다단계 프로그래밍 해커를 위한 타입 시스템

A Polymorphic Modal Type System for Lisp-like Multi-Staged Languages [1]

이 광근

서울대

11/19/2005 SIGPL@항공대

[1] 김 익순과 공동작업 + C. Calcagno

그동안

김 익순 박사와

그동안

김 익순 박사와

- 3년간의 어둡고 긴 터널

그동안

김 익순 박사와

- 3년간의 어둡고 긴 터널

김: "스스로 진화하는 프로그램을 작성하고 실행시키는 시스템을 만들고 싶어요."

그동안

김 익순 박사와

- 3년간의 어둡고 긴 터널

김: "스스로 진화하는 프로그램을 작성하고 실행시키는 시스템을 만들고 싶어요."

이: "다단계 프로그래밍 언어 시스템이란건데."

그동안

김 익순 박사와

- 3년간의 어둡고 긴 터널

김: "스스로 진화하는 프로그램을 작성하고 실행시키는 시스템을 만들고 싶어요."

이: "다단계 프로그래밍 언어 시스템이란건데."

김: "그런 시스템으로 (이런일저런일)" (공상과학)

그동안

김 익순 박사와

- 3년간의 어둡고 긴 터널

김: "스스로 진화하는 프로그램을 작성하고 실행시키는 시스템을 만들고 싶어요."

이: "다단계 프로그래밍 언어 시스템이란건데."

김: "그런 시스템으로 (이런일저런일)" (공상과학)

이: "제대로 된 타입 시스템 우선 만들자."

그동안

김 익순 박사와

- 3년간의 어둡고 긴 터널

김: "스스로 진화하는 프로그램을 작성하고 실행시키는 시스템을 만들고 싶어요."

이: "다단계 프로그래밍 언어 시스템이란건데."

김: "그런 시스템으로 (이런일저런일)" (공상과학)

이: "제대로 된 타입 시스템 우선 만들자."

- 그리곤 터널로 빨려들어가다.

다단계 프로그래밍 Multi-Staged Programming (1/2)

프로그램 텍스트가 프로그램의 데이타

program texts are the first class objects

meta programming

- 단계0: 일반 프로그래밍
- 단계$n+1$: 단계n 프로그램이 만들어내는 프로그램

다단계 프로그래밍 Multi-Staged Programming (2/2)

이미 고안된 여러가지 것들을 포섭하는 일반 개념

- Lisp/Scheme의 준인용(quasi-quatation)
- 부분계산(partial evaluation)
- 실행중에 실행코드 만들기(runtime code generation)
- 매크로(macro)
- “진화하는 코드(evolutionary code)” “알아서 잘하는 코드(autonomous code)”

다단계 프로그래밍 언어

$$
\begin{array}{rcll}
e & ::= & c \mid x \mid \lambda x.e \mid e\ e & \\
 & \mid & \text{`}e & \text{code constant} \\
 & \mid & \text{,}e & \text{code composition} \\
 & \mid & \texttt{run}\ e & \text{execute code} \\
 & \mid & \texttt{lift}\ e & \text{value to code}
\end{array}
$$

다단계 프로그래밍 예(1/5)

단순 매크로 프로그래밍

```
let NULL = `0
let body = `(if x= ,NULL then abort() else ...)
in run body
```

다단계 프로그래밍 예(2/5)

인자를 가진 매크로 프로그래밍

```
let or(a,b) = ‘(let v = ,a in if v then v else ,b end)
in if run(or code1 code2) then ... else ...
```

다단계 프로그래밍 예(2/5)

인자를 가진 매크로 프로그래밍

```
let or(a,b) = ‘(let v = ,a in if v then v else ,b end)
in if run(or code1 code2) then ... else ...
```

또는

```
let repeatUntil(s,c) = ‘(,s; while ,c do ,s)
let xloop = repeatUntil(‘(x = x+1), ‘(x<10))
let x = 0
in run xloop
```

다단계 프로그래밍 예(3/5)

"자동진화"/특화기(specialization)/부분 계산(partial evaluation)

```
power(n,x) = if n=0 then 1 else x * power(n-1,x)
```

대

```
power(3,x) = x*x*x
```

다단계 프로그래밍 예(3/5)

"자동진화"/특화기(specialization)/부분 계산(partial evaluation)

```
power(n,x) = if n=0 then 1 else x * power(n-1,x)
```

대

```
power(3,x) = x*x*x
```

이렇게 준비

```
let spower(n) =
  if n=0 then `1 else `(x * ,(spower (n-1)) )
let fastpower100 = `(λx.  ,(spower 100))
in (run fastpower100)(10); (run fastpower100)(20)
```

