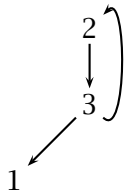
Semantics, WS 2003:
Solutions for assigment 11

Prof. Dr. Gert Smolka, Dipl.-Inform. Guido Tack

Exercise 11.1: Rule Sets



(a)

(b) $\{1\}, \{1,2,3\}$

(c) $\{1\}, \{1,2,3\}$

(d) $\mu\hat{R} = \{1\}$, $\nu\hat{R} = \{1,2,3\}$

(e) $\mu\hat{R}$, $\nu\hat{R}$

Exercise 11.2: Decomposition Algorithm

(a) R_1 is invertible, R_2 is not invertible.

(b)

$(\emptyset, [1])$

$(\{1\}, [2,3])$

$(\{1,2\}, [3,3])$

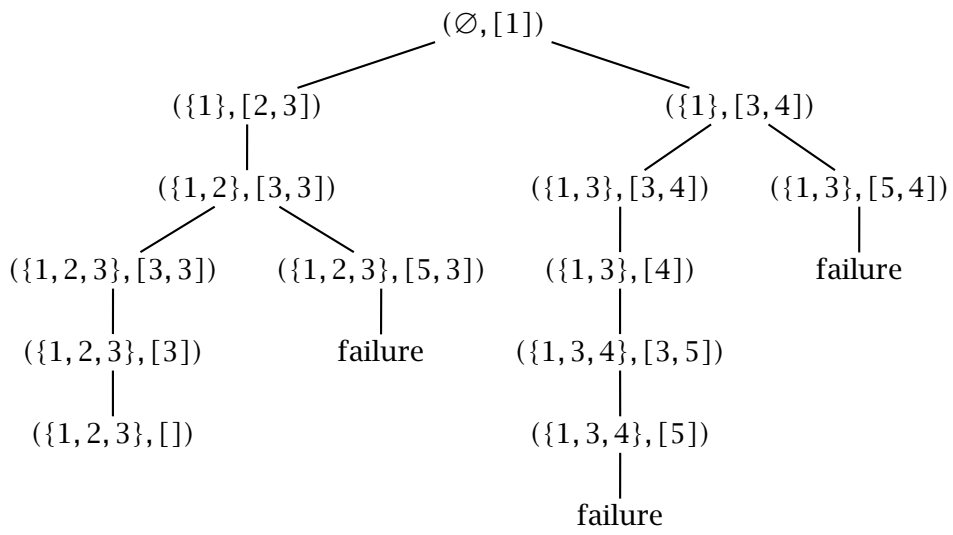
$(\{1,2,3\}, [3,3])$

$(\{1,2,3\}, [3])$

$(\{1,2,3\}, [])$

(c) $\mu\hat{R_1} = \emptyset$, $\nu\hat{R_1} = \{1,2,3\}$

(d)



(e) $\mu\hat{R}_2 = \emptyset$, $\nu\hat{R}_2 = \{1, 2, 3\}$

Exercise 11.3: Decomposition Algorithm in SML

```
type node = int
datatype info = T | P of node * node

type graph = node -> info

fun contains x (y::yr) = if x=y then true else contains x yr

fun test _ _ [] = true
  | test (g:graph) d ((p as (x1,x2))::xr) =
    if contains p d
    then test g d xr
    else case (g x1, g x2) of
             (_,T) => test g (p::d) xr
           | (T,_) => false
           | (P(x11,x12),
              P(x21,x22)) =>
             test g (p::d) ((x11,x21)::(x12,x22)::xr)
```

Exercise 11.4

(a) (i) $(\{X : \mathsf{Nat}, G : \mathsf{Nat} \to \mathsf{Nat}, F : \mathsf{Nat} \to \mathsf{Nat}\}, \mathsf{Nat})$

(ii) $(\{X : \mathsf{Bool}, G : \mathsf{Bool} \to \mathsf{Nat}, F : \mathsf{Nat} \to \mathsf{Bool}\}, \mathsf{Bool})$

(iii) $(\{X : \mathsf{Bool} \to \mathsf{Bool}, G : (\mathsf{Bool} \to \mathsf{Bool}) \to \mathsf{Nat}, F : \mathsf{Nat} \to \mathsf{Bool} \to \mathsf{Bool}\}$, $\mathsf{Bool} \to \mathsf{Bool})$

(b) Constraint synthesis yields

$T := F \to G \to X \to Z_2$

$C := \{G = X \to Z_1, F = Z_1 \to Z_2\}$

$V := \{Z_1, Z_2\}$

(c) $\{G = X \to Z_1, F = Z_1 \to Z_2\}$

Exercise 11.5

```
type typvar = int
datatype typ = TV of typvar | Arrow of typ * typ
type var = int
datatype term = V of var | L of var * typ * term | A of term * term
type gamma = var -> typ
type cset = (typ * typ) list

fun empty _ = raise Empty

fun adjoin gamma v t = fn x => if x=v then t else gamma x

fun nextvar new = new-1

fun constr' new gamma (V v) = (new, gamma v, [])
  | constr' new gamma (L (v,ty,ter)) =
    let
        val (new', t2, c) = constr' new (adjoin gamma v ty) ter
    in
        (new', Arrow(ty,t2),c)
    end
  | constr' new gamma (A (t1, t2)) =
    let
        val (new', ty1, c1) = constr' new gamma t1
        val (new'', ty2, c2) = constr' new' gamma t2
        val x = TV new''
    in
        (nextvar new'', x , (ty1,Arrow(ty2,x))::(c1@c2))
    end

fun constr gamma t = constr' ~1 gamma t
```