Homework 5

SNU 4541.664A

Due: 04/30, 24:00

Kwangkeun Yi

이 숙제의 목표는

- 앞으로 구현할 분석기들이 사용할 다양한 요약도메인(abstract domain)들을 자동으로 구현해 줄 모듈함수(functor)들을 정의한다.
- 배운 이론에따라 간단한 분석기를 디자인하고 안전성을 증명해본다.
- 디자인된 분석기를 구현하고 돌려본다.

Exercise 1 "요약 도메인 자동구현기"

프로그램 분석기에서 사용할 요약 도메인(abstract domain, cpo)들을 구현해 줄 모듈함수(functor)들을 구현한다.

예를들어 다음과 같이 정의된 도메인 $\hat{D}_i$들은

$$\begin{array}{rcll} A & = & \{a,b,c\} & \text{set} \\ B & = & \{0,1,2,3\} & \text{set} \\ C & = & A \xrightarrow{\text{fin}} B & \text{set} \\ \hat{D}_1 & = & C \cup \{\bot, \top\} & \text{lifted, flat domain} \\ \hat{D}_2 & = & 2^B & \text{powerset domain} \\ \hat{D}_3 & = & \hat{D}_1 \times \hat{D}_2 & \text{product domain} \\ \hat{D}_4 & = & A \to \hat{D}_3 & \text{atomic function domain} \end{array}$$

다음과 같이 손쉽게 구현될 것이다:

```
module A = PrimitiveSet(struct type t = string
                            let compare = compare
                            exception TooMany
```

```
                            val all = fun () -> ["a"; "b"; "c"]
                        end)

module B = PrimitiveSet(struct type t = int
                            let compare = compare
                            exception TooMany
                            val all = fun () -> [0; 1; 2; 3]
                        end)
module C = FunctionSet (A) (B)
module D1 = FlatDomain (C)
module D2 = PowersetDomain (B)
module D3 = ProductDomain (D1) (D2)
module D4 = FunDomain (A) (D3)
```

요약도메인 모듈들을 만들어 주는 모듈함수들을 구현하라. 아래의 코드에서 빈칸("...")을 완성하면 된다. (요약도메인은 집합으로 부터 만들어 질 것이므로, 집합 모듈들을 만들어주는 모듈함수(functor)들은 아래와 같이 제공된다.)

주의: 모듈 함수 `FunDomain (A) (B)`에 의해서 만들어 지는 $A \to B$는 "atomic function domain"이라고 불리는데, A는 유한집합이어야 하고 B는 cpo로서 다음의 성질을 가지는 cpo이다:

