**Checking Initialization Using Data-Flow Analysis**

We follow the methodology of [Designing Correct Data-Flow Analyses](http://lara.epfl.ch/w/cc09:designing_correct_data-flow_analyses)

**Defining Properties of Interest**

We represent program execution as a sequence of program states and simple statements.

Program state is $(u_i,w_i)$where

1. $u_i$is a program point
2. $w_i$is the map from variables to their values

Given a sequence   
\begin{displaymath}
   (u_1,w_1), s_1, (u_2,w_2), s_2, \ldots, s_{k-1}, (u_k,w_k), \ldots, s_n, (u_{n+1},w_{n+1})
\end{displaymath}  
we say that variable $x$is **initialized** at step $k$if one of the statements $s_1,\ldots,s_{k-1}$is an assignment to $x$.

Note that if the variable is initialized, it remains initialized during that execution.

We say that a variable $x$is definitely initialized at program point $u$, if for every execution of length $k$such that $u_k=u$, we have that $x$is initialized at step $k$.

We would like to compute, for each program point, which variables at that point are definitely initialized.

Then, we will emit a warning if a variable is not definitely initialized at some program point, but there is a statement reading its value that can be executed at this program point.

**Reformulation of the Property**

To avoid talking about previous states in a sequence and talk only about states, we use the following trick

* (an instance of a general technique of *history variables*)

For each variable, introduce an additional boolean flag (0 or 1) that indicates whether the variable is initialized

We have

* $Var_X = \{ x_1, \ldots, x_n \}$is the set of original variables
* $Var_B = \{ b_1, \ldots, b_n \}$the corresponding boolean flags

The state is map $c : (Var_X \cup Var_B) \to Int$

Then:

* at the beginning, all flags are 0
* at each assignment, we set the flag to 1

Desired property: *if a statement reads a variable, then the variable’s flag must be 1*

**Defining Semilattice to Express Properties**

We use

* $\bot$to represent that the case when there are no states of initialization flag set to 0 (either there are no states at all, or there are only states with flag 1)
  + in other words: initialized variable
* $\top$to represent states where initialization flag can be anything (0 or 1)
  + in other words: possibly uninitialized variable

Lattice element is map $m : Var_X \to \{\bot,\top\}$

We use a pointwise lattice

**Specifying Meaning of Lattice Elements**

\begin{displaymath}
   \gamma(m) = \{ c \mid \forall b_i \in Var_B.\ (m(x_i)=\bot \rightarrow c(b_i)=1) \}
\end{displaymath}

**Example:** Let $Var = \{ x, y \}$and let $m = \{(x_1,\bot),(x_2,\top)\}$.

Then $\gamma(m)$is the set of concrete states $c$such that $m(b_1) = 1$i.e. $x_1$is initialized.

There is no constraint on $x_2$, because $m(x_2)=\top$.

**Checking monotonicity:** if $m \sqsubseteq m'$then $\gamma(m) \subseteq \gamma(m')$

* note that if element in map goes from $\bot$to $\top$, then $\gamma$value gets only bigger

**Initial Lattice Element**

All points except entry get value $\bot$for each variable - as usual

What can we assign to entry?

* all variables are uninitialized
* so all elements are $\top$

**Transfer Functions**

Two kinds of statements in CFG (ignoring procedure calls):

* dooes not change state (e.g. test) - initialization remains same
* they assign to some variable x - set initialization of x to $\bot$(it is initialized)

Formally,   
\begin{displaymath}
    transferFun(x=e,a) = a[x:=\bot]
\end{displaymath}  
and   
\begin{displaymath}
   transferFun(s,a) = a
\end{displaymath}  
if $s$does not assign to any variable.

**Note on Procedures**

What about procedure calls?

* for local variables - no change, procedure cannot change them
* for global variables
  + who knows what procedure might do
  + a safe thing is to set all global variables to $\top$

**Intraprocedural** analysis: analyzes one procedure at time

**Interprocedural** analysis: descend into procedures at call site

**Using Results of Initialization Analysis**

If at node $v$we have $m(x)=\top$and there is a statement from $v$that reads $x$, report error “reading a possibly uninitialized variable x”.