4471028: 프로그래밍언어

Lecture 9 — 변경 가능한 상태
Mutable State

임현승
2020 봄학기

왜 필요할까?

- 프로그램이 실행되는 동안 함수 f가 몇 번 호출되는지 알고 싶다면?

```
let f = fun x -> x in
f (f 1)
```

- 다음과 같이 작성하면 될까?

```
let counter = 0 in
let f = fun x ->
            (let counter = counter + 1 in x)
in
let a = f (f 1) in
counter
```

- 프로그래밍 언어에서 side effect*를 지원해야 함
- Effects는 메모리(memory, store)와 주소(location, reference)를 이용하여 구현 가능

*side effect 부수 효과, 수반되는 효과, 메모리 반응 등을 의미

계산 효과(Computational Effect)

프로그래밍 언어에서 effects는 명시적(explicitly) 또는 암시적(implicitly)으로 지원됨

- ML처럼 명시적인 참조(explicit reference)를 지원하는 언어에서는 메모리 할당(allocation) 및 메모리에 저장된 값을 읽어오거나(dereference) 값을 변경(mutation)하기 위한 별도의 연산을 제공
- C나 Java처럼 암시적 참조(implicit reference)를 지원하는 언어에서는 이와 같은 연산들이 내장되어 있음

명시적 참조를 지원하는 언어 $\mathcal{L}_{exp}^{ref}$

$$
\begin{array}{rcl}
P & ::= & E \\
E & ::= & n \mid x \mid E + E \mid E - E \\
 & \mid & \texttt{iszero}\ E \mid \texttt{if}\ E\ \texttt{then}\ E\ \texttt{else}\ E \\
 & \mid & \texttt{let}\ x = E\ \texttt{in}\ E \\
 & \mid & \texttt{fun}\ x\ \texttt{->}\ E \mid E\ E \\
 & \mid & \texttt{ref}\ E \\
 & \mid & \texttt{!}\ E \\
 & \mid & E := E \\
 & \mid & E; E
\end{array}
$$

- **ref** E는 새로운 주소를 할당받아, 그 주소에 E를 계산한 값을 저장
- ! E는 E가 가리키는 주소에 저장되어 있는 값을 반환
- $E_1 := E_2$는 E_1이 가리키는 주소에 저장되어 있는 값을 E_2의 계산 결과로 변경

예제

- ```
  let counter = ref 0 in
  let f =
    fun x -> (counter := !counter + 1; !counter)
  in
  let a = f 0 in
  let b = f 0 in
  a - b
  ```

- ```
  let f =
    let counter = ref 0 in
    fun x -> (counter := !counter + 1; !counter)
  in
  let a = f 0 in
  let b = f 0 in
  a - b
  ```

예제

```
let f =
  fun x ->
    let counter = ref 0 in
    (counter := !counter + 1; !counter)
in
let a = f 0 in
let b = f 0 in
a - b
```

예제

- ```
  let f =
    fun x ->
      let counter = ref 0 in
      (counter := !counter + 1; !counter)
  in
  let a = f 0 in
  let b = f 0 in
  a - b
  ```

