計算機科学実験レポート 3 タスク 10

1029-28-9483 勝田 峻太朗

2018 年 10 月 30 日

課題 10 (任意: バックエンドの移植)

ARM アセンブリコードではなく C 言語コードへ変換するコード生成器を作成し, 性能比較を行いなさい.

C 言語への変換の方針

代入文

```
<type_of_src> dst = src
```

ただし,<type_of_src> には,"closure"(クロージャ), "int"(数), "int*"(配列), ""(すでに定義されている) のいずれかが入る.

例として, ソースコード 1 の変換結果は ソースコード 2 のようななる.

```
let a = 3 in let b = a in 0;;
```

ソースコード 1　ML コードの例

```
int main(){

int var00 = 3;
int var10 = var00;
int var30 = 0;
printf("%d\n", var30);
return 0;
}
```

ソースコード 2　変換後の C 言語

label と goto 文

```
<labelname>:;
<some operation>
<some operation>
goto <labelname>;
```

のような形で,C 言語の goto 文を用いて実装する.

クロージャと関数呼び出し

クロージャは, 以下のような struct を用いて実装した.

```c
typedef struct{
    int (*f)(const int*, const int);
    int* vars;
    int length;

} closure;
```

ソースコード 3　C 言語でのクロージャの実装

f は, 第 1 要素目の関数ポインタ (クロージャの第 0 要素) を表し,$vars$ は, スコープ外変数の列 (クロージャの第 1 要素以降) を表す. $length$ は,$vars$ の長さを保持する.(ただし実装では用いられていない.)

また, 混乱を避けるため, クロージャの第 1 要素は $vars$ の第 0 要素ではなく, 第 1 要素に格納される.

例えば, ソースコード 4 は, ソースコード 5 のように変換される.

```
let a = 1 in let b = 2 in let rec f x = x + a + b in f 5;;
```

ソースコード 4　クロージャを含む ML コード

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct{
    int (*f)(const int*, const int);
    int* vars;
    int length;

} closure;

int _b__recf10(const int *param_0, const int param_1){

int var02 = param_0[2];
int var12 = param_1;
int var21 = param_0[1];
int var32 = var12 + var02;
int var51 = var32 + var21;
return var51;
}

int main(){

int var03 = 1;
int var13 = 2;
closure var22;
var22.f = _b__recf10;
var22.length = 3;
int params[3];
params[1] = var13;
params[2] = var03;
```

```
var22.vars = params;
closure var33 = var22;
int var40 = 5;
int (*var52)(const int*, const int);
var52 = var33.f;
int var70 = var52(var33.vars, var40);
printf("%d\n", var70);
return 0;
}
```

ソースコード 5　クロージャの変換例

C 言語の構造の定義

C 言語上で使用する命令を, *c_spec.ml* において, 定義した.

```
(* variable names *)
type id = string
type label = string
type imm = int
type binop = Syntax.binOp

type ty =
  | Int
  | Closure
  | Tuple
  | Defined

type op =
  | Var of id
  | Imm of imm

type exp =
  | Decl of ty * id * op (* int id = op *)
  | Exp of op (* operand *)
  | Bin of id * binop * exp * exp
  | If of op * exp
  | Write of id * int * op (* id[i] = op *)
  | Return of op (* return op *)
  | Print of op
  | Read of id * op * int (* id = op[int] *)
  | Label of label
  | Goto of label
  | Call of id * id * id * op (* id = id(id.vars, x) *)
  | DeclareTuple of id * int
  | SetTupleValue of id * int * op
  | DeclareClosure of id (* closure aru_closure; *)
```

```
32    | SetClosurePointer of id * label (* aru_closure.f = b__recf00 *)
33    | SetClosureLength of id * int (* aru_closure.length = 2 *)
34    | DeclareClosureParams of int (* int params[i]; *)
35    | StoreClosureParams of int * op (* params[i] = op *)
36    | SetClosureParams of id (* aru_closure.vars = params *)
37    | DeclarePointer of id
38    | AssignPointer of id * id (* id = id.f *)
39    | Exit
40
41  type funct = Funct of id * (id list) * (exp list)
```

ソースコード 6　c_spec.ml における C 言語の構造定義

VM コードから C 言語への変換

VM コードから `c_spec.ml` で示した C 言語の構造への変換は `c_of_decl` 関数を用いて行った.
ただし, `ref` 型の値は以下の内容を持つ.

var_assoc
: $Vm.id$ か id への変換を保持する.

closure_var
: 中身がクロージャである id 集合を保持する.

tuple_var
: 中身が (クロージャではなく)tuple である id 集合を保持する.

defined_var
: すでに定義された変数集合を保持する.

