⟨*prog*⟩ ::= ⟨*funcs*⟩
⟨*funcs*⟩ ::= ⟨*func*⟩ | ⟨*func*⟩ ⟨*funcs*⟩
⟨*func*⟩ ::= ⟨*type*⟩ ⟨*func-name*⟩`(`⟨*deco*⟩`)` ⟨*func-stm*⟩
⟨*func-stm*⟩ ::= ⟨*deco*⟩ ⟨*stm*⟩
⟨*deco*⟩ ::= ε | ⟨*type*⟩ ⟨*var*⟩`;` | ⟨*type*⟩ ⟨*var*⟩`;` ⟨*deco*⟩
⟨*stm*⟩ ::= `;` | ⟨*exp-stm*⟩ | ⟨*if-stm*⟩ | ⟨*iter-stm*⟩ | ⟨*stms*⟩ | `return`⟨*exp*⟩`;`
⟨*stms*⟩ ::= ⟨*stm*⟩ | ⟨*stm*⟩ ⟨*stms*⟩
⟨*exp-stm*⟩ ::= ⟨*b-exp*⟩`;` | ⟨*var*⟩ ⟨*upd*⟩ ⟨*b-exp*⟩`;`
⟨*upd*⟩ ::= `=` | `+=` | `-=` | `*=` | `/=`
⟨*b-exp*⟩ ::= ⟨*exp*⟩ | ⟨*b-exp*⟩ ⟨*b-bin*⟩ ⟨*b-exp*⟩
⟨*b-bin*⟩ ::= `||` | `&&` | `<` | `<=` | `>` | `>=`
⟨*exp*⟩ ::= ⟨*cons*⟩ | ⟨*var*⟩ | ⟨*func-name*⟩`(`⟨*augs*⟩`)` | ⟨*exp*⟩ ⟨*bin*⟩ ⟨*exp*⟩
⟨*augs*⟩ ::= ε | ⟨*var*⟩ | ⟨*var*⟩`,` ⟨*augs*⟩
⟨*bin*⟩ ::= `+` | `-` | `*` | `/`
⟨*if-stm*⟩ ::= `if(`⟨*b-exp*⟩`)` ⟨*func-stm*⟩ | `if(`⟨*b-exp*⟩`)` ⟨*func-stm*⟩ `else` ⟨*func-stm*⟩
⟨*iter-stm*⟩ ::= `while(`⟨*b-exp*⟩`)` ⟨*func-stm*⟩
⟨*type*⟩ ::= `void` | `int`

1 はじめに

2 関連研究

- **TODO:** これまでに読んだ文献を，**cite** コマンドを用いて，すべて引用してください
- `https://topps.diku.dk/pirc/janus-playground/` の Janus から C++ への変換についても調べてください。

3 【準備の章】

用語の定義をすること

- 順実行，逆実行

4 C 言語の構文

TODO: BNF で C 言語の構文を記述すること.

```
T[[if (e)      = {                    R[[if (e)      = {
    s1                 char c = !!e;          s1                 char c;
  else                 if (c)               else                 RESTORE(c);
    s2]]                 T[[s1]]              s2]]                 if (c)
                       else                                        R[[s1]]
                         T[[s2]]                                   else
                       SAVE(c);                                    R[[s2]]
                     }                                         }
```

(c は fresh)

図 1: if(仮)

5 翻訳

if 文は，分岐の処理を行う構文である．if 文に記述されている条件が真であるとき，then 節が実行され，偽であるとき，else 節が実行される．else 節の記述は任意であり，else 節を記述せず，if 文の条件が偽であるとき，何も実行されない．if 文が実行されたとき，if 文の条件の真理値は記録されない．if 文の条件の真理値が記録されなければ，if 文を逆実行したとき，どちらの節を実行してよいか分からない．したがって，if 文を可逆化するためには，if 文の条件の真理値を記録する必要がある．

順実行を行うための変換では，if 文の条件の真理値を記録するための fresh な変数を用意している．更に，条件式を二重否定することで真理値へと変換し，fresh な変数をその真理値で初期化することで，if 文の条件の真理値を記録できる．また，fresh な変数には if 文の条件の真理値が記録されているので，if 文の条件の記述を fresh な変数へと書き換えている．逆実行を行うための変換では，fresh な変数に真理値を読み出し，変数を if 文の条件とすることで，順実行と同じ節を実行することができる．あとは，then 節と else 節に変換を再帰的に適用することで，if 文全体を変換できる．また，真理値は 1bit の情報であるため，fresh な変数を char 型で宣言することで，確保されるメモリを比較的小さいサイズに抑えている．

```
T⟦while (e) = {          R⟦while (e) = {
    s⟧                       s⟧
                               int c = 0;                  int c;
                               while (e){                  RESTORE(c);
                                 c++;                      while (c > 0){
                                 T⟦s⟧                        R⟦s⟧
                               }                             --c;
                               SAVE(c);                    }
                             }                           }
```

