A security types preserving compiler in Haskell

ALBERTO PARDO

Instituto de Computación
Universidad de la República
Montevideo - Uruguay

joint with Cecilia Manzino, Univ. Nac. de Rosario, Arg.

Noninterference

$$P \in \mathcal{L}$$

$$[\![P]\!] : \mathit{State} \rightarrow \mathit{State}$$

$$\forall s, s' \in \mathit{State}.$$
$$s \cong_L s' \;\wedge\; [\![P]\!]s \Downarrow \;\wedge\; [\![P]\!]s' \Downarrow \implies [\![P]\!]s \cong_L [\![P]\!]s'$$

Security-preserving compilation

Let $\boldsymbol{C} : \mathcal{L} \to \mathcal{L}'$.

Security-preserving compilation

Let $\boldsymbol{C} : \mathcal{L} \rightarrow \mathcal{L}'$.

$$\begin{aligned} &\forall s, s' \in State. \\ &\quad s \cong_L s' \;\wedge\; [\![P]\!]s \Downarrow \;\wedge\; [\![P]\!]s' \Downarrow \\ &\quad\quad \Longrightarrow\; [\![P]\!]s \cong_L [\![P]\!]s' \end{aligned}$$

Security-preserving compilation

Let $\boldsymbol{C} : \mathcal{L} \to \mathcal{L}'$.

$$\begin{aligned} &\forall s, s' \in State. \\ &\quad s \cong_L s' \;\wedge\; [\![P]\!] s \Downarrow \;\wedge [\![P]\!] s' \Downarrow \\ &\quad\quad \Longrightarrow \; [\![P]\!] s \cong_L [\![P]\!] s' \end{aligned}$$

$\Longrightarrow$

$$\begin{aligned} &\forall s, s' \in State. \\ &\quad s \cong_L s' \;\wedge\; [\![\boldsymbol{C}\; P]\!] s \Downarrow \;\wedge [\![\boldsymbol{C}\; P]\!] s' \Downarrow \\ &\quad\quad \Longrightarrow \; [\![\boldsymbol{C}\; P]\!] s \cong_L [\![\boldsymbol{C}\; P]\!] s' \end{aligned}$$

Source language

Expressions

$$
\begin{aligned}
e ::= &\ n \\
\mid &\ x_L \\
\mid &\ x_H \\
\mid &\ e + e
\end{aligned}
$$

Statements

$$
\begin{aligned}
S ::= &\ x_L := e \\
\mid &\ x_H := e \\
\mid &\ \textbf{\textit{if}}\ e\ \textbf{\textit{then}}\ S\ \textbf{\textit{else}}\ S \\
\mid &\ \textbf{\textit{while}}\ e\ \textbf{\textit{do}}\ S \\
\mid &\ S\textbf{;}S
\end{aligned}
$$

Abstract syntax

```
data ASTExp where
  INTVAL :: Int  -> ASTExp
  VARL   :: RefL -> ASTExp
  VARH   :: RefH -> ASTExp
  ADD    :: ASTExp -> ASTExp -> ASTExp


data ASTCom where
  ASSIGNL :: RefL -> ASTExp -> ASTCom
  ASSIGNH :: RefH -> ASTExp -> ASTCom
  IF0     :: ASTExp -> ASTCom -> ASTCom -> ASTCom
  WHILE   :: ASTExp -> ASTCom -> ASTCom
  SEQ     :: ASTCom -> ASTCom ->  ASTCom
```

Security types: expressions

$$\vdash n : low \qquad \vdash x_L : low \qquad \vdash x_H : high$$

$$\frac{\vdash e_1 : st_1 \qquad \vdash e_2 : st_2}{\vdash e_1 + e_2 : \mathbf{max}(st_1, st_2)}$$

Security types: expressions

$$\vdash n : low \qquad \vdash x_L : low \qquad \vdash x_H : high$$

$$\frac{\vdash e_1 : st_1 \qquad \vdash e_2 : st_2}{\vdash e_1 + e_2 : \mathbf{max}(st_1, st_2)}$$

```
data Exp st where
 IntVal :: Int -> Exp Low
 VarL   :: RefL -> Exp Low
 VarH   :: RefH -> Exp High
 Add    :: Exp st1 -> Exp st2 -> Exp (Max st1 st2)
```

Security types

```
data Low
data High

type family Max n m
type instance Max Low  x = x
type instance Max High x = High

type family Min n m
type instance Min Low  x = Low
type instance Min High x = x
```

Security types: statements

$$\frac{\vdash e : low}{low \vdash x_L := e} \qquad pc \vdash x_H := e$$

$$\frac{\vdash e : st \quad pc_1 \vdash S_1 \quad pc_2 \vdash S_2 \quad st \leq pc_1 \quad st \leq pc_2}{\mathbf{min}(pc_1, pc_2) \vdash \mathbf{if}\ e\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2}$$

$$\frac{\vdash e : st \quad pc \vdash S \quad st \leq pc}{pc \vdash \mathbf{while}\ e\ \mathbf{do}\ S}$$

$$\frac{pc_1 \vdash S_1 \quad pc_2 \vdash S_2}{\mathbf{min}(pc_1, pc_2) \vdash S_1 \mathbf{;} S_2}$$

