SNU 4190.310 Programming Language, Fall 2004
Final Exam
12/14/2004, 14:30 - 16:00

Problem 1 [20pts] O/X questions (1pts/each)

- Any progam with dynamic scoping can be encoded by a program with static scoping.
- C's pointers, records, set-jumps/long-jumps, gotos, local blocks, and loops are syntactic sugars of eager-evaluating λ-calculus.
- We can design a programming language for which the garbage collection can be complete.
- Function call is expensive because we have to remember the context to switch after the call.
- 환경은 프로그램식에 나타난 이름의 실체를 결정한다.
- 프로그램 실행중에 잠깐 멈추어 서서, 지금까지는 사용되었으나 앞으로는 사용되지 않을 메모리를 족집게같이 찝어낼 방법은 경우에 따라서는 가능하다.
- 적극적인 계산법(eager evaluation)은 소극적인 계산법(lazy evaluation)보다 빨리 프로그램 결과를 계산한다.
- 모든 정수 n에 대해서 `fac` n과 `fac'(`n`,1)` 은 같은 값을 계산한다.
```
fun fac n =
 if n <= 0 then 1 else n * fac(n-1)
fun fac'(n, r) =
 if n <= 0 then r else fac'(n-1, n*r)
```
- C, Java, ML로 짤 수 있는 모든 재귀함수들은 모두 그 언어로 된 끝재귀호출(tail recursive) 함수로 변경할 수 있다.
- 따로따로 포장하기(modularity), 계층구조를 형성하기(hierarchy), 속내용을 신경쓰지않게하기(data abstraction) 등을 프로그래머가 쉽게 운용 할 수 있도록 프로그래밍 언어에서 도와줄 수 있다.
- 그 도움이란 프로그래머가 그러한 프로그래밍 기술들을 제대로 잘 적용하고 있는지를 컴파일러가 자동으로 엄밀하게 검증해주는 것을 말한다.
- 물건중심의 프로그래밍 언어(object-oriented programming language)에서 다른 방식의 언어와 다른 유일한 것은 inheritance이다.
- 아래 문법을 가지는 프로그램을 생각합시다

$$\begin{array}{rcl} E & \rightarrow & x \texttt{ := } E \mid E \texttt{ ; } E \mid \texttt{while } E \texttt{ do } E \\ & \mid & n \mid x \mid E \texttt{ + } E \mid \texttt{true} \mid \texttt{false} \mid E \texttt{ < } E \end{array}$$

  프로그램 `while (true < false) do x := 2+true` 는 제대로 생긴 프로그램이다
- 위의 프로그램은 말이 된다 (잘 돌아간다; 정의되는 의미가 있다).

- 위의 문법을 좀 고쳐서

$$\begin{array}{lcl} S & \rightarrow & x := E \mid S \ ; \ S \mid \texttt{while}\ E\ \texttt{do}\ S \\ E & \rightarrow & n \mid x \mid E + E \mid \texttt{true} \mid \texttt{false} \mid E < E \end{array}$$

  라고 했을 때, 제대로 생긴 프로그램은 말이되는 프로그램이다.

- 위의 문법을 좀 고쳐서

$$\begin{array}{lcl} S & \rightarrow & x := E \mid S \ ; \ S \mid \texttt{while}\ B\ \texttt{do}\ S \\ E & \rightarrow & Z \mid B \\ Z & \rightarrow & n \mid Z + Z \mid x \\ B & \rightarrow & \texttt{true} \mid \texttt{false} \mid Z < Z \mid x \end{array}$$

  라고 했을 때, 제대로 생긴 프로그램은 말이되는 프로그램이다.

- 위의 문법을 좀 고쳐서

$$\begin{array}{lcl} S & \rightarrow & x := E \mid S \ ; \ S \mid \texttt{while}\ B\ \texttt{do}\ S \\ E & \rightarrow & Z \mid B \\ Z & \rightarrow & n \mid Z + Z \mid x \\ B & \rightarrow & \texttt{true} \mid \texttt{false} \mid Z < Z \end{array}$$

  라고 했을 때, 제대로 생긴 프로그램은 말이되는 프로그램이다.