(c は fresh. 繰り返しは $2^{31}-1$ 以下)

図 2: while(仮)

while 文は，繰り返しの処理を行う構文である．while 文に記述されている条件が真である間，処理が繰り返し行われるが，処理を繰り返した回数は記録されない．繰り返した回数が記録されなければ，while 文を逆実行したとき，繰り返し処理の初期状態が分からない．したがって，while 文を可逆化するためには，繰り返しの回数を記録する必要がある．

順実行を行うための変換では，繰り返しの回数を記録するための fresh な変数を用意し，while 文の本体の最初で変数の値を 1 つ増やしている．この変換により，while 文の処理が行われた回数を記録することができる．逆実行を行うための変換では，fresh な変数に繰り返しの回数を読み出し，while 文の本体の最後で変数の値を 1 つ減らしている．更に，while 文の条件を fresh な変数 > 0 とすることで，繰り返し回数の分だけ繰り返し処理の逆実行が行われる．したがって，while 文の逆実行の終了時に，繰り返し処理の初期状態に戻ることができる．あとは，while 文の本体に変換を再帰的に適用することで，while 文全体を変換することができる．しかし，fresh な変数は int 型で宣言されるので，繰り返しの回数が $2^{31}-1$ 回を超えるとオーバーフローする．int 型のオーバーフロー時の動作は未定義であるため，そのときの変換後の動作は可逆性が保証されない．

6　むすび

$\mathcal{T}[\![s_1\ s_2]\!]$ = $\mathcal{T}[\![s_1]\!]\ \mathcal{T}[\![s_2]\!]$

$\mathcal{T}[\![x$ `=` e`;`$]\!]$ =
```
SAVE(x);
x = e;
```
$\mathcal{T}[\![e$`;`$]\!]$ = e`;` (式が副作用を持たない場合)
(式の副作用は高々 1 つ)

$\mathcal{T}[\![$`{`
```
  s1
  goto label;
  s2
  goto label;
    ...
  goto label;
  sn
label:
  s
}
```
$]\!]$ =
```
{
  char c;
  T[[s1]]
  c = 1; SAVE(c);
  goto label;
  T[[s2]]
  c = 2; SAVE(c);
  goto label;
    ...
  c = n; SAVE(c);
  T[[sn]]
  c = 0; SAVE(c);
label:
  T[[s]]
}
```
(c は fresh, かつ n ≤ 127)

$\mathcal{T}[\![$`{`
```
  T x1, x2, ..., xn;
  s
}
```
$]\!]$ =
```
{
  T x1, x2, ..., xn;
  T[[s]]
  SAVE(x1);
  SAVE(x2);
    ...
  SAVE(xn);
}
```

図 3: Translater

$\mathcal{T}[\![$`if (`e`)`
  s_1
`else`
  $s_2$$]\!]$ = `{`
  `char` c `= !!`e`;`
  `if (`c`)`
    $\mathcal{T}[\![s_1]\!]$
  `else`
    $\mathcal{T}[\![s_2]\!]$
  `SAVE(`c`);`
`}`

(c は fresh)

$\mathcal{T}[\![$`switch (`e`)`
`{`
  `case` v_1`:`
    s_1
  `case` v_2`:`
    s_2
      $\cdots$
  `case` v_n`:`
    s_n
  `default:`
    s_{n+1}
`}`$]\!]$ = `{`
  `char` c `= 0,` d `= 0;`
  `switch (`e`)`
  `{`
    `case` v_1`:` c `= 1;` d`++;`
      $\mathcal{T}[\![s_1]\!]$
    `case` v_2`:` c `= 2;` d`++;`
      $\mathcal{T}[\![s_2]\!]$
      $\cdots$
    `case` v_n`:` c `= n;` d`++;`
      $\mathcal{T}[\![s_n]\!]$
    `default:` c `= 0;` d`++;`
      $\mathcal{T}[\![s_{n+1}]\!]$
  `}`
  `SAVE(`c`);`
  `SAVE(`d`);`
`}`

(c, d は fresh, かつ n ≤ 127)

$\mathcal{T}[\![$`while (`e`)`
  $s$$]\!]$ = `{`
  `int` c `= 0;`
  `while (`e`)`
  `{`
    c`++;`
    $\mathcal{T}[\![s]\!]$
  `}`
  `SAVE(`c`)`
`}`

(c は fresh)

図 4: Translater