Security types: statements

```
data Com pc  where
 AssL  :: RefL -> Exp Low -> Com Low
 AssH  :: RefH -> Exp st -> Com High
 If0   :: Exp st -> Com pc1 -> Com pc2
                 -> LEq st (Min pc1 pc2)
                 -> Com (Min pc1 pc2)
 While :: Exp st -> Com pc -> LEq st pc -> Com pc
 Seq   :: Com pc1 -> Com pc2 -> Com (Min pc1 pc2)
```

```
data LEq st st' where
   L1 :: LEq Low x
   L2 :: LEq High High
```

Target language

Instructions

$$
\begin{aligned}
c ::= \;& \textbf{\textit{push}}\ n \\
\mid\;& \textbf{\textit{add}} \\
\mid\;& \textbf{fetch}_{\textbf{L}}\ x_L \\
\mid\;& \textbf{fetch}_{\textbf{H}}\ x_H \\
\mid\;& \textbf{store}_{\textbf{L}}\ x_L \\
\mid\;& \textbf{store}_{\textbf{H}}\ x_H \\
\mid\;& \textbf{\textit{branch}}(c, c) \\
\mid\;& \textbf{\textit{loop}}(c, c) \\
\mid\;& c\textbf{;}c \\
\mid\;& \textbf{\textit{noop}}
\end{aligned}
$$

Operational semantics

$$\langle c, vs, s\rangle \triangleright \langle c', vs', s'\rangle$$

$$\langle c, vs, s\rangle \triangleright (vs', s')$$

Operational semantics

$$\langle \textbf{push}\ n, vs, s\rangle \triangleright (\mathcal{N}[\![n]\!] : vs,\ s)$$

$$\langle \textbf{add}, z_1 : z_2 : vs, s\rangle \triangleright (z_1 + z_2 : vs,\ s)$$

$$\langle \textbf{fetch}_\textbf{L}\ x_L, vs, s\rangle \triangleright (s\ x_L : vs,\ s)$$

$$\langle \textbf{store}_\textbf{L}\ x_L, z : vs, s\rangle \triangleright (vs, s[x_L \mapsto z])$$

$$\langle \textbf{branch}(c_1, c_2), z : vs, s\rangle \triangleright \langle c_1, vs, s\rangle \quad \texttt{if}\ \ z = 0$$
$$\langle \textbf{branch}(c_1, c_2), z : vs, s\rangle \triangleright \langle c_2, vs, s\rangle \quad \texttt{if}\ \ z \neq 0$$

$$\langle \textbf{loop}(c_1, c_2), vs, s\rangle \triangleright \langle c_1 ; \textbf{branch}(c_2 ; \textbf{loop}(c_1, c_2), \textbf{noop}), vs, s\rangle$$

Compilation

$$\boldsymbol{C}[\![n]\!] = \textbf{push}\ n$$

$$\boldsymbol{C}[\![x_L]\!] = \textbf{fetch}_{\textbf{L}}\ x_L$$

$$\boldsymbol{C}[\![x_H]\!] = \textbf{fetch}_{\textbf{H}}\ x_H$$

$$\boldsymbol{C}[\![e_1 + e_2]\!] = \boldsymbol{C}[\![e_1]\!]\,;\,\boldsymbol{C}[\![e_2]\!]\,;\,\textbf{add}$$

$$\boldsymbol{C}[\![x_L := e]\!] = \boldsymbol{C}[\![e]\!]\,;\,\textbf{store}_{\textbf{L}}\ x_L$$

$$\boldsymbol{C}[\![x_H := e]\!] = \boldsymbol{C}[\![e]\!]\,;\,\textbf{store}_{\textbf{H}}\ x_H$$

$$\boldsymbol{C}[\![\textbf{if}\ e\ \textbf{then}\ c_1\ \textbf{else}\ c_2]\!] = \boldsymbol{C}[\![e]\!]\,;\,\textbf{branch}(\boldsymbol{C}[\![c_1]\!], \boldsymbol{C}[\![c_2]\!])$$

$$\boldsymbol{C}[\![\textbf{while}\ e\ \textbf{do}\ c]\!] = \textbf{loop}(\boldsymbol{C}[\![e]\!], \boldsymbol{C}[\![c]\!])$$

$$\boldsymbol{C}[\![c_1 ; c_2]\!] = \boldsymbol{C}[\![c_1]\!] ; \boldsymbol{C}[\![c_2]\!]$$

Properties

Property 1

$$\langle \mathbf{C}[\![e]\!], vs, s\rangle \rhd^* (v : vs, s) \;\text{ if }\; [\![e]\!]s = v$$

Property 2

$$\langle \mathbf{C}[\![S]\!], vs, s\rangle \rhd^* (vs, s') \;\text{ if }\; [\![S]\!]s = s'$$

