研究会

大元 武

2015/6/22

参考文献

1. Kohei Suenaga, Naoki Kobayashi: Fractional Ownerships for Safe Memory Deallocation. APLAS 2009: 128-143

2. Xavier Leroy: Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107-115, 2009.

言語

型検査は、CompCertの中間言語の一つである Clight を対象に行う。Clight は

- long long や long double などの拡張された数
- goto 構文
- unstructured switch
- 可変長引数の関数
- side-effect
- block-scoped variable

などを排除している。

Expression

Expressionは以下の様な定義である。

expr :: = const | x | temp | *x | &x | unop e |
e_1 binop e_2 | (type)e | e.f | e -> f

Expression

- *xは、ポインタの参照先から値をとってくる操作。
  *xが左辺の場合、書き込みを行っているので1の所有権が、
  右辺の場合、読み込みを行っているので0より大きい所有権
  が必要になる。

- &xは、変数のアドレスを取ってくる操作。
  なんらかの制約式が必要になると思われるが、今は未実装。

- キャストも未実装。

Statement

Statementは以下の様な定義である。

stmt :: = skip | $e_1 = e_2$ | temp = e | temp = f (e_1, e_2, …, e_n) |
f (e_1, e_2, …, e_n) | free(e) | temp = malloc(e) |
assert (e_1, e_2) | s_1 ; s_2 | if (e) then s_1 else s_2 |
switch (e) case 1: s_1 … | loop (s_1) s_2 | break | continue |
return e | label: s | goto l |

Assign, Set

$e_1 = e_2$ からは以下の制約式が生成される。

- 実行前の e1の外側の所有権が1である (書き込み可能)
- 実行前の e1の内側の所有権がemptyである
- 実行後の e1とe2の所有権の和が、実行前のe2の所有権と等しい

temp = e からは以下の制約式が生成される。

- 実行前の、tempの所有権がemptyである

関数呼び出し

f (e_1, e_2, …, e_n) からは以下の制約式が生成される。

- 実行前の e_1,e_2, …, e_nの型が仮引数の型と等しい
- 実行後の e_1,e_2, …, e_nの型が仮引数の型と等しい
- 関数の返り値の所有権がemptyである

temp = f (e_1, e_2, …, e_n) からは、以下の制約式が生成される。

- 実行前の e_1,e_2, …, e_nの型が仮引数の型と等しい
- 実行後の e_1,e_2, …, e_nの型が仮引数の型と等しい
- 実行前の tempの所有権がemptyである
- 実行後の tempの型が関数の返り値の型と等しい

free, malloc

free(e)からは以下の制約式が生成される。

- 実行前の e の外側の所有権が1である
- 実行前の e の内側の所有権がemptyである
- 実行後の e の所有権がemptyである

temp = malloc(e) からは以下の制約式が生成される。

- 実行前の temp の所有権がemptyである
- 実行後の temp の外側の所有権が1である
- 実行後の temp の内側の所有権がemptyである

assert

assert (e_1, e_2) からは以下の制約式が生成される。

- 実行前のe1, e2の所有権の和と実行後のe1,e2の所有権の和が等しい

if, switch, loop

if (e) then s_1 else s2 からは以下の制約式が生成される

- s_1を実行した後の環境と、s_2を実行した後の環境が等しい

switch (e) case 1: s_1 … も同様

loop (s_1) s2 からは以下の制約式が生成される。

- 実行前の環境と、s2の実行後の環境が等しい

break, continue

breakは、強制的にブロック (分岐や再帰) から抜け出す。
そのため、分岐や再帰の型付けをする際に、予め実行後の環境を生成しておき、それを引数として渡す。そして、分岐や再帰中に break が呼ばれたら、引数と渡しておいた環境を返すようにする。

continueは、再帰中にそれ以降の処理を中止し、次の再帰に入る。こちらも、再帰の型付けをする前に、再帰実行前の環境を引数として渡しておき、continueが呼ばれたら、それを返すようにする。

return

return e からは以下の制約式が生成される

- 実行前のeの所有権と、実行後のeの所有権と関数の返り値の所有権の和とが等しい

全体の流れ

1. 関数の引数、関数内で宣言されている変数にfreshな所有権を割り当てて、それを環境に追加。
2. 関数の本体について、制約式を生成する

ということを、それぞれの関数定義に適用し、生成された制約式をまとめてソルバーに投げる。
結果が、satならプログラムに型がつき、unsatなら型がつかない。

例

ToDo

1. 構造体
2. freeの自動挿入
3. 実験
4. …