Homework 6

SNU 4541.664A, 2009 봄

Due: 4/23 24:00

Exercise 1 "의미방정식 풀기 `fixpoint`"

주어진 K 프로그램에서 도출된 의미방정식을 푸는 `fixpoint`를 구현하라. 함수 `fixpoint`는 숙제 5에서 만들어진 의미 방정식 도출기에서 나오는 방정식을 받아 그 해를 찾아준다. 즉, 의미 연립 방정식을 다음과 같이 표현하면

$$\vec{X} = \mathcal{F}(\vec{X})$$

`fixpoint`는 다음을 계산하게 된다:

$$\bigsqcup_{i \geq 0} \mathcal{F}^i(\bot).$$

방정식의 해가 돌아다니는 공간(아래 `CPO`타입의 모듈)의 정의에 따라서, 어떤 프로그램에 대해서는 그 의미방정식의 해를 유한시간내에 계산 못할 수 도 있다.

아래의 모듈함수 `Analyzer`를 구현하면 된다.

- `Aanalyzer`는 분석 방정식 해의 핵심 공간(모듈 타입 `CPO`)을 인자로 받도록 구성되어 있다.
- `Analyzer`는 최소한 모듈 타입 `ANALYSIS`를 만족하도록 구현하면 된다.

```
module type ANALYZER =
 sig
   (* K program part *)
   exception Error of string
   type id
   type label
   type cmd = label * stmt
   and stmt = SKIP
            | ASSIGN of id * exp          (* x := exp *)
            | ASSIGNSTAR of id * exp      (* *x := exp *)
            | SEQ of cmd * cmd            (* c;c *)
            | IF of exp * cmd * cmd       (* if e c c *)
            | WHILE of exp * cmd          (* while e c *)

   and exp = NUM of int
           | ADD of exp * exp        (* e+e *)
           | MINUS of exp            (* -e *)
           | VAR of id               (* x *)
           | STAR of id              (* *x *)
           | AMPER of id             (* &x *)
           | READ

   type program = cmd

   (* equation part *)
   type eqn_var
   type rhs = Var of eqn_var
            | Cup of rhs * rhs
            | Restrict of rhs * exp
            | Nrestrict of rhs * exp
            | Update of rhs * exp * exp * rhs
            | Bottom
            | Top
   type equation = eqn_var -> rhs
   val eqnPprint: equation -> unit
```

```
   val semanticEqns: program -> equation

   (* solution space part *)
   type state
   type solution = eqn_var -> state
   val statePprint: state -> unit
   val solnPprint: solution -> unit

   (* equation interpretation part *)
   val evalExp: state -> exp -> v
   val evalRhs: solution -> rhs -> state

   (* equation solver part *)
   val fixpoint: equation -> solution

   (* main part *)
   val analyze: program -> solution
end

module type CPO =
 sig
   type t = loc -> v         (* the state type, location to value *)
   and loc
   and v                     (* the value type = loc and z *)
   val order: t -> t -> bool (* the partial order *)
   val lub: t -> t -> t      (* least upper bound *)
   val bottom: t             (* the bottom element *)
   val top: t                (* the top element *)
   val update: t -> loc -> v -> t (* update operation, x{l |-> v} *)
   val add: v -> v -> v      (* v + v *)
   val neg: v -> v           (* -v *)
   val locOfV: v -> loc      (* extract the loc from a value *)
   val locToV: loc -> v      (* embed a loc to a value *)
   val intToV: int -> v      (* embed an int to a value *)
   val stringToLoc: string -> loc (* a new location from a string *)
```

```
   val locPprint: loc -> unit     (* pretty printer *)
   val vPprint: v -> unit         (* pretty printer *)
   val pprint: t -> unit          (* pretty printer *)
 end

module Analyzer(D: CPO): ANALYZER with type state = D.t =
struct
  ...
end
```

□