Security types

$$pc, ls \vdash \textbf{push}\ n \leadsto low :: ls$$

$$pc, ls \vdash \textbf{fetch}_\textbf{L}\ x_L \leadsto low :: ls$$

$$pc, ls \vdash \textbf{fetch}_\textbf{H}\ x_H \leadsto high :: ls$$

$$low, low :: ls \vdash \textbf{store}_\textbf{L}\ x_L \leadsto ls$$

$$pc, st :: ls \vdash \textbf{store}_\textbf{H}\ x_H \leadsto ls$$

$$pc, st_1 :: st_2 :: ls \vdash \textbf{add} \leadsto \textbf{max}(st_2, st_1) :: ls$$

$$\frac{pc_1, ls \vdash c_1 \leadsto ls' \qquad pc_2, ls' \vdash c_2 \leadsto ls''}{\textbf{min}(pc_1, pc_2), ls \vdash c_1 ; c_2 \leadsto ls''}$$

Security types

$$\frac{pc_1, ls \vdash c_1 \rightsquigarrow ls \qquad pc_2, ls \vdash c_2 \rightsquigarrow ls \qquad st \leq \mathbf{min}(pc_1, pc_2)}{\mathbf{min}(pc_1, pc_2), st :: ls \vdash \mathbf{branch}(c_1, c_2) \rightsquigarrow ls}$$

$$\frac{pc_1, ls \vdash c_1 \rightsquigarrow st :: ls' \qquad pc_2, ls' \vdash c_2 \rightsquigarrow ls \qquad st \leq pc_2}{\mathbf{min}(pc_1, pc_2), ls \vdash \mathbf{loop}(c_1, c_2) \rightsquigarrow ls'}$$

Security types in Haskell

```
data CodeS env pc env' where
  Push   :: Int  -> CodeS env High (Cons Low env)
  FetchL :: RefL -> CodeS env High (Cons Low env)
  FetchH :: RefH -> CodeS env High (Cons High env)
  StoreL :: RefL -> CodeS (Cons Low env) Low env
  StoreH :: RefH -> CodeS (Cons st env) High env
  Sum    :: CodeS (Cons st1 (Cons st2 env)) High
                  (Cons (Max st2 st1) env)
  App    :: CodeS env pc1 env1
         -> CodeS env1 pc2 env2
         -> CodeS env (Min pc1 pc2) env2
  ...
```

Security types en Haskell

```
  ...
  Branch :: CodeS env pc1 env
         -> CodeS env pc2 env
         -> LEq st (Min pc1 pc2)
         -> CodeS (Cons st env) (Min pc1 pc2) env
  Loop   :: CodeS env pc1 (Cons st env')
         -> CodeS env' pc2 env
         -> LEq st pc2
         -> CodeS env (Min pc1 pc2) env'

data Nil
data Cons st env
```

Compilation

$$\mathcal{C}[\![n]\!] = \textbf{push}\ n$$
$$\mathcal{C}[\![x_L]\!] = \textbf{fetch}_{\textbf{L}}\ x_L$$
$$\mathcal{C}[\![x_H]\!] = \textbf{fetch}_{\textbf{H}}\ x_H$$
$$\mathcal{C}[\![e_1 + e_2]\!] = \mathcal{C}[\![e_1]\!]\,;\mathcal{C}[\![e_2]\!]\,;\textbf{add}$$
$$\mathcal{C}[\![x_L := e]\!] = \mathcal{C}[\![e]\!]\,;\textbf{store}_{\textbf{L}}\ x_L$$
$$\mathcal{C}[\![x_H := e]\!] = \mathcal{C}[\![e]\!]\,;\textbf{store}_{\textbf{H}}\ x_H$$
$$\mathcal{C}[\![\textbf{if}\ e\ \textbf{then}\ c_1\ \textbf{else}\ c_2]\!] = \mathcal{C}[\![e]\!]\,;\textbf{branch}(\mathcal{C}[\![c_1]\!], \mathcal{C}[\![c_2]\!])$$
$$\mathcal{C}[\![\textbf{while}\ e\ \textbf{do}\ c]\!] = \textbf{loop}(\mathcal{C}[\![e]\!], \mathcal{C}[\![c]\!])$$
$$\mathcal{C}[\![c_1 ; c_2]\!] = \mathcal{C}[\![c_1]\!] ; \mathcal{C}[\![c_2]\!]$$

Compilation in Haskell

```
compileExp :: Exp st
           -> CodeS env High (Cons st env)

compileExp (IntVal n)   = Push n
compileExp (VarL var)   = FetchL var
compileExp (VarH var)   = FetchH var
compileExp (Add e1 e2)  = App (App (compileExp e1)
                                   (compileExp e2))
                              Sum
```

Compilation in Haskell

```
compiler :: Com pc -> CodeS env pc env
compiler (AssL var e)     = App (compileExp e)
                                (StoreL var)
compiler (AssH var e)     = App (compileExp e)
                                (StoreH var)
compiler (Seq c1 c2)      = App (compiler c1)
                                (compiler c2)
compiler (If0 e c1 c2 p) = App (compileExp e)
                                (Branch (compiler c1
                                        (compiler c2
                                       p)
compiler (While e c p)    = Loop (compileExp e)
                                 (compiler c)
                                p
```