- 소프트웨어를 품고있는 상품(자동차, 휴대폰, 의료기기, 가전제품, 인공위성, 무기 등등)의 질은 그 소프트웨어의 품질(버그 없음, 생각대로 실행됨)에 결정적으로 의존할 것이다.

- 다음과 같은 두개의 nML 함수는 같은 일을 한다 (즉, `map` 이 사용된 곳에 `imperative_map`을 쓸 수 있고 그 반대도 성립한다.)

```
fun map f nil  = nil
  | map f (a::r) = (f a)::(map f r)



fun imperative_map f l =
  let
   val arg = ref l
   val result = ref nil
  in
   while not(!arg = nil) do
     result := (f (List.hd !arg)) :: !result;
     arg := List.tl !arg
   end;
   List.rev !result
  end
```

- 현재의 소프트웨어 문제점을 없애는 데 도움이 되는 프로그래밍 언어나 프로그래밍 시스템은 꾸준히 연구되고 만들어 질 것이다.

- 프로그래밍 언어 디자인 및 구현에 대한 좋은 연구 결과들은 당장 많이 쓰이는 프로그래밍 언어들 속에 재빨리 반영되지는 못하지만, 시간차를 두면서 다음세대의 프로그래밍 언어속에 녹아들고 있다.

Problem 2 [10pts]

λ식을 계산하는 다음의 기계 (e, S, E) 를 생각하자 (Krivine Machine). 기계의 각 부품들은

$$\begin{array}{lcll} e & \in & \lambda\text{식} & \\ S & \in & \lambda\text{식}^* & \text{stack} \\ E & \in & \text{Id} \xrightarrow{fin} \lambda\text{식} & \text{environment} \end{array}$$

이고, 기계의 동작

$$(e, S, E) \rightarrow (e', S', E')$$

은 다음과 같이 정의된다:

$$\begin{array}{rcl} (x,\ S,\ E) & \rightarrow & (E(x),\ S,\ E) \\ (\lambda x.e,\ e' \cdot S,\ E) & \rightarrow & (e,\ S,\ E[e'/x]) \\ (e_1\ e_2,\ S,\ E) & \rightarrow & (e_1,\ e_2 \cdot S,\ E) \end{array}$$

위의 기계는 λ 계산법에서 두개의 계산(reduction)방식중 무슨 방식을 따르는가? 그 이유를 설명하라.

Problem 3 [10pts] 적극적으로 계산하는(eager-order evaluation) λ 계산법에서 true, false, if-then-else 를 정의하라.

Problem 4 [15pts] 다음의 손바닥 계산기에서 제공하는 언어 E를 생각하자:

E	$\rightarrow$	n	number
	$\mid$	x	variable
	$\mid$	$E + E$	addition
	$\mid$	$E - E$	subtraction
	$\mid$	`let` x `=` E `in` E	binding

이 언어의 프로그램을 다음의 $\langle S, E, C\rangle$-machine으로 돌리려고 한다. S는 값들의 스택 (ordered sequence), E는 변수들의 값을 보관하는 함수, 명령어 C는 다음으로 구성된다:

C	$\rightarrow$	add
	$\mid$	sub
	$\mid$	bind(x)
	$\mid$	unbind(x)
	$\mid$	push(x)
	$\mid$	push(n)
	$\mid$	$C.C$

E 언어와 C 명령어의 의미구조(semantics)들을 정의하고, E 프로그램을 C 명령어로 변환하는 컴파일러 함수를 정의하라.

Problem 5 [5pts/-2pts] O/X

다음과 같은 언어를 생각하자. 적극적인 계산법(eager evaluation)을 갖추고 있고, 정수/함수/리스트를 다루는 방법을 제공하는 언어이다.

e	::=	n	integer
	$\mid$	`nil`	empty list
	$\mid$	`cons` e e	list construction
	$\mid$	`car` e	first element in list
	$\mid$	`cdr` e	rest element in list
	$\mid$	`ifnil` e e e	if-nil branch
	$\mid$	x	variable
	$\mid$	$\lambda x.e$	function
	$\mid$	e e	application

실행중에 타입에러를 만들어내는 프로그램의 예는 "$(\lambda x.1)$ (`car 1`)"가 있다.