```
1   let c_of_decl (Vm.ProcDecl(lbl, local_var, instr_list)): funct =
2     (* helper definitions and functions *)
3     let var_assoc = ref (MyMap.empty: (V.id, id) MyMap.t) in
4     let closure_var = ref (MySet.empty: id MySet.t) in
5     let tuple_var = ref (MySet.empty: id MySet.t) in
6     let defined_var = ref (MySet.empty: id MySet.t) in
7     let id_is_closure (id: id) = MySet.member id !closure_var in
8     let id_is_tuple (id: id) = MySet.member id !tuple_var in
9     let id_is_defined (id:  id) = MySet.member id !defined_var in
10    let op_is_closure = function
11      | Var id -> id_is_closure id
12      | Imm _ -> false in
13    let op_is_tuple = function
14      | Var id -> id_is_tuple id
15      | Imm _ -> false in
16    let append_closure id = closure_var := MySet.insert id !closure_var in
17    let append_tuple id = tuple_var := MySet.insert id !tuple_var in
18    let append_defined id = defined_var := MySet.insert id !defined_var in
19    let convert_id id =
20      match MyMap.search id !var_assoc with
```

```
21      | Some id' -> id'
22      | None -> let id' = fresh_var (string_of_int id) in
23        var_assoc := MyMap.append id id' !var_assoc;
24        id' in
25    let id_of_op op =
26      match op with
27      | V.Local id -> convert_id id
28      | _ -> err "id_of_exp: unexpected input" in
29    let convert_op op =
30      match op with
31      | V.Param i -> if i = 0 then Var param0_name
32        else Var param1_name
33      | V.Local id -> Var (convert_id id)
34      | V.IntV i -> Imm i
35      | V.Proc l -> Var ("unexpected_" ^ l) in
36    (* end helper definition and functions *)
37    let rec c_of_instr instr =
38      match instr with
39      | V.Move(id, op) ->
40        let id' = convert_id id in
41        let op' = convert_op op in
42        let id_was_defined = id_is_defined id' in
43        let is_closure = op_is_closure op' in
44        let is_tuple = op_is_tuple op' in
45        if is_closure then append_closure id';
46        if is_tuple then append_tuple id';
47        append_defined id';
48        let ty = if id_was_defined then Defined
49          else (if is_closure then Closure else if is_tuple then Tuple else
                Int) in
50        [Decl(ty, convert_id id, convert_op op)]
51      | V.BinOp(id, binop, op1, op2) -> [Bin(convert_id id, binop,
            Exp(convert_op op1), Exp(convert_op op2))]
52      | V.Label l -> [C_spec.Label l]
53      | V.BranchIf(op, l) -> [If(convert_op op, Goto(l))]
54      | V.Goto l -> [Goto l]
55      | V.Call(dest, op, opl) ->
56        (match opl with
57         | closure:: x:: [] -> [Call(convert_id dest, id_of_op op, id_of_op
                closure, convert_op x)]
58         | _ -> err "unexpected function call")
59      | V.Return op -> if lbl = "_toplevel"
60        then [Print(convert_op op); Exit]
61        else [Return(convert_op op)]
62      | V.Malloc(id, opl) ->
```

```
63        (match opl with
64         | pointer:: vars ->
65           if is_vm_proc pointer then (* if tuple is closure *)
66             let closure_name = convert_id id in
67             append_closure closure_name;
68             append_defined closure_name;
69             let funct_name = label_of_proc pointer in
70             let var_len = List.length vars in
71             [DeclareClosure closure_name; SetClosurePointer(closure_name,
                   funct_name);
72              SetClosureLength (closure_name, var_len + 1);
                    DeclareClosureParams(var_len + 1)] @
73             (List.mapi (fun i op -> StoreClosureParams(i+1, convert_op op))
                   vars) @
74             [SetClosureParams closure_name;]
75           else (* if tuple is not closure (a normal tuple) *)
76             let len = List.length opl in
77             let id' = convert_id id in
78             append_tuple id';
79             [DeclareTuple(id', len)] @
80             List.mapi (fun i op -> SetTupleValue(id', i, convert_op op)) opl
81         | _ -> err "undefined")
82      | V.Read(id, op, i) ->
83        if op_is_closure (convert_op op) then
84          (if i = 0
85           then [DeclarePointer(convert_id id); AssignPointer(convert_id id,
                    id_of_op op)]
86           else [Read(convert_id id, convert_op op, i)])
87        else [Read(convert_id id, convert_op op, i)]
88      | V.BEGIN _ -> err "error"
89      | V.END _ -> err "error" in
90    let instrs = List.concat (List.map c_of_instr instr_list) in
91    if lbl = "_toplevel"
92    then Funct("main", [], instrs)
93    else Funct(lbl, [param0_name; param1_name], instrs)
94
95
96  let convert_c = List.map c_of_decl
```

ソースコード 7　backend.ml - C 言語への変換

コマンドラインオプション

-C　このオプションをつけると, 生成された C プログラムが gcc によってコンパイルされる. ./a.out を実行することにより, プログラムを実行できる.

-o filename　結果を<filename>に書き込む.

```
$ ./minimlc -C -o sigma.c
# loop v = (1, 0) in
if v.1 < 101 then
  recur (v.1 + 1, v.1 + v.2)
else
  v.2;;
compile c program in sigma.c => success
# ^C
$./a.out
5050
```

ソースコード 8　実行例