다단계 프로그래밍 예(4/5)

"자동진화"/특화기(specialization)/부분 계산(partial evaluation)

```
let map [] f = []
  | map (x::r) f = (f x)::(map r f)
```

대

```
let map [x,y,z] f = [f x, f y, f z]
```

다단계 프로그래밍 예(4/5)

"자동진화"/특화기(specialization)/부분 계산(partial evaluation)

```
let map [] f = []
  | map (x::r) f = (f x)::(map r f)
```

대

```
let map [x,y,z] f = [f x, f y, f z]
```

이렇게 준비:

```
let smap [] = `[]
  | smap (x::r) = `((f ,x) ::  ,(smap r))
let fastmap3 = `(λ f.  ,(smap [1,2,3]))
in (run fastmap3) inc; (run fastmap3) dec
```

다단계 프로그래밍 예(5/5)

코드가 저장되고 갱신되고

```
let a = ref ‘1
    f = ‘(λ x.  ,(a := ‘(x+1); ‘2))
in !a
```

다단계 프로그래밍 언어 연구

다단계 프로그래밍을 즐겁게 할 수 있는 언어는 뭘까?

다단계 프로그래밍 언어 연구

다단계 프로그래밍을 즐겁게 할 수 있는 언어는 뭘까?

- 다단계 프로그래밍은 정신없다

다단계 프로그래밍 언어 연구

다단계 프로그래밍을 즐겁게 할 수 있는 언어는 뭘까?

- 다단계 프로그래밍은 정신없다
- 내가 제정신이었는 지 확인해 주는 도구가 있었으면

다단계 프로그래밍 언어 연구

다단계 프로그래밍을 즐겁게 할 수 있는 언어는 뭘까?

- 다단계 프로그래밍은 정신없다
- 내가 제정신이었는 지 확인해 주는 도구가 있었으면

하나의 답: 다단계 언어의 안전한 타입 시스템

논문의 한일

다단계 프로그래밍을 위한 타입 시스템 고안

- 넉넉한 타입 시스템
  - 다단계 프로그래밍의 실제를 모두 허용
- 안전한 타입 시스템
  - 믿어도 좋다
- 알고리즘이 좋은 타입 시스템
  - 충실한 구현

좋은 타입 시스템(1/2)

다단계 프로그래밍의 실제를 모두 허용해야(pragmatism)

- 제약이 없는 코드 데이타(code as 1st class object)
- 코드안에 자유로운 이름들(open code): `‘(x+1)`
- 프로그래머가 선택하도록:
  - 묶인 이름의 변동에 민감(non alpha-equivalence, unhygienic manipulation of code): 단계 > 0
  - 묶인 이름의 변동에 불감(alpha-equivalence, hygienic manipulation of code): 단계 0

좋은 타입 시스템(2/2)

지금까지의 성공을 계속 유지해야(conservative extension)

- 널리 쓰이는, 실용적인 안전한 타입 시스템 유지
- ML의 let-polymorphism등
- 다단계 타입 시스템 때문에 예전에 좋던것이 없어지진 말아야

다른 시스템과 차이

1 closed code and eval	2 open code
3 reference	4 type inference
5 variable-capturing substitution	6 capture-avoiding substitution

Our System	+1+2+3+4+5+6	
$\lambda^{\Box}$	+1(+3+4)-2-5-6	(1996, 2001)
$\lambda^{\bigcirc}$	+2+6-1-3-4-5	(1996)
MetaML, MetaOcaml	+1+2+4+6-3-5	(2000, 2001)
Nanevsky-Pfenning	+1+2(+3)+6-4-5	(2002)
λ^{+}_{code}	+1+2+3+5-4-6	(2003)
λ^{i}	+1+2+4+6-3-5	(2004)

단순 타입 시스템 λ^{sim}_{open} (1/3)

- 코드의 타입: 코드의 자유변수에 대한 타입들을 가진다

$$\text{`(x+1)} : \Box(\{x : int\}\rho \triangleright int)$$

단순 타입 시스템 λ^{sim}_{open} (2/3)

$$
\begin{array}{rrcl}
Types & A, B & ::= & \iota \mid A \to B \mid \Box(\Gamma \triangleright A) \mid A\ \mathtt{ref} \\
TypeEnvironments & \Gamma & \in & \mathit{TyEnv} = \mathit{Var} \xrightarrow{fin} \mathit{Type} \\
StoreTypings & \Sigma & \in & \mathit{ST} = \mathit{Loc} \xrightarrow{fin} \mathit{Type}
\end{array}
$$

