Static Analysis of Multi-Staged Programs via Unstaging Translation

Wontae Choi[1]   Baris Aktemur[2]   Kwangkeun Yi[1]   Makoto Tatsuta[3]

[1] Seoul National University
Korea

[2] Ozyegin University/UIUC
Turkey

[3] National Institute of Informatics
Japan

POPL 2011 @ Austin, USA

Multi-Staged Programming

Program codes are first class objects
"meta programming"

Multi-Staged Programming

A general concept that subsumes

- C++ and Haskell templates
- web programming's runtime code generation
- macro
- Lisp's quasi-quotation
- partial evaluation

Multi-Staged Programming

Divides a computation into stages

- stage 0 program : conventional program
- stage n+1 program : code value at stage n

Multi-Staged Programming

In presentation, we are going to use Lisp-like syntax + 2 stages

```
e  :=  . . .
    |  ‘e          code as a data
    |  ,e          code composition
    |  run  e      code execution
```

Multi-Staged Programming Examples

- code as a value

```
‘(1+1)
```

- open code

```
‘(x+1)
```

- code composition and intentional variable capturing

```
let y = ‘(x+1) in ‘(λx. ,y)  →  ‘(λx.x+1)
```

- code execution

```
run ‘(1+1)
```

Contents

- Problem in Static Analysis
- Translation
- Projection
- Conclusion

Problem in Static Analysis

- Program text to analyze is dynamic
- Conventional analysis may fail to handle "run"

```
let spow n = if (n=0) then `1 else `(x* ,(spow (n-1)))
in let pow = `(λx. ,(spow input))
in (run pow) 2
```

Problem in Static Analysis

- Program text to analyze is dynamic
- Conventional analysis may fail to handle "run"

```
{‘1, ‘(x*1), ‘(x*x*1), ...}
```

```
let spow n = if (n=0) then ‘1 else ‘(x* ,(spow (n-1)))
in let pow = ‘(λx. ,(spow input))
in (run pow) 2
```

Problem in Static Analysis

- Program text to analyze is dynamic
- Conventional analysis may fail to handle “run”

static estimation

```
{‘1, ‘(x*1), ‘(x*x*1), ...}  →  S -> 1 | x*S
```

```
let spow n = if (n=0) then ‘1 else ‘(x* ,(spow (n-1)))
in let pow = ‘(λx. ,(spow input))
in (run pow) 2
```

Problem in Static Analysis

- Program text to analyze is dynamic
- Conventional analysis may fail to handle “run”

static estimation

{‘1, ‘(x*1), ‘(x*x*1), ...}

S -> 1 | x*S

```
let spow n = if (n=0) then ‘1 else ‘(x* ,(spow (n-1)))
in let pow = ‘(λx. ,(spow input))
in (run pow) 2
```

pow -> λx.S

Problem in Static Analysis

- Program text to analyze is dynamic
- Conventional analysis may fail to handle "run"

static estimation

S -> 1 | x*S

{‘1, ‘(x*1), ‘(x*x*1), ...}

```
let spow n = if (n=0) then ‘1 else ‘(x* ,(spow (n-1)))
in let pow = ‘(λx. ,(spow input))
in (run pow) 2
```

pow -> λx.S

{λx.1, λx.x*1, λx.x*x*1, ...}

concretization

Problem in Static Analysis

- Program text to analyze is dynamic
- Conventional analysis may fail to handle "run"

static estimation

```
{‘1, ‘(x*1), ‘(x*x*1), ...}   ->   S -> 1 | x*S
```

```
let spow n = if (n=0) then ‘1 else ‘(x* ,(spow (n-1)))
in let pow = ‘(λx. ,(spow input))
in (run pow) 2
```

```
pow -> λx.S
```

```
{λx.1, λx.x*1, λx.x*x*1, ....}
```

concretization

Unrealizable!

