CS 565 Out: October 23, 1996
Jens Palsberg Due: October 30, 1996

Homework 9:

Type Inference

1. Do exercises 1,2,7 on pages 98–99 in *OOTS*.

2. Implement the solver from lecture note 8 (also known as "Types for the language designer") pages 58–62. Constraints must be represented as follows:

```
(define-record start-constraint (const var))
(define-record prop-constraint (var1 var2))
(define-record cond-constraint (const var1 var2 var3))
```

Here, `const` is a constant of some kind (say, a class name), and `var` is a type variable.

The result of the application `(make-start-constraint` c v`)` represents the start constraint $c \in v$. Notice that page 56 of lecture note 8 uses start constraints of the form $C \subseteq X$ where C is a set of constants and X is a type variable. Clearly, we can represent

$$\{c_1, \ldots, c_n\} \quad \subseteq \quad X$$

as the n constraints

$$\begin{array}{ccc} c_1 & \in & X \\ & \ldots & \\ c_n & \in & X \end{array}$$

The result of the application `(make-prop-constraint` v v'`)` represents the propagation constraint $v \subseteq v'$.

The result of the application `(make-cond-constraint` c w v v'`)` represents the conditional constraint $c \in w \Rightarrow v \subseteq v'$.

The solver must be programmed in the style of lecture note 3, page 8. This means that your implementation must have the following structure:

```
(define solver
  (let ((graph ...) ...)
    (lambda (message)
      (case message
        ((insert)
           (lambda (constraint)
             (record-case constraint
               (start-constraint (const var) ...)
               (prop-constraint (var1 var2) ...)
               (cond-constraint (const var1 var2 var3) ...)
               (else (error ...)))))
        ((read)
           (lambda (var) ...))
        (else (error ...))))))
```

This will allow us to write things like:

```
((solver 'insert) a-constraint)
((solver 'insert) another-constraint)
((solver 'read) some-variable)
```

The operation `read` returns a list of constants. More precisely, the result of the application `((solver 'read)` v`)` is a list that represents the set of constants currently assigned to the variable v.

You are welcome to program other operations than `insert` and `read`; you may need some operations for testing purposes.

Efficiency will not play any role when grading your solution.

Submit both your program and at least one convincing test case.