Challenge Set II

SNU 4190.310, 2006 가을

이 광근

Due: 12/18 10:30

한 문제라도 올바른 답을 제출한 사람은 최종 성적이 상승합니다.

Challenge 1 (1 단계 상승)"메모리는 설탕"

메모리 반응식(imperative operations)들은 모두 설탕들인가?

아래는 적극적 계산법(eager evaluation)으로 실행되는 람다 언어이다. 메모리 명령문들의 실행은 익히 알고 있는 방식이다.

e	::=	x	variable
	|	$\lambda x.e$	function
	|	$e\,e$	application
	|	`ref` e	malloc
	|	e `:=` e	assignment
	|	`!`e	dereference

강의에서 이야기 한 바, 주어진 프로그램에 있는 메모리 명령문들을 나머지 다른 방식의 식들로 녹여서 같은 일을 하는 프로그램으로 바꿀 수 있다. 그 방안을 생각해 보자.

- 위와 같은 메모리 반응 식들은 메모리를 필요로한다.
- 그리고, 메모리에 반응하는 순서가 중요해 진다.

따라서 변환할 때는

- 프로그램식들의 실행순서를 존중하면서

- 각 식들이, 이전 순서 식의 결과값과 메모리를 받아서, 결과로 현재 식의 값과 결과 메모리를 리턴하도록 하면 된다.

그래서,

- 모든 식들이 함수가 된다: 이전식의 값과 메모리를 받아서 현재 식의 값과 반응이 일어난 메모리를 결과로 내놓는.

이제 남은 세 개의 질문:

1. 순서대로 계산되는 과정을 함수로 표현하는 방법은?
2. 메모리를 다른 식으로 어떻게 표현?
3. 그리고 나면, `ref` e, e `:=` e, `!`e는 어떻게 표현?

1답 : 위와같은 변환(할일의 순서를 드러내서 함수로 표현하는 변환)을 CPS 변환(continuation-passing-style transformation)이라고 한다.

2답 : 많은 방법이 있을 수 있다. 예를 들어, 짝으로:

$$\text{counter} \times \text{list of } (\text{주소} \times \text{값})$$

표현할 수 있을 것이다. counter는 현재 메모리에 할당된 메모리 갯수.

3답 : 프로그램 식 e가 변환된 것을 $\underline{e}$로 표현하면, 다음과 같이 메모리 설탕을 녹여낼 수 있을 것이다:

$$\underline{\texttt{ref}\ e} = \lambda(v,(c,S)).\texttt{let}\ (v,(c',S')) = \underline{e}(v,(c,S))\ \texttt{in}\ (c'+1, (c', v)::S')$$

나머지 경우도 위와 보조가 맞도록 프로그램 식 e의 메모리 설탕을 녹이는 룰을 찾아라. □

Challenge 2 (2 단계 상승)"처리안된 예외는 없슴"

다음과 같은 적극적인 계산법으로 실행되는 언어를 생각하자.

e	::=	x	variable
	|	$\lambda x.e$	function
	|	$e\ e$	application
	|	n	integer
	|	e `+` e	addition
	|	`if0` $e\ e\ e$	branch
	|	`raise`	exception raise
	|	e `handle` e	exception handling

"raise"는 예외상황을 발생시키는 명령이고, "e_1 handle e_2"는 e_1을 실행하다가 예외가 발생되면 e_2를 실행하게 된다. e_1의 실행중에 예외가 발생되지 않으면 e_1의 값이 전체 식의 결과값이 된다. "if0"식은 첫째 식의 값이 0이면 둘째 식을 계산하고 아니면 세째 식을 계산한다.

예를들어,

$$10 + 1 \texttt{ handle } 0$$

은 11을 계산한다.

$$((\lambda x.\texttt{if0}\ x\ \texttt{raise}\ 10)\ 0)\ \texttt{handle}\ 8$$

은 8을 계산한다.

$$((\lambda x.\texttt{if0}\ x\ (\texttt{raise handle}\ 1)\ 10)\ 0)\ \texttt{handle}\ 8$$

은 1을 계산한다.

$$(\lambda x.\texttt{if0}\ x\ 1\ raise)\ 1$$

은 발생한 예외를 처리하는 식이 없으므로, 실행이 값자기 멈춘다.

let-설탕을 써서 예를 더 들면,

```
let
  f = λ x. if0  x  raise  2
in
  ((f 0) handle λ x.x)
  (f 1)
end
```

은 2를 계산한다.

```
let
  k = (λ f. f  λ x.x)
in
  ((k λ x.x) handle λ x.x)
  (k λ x. raise)
end
```

는 발생한 예외를 처리하지 못하고 실행이 값자기 멈춘다.

위와 같은 언어에 대해서, 수업시간에 다룬 단순 타입 시스템(simple type system)을 확장하라. 그래서 확장된 타입 시스템을 통과한 프로그램은 실행중에 처리되지 않는 예외가 없다는 것이 보장되는. 그리고 그 타입 시스템을 안전하게 구현하는 알고리즘을 정의하라. 그래서 위에 예를 든 프로그램들에 대

해서, 제대로 도는 프로그램들 네개중 적어도 세개는 받아들일 수 있어야 하고, 제대로 돌지 않는 프로그램들은 모두 받아들이지 말아야 한다. □