- A chain of references:

  ```
  let x = ref (ref 0) in
  (!x := 11; !(!x))
  ```
  ```

실행 의미구조

메모리는 주소(location)로부터 값(value)으로의 부분 함수(finite map)

$$\begin{aligned} \mathit{Val} &= \mathbb{Z} + \mathit{Bool} + \mathit{Closure} + \mathit{Loc} \\ \mathit{Closure} &= \mathit{Var} \times \mathit{Exp} \times \mathit{Env} \\ \rho \in \mathit{Env} &= \mathit{Var} \to \mathit{Val} \\ \sigma \in \mathit{Mem} &= \mathit{Loc} \to \mathit{Val} \end{aligned}$$

메모리 효과를 기술하기 위해 다음과 같은 형태의 판단문를 이용:

$$\rho, \sigma \vdash E \Rightarrow v, \sigma'$$

실행 환경이 ρ이고 메모리 상태가 σ일 때, 표현식 E는 값 v로 계산되고 메모리 상태는 σ'으로 업데이트된다.

실행 의미구조

기존 추론 규칙들은 메모리 σ를 추가하기만 하면 됨

$$\frac{}{\rho, \sigma \vdash n \Rightarrow n, \sigma} \qquad \frac{}{\rho, \sigma \vdash x \Rightarrow \rho(x), \sigma}$$

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow n_1, \sigma_1 \qquad \rho, \sigma_1 \vdash E_2 \Rightarrow n_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 + E_2 \Rightarrow n_1 + n_2, \sigma_2}$$

$$\frac{\rho, \sigma_0 \vdash E \Rightarrow 0, \sigma_1}{\rho, \sigma_0 \vdash \texttt{iszero}\ E \Rightarrow \mathit{true}, \sigma_1} \qquad \frac{\rho, \sigma_0 \vdash E \Rightarrow n, \sigma_1}{\rho, \sigma_0 \vdash \texttt{iszero}\ E \Rightarrow \mathit{false}, \sigma_1}\ (n \neq 0)$$

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow \mathit{true}, \sigma_1 \qquad \rho, \sigma_1 \vdash E_2 \Rightarrow v, \sigma_2}{\rho, \sigma_0 \vdash \texttt{if}\ E_1\ \texttt{then}\ E_2\ \texttt{else}\ E_3 \Rightarrow v, \sigma_2}$$

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow v_1, \sigma_1 \qquad [x \mapsto v_1]\rho, \sigma_1 \vdash E_2 \Rightarrow v, \sigma_2}{\rho, \sigma_0 \vdash \texttt{let}\ x = E_1\ \texttt{in}\ E_2 \Rightarrow v, \sigma_2}$$

$$\frac{}{\rho, \sigma \vdash \texttt{fun}\ x \texttt{->}\ E \Rightarrow (x, E, \rho), \sigma}$$

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow (x, E, \rho'), \sigma_1 \qquad \rho, \sigma_1 \vdash E_2 \Rightarrow v, \sigma_2 \qquad [x \mapsto v]\rho', \sigma_2 \vdash E \Rightarrow v', \sigma_3}{\rho, \sigma_0 \vdash E_1\ E_2 \Rightarrow v', \sigma_3}$$

실행 의미구조

새로운 언어 요소(construct)들을 위한 추론 규칙:

$$\frac{\rho, \sigma_0 \vdash E \Rightarrow v, \sigma_1}{\rho, \sigma_0 \vdash \texttt{ref}\ E \Rightarrow l, [l \mapsto v]\sigma_1}\ (l \notin \mathrm{Dom}(\sigma_1))$$

$$\frac{\rho, \sigma_0 \vdash E \Rightarrow l, \sigma_1}{\rho, \sigma_0 \vdash\ !\, E \Rightarrow \sigma_1(l), \sigma_1}$$

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow l, \sigma_1 \qquad \rho, \sigma_1 \vdash E_2 \Rightarrow v, \sigma_2}{\rho, \sigma_0 \vdash E_1 := E_2 \Rightarrow v, [l \mapsto v]\sigma_2}$$

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow v_1, \sigma_1 \qquad \rho, \sigma_1 \vdash E_2 \Rightarrow v_2, \sigma_2}{\rho, \sigma_0 \vdash E_1; E_2 \Rightarrow v_2, \sigma_2}$$

예제

$\rho, \sigma_0 \vdash$ `let x = ref (ref 0) in (!x := 11; !(!x))` $\Rightarrow$

암시적 참조를 지원하는 언어 $\mathcal{L}_{imp}^{ref}$

$$
\begin{array}{rcl}
P & ::= & E \\
E & ::= & n \mid x \mid E + E \mid E - E \\
 & \mid & \texttt{iszero}\ E \mid \texttt{if}\ E\ \texttt{then}\ E\ \texttt{else}\ E \\
 & \mid & \texttt{let}\ x = E\ \texttt{in}\ E \\
 & \mid & \texttt{fun}\ x\ \texttt{->}\ E \mid E\ E \\
 & \mid & \texttt{set}\ x = E \\
 & \mid & E; E
\end{array}
$$

- 모든 변수는 변경 가능(mutable, changeable)
- **set** $x = E$는 x가 가리키는 주소에 저장되어 있는 값을 E의 계산 결과로 변경
- 각 바인딩 연산이 실행될 때마다 주소가 생성됨: 함수 적용식과 **let**-표현식

예제