Our Contribution

- An unstaging translation which preserves the semantics
- An analysis framework based on the translation

$$
\begin{tikzcd}
e : \text{staged program} \arrow[d, mapsto, "\textbf{translation}"] & \text{analysis result } [\![\hat{e}]\!] \text{ for } e \\
\underline{e} : \text{conventional program} \arrow[r, "\textbf{conventional analysis}"'] & \text{analysis result } [\![\hat{\underline{e}}]\!] \text{ for } \underline{e} \arrow[u, "\textbf{projection}"']
\end{tikzcd}
$$

Theorems

- Simulation

$$\begin{tikzcd} e \arrow[r] \arrow[d, mapsto] & e' \arrow[d, mapsto] \\ \underline{e} & \underline{e'} \end{tikzcd} \quad \Longrightarrow \quad \underline{e} \longrightarrow \underline{e'}$$

- Inversion

$$e \longrightarrow e' \quad \Longrightarrow \quad \begin{tikzcd} e \arrow[d, mapsto] & e' \\ \underline{e} \arrow[r] & \underline{e'} \arrow[u, mapsto] \end{tikzcd}$$

- Sound Projection

$$\begin{tikzcd} e \arrow[d, mapsto] \\ \underline{e} \end{tikzcd} \qquad \begin{tikzcd} \llbracket e \rrbracket \in D \arrow[r, shift right, "\alpha"'] & \hat{D} \ni \llbracket \hat{e} \rrbracket \arrow[l, shift right, "\gamma"'] \\ \llbracket \underline{e} \rrbracket \in \underline{D} \arrow[u, "\pi"] \arrow[r, shift right, "\underline{\alpha}"'] & \underline{\hat{D}} \ni \llbracket \underline{\hat{e}} \rrbracket \arrow[u, "\hat{\pi}"'] \arrow[l, shift right, "\underline{\gamma}"'] \end{tikzcd}$$

Languages

Source Staged Language λ_S

```
e := λx.e
   | e e
   | x
   | ‘e
   | ,e
   | run e
```

$\longmapsto$

Target Unstaged Language λ_R

```
e := λx.e
   | e e
   | x
   | {}
   | e{x=e}
   | e.x
```

Translation Ideas (1/2)

- code expression to function expression

$$`(1+1) \longmapsto \lambda\rho.1+1$$

- free variable to record lookup

$$`(x+1) \longmapsto \lambda\rho.(\rho.x)+1$$

- variable capturing to record passing

$$`(\lambda x.\ ,(\ `(x+1)\ )\ ) \longmapsto \lambda\rho_1.\lambda x.((\lambda\rho_2.(\rho_2.x)+1)\ (\rho_1\{x=x\}))$$