$$A \to B \quad \equiv \quad \underbrace{B \times \cdots \times B}_{|A|}$$

즉, $A \to B$의 원소들은 A의 원소들이 키가 되는 B 테이블이다.

```
(* set functors for
 * SNU 4541.664A Program Analysis
 * Kwangkeun Yi, 2010
 *)
module type SET = sig
                   include Set.S
                   exception TooMany
                   val all: unit -> t
                  end

module PrimitiveSet (A: sig
                            type t val compare: t -> t -> int
                            exception TooMany
                            val all: unit -> t list
                          end) =
 struct
  include Set.Make (A)
  exception TooMany
  let all = fun () -> try
                        List.fold_left (fun s x -> add x s) empty (A.all())
                      with A.TooMany -> raise TooMany
 end

module ProductSet (A: SET) (B: SET) =
 struct
  include Set.Make (struct type t = A.elt * B.elt let compare = compare end)
  exception TooMany
  let all = fun () -> try
                        A.fold (fun a c ->
                                          B.fold (fun b c -> add (a,b) c)
                                                 (B.all()) c
                                        ) (A.all()) empty
                       with A.TooMany -> raise TooMany
                          | B.TooMany -> raise TooMany
 end
```

```
module PowerSet (A: SET) =
 struct
  include Set.Make (struct type t = A.t let compare = compare end)
  exception TooMany
  let all = fun () -> raise TooMany
 end

module FunctionSet (A: SET) (B: SET) =
 struct
  module F = Map.Make (struct type t = A.elt let compare = compare end)
  include Set.Make (struct type t = B.elt F.t let compare = compare end)
  exception TooMany
  let all = fun () -> raise TooMany
  let domain_all = A.all
  let range_all = B.all
 end
```

```
(* domain functors for
 * SNU 4541.664A Program Analysis
 * Kwangkeun Yi, 2010
 *)

module type DOMAIN =
 sig
   type elt      (* the type of abstract domain elements *)
   val top: elt
   val bot: elt
   val join: elt -> elt -> elt
   val leq: elt -> elt -> bool
 end

module type FLAT_DOMAIN =
 sig
   include DOMAIN
   type atom
   val make: atom -> elt
 end

module type PRODUCT_DOMAIN =
 sig
  include DOMAIN
  type lelt
  type relt
  val l: elt -> lelt     (* left *)
  val r: elt -> relt     (* right *)
  val make: lelt -> relt -> elt
 end
```

```
module type POWERSET_DOMAIN =
 sig
  include DOMAIN
  type atom
  val union: elt -> elt -> elt
  val inter: elt -> elt -> elt
  val diff: elt -> elt -> elt
  val remove: atom -> elt -> elt
  val mem: atom -> elt -> bool
  val map: (atom -> atom) -> elt -> elt
  val fold: (atom -> 'a -> 'a) -> elt -> 'a -> 'a
  val make: atom list -> elt
 end

module type FUNCTION_DOMAIN =
 sig
  include DOMAIN
  type lelt
  type relt
  val image: elt -> lelt -> relt
  val update: elt -> lelt -> relt -> elt
  val map: (lelt -> relt -> lelt * relt) -> elt -> elt
  val fold: (lelt -> relt -> 'a -> 'a) -> elt -> 'a -> 'a
  val make: (lelt * relt) list -> elt
 end

module type INTERVAL_DOMAIN =
 sig
  include DOMAIN
  exception Undefined
  type bound = Z of int | Pinfty | Ninfty
  val l: elt -> bound  (* lower bound *)
  val u: elt -> bound  (* upper bound *)
  val make: bound -> bound -> elt
 end
```

```
module FlatDomain (A: SET) : FLAT_DOMAIN
 with type atom = A.elt =
 struct
  type elt = BOT | TOP | ELT of A.elt
  type atom = A.elt
  let bot = BOT
  let top = TOP
  let join x y = match (x, y)
         with (BOT, _) -> y
            | (_, BOT) -> x
            | (TOP, _) -> TOP
            | (_, TOP) -> TOP
            | (x, y) -> if x=y then x else TOP
  let leq x y = match (x, y)
        with (BOT, _) -> true
           | (_, TOP) -> true
           | (ELT a, ELT b) -> a=b
           | _ -> false
  let make a = ELT a
 end
```

```
module ProductDomain (A: DOMAIN) (B: DOMAIN) : PRODUCT_DOMAIN
 with type lelt = A.elt and type relt = B.elt =
 struct
  type elt = BOT | TOP | ELT of A.elt * B.elt
  type lelt = A.elt
  type relt = B.elt
  let bot = BOT
  let top = TOP
  let join x y = match (x,y)
                 with (BOT,_) -> y
                    | (TOP,_) -> TOP
                    | (_,BOT) -> x
                    | (_,TOP) -> TOP
                    | (ELT(a,b), ELT(a',b')) -> ELT(A.join a a', B.join b b')
  let leq x y = match (x,y)
                with (BOT,_) -> true
                   | (TOP,_) -> false
                   | (_,BOT) -> false
                   | (_,TOP) -> true
                   | (ELT(a,b), ELT(a',b')) -> (A.leq a a') && (B.leq b b')
  let l x = match x with TOP -> A.top | BOT -> A.bot | ELT(a,b) -> a
  let r x = match x with TOP -> B.top | BOT -> B.bot | ELT(a,b) -> b
  let make a b = ELT (a,b)
 end
```

```
module PowersetDomain (A: SET) : POWERSET_DOMAIN
 with type atom = A.elt =
 struct
  type elt = BOT | TOP | ELT of A.t
  type atom = A.elt
  let bot = BOT
  let top = TOP
  let join x y = match (x,y)
                 with (BOT, _) -> y
                    | (_, BOT) -> x
                    | (TOP, _) -> TOP
                    | (_, TOP) -> TOP
                    | (ELT s, ELT s') -> ELT (A.union s s')
  let mem a s = match s with BOT -> false
                           | TOP -> true
                           | ELT s -> A.mem a s
  let fold f x a = match x with BOT -> a
                              | TOP -> A.fold f (A.all()) a
                              | ELT s -> A.fold f s a
  let map f x = match x with BOT -> BOT
                           | TOP ->
                     (ELT (A.fold (fun a s' -> A.add (f a) s') (A.all()) A.empty))
                           | ELT s ->
                     (ELT (A.fold (fun a s' -> A.add (f a) s') s A.empty))

  let make lst = match lst
                  with [] -> BOT
                     | l -> ELT
                             (List.fold_left (fun s x -> A.add x s)
                                             A.empty l
                             )
  ...
 end
```

```
module FunDomain (A: SET) (B: DOMAIN) : FUNCTION_DOMAIN
 with type lelt = A.elt and type relt = B.elt =
 struct
  module Map = Map.Make(struct type t = A.elt let compare = compare end)
  type elt = BOT | TOP | ELT of B.elt Map.t
  type lelt = A.elt
  type relt = B.elt
  let bot = BOT
  let top = TOP
  ...
 end

module Zintvl: INTERVAL_DOMAIN =
 struct
  ...
 end

module Zparity: DOMAIN =
 struct
  ...
 end
```