- ```
  let f =
    let count = 0 in
    fun x -> (set count = count + 1; count)
  in
  let a = f 0 in
  let b = f 0 in
  a - b
  ```

- ```
  let f =
    fun x -> fun y -> (set x = x + 1; x - y)
  in (f 44) 33
  ```

실행 의미구조

모든 변수는 주소(reference, location)를 의미:

$$
\begin{aligned}
\mathit{Val} &= \mathbb{Z} + \mathit{Bool} + \mathit{Closure} \\
\mathit{Closure} &= \mathit{Var} \times \mathit{Exp} \times \mathit{Env} \\
\rho \in \mathit{Env} &= \mathit{Var} \to \mathit{Loc} \\
\sigma \in \mathit{Mem} &= \mathit{Loc} \to \mathit{Val}
\end{aligned}
$$

실행 의미구조

$$\frac{}{\rho, \sigma \vdash n \Rightarrow n, \sigma} \qquad \frac{}{\rho, \sigma \vdash x \Rightarrow \sigma(\rho(x)), \sigma}$$

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow n_1, \sigma_1 \qquad \rho, \sigma_1 \vdash E_2 \Rightarrow n_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 + E_2 \Rightarrow n_1 + n_2, \sigma_2}$$

$$\frac{\rho, \sigma_0 \vdash E \Rightarrow 0, \sigma_1}{\rho, \sigma_0 \vdash \texttt{iszero}\ E \Rightarrow \mathit{true}, \sigma_1} \qquad \frac{\rho, \sigma_0 \vdash E_1 \Rightarrow \mathit{true}, \sigma_1 \qquad \rho, \sigma_1 \vdash E_2 \Rightarrow v, \sigma_2}{\rho, \sigma_0 \vdash \texttt{if}\ E_1\ \texttt{then}\ E_2\ \texttt{else}\ E_3 \Rightarrow v, \sigma_2}$$

$$\frac{}{\rho, \sigma \vdash \texttt{fun}\ x\ \texttt{->}\ E \Rightarrow (x, E, \rho), \sigma} \qquad \frac{\rho, \sigma_0 \vdash E \Rightarrow v, \sigma_1}{\rho, \sigma_0 \vdash \texttt{set}\ x = E \Rightarrow v, [\rho(x) \mapsto v]\sigma_1}$$

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow v_1, \sigma_1 \qquad [x \mapsto l]\rho, [l \mapsto v_1]\sigma_1 \vdash E_2 \Rightarrow v, \sigma_2}{\rho, \sigma_0 \vdash \texttt{let}\ x = E_1\ \texttt{in}\ E_2 \Rightarrow v, \sigma_2}\ (l \notin \mathrm{Dom}(\sigma_1))$$

$$\frac{\begin{array}{c}\rho, \sigma_0 \vdash E_1 \Rightarrow (x, E, \rho'), \sigma_1 \qquad \rho, \sigma_1 \vdash E_2 \Rightarrow v, \sigma_2 \\ [x \mapsto l]\rho', [l \mapsto v]\sigma_2 \vdash E \Rightarrow v', \sigma_3\end{array}}{\rho, \sigma_0 \vdash E_1\ E_2 \Rightarrow v', \sigma_3}\ (l \notin \mathrm{Dom}(\sigma_2))$$

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow v_1, \sigma_1 \qquad \rho, \sigma_1 \vdash E_2 \Rightarrow v_2, \sigma_2}{\rho, \sigma_0 \vdash E_1; E_2 \Rightarrow v_2, \sigma_2}$$

예제

```
let f =
  let count = 0 in
  fun x -> (set count = count + 1; count)
in
let a = f 0 in
let b = f 0 in
a - b
```

값에 의한 호출(Call-By-Value)

아래 프로그램을 계산한 결과 값은?