타입 판단(typing judgment)

$$\boxed{\Sigma; \Gamma_0 \cdots \Gamma_n \vdash e : A}$$

단순 타입 시스템 λ^{sim}_{open} (3/3)

(TSVAR)
$$\frac{\Gamma_n(x) = A}{\Sigma; \Gamma_0 \cdots \Gamma_n \vdash x : A}$$

(TSABS)
$$\frac{\Sigma; \Gamma_0 \cdots \Gamma_n + x : A \vdash e : B}{\Sigma; \Gamma_0 \cdots \Gamma_n \vdash \lambda x.e : A \to B}$$

(TSAPP)
$$\frac{\Sigma; \Gamma_0 \cdots \Gamma_n \vdash e_1 : A \to B \qquad \Sigma; \Gamma_0 \cdots \Gamma_n \vdash e_2 : A}{\Sigma; \Gamma_0 \cdots \Gamma_n \vdash e_1 e_2 : B}$$

(TSBOX)
$$\frac{\Sigma; \Gamma_0 \cdots \Gamma_n \Gamma \vdash e : A}{\Sigma; \Gamma_0 \cdots \Gamma_n \vdash \text{`}e : \Box(\Gamma \triangleright A)}$$

(TSUNBOX)
$$\frac{\Sigma; \Gamma_0 \cdots \Gamma_{n-1} \vdash e : \Box(\Gamma_n \triangleright A) \qquad k > 0}{\Sigma; \Gamma_0 \cdots \Gamma_n \vdash \text{,}e : A}$$

(TSEVAL)
$$\frac{\Sigma; \Gamma_0 \cdots \Gamma_n \vdash e : \Box(\varnothing \triangleright A)}{\Sigma; \Gamma_0 \cdots \Gamma_n \vdash \mathtt{run}\, e : A}$$

다형(polymorphic) 타입 시스템 λ^{poly}_{open} (1/3)

ML의 다형 타입 시스템을 Rémy의 레코드 타입시스템으로 확장

- `if` e `then ‘(x+1) else ‘1`: $\boxed{\Box(\{x : int\}\rho \triangleright int)}$
  - else-branch: $\Box(\rho' \triangleright int)$
  - then-branch: $\Box(\{x : int\}\rho \triangleright int)$
- `let val x = ‘y in ‘(,x + 1); ‘((,x 1) + z) end`:
  `x`: $\boxed{\forall\alpha\forall\rho.\Box(\{y : \alpha\}\rho \triangleright \alpha)}$
  - first `x`: $\Box(\{y : int\} \triangleright int)$
  - second `x`: $\Box(\{y : int \rightarrow int, z : int\} \triangleright int \rightarrow int)$

다형 타입 시스템 λ^{poly}_{open} (2/3)

Store Typings	Σ	$\in$	*StoTy*	$=$	*Loc* $\xrightarrow{fin}$ *Type*	
Type Environments	Γ	$\in$	*TyEnv*	$=$	*Var* $\xrightarrow{fin}$ *Field*	
				$::=$	$\{x_i : A_i\}_1^m \mid \{x_i : A_i\}_1^k \rho$	
Type Env. Vars	ρ	$\in$	*TyEnvVar*			
Type Schemes	τ, σ	$\in$	*TyScheme*	$::=$	$\forall \xi.\tau \mid A$	
Type Scheme Env	Δ	$\in$	*TySchemeEnv*	$=$	*Var* $\xrightarrow{fin}$ *FieldScheme*	
				$::=$	$\{x_i : \mu_i\}_1^m \mid \{x_i : \mu_i\}_1^k \rho$	

다형 타입 시스템 λ^{poly}_{open} (2/3)

Store Typings	Σ	$\in$	*StoTy*	$=$	*Loc* $\xrightarrow{fin}$ *Type*	
Type Environments	Γ	$\in$	*TyEnv*	$=$	*Var* $\xrightarrow{fin}$ *Field*	
				$::=$	$\{x_i : A_i\}_1^m \mid \{x_i : A_i\}_1^k \rho$	
Type Env. Vars	ρ	$\in$	*TyEnvVar*			
Type Schemes	τ, σ	$\in$	*TyScheme*	$::=$	$\forall \xi.\tau \mid A$	
Type Scheme Env	Δ	$\in$	*TySchemeEnv*	$=$	*Var* $\xrightarrow{fin}$ *FieldScheme*	
				$::=$	$\{x_i : \mu_i\}_1^m \mid \{x_i : \mu_i\}_1^k \rho$	