- run expression to application expression

$$\mathtt{run}\ `(1+1) \longmapsto (\lambda\rho.1+1)\ \{\}$$

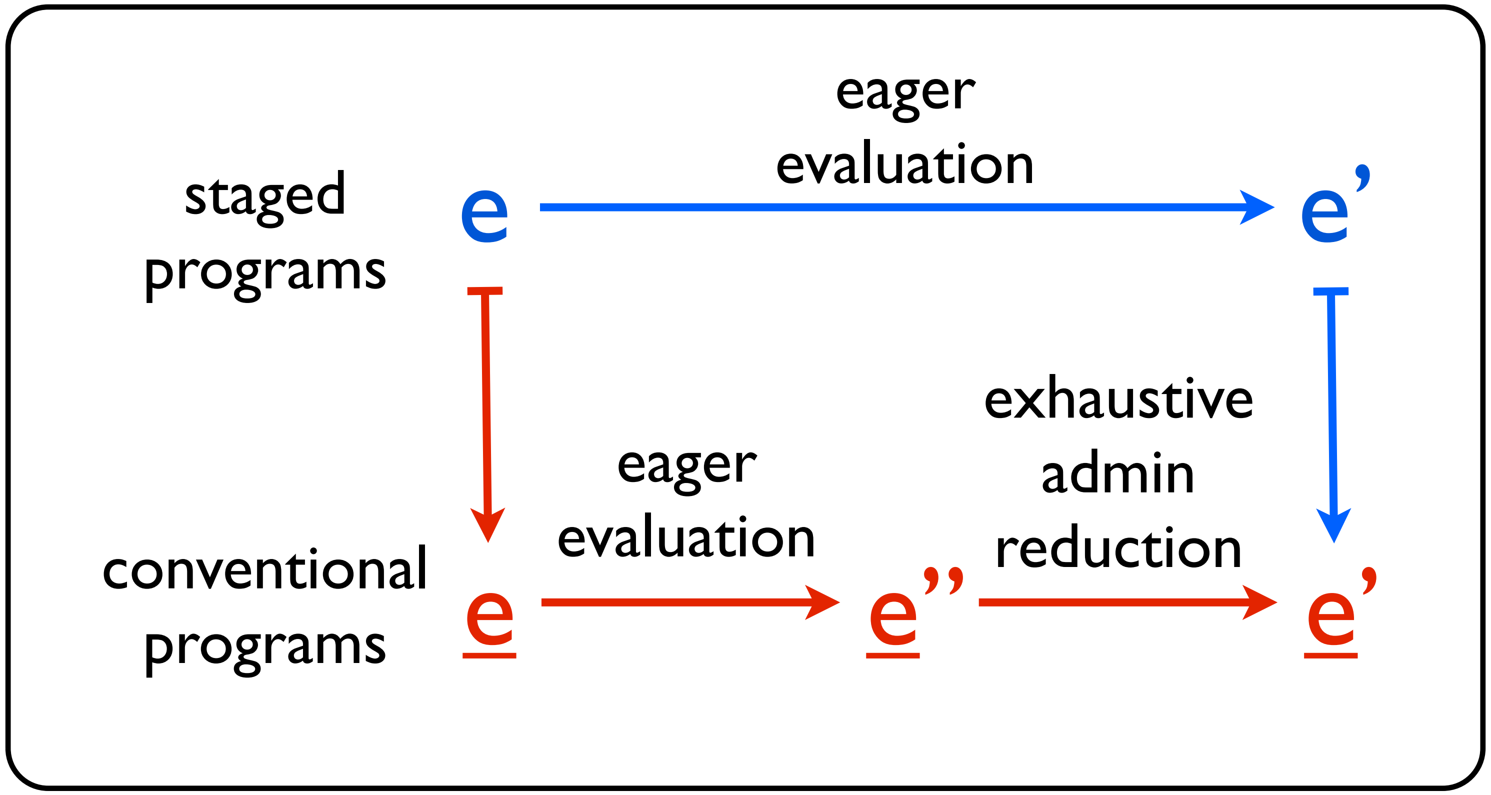
Translation Ideas (2/2)

- to preserve the evaluation order

$$\underbrace{\texttt{box}\ (\ldots\texttt{unbox}\ (\boxed{\cdot})\ldots\texttt{unbox}\ (\boxed{\cdot})\ldots)}$$

$$(\lambda h_1.(\lambda h_2.(\lambda\rho.\ \ \ldots\ \ h_1\ \rho\ \ \ldots\ \ h_2\ \rho\ \ \ldots))(\boxed{\cdot}))(\boxed{\cdot})$$

