2020 봄학기 프로그래밍언어
프로그래밍 과제 #5 (100점)

지수환 93suhwan@gmail.com
임현승 hsim@kangwon.ac.kr

제출 마감: 6월 17일 수요일 11:59pm

1 개요

- 숙제를 진행하기에 앞서 본 문서를 꼼꼼히 읽어보시길 바랍니다.
- 숙제에 대해서 친구들과 토론하는 것은 괜찮지만 숙제는 반드시 혼자서 작성하시기 바랍니다. **특히 코딩하는 중에 토론을 병행하거나 숙제를 완성한 이후에 다른 학생들에게 도움을 줄 경우 자칫하면 서로 매우 유사한 답안이 도출되어 표절로 판단될 수 있으니 주의하시기 바랍니다. 이 경우 설사 친구에게 소스코드를 보여주지 않았더라도 표절로 판단합니다.** 또한 다른 친구의 소스코드를 직접 보고 고쳐주는 경우도 부정행위에 해당합니다. **숙제가 표절로 판단될 경우 정보를 제공한 사람(copyee)과 정보를 도용한 사람(copier) 모두 0점 처리합니다.** 여러분이 제출한 숙제는 프로그램 표절 검사기(clone-checker)를 이용해서 표절 여부를 검사합니다.
- 숙제가 어렵거나 이해가 안가는 부분이 있다면 가급적 과목 웹페이지 질의응답 게시판을 활용하세요. TA나 제가 최대한 친절히 도와드리겠습니다. 단, 정답은 가르쳐 드리지 않습니다.

숙제를 진행하기 위해 먼저 과목 웹페이지(`http://eruri.kangwon.ac.kr`) 과제 게시판에서 `hw5.zip` 파일을 다운받아 압축을 풉니다. 다운받은 파일 중에서 반드시 `hw5.ml` 파일만 수정하세요. `hw5.ml` 파일에는 총 4개의 문제가 있으며, 각 문제에 대한 설명은 다음 섹션 참고하기 바랍니다.

2 문제

2.1 람다 계산식 [20점]

`lambda.ml` 파일에는 다음과 같은 표현식 언어가 정의되어 있다.

```
type exp =
  Var of var
| Lambda of var * exp
| App of exp * exp
and var = string
```

위 정의는 아래 *람다 계산식*(λ-calculus)의 추상 문법 구조를 구현한 것이다.

$$e \quad ::= \quad x \mid \lambda x.e \mid e\ e$$

수업 시간에 배운 이름 없는 함수 `fun x -> e`는 $\lambda x.e$의 다른 표현이다. 람다 계산식은 굉장히 간단해 보이지만 튜링 완전한(Turing-complete) 언어이며 모든 OCaml 프로그램은 의미가 동일한 람다 계산식 프로그램으로 변환될 수 있다. 람다 계산식에서 프로그램은 변수(variable)거나, 함수(function or lambda abstraction), 또는 함수 적용식(function application)이다.

이번 숙제의 첫 번째 문제는 `hw5.ml` 파일에 있는 `Problem1` 모듈에 정의된 `check` 함수를 완성하는 것이다.

```
module Problem1 = struct
  open Lambda

  let rec check : exp -> bool =
    fun _ -> raise NotImplemented
end
```

`check` 함수는 람다 표현식이 닫혀있는지(closed) 검사한다. 즉, 인자로 주어진 표현식에 자유 변수(free variable)가 없으면 `true`를, 있으면 `false`를 반환한다. 다음은 닫혀있는 표현식의 예이다.

- `Lambda ("a", Var "a")`
- `Lambda ("a", Lambda ("a", Var "a"))`
- `Lambda ("a", Lambda ("b", App (Var "a", Var "b")))`
- `Lambda ("a", App (Var "a", Lambda ("b", Var "a")))`

다음은 닫혀있지 않은(unclosed) 표현식의 예이다.

- `Lambda ("a", Var "b")`
- `Lambda ("a", App (Var "a", Lambda ("b", Var "c")))`
- `Lambda ("a", Lambda ("b", App (Var "a", Var "c")))`

2.2 KML 해석기 [30점]

다음은 수업 시간에 배운 $\mathcal{L}^{rec}$ 언어를 확장한 KML(Kangwon ML)의 추상 문법 구조이다.