```
let p = fun x -> (set x = 4) in
let a = 3 in
(p a; a)
```

함수 적용식 계산 규칙:

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow (x, E, \rho'), \sigma_1 \qquad \rho, \sigma_1 \vdash E_2 \Rightarrow v, \sigma_2 \qquad [x \mapsto l]\rho', [l \mapsto v]\sigma_2 \vdash E \Rightarrow v', \sigma_3}{\rho, \sigma_0 \vdash E_1\ E_2 \Rightarrow v', \sigma_3} \ (l \notin \mathrm{Dom}(\sigma_2))$$

값에 의한 호출 인자 전달 방식:

- 형식 인자(formal parameter)는 새로운 메모리 주소를 할당받고, 할당받은 주소에 실질 인자(actual argument)의 값을 저장
- 가장 많이 이용되는 인자 전달 방식

참조에 의한 호출(Call-By-Reference)

함수 호출시 실질 인자로 주어진 변수에 저장되어 있는 값이 아닌, 변수의 주소 자체를 넘김

- 문법을 확장하고:

$$\begin{array}{rcl} E & ::= & \ldots \\ & | & E\ E \\ & | & E\ \langle y \rangle \end{array}$$

- 계산 규칙을 정의:

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow (x, E, \rho'), \sigma_1 \qquad [x \mapsto \rho(y)]\rho', \sigma_1 \vdash E \Rightarrow v', \sigma_2}{\rho, \sigma_0 \vdash E_1\ \langle y \rangle \Rightarrow v', \sigma_2}$$

예제

- ```
  let p = fun x -> (set x = 4) in
  let a = 3 in
  (p <a>; a)
  ```

- ```
  let f = fun x -> (set x = 44) in
  let g = fun y -> f <y> in
  let z = 55 in
  (g <z>; z)
  ```

- ```
  let swap =
    fun x -> fun y -> let temp = x in
                      (set x = y; set y = temp)
  in
  let a = 33 in
  let b = 44 in
  ((swap <a>) <b>; a - b)
  ```
  ```

별칭, 별명(Variable Aliasing)

두 개 이상의 참조에 의해 호출된 형식 인자가 같은 주소를 가리킬 때

```
let b = 3 in
let p = fun x -> fun y -> (set x = 4; y) in
(p <b>) <b>
```

- 변수의 별칭(variable aliasing)이 만들어짐: `x`와 `y`가 같은 주소를 가리킨다.
- 별칭이 있을 경우, 변수 하나의 값만 변경하더라도 다른 변수의 값도 변경될 수 있기 때문에, 프로그램이 어떻게 동작하는지 추론하기가 어려워진다.

적극적인 vs 느긋한 계산법

```
let rec infinite-loop (x) = infinite-loop (x) in
let f = fun x -> 1 in
f (infinite-loop 0)
```

- 적극적인 계산 방식(eager evaluation)에서는 함수의 실질 인자가 함수에 전달되기 전에 먼저 완전히 값으로 계산함
- 느긋한 계산 방식(lazy evaluation)에서는 함수의 몸체에서 실질 인자의 값을 필요로 할 때까지 인자의 계산을 뒤로 미룸

요약

지금까지 살펴 본 언어 요소들:

$$\mathcal{L}^{let} : \text{간단한 표현식 언어}$$
$$\downarrow$$
$$\mathcal{L}^{fun} = \mathcal{L}^{let} + \text{고차 함수}$$
$$\downarrow$$
$$\mathcal{L}^{rec} = \mathcal{L}^{fun} + \text{재귀 함수}$$
$$\downarrow$$
$$\mathcal{L}^{ref} = \mathcal{L}^{fun} + \text{변경 가능한 상태}$$
$$\downarrow$$
$$\mathcal{L}^{typed} = \mathcal{L}^{fun} + \text{타입 검사}$$

- Effects는 명시적 또는 암시적으로 지원 가능
- 인자 전달 방식: 값에 의한 호출, 참조에 의한 호출