Problem 6 [10pts/-5pts] O/X

관계식 $\Gamma \vdash e : \tau$ 를 결정하는 논리시스템을 아래와 같이 고안하였다.

$$\begin{array}{lcl} \tau \in \mathit{Type} & = & \mathit{int} \\ & \mid & \tau\ \mathit{list} \\ & \mid & \tau \rightarrow \tau \\ \Gamma \in \mathit{TypeEnv} & = & \mathit{Id} \xrightarrow{\text{fin}} \mathit{Type} \end{array}$$

$$\frac{}{\Gamma \vdash n : int} \qquad \frac{}{\Gamma \vdash nil : \tau\ list}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau\ list}{\Gamma \vdash \texttt{cons}\, e_1\, e_2 : \tau\ list}$$

$$\frac{\Gamma \vdash e : \tau\ list}{\Gamma \vdash \texttt{car}\, e : \tau} \qquad \frac{\Gamma \vdash e : \tau\ list}{\Gamma \vdash \texttt{cdr}\, e : \tau\ list}$$

$$\frac{\Gamma \vdash e_1 : \tau\ list \quad \Gamma \vdash e_2 : \tau' \quad \Gamma \vdash e_3 : \tau'}{\Gamma \vdash \texttt{ifnil}\, e_1\, e_2\, e_3 : \tau'}$$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$$

$$\frac{\Gamma + x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x.e : \tau_1 \to \tau_2}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\, e_2 : \tau_2}$$

임의의 프로그램 e에 대해서 $\emptyset \vdash e : \tau$가 증명가능하면, 프로그램 e는 실행중에 잘못되는 경우가 발생하지 않는다.

Problem 7 [15pts/-10pts] O/X

위의 언어로 짠 프로그램은 실행중에 `cons`문을 실행할 때만, 메모리 하나씩을 소모한다고 가정하자. 위에서 고안한 타입시스템을 확장해서 관계식 $\Gamma \vdash e : \tau \uparrow c$ 를 결정하는 논리시스템을 아래와 같이 고안하였다.

$$\frac{}{\Gamma \vdash n : int \uparrow 0} \qquad \frac{}{\Gamma \vdash nil : \tau\ list \uparrow 0}$$

$$\frac{\Gamma \vdash e_1 : \tau \uparrow c_1 \qquad \Gamma \vdash e_2 : \tau\ list \uparrow c_2}{\Gamma \vdash \texttt{cons}\, e_1\, e_2 : \tau\ list \uparrow c_1 + c_2 + 1}$$

$$\frac{\Gamma \vdash e : \tau\ list \uparrow c}{\Gamma \vdash \texttt{car}\, e : \tau \uparrow c} \quad \frac{\Gamma \vdash e : \tau\ list \uparrow c}{\Gamma \vdash \texttt{cdr}\, e : \tau\ list \uparrow c}$$

$$\frac{\Gamma \vdash e_1 : \tau\ list \uparrow c_1 \quad \Gamma \vdash e_2 : \tau' \uparrow c_2 \quad \Gamma \vdash e_3 : \tau' \uparrow c_3}{\Gamma \vdash \texttt{ifnil}\, e_1\, e_2\, e_3 : \tau' \uparrow c_1 + max(c_2, c_3)}$$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau \uparrow 0}$$

$$\frac{\Gamma + x : \tau_1 \vdash e : \tau_2 \uparrow c}{\Gamma \vdash \lambda x.e : \tau_1 \to \tau_2 \uparrow c}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \uparrow c_1 \quad \Gamma \vdash e_2 : \tau_1 \uparrow c_2}{\Gamma \vdash e_1\, e_2 : \tau_2 \uparrow c_1 + c_2}$$

임의의 프로그램 e에 대해서 $\emptyset \vdash e : \tau \uparrow c$ 가 증명 가능하면, 프로그램 e의 실행중 총 메모리 사용량은 c개 이하이다.

Problem 8 [15pts] (7.9 in Textbook)

Explain how K's simple type inference system (lecture note's) relies on static scope. Would the K's type inference system work properly if the language were changed to use dynamic scope?