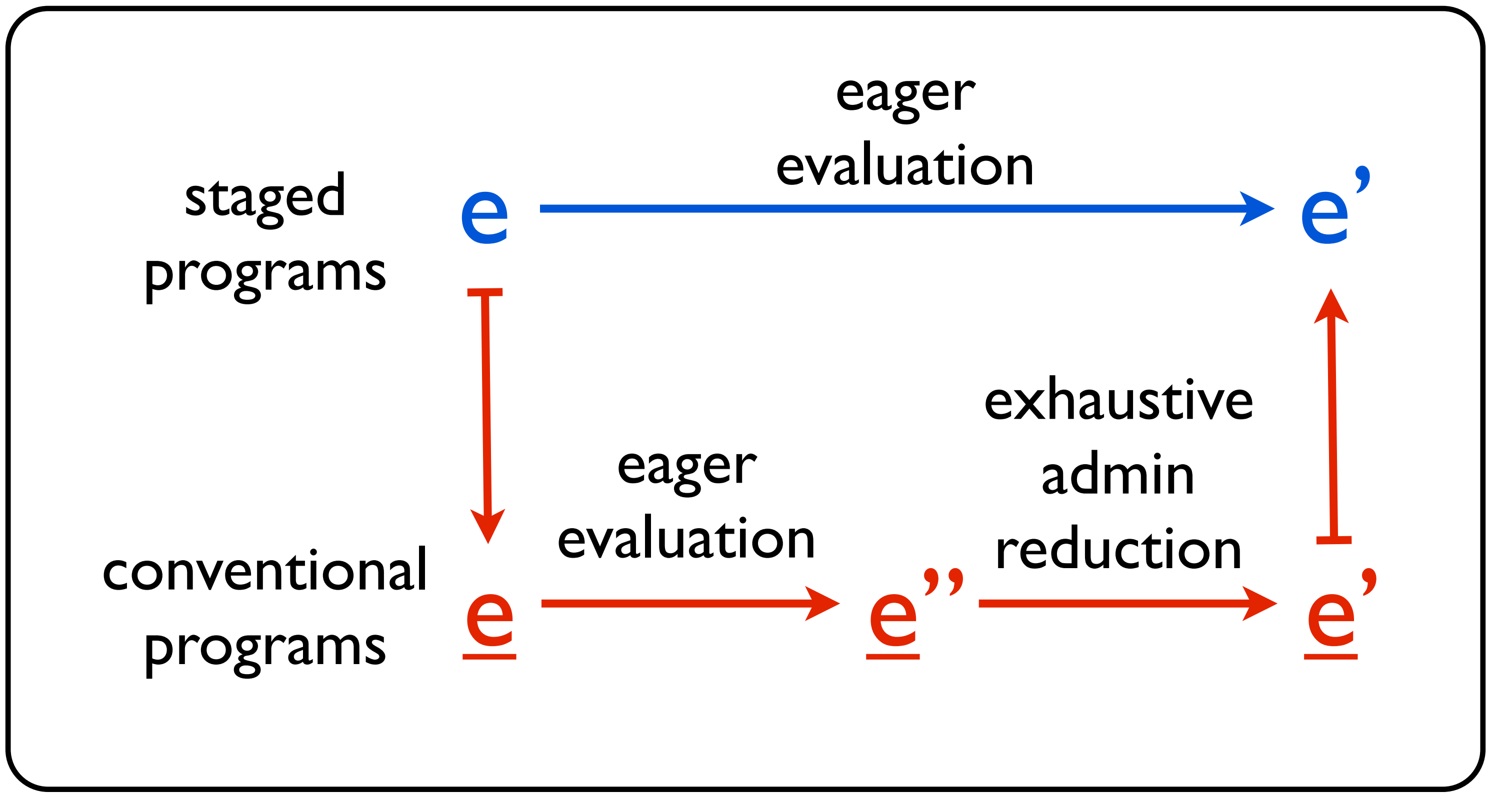
Simulation



evaluation + translation

≅ translation + evaluation + admin reduction

Inversion



evaluation

≅ translation + evaluation + admin reduction + inversion

Static Analysis Framework

$$\begin{tikzcd}
e \arrow[d, mapsto] & & {[\![e]\!] \in D} \arrow[r, shift right, "\alpha"'] & \hat{D} \arrow[l, shift right, "\gamma"'] \\
\underline{e} & & & {\underline{\hat{D}} \ni \widehat{[\![\underline{e}]\!]}} \arrow[u, "\hat{\pi}"]
\end{tikzcd}$$

Implementation	$e \mapsto \underline{e}$	$\widehat{[\![\underline{e}]\!]}$	$\hat{\pi}$
Requirement	$\alpha[\![e]\!] \sqsubseteq \hat{\pi}\widehat{[\![\underline{e}]\!]}$		

Static Analysis Framework

$$
\begin{tikzcd}
e \arrow[d, mapsto] & \llbracket e \rrbracket \in D \arrow[r, shift right, "\alpha"'] & \hat{D} \arrow[l, shift right, "\gamma"'] \\
\underline{e} & \llbracket \underline{e} \rrbracket \in \underline{D} \arrow[u, "\pi"] \arrow[r, shift right, "\underline{\alpha}"'] & \underline{\hat{D}} \ni \widehat{\llbracket \underline{e} \rrbracket} \arrow[l, shift right, "\underline{\gamma}"'] \arrow[u, "\hat{\pi}"']
\end{tikzcd}
$$

Implementation $\quad e \mapsto \underline{e} \qquad \widehat{\llbracket \underline{e} \rrbracket} \qquad \hat{\pi}$