□

Exercise 2 "분석기I: 디자인과 구현"

다음 언어로 짜여진 프로그램을 입력으로 받아서, 각 명령문이 실행된 직후의 메모리 상태를 모두 포섭해서 예측하는 분석기를 디자인하고 구현한다.

$$
\begin{array}{rcl}
C & \to & \texttt{skip} \\
& | & x \texttt{ := } E \\
& | & C \texttt{ ; } C \\
& | & \texttt{if } E\ C\ C \\
& | & \texttt{while } E\ C \\
E & \to & n \quad (n \in \mathbb{Z}) \\
& | & E \texttt{ + } E \\
& | & \texttt{- } E \\
& | & x
\end{array}
$$

분석하고자 하는 성질은 변수들이 가지는 정수가 짝수일지 홀수일지 여부이다. 분석기는 의미있는 프로그램("잘 도는" 프로그램)만 입력으로 받는다고 가정한다.

제출할 것은 디자인 리포트(PDF)와 구현된 코드이다

- 디자인: 분석기에 해당하는 "요약 의미구조"(abstract semantics)를 조립식으로 정의하고 그 안전성을 증명한 것을 리포트로 제출한다.
- 구현: 분석기 프로그램

  $$\texttt{parity} : \texttt{K.cmd} \to \texttt{unit}$$

  을 구현해서 제출한다. 분석기 parity는 주어진 프로그램을 분석한 후 그 결과를 출력한다. 분석결과의 출력은: 프로그램을 "depth-first-traversal"하면서 만나는 명령문 순서로, 분석된 결과(메모리 상태)를 출력하도록 한다.

  위에서 모듈 K는 다음의 모듈타입 KMINUS를 만족하도록 한다.

```
module type KMINUS =
sig
  exception Error of string
  type id = string
  type exp = NUM of int
           | VAR of id
           | ADD of exp * exp
```

```
            | MINUS of exp
  type cmd = SKIP
            | ASSIGN of id * exp          (* assign to variable *)
            | SEQ of cmd * cmd            (* sequence *)
            | IF of exp * cmd * cmd       (* if-then-else *)
            | WHILE of exp * cmd          (* while loop *)
end
```

□