P	::=	E	
E	::=	n	정수
	|	x	변수
	|	$E + E$	덧셈식
	|	$E - E$	뺄셈식
	|	$E * E$	곱셈식
	|	E/E	정수 나눗셈, 예: $5/2 = 2$
	|	$-E$	단항 마이너스, 예: -1, $-(1+3)$
	|	$E == E$	두 정수식 값이 같으면 true, 그렇지 않으면 false
	|	$E <= E$	왼쪽 식 값이 오른쪽 식 값보다 작거나 같으면 true, 그렇지 않으면 false
	|	$E < E$	왼쪽 식 값이 오른쪽 식 값보다 작으면 true, 그렇지 않으면 false
	|	$E >= E$	왼쪽 식 값이 오른쪽 식 값보다 크거나 같으면 true, 그렇지 않으면 false
	|	$E > E$	왼쪽 식 값이 오른쪽 식 값보다 크면 true, 그렇지 않으면 false
	|	`not` E	논리 부정 NOT
	|	E `||` E	논리합 OR
	|	E `&&` E	논리곱 AND
	|	`iszero` E	0 테스트
	|	`if` E `then` E `else` E	조건식
	|	`let` $x = E$ `in` E	지역 변수 선언식
	|	`let rec` $f\ x = E$ `in` E	자기 호출 함수 선언식
	|	`fun` x `->` E	이름 없는 함수
	|	$E\ E$	함수 적용식, 함수 호출

연산자 우선 순위와 결합 규칙은 OCaml에 정의된 것과 동일하다.

kml.ml 파일에는 다음과 같이 KML의 추상 문법 구조가 OCaml 데이터 타입으로 정의되어 있다. (위 문법에 정의된 순서대로 생성자가 정의되어 있음.)

```
type program = exp
and exp =
  NUM of int
| VAR of var
| ADD of exp * exp
| SUB of exp * exp
| MUL of exp * exp
| DIV of exp * exp
| UMINUS of exp
| EQ of exp * exp
| LE of exp * exp
| LT of exp * exp
| GE of exp * exp
| GT of exp * exp
| NOT of exp
| OR of exp * exp
| AND of exp * exp
| ISZERO of exp
| IF of exp * exp * exp
| LET of var * exp * exp
| LETREC of var * var * exp * exp
| FUN of var * exp
| APP of exp * exp
and var = string
```

본 문제에서는 Problem2 모듈에 정의된 run 함수를 완성해야 한다.

```
module Problem2 = struct
  open Kml
  exception Error

  let rec run : program -> value =
    fun _ -> raise NotImplemented
end
```

run 함수는 Kml.program 타입의 프로그램을 입력으로 받아 이를 실행하고, 그 결과 값을 Kml.value 타입의 값으로 반환한다. 강의 슬라이드에 정의되어 있는 정적 유효 범위(static scoping)를 사용하는 실행 의미 구조(operational semantics)를 참고하여 run 함수를 완성하라. 새롭게 추가된 비교식, 논리식 등에 대한 실행 의미 구조는 다른 표현식의 의미 구조를 참고하여 구현하라.

아래 코드는 KML로 구현한 double 함수와 이에 대응되는 OCaml AST이다.

```
let rec double x =
  if iszero x then 0 else (double (x - 1)) + 2
in double 6

LETREC ("double", "x",
  IF (ISZERO (VAR "x"),
      CONST 0,
      ADD (APP (VAR "double", SUB (VAR "x", CONST 1)), CONST 2)),
  APP (VAR "double", CONST 6))
```

위 OCaml 값을 run 함수의 인자로 넘겨서 실행하면 Int 12 값으로 계산되어야 한다.