Requirement $\quad \alpha \llbracket e \rrbracket \sqsubseteq \hat{\pi} \widehat{\llbracket \underline{e} \rrbracket}$

Theorem
$$
\left.\begin{array}{l} \llbracket e \rrbracket \sqsubseteq \pi \llbracket \underline{e} \rrbracket \\ \alpha \circ \pi \circ \underline{\gamma} \sqsubseteq \hat{\pi} \end{array}\right\} \Longrightarrow \alpha \llbracket e \rrbracket \sqsubseteq \hat{\pi} \widehat{\llbracket \underline{e} \rrbracket}
$$

$[\![e]\!] \in D$

Example : Value Analysis

Setting 1) collecting analysis $[\![e]\!]$ for the staged program (uncomputable)

staged program

```
let
  x = `0                 (* indexed as ρ1 *)
  repeat
    x = `(,x+2)          (* indexed as ρ2 *)
  until ?
in
  run x
```

```
      x    has   {`0, `(0+2), `(0+2+2), ....}
(run x)    has   {0, 2, 4, 6, ....}
```

$[\![e]\!] \in D$

$[\![e]\!] \in \underline{D}$

Example : Value Analysis

Setting 2) collecting analysis $[\![\underline{e}]\!]$ for it's translated version (uncomputable)

translated program

```
let
  x = (λρ1.0)
  repeat
    x = ((λh.λρ2.(h ρ2)+2) x)
  until ?
in
  x {}
```

x, h	has	$\{\langle\lambda\rho_1.0,\emptyset\rangle, \langle\lambda\rho_2.(h\ \rho_2)+2, \{h\mapsto\langle\lambda\rho_1.0,\emptyset\rangle\}\rangle, \ldots .\}$
ρ_1, ρ_2	has	{ }
(x {})	has	{0, 2, 4, 6, ...}

$$\begin{array}{c} [\![e]\!] \in D \\ \pi \uparrow \\ [\![e]\!] \in \underline{D} \end{array}$$

Example : Value Analysis

Setting 3) collecting projection π (uncomputable)

translation + collecting analysis (part of)

$$\begin{array}{lll} x, h & \text{has} & \{ \langle \lambda\rho_1.0, \emptyset \rangle, \\ & & \langle \lambda\rho_2.(h\ \ \rho_2)+2, \{h \mapsto \langle \lambda\rho_1.0, \emptyset \rangle\} \rangle, \\ & & \ldots.\ \} \\ \rho_1, \rho_2 & \text{has} & \{ \} \end{array}$$

$\xrightarrow{\pi}$

projection result

$$\begin{array}{lll} x & \text{has} & \{ `0, `(0+2), \\ & & \ldots. \} \end{array}$$

- inverse translation + removing unnecessary stuff
- intuition : $\begin{array}{l} \text{“}\lambda\rho\text{”} \\ \text{“h } \rho\text{”} \end{array} \xrightarrow{\hat{\pi}} \begin{array}{l} \text{“code } \rho\text{”} \\ \text{“code-filling by h”} \end{array}$
- π satisfies $\hat{\pi}$'s first safety condition : $[\![e]\!] \sqsubseteq \pi [\![e]\!]$

$$
\begin{tikzcd}
[\![e]\!] \in D & \\
[\![e]\!] \in \underline{D} \arrow[u, "\pi"] \arrow[r, shift right, "\underline{\alpha}"'] & \hat{\underline{D}} \ni [\![\hat{e}]\!] \arrow[l, shift right, "\underline{\gamma}"']
\end{tikzcd}
$$

Example : Value Analysis

(computable) **static** analysis $[\![\hat{e}]\!]$ for the translated version

translated program

```
let
  x = (λρ1.0)
  repeat
    x = ((λh.λρ2.(h ρ2)+2) x)
  until ?
in
  x {}
```

x	has	$\lambda\rho_1.0$
x	has	$\lambda\rho_2.(h\ \rho_2)+2$
h	has	$\lambda\rho_1.0$
h	has	$\lambda\rho_2.(h\ \rho_2)+2$
ρ_1, ρ_2	has	{}
(x {})	has	0
(x {})	has	$(h\ \rho_2) + 2$
$(h\ \rho_2)$	has	0
$(h\ \rho_2)$	has	$(h\ \rho_2) + 2$

set-constraint style 0-CFA

$$\begin{tikzcd}
\llbracket e \rrbracket \in D & & \\
\llbracket e \rrbracket \in \underline{D} \arrow[u, "\pi"] \arrow[r, shift right, "\underline{\alpha}"'] & \underline{\hat{D}} \ni \underline{\llbracket \hat{e} \rrbracket} \arrow[l, shift right, "\underline{\gamma}"'] &
\end{tikzcd}$$

