TDA342
Security - Part I

QUFEI WANG
JACOB J. NILSSON

March 21, 2020

Task 1

The definition of data type *Sensitivity* as a monoid is as follows:

```haskell
data Sensitivity = Secret | Public deriving (Show, Eq)

instance Monoid Sensitivity where
  mempty = Public
  mappend Public Public = Public
  mappend _ _ = Secret

instance Semigroup Sensitivity where
  (<>) = mappend
```

From this definition we have the following two premises:

1. $mempty = Public$
2. Only *Public* '*mappend*' *Public* yields *Public*, in all other situations we have *Secret* as the reuslt of *mappend* operation.

From these two premises, we have the following 4 conclustions.

1. $mappend\ mempty\ a == a$.

   Proof. From premise 2 we have,

$$mappend\ Public\ Public = Public \tag{1}$$
$$mappend\ Public\ Secret = Secret \tag{2}$$

From premise 1, we have

$$Public = mempty \tag{3}$$

Substitute $Public$ in $(1),(2)$ with $mempty$, we have

$$mappend\ mempty\ Public = Public \tag{4}$$
$$mappend\ mempty\ Secret = Secret \tag{5}$$

Then we conclude that $mappend\ mempty\ a == a$.

2. $mappend\ a\ mempty == a$.

   Proof. Similar with the proof above, just replace formula (2) with

   $$mappend\ Secret\ Public = Secret$$

   , and apply the same logic to the rest.

3. $mappend\ a\ (mappend\ b\ c) == mappend\ (mappend\ a\ b)\ c$.

   Proof. Assume $a = mempty$, by the first conclusion, we have

   $$mappend\ a\ (mappend\ b\ c) = mappend\ b\ c \tag{6}$$
   $$mappend\ (mappend\ a\ b)\ c = mappend\ b\ c \tag{7}$$

   , such that

   $$a = mempty \implies mappend\ a\ (mappend\ b\ c) == mappend\ (mappend\ a\ b)\ c \tag{8}$$

   . Similarly, by assuming $b = mempty$ and $c = mempty$, using the first two conclusions we already proved, we have

   $$b = mempty \implies mappend\ a\ (mappend\ b\ c) == mappend\ (mappend\ a\ b)\ c \tag{9}$$

   $$c = mempty \implies mappend\ a\ (mappend\ b\ c) == mappend\ (mappend\ a\ b)\ c \tag{10}$$

   Combining $(8),(9),(10)$, we know that the equation holds if any of a, b, c is $mempty$. On the other hand, if all of a, b, c are $Secret$, then by premise (2), we have both left and right hand side of the equation is $Secret$. So the equation is proven.

4. $mappend\ a\ b == Public \iff (a == Public\ \&\&\ b == Public)$. This is a direct conclusion of premise (2).

By the 4 conclusions we've proven above, we know that the data type $Sensitivity$ is a monoid.

Task 2

The modifications made to achieve the desired effect on file *Basic.hs* is listed as follows:

```
data Expr = Lit Integer
               ...
          | Sec Expr
  deriving (Show)

type Env = Map Name (Labeled Value)

lookupVar :: Name -> Eval (Labeled Value)

language = P.Lang
  { ...
  , P.lNewSecretexp = Sec
  , ...
  }

eval :: Expr -> Eval (Labeled Value)
eval (Lit n)        = return (LV Public n)
eval (a :+: b)      = do
  LV s1 v1 <- eval a
  LV s2 v2 <- eval b
  return $ LV (s1 `mappend` s2) (v1 + v2)
eval (Var x)        = lookupVar x
eval (Let n e1 e2) = do v <- eval e1
                         localScope n v (eval e2)
eval (Sec e)        = do LV _ v <- eval e
                         return $ LV Secret v
```

1. We add a value constructor *Sec Expr* in type *Expr* to incorporate the 'secret' keyword.

2. We changed the type of *Env* to *Map Name* (*Labeled Value*), since we need to track the sensitivity level of our expression.

3. As a consequence of 2, we also need to change the signature of function *lookupVar*. However the implementation of this function doesn't need to change since the definition of *Eval* remains the same.

4. We modified the value of *P.lNewSecretexp* in *language* to *Sec* to guarantee the parser works correctly.

5. Finally, we changed the signature of function *eval* and altered parts of the implementation to have the final result being 'labeled'. Notice that we properly used the function *mappend* to make the semantics of sensitivity consistent.

A testing function named $testLabeledValue$ is added to the source file to verify the correctness of our implementation.