2.3 이름 없는 KML 프로그램으로 번역하기 [25점]

`nameless.ml` 파일에는 자기 호출 함수 선언문(recursive function declaration)을 제외한 KML의 추상 문법 구조가 다음과 같이 OCaml 데이터 타입으로 정의되어 있다.

```
type program = exp
and exp =
  NUM of int
| VAR of var
| ADD of exp * exp
| SUB of exp * exp
| MUL of exp * exp
| DIV of exp * exp
| UMINUS of exp
| EQ of exp * exp
| LE of exp * exp
| LT of exp * exp
| GE of exp * exp
| GT of exp * exp
| NOT of exp
| OR of exp * exp
| AND of exp * exp
| ISZERO of exp
| IF of exp * exp * exp
| LET of var * exp * exp
| FUN of var * exp
| APP of exp * exp
and var = string
```

또한 이름 없는(nameless) KML 프로그램의 추상 문법 구조가 다음과 같이 정의되어 있다.

```
type nl_program = nl_exp
and nl_exp =
  NL_NUM of int
| NL_VAR of int
| NL_ADD of nl_exp * nl_exp
| NL_SUB of nl_exp * nl_exp
| NL_MUL of nl_exp * nl_exp
| NL_DIV of nl_exp * nl_exp
| NL_UMINUS of nl_exp
| NL_EQ of nl_exp * nl_exp
| NL_LE of nl_exp * nl_exp
| NL_LT of nl_exp * nl_exp
| NL_GE of nl_exp * nl_exp
| NL_GT of nl_exp * nl_exp
| NL_NOT of nl_exp
| NL_OR of nl_exp * nl_exp
| NL_AND of nl_exp * nl_exp
| NL_ISZERO of nl_exp
| NL_IF of nl_exp * nl_exp * nl_exp
| NL_LET of nl_exp * nl_exp
| NL_FUN of nl_exp
| NL_APP of nl_exp * nl_exp
```

본 문제에서는 `Problem3` 모듈에 정의된 `translate` 함수를 완성해야 한다.

```
module Problem3 = struct
  open Nameless
  exception CannotTranslate

  let rec translate : program -> nl_program =
    fun _ -> raise NotImplemented
end
```

translate 함수는 Nameless.program 타입의 KML 프로그램을 인자로 받아 Nameless.nl_program 타입의 이름 없는 KML 프로그램을 반환한다. 수업 시간에 배운 내용을 바탕으로 translate 함수를 완성하라. 예를 들어, 다음 프로그램은

```
LET ("x", CONST 37,
     FUN ("y", LET ("z", SUB (VAR "y", VAR "x"),
                    SUB (VAR "x", VAR "y"))))
```

아래 이름 없는 프로그램으로 변환된다:

```
NL_LET (NL_CONST 37,
        NL_FUN (NL_LET (NL_SUB (NL_VAR 0, NL_VAR 1),
                        NL_SUB (NL_VAR 2, NL_VAR 1))))
```

2.4 이름 없는 KML 해석기 [25점]

마지막 문제는 Problem4 모듈에 정의된 nl_run 함수를 완성하는 것이다.

```
module Problem4 = struct
  open Nameless
  exception Error
  let rec nl_run : nl_program -> nl_value =
    fun _ -> raise NotImplemented
end
```

nl_run 함수는 Nameless.nl_program 타입의 프로그램을 입력으로 받아 이를 실행하고, 그 결과 값을 Nameless.nl_value 타입의 값으로 반환한다. 강의 슬라이드에 정의되어 있는 실행 의미 구조를 참고하여 nl_run 함수를 완성하라. 새롭게 추가된 비교식, 논리식 등에 대한 실행 의미 구조는 다른 표현식의 의미 구조를 참고하여 구현하라.

본 섹션까지의 숙제를 끝마치고 나면, (1) 임의의 KML 프로그램(자기 호출 함수가 없는)을 Problem2.run 함수로 실행해보고, (2) Problem3.translate을 이용하여 이름 없는 KML 프로그램으로 번역한 다음, (3) Problem4.nl_run을 이용하여 번역된 프로그램을 실행해볼 수 있다. 예를 들어, 아래 KML 프로그램

```
let pgm = LET ("x", CONST 1, VAR "x")
```

을 Problem2.run pgm으로 실행하면 그 결과는 Int 1이 되며, Problem4.nl_run (Problem3.translate pgm)으로 실행하면 그 결과는 NL_Int 1이 된다.

3 제출

숙제 작성을 마친 후 반드시 make를 실행하여 숙제 프로그램이 컴파일이 잘 되는지 확인하세요. 컴파일이 안 되는 코드를 제출하면 본 숙제는 0점입니다. 끝으로 완성한 hw5.ml 파일을 과제 게시판에 업로드하면 됩니다.