Example : Value Analysis

(computable) **static** analysis $\llbracket \hat{e} \rrbracket$ for the translated version

translated program

```
let
  x = (λρ1.0)
  repeat
    x = ((λh.λρ2.(h ρ2)+2) x)
  until ?
in
  x {}
```

x	has	λρ1.0
x	has	λρ2.(h ρ2)+2
h	has	λρ1.0
h	has	λρ2.(h ρ2)+2
ρ1, ρ2	has	{}
(x {})	has	0
(x {})	has	(h ρ2) + 2
(h ρ2)	has	0
(h ρ2)	has	(h ρ2) + 2

(x {})'s values in grammar : V -> 0 | V+2

$$\begin{tikzcd} {[\![e]\!]} \in D \arrow[r, shift left, "\alpha"'] & \hat{D} \arrow[l, shift left, "\gamma"'] \\ {[\![e]\!]} \in \underline{D} \arrow[u, "\pi"] \arrow[r, shift left, "\underline{\alpha}"'] & \underline{\hat{D}} \ni [\![\hat{e}]\!] \arrow[u, "\hat{\pi}"'] \arrow[l, shift left, "\underline{\gamma}"'] \end{tikzcd}$$

Example : Value Analysis

(computable) **abstract** projection

static analysis for the translated program

x	has	$\lambda\rho_1.0$
x	has	$\lambda\rho_2.(h\ \rho_2)+2$
h	has	$\lambda\rho_1.0$
h	has	$\lambda\rho_2.(h\ \rho_2)+2$
(x {})	has	V -> 0 | V+2

$\xrightarrow{\hat{\pi}}$

abstract projection result

x	has	S_1 -> ρ_1
x	has	S_2 -> ρ_2(S)
		S -> ρ_1 | ρ_2(S)
(run x)	has	V -> 0 | V+2

- intuition : "λρ" $\xrightarrow{\hat{\pi}}$ "code ρ"
  "h ρ" "code-filling by h"
- $\hat{\pi}$ satisfies the second safety condition : $\alpha \circ \pi \circ \underline{\gamma} \sqsubseteq \hat{\pi}$

$$
\begin{tikzcd}
\llbracket e \rrbracket \in D \arrow[r, shift left, "\alpha"'] & \hat{D} \arrow[l, shift left, "\gamma"'] \\
\llbracket \underline{e} \rrbracket \in \underline{D} \arrow[u, "\pi"] \arrow[r, shift left, "\underline{\alpha}"'] & \hat{\underline{D}} \ni \llbracket \hat{\underline{e}} \rrbracket \arrow[u, "\hat{\pi}"'] \arrow[l, shift left, "\underline{\gamma}"']
\end{tikzcd}
$$

Example : Value Analysis

final result for the staged program

staged program

```
let
  x = `0            (* indexed as ρ1 *)
  repeat
    x = `(,x+2)   (* indexed as ρ2 *)
  until ?
in
  run x
```

translation + static analysis + projection

```
      x    has   S1  ->  ρ1
      x    has   S2  ->  ρ2(S)
                 S  ->  ρ1  |  ρ2(S)
(run x)    has   V  ->  0  |  V+2
```

"translation + static analysis + projection" is sound

$$\alpha\llbracket e \rrbracket \sqsubseteq \hat{\pi}\llbracket \hat{\underline{e}} \rrbracket$$

Conclusion

- Semantics-preserving translation from staged programs to conventional programs
- Sound analysis framework using the translation

Conclusion

- Semantics-preserving translation from staged programs to conventional programs
- Sound analysis framework using the translation

Unstaging + Conventional static analysis
That's sufficient!

Thank you