(TVAR) $$\frac{\Delta_n(x) \succ A}{\Sigma; \Delta_0 \cdots \Delta_n \vdash x : A}$$

(TABS) $$\frac{\Sigma; \Delta_0 \cdots \Delta_n + x : A \vdash e : B}{\Sigma; \Delta_0 \cdots \Delta_n \vdash \lambda x.e : A \to B}$$

(TAPP) $$\frac{\Sigma; \Delta_0 \cdots \Delta_n \vdash e_1 : A \to B \qquad \Sigma; \Delta_0 \cdots \Delta_n \vdash e_2 : A}{\Sigma; \Delta_0 \cdots \Delta_n \vdash e_1 e_2 : B}$$

다형 타입 시스템 λ^{poly}_{open} (3/3)

(TBOX)
$$\frac{\Sigma; \Delta_0 \cdots \Delta_n \Gamma \vdash e : A}{\Sigma; \Delta_0 \cdots \Delta_n \vdash \text{`}e : \Box(\Gamma \triangleright A)}$$

(TUNBOX)
$$\frac{\Sigma; \Delta_0 \cdots \Delta_n \vdash e : \Box(\Gamma \triangleright A) \quad \Delta_{n+1} \succ \Gamma \quad k > 0}{\Sigma; \Delta_0 \cdots \Delta_{n+1} \vdash \text{,}e : A}$$

(TEVAL)
$$\frac{\Sigma; \Delta_0 \cdots \Delta_n \vdash e : \Box(\varnothing \triangleright A)}{\Sigma; \Delta_0 \cdots \Delta_n \vdash \texttt{run}\, e : A}$$

(TLETAPP)
$$\frac{\begin{array}{l}\neg \,\text{expansive}^n(e_1) \\ \Sigma; \Delta_0 \cdots \Delta_n \vdash e_1 : A \\ \Sigma; \Delta_0 \cdots \Delta_n + x : \mathit{GEN}_A(\Sigma, \Delta_0 \cdots \Delta_n) \vdash e_2 : B\end{array}}{\Sigma; \Delta_0 \cdots \Delta_n \vdash \texttt{let}\ (x\ e_1)\ e_2 : B}$$

타입 시스템의 안전성

Lemma (Preservation)

Let $\models \mathcal{S} : \Sigma$, $\Sigma \models \mathcal{E} : \Delta_0$. *If*

$$\Sigma; \Delta_0 \cdots \Delta_n \vdash e : A, \quad \textit{and} \quad \mathcal{E}, \mathcal{S}, \mathcal{V} \vdash e \xrightarrow{n} (r, \mathcal{S}', \mathcal{V}'),$$

then,

$$\Sigma'; \varnothing \Delta_1 \cdots \Delta_n \vdash r : A \quad \textit{and} \quad \models \mathcal{S}' : \Sigma' \textit{ for some } \Sigma' \supseteq \Sigma.$$

타입 시스템의 알고리즘

Lemma (Soundness)

If $\texttt{infer}(\Delta_0 \cdots \Delta_n, e, A, \mathcal{Q})$ *succeeds with* S*, then*

$$\varnothing; (S\Delta_0) \cdots (S\Delta_n) \vdash e : SA.$$

Lemma (Completeness)

If $\varnothing; (R\Delta_0) \cdots (R\Delta_n) \vdash e : RA$ *then* $\texttt{infer}(\Delta_0 \cdots \Delta_n, e, A, \mathcal{Q})$ *succeeds with* S *such that* $R|_{\mathcal{Q}}^{-} = TS|_{\mathcal{Q}}^{-}$ *for some* T *and* $\mathrm{RV}(S) \subseteq \mathcal{Q} \cup \mathrm{FV}(A) \cup \bigcup_{i=0}^{n} \mathrm{FV}(\Delta_i)$.

계획

계획

- λ^{poly}_{open} 타입 시스템을 nML에 장착: nML 2.0
  - "Staged nML" 혹은 "MetanML" :-)

계획

- λ^{poly}_{open} 타입 시스템을 nML에 장착: nML 2.0
  - "Staged nML" 혹은 "MetanML" :-)
- 프로그램의 분석 문제: 금광
  - cfa, exception analysis, secure flow analysis, $\cdots$

계획

- λ^{poly}_{open} 타입 시스템을 nML에 장착: nML 2.0
  - "Staged nML" 혹은 "MetanML" :-)
- 프로그램의 분석 문제: 금광
  - cfa, exception analysis, secure flow analysis, $\cdots$

감사합니다