorgangers

slide 8: verkooppraatje

slide 9: Overzicht componenten

slide 10: Abstract Machine Notion
Uitleggen: Sate based. type checking. obligations. consistentie.

slide: 11 Abstract Machine notiation Abstract Machine: voorbeeld is turing machine.

slide: 12 **Meer uitleg geven. Even opzoeken**

slide: 13 Gewoon uitleggen.

slide: 14 Programmeren!

slide: 15 Refinement. Wat doet het waarom etc.

slide: 16 Refinement uitleg

slide: 17 Po. Denk aan semantiek en correctheid

slide: 18 Uitleg PO

slide: 19 Welke POs komen er

slide: 20 Uitleg

slide: 21 Nog meer uitleg

slide: 22 Type

slide: 23 Type uitleg

slide: 24 Iets meer uitleg nodig,

A concept or idea not associated with any specific instance. The term abstract data applies to an abstract constant, an abstract variable or data introduced by an ANY, LET.

The typing predicates for abstract data are specific elementary predicates. Each typing predicate is used to set the type of one or more abstract data elements. It is separated from the preceding and successive predicates by a conjunction.

slide: 25 Structuren

slide: 26 Bewijs assistent

Demo

```
OPERATIONS
  ll, ii <-- find_nth (aa, bb, pp, nn) =
  PRE
      aa : INT &
      bb : INT &
      aa <= bb &
      aa-1 : INT &
      pp : INT --> BOOL &
      aa..bb <: dom(pp) &
      nn : INT &
      nn > 0
  THEN
      ll, ii :(ll = bool(card({kk | kk : aa..bb &
                      pp(kk) = TRUE}) >= nn) &
          ii : aa..bb &
          (ll = TRUE => pp(ii) = TRUE
                & card({kk | kk : aa..ii-1
                        & pp(kk) = TRUE}) = nn-1))
  END
END
```

```
REFINES
   find
OPERATIONS
  ll, ii <-- find_nth (aa, bb, pp, nn) =
  BEGIN
    ii := aa - 1;
    ll := FALSE;
    VAR
      cc
    IN
      cc := 0;
      WHILE ii < bb & ll = FALSE
      DO
        IF pp(ii + 1) = TRUE THEN
          cc := cc + 1
        END;
        IF cc = nn THEN
          ll := TRUE
        END;
        ii := ii + 1
    END
  END
END
```

Waarom werkt dit niet. Halting probleem! Zoek maar raak als je iets vind is het goed.

Toe te voegen. Een invariant.

Om terminatie te bewijzen heeft B een VARIANT nodig. Dit is een expressie die naar een niet-negatieve waarde evalueert aan het begin van elke iteratie en elke iteratie verminderd de variant. Omdat de natuurlijke getallen welgeordende (strikt totaal georderend) zijn door = betekend dit dat er maar eindig aantal stappen zijn en daarom terminatie bb-ii is een goede variant omdat we stap voor stap gaan. Dit zin dan de elementen die nog gecontroleerd moeten worden.

Implementatie

```
REFINES
   find
OPERATIONS
  ll, ii <-- find_nth (aa, bb, pp, nn) =
  BEGIN
    ii := aa - 1;
    ll := FALSE;
    VAR
      cc
    IN
      cc := 0;
      WHILE ii < bb & ll = FALSE
      DO
        IF pp(ii + 1) = TRUE THEN
          cc := cc + 1
        END;
        IF cc = nn THEN
          ll := TRUE
        END;
        ii := ii + 1
      INVARIANT
        ii : aa-1..bb &
        cc : 0..nn &
        cc = card({kk | kk : aa..ii & pp(kk) = TRUE}) &
        ll = bool(card({kk | kk : aa..ii & pp(kk) = TRUE}) = nn) &
        (ll = FALSE <=> cc < nn) &
        (ll = TRUE => ii : aa..bb & pp(ii)=TRUE) &
        card({kk | kk : aa..ii-1 & pp(kk)=TRUE}) < nn
      VARIANT
        bb - ii
      END
    END
  END
END
```

Proof ziet er een beetje maf uit. Maar dit is de preconditie, type beperkingen, loop invariant en de negatie van de loop conditie (omdat we iets willen bewijzen nadat de loop afgelopen is

Alles voor de =¿ is de hypothese. Alles erna is wat je mag bewijzen iiz$7777 is de variabele ii nadat de loop getermineerd is

```
dd
sh(iiz$7777)
dc(card({kk | kk : aa..iiz$7777 & pp(kk) = TRUE}) = nn)
dd
ah(bool(card({kk | kk : aa..iiz$7777 & pp(kk) = TRUE}) = nn) = TRUE)
pr
dd
eh(bool(card({kk | kk : aa..iiz$7777 & pp(kk) = TRUE}) = nn))
ss
eh( nn )
ah(aa..iiz$7777 <: aa..bb)
pr
pp(5000)
dd
ah({kk | kk : aa..iiz$7777 & pp(kk) = TRUE} <: {kk | kk : aa..bb & pp(kk) = TRUE})
pr
dd
pr
dd
ah(bool(card({kk | kk :  aa..iiz$7777 &  pp(kk) = TRUE}) = nn) = FALSE)
pr
dd
eh(bool(card({kk | kk :  aa..iiz$7777 &  pp(kk) = TRUE}) = nn))
ss
ah(iiz$7777 = bb)
sh(iiz$7777 _and bb)
sh(card({kk | kk : aa..iiz$7777 &  pp(kk) = TRUE}))
ct
ah(iiz$7777 < bb)
pp(5000)
dd
ah(card({kk | kk : aa.iiz$7777 & pp(kk) = TRUE}) = nn)
pr
pr
pr
pr
```