6L-7

証明支援系を用いた Morrisの二分木走査アルゴリズムの実装の検証*

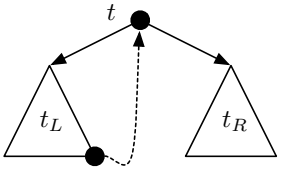
山田一宏　　渡部卓雄　　森口草介　　西崎真也
東京工業大学・大学院情報理工学研究科・計算工学専攻

概要　我々は，ポインタ操作を伴うCプログラムへの検証支援ツール適用のケーススタディとして，再帰やスタックを用いずに二分木走査を行うMorrisのアルゴリズムの実装の検証を試みた．今回我々が用いたツールは，C用の検証支援系Caduceus，自動証明系Simplifyおよび対話的証明支援系Coqである．本稿では，これらによる検証手法とその結果の概要を述べる．

1　Morrisのアルゴリズム

検証の対象は，Morrisによる二分木走査アルゴリズム[4]のCによる実装(プログラム1)である．このアルゴリズムは二分木を通りがけ順(in-order)で走査する．特徴として，再帰やスタックを必要としないこと，および節点の構造体内にポインタ変更を表すフラグ等の余分なフィールドが不要であることが挙げられる．また，ポインタの変更を行うために実行中は一時的に木の形が崩れるが，走査終了後には元の木が復元される．

二分木tについて，その根の左(右)の子を根とする部分木をtの左(右)部分木と呼び，t_L (t_R) と表記する．木tを通りがけ順で走査するためにはt_Lの走査後にtの根に戻る必要がある．そのために，本アルゴリズムではt_Lの最も右にある節点(t_Lを通りがけ順で走査したときに最後に訪れる節点)からtの根に戻るための辺(下図の点線)を一時的に作成する．



この辺は木を構成する本来の辺と区別するために**糸(thread)**と呼ばれる．糸が出る節点は右の子を持たないという性質があるので，本アルゴリズムでは節点を表す構造体(bintree_node)に糸のためのフィールドを追加せず，右の子のフィールド(tr)で代用している．そして，右の子への辺が本来の辺か糸かを区別するため，t_Lをその根から右方向に辿っていってtの根に到達すれば，tの根に入る辺は糸であるという性質を利用する(14〜16行めのループおよび21〜24行目)．

```
 1 typedef struct bintree_node {
 2   val_t val;
 3   struct bintree_node *tl, *tr;
 4 } *bintree;
 5 void visit_node(bintree node);
 6 void traverse(bintree t) {
 7   while (t != NULL) {
 8     if (t->tl == NULL) {
 9       visit_node(t);
10       t = t->tr;
11     }
12     else {
13       bintree rm = t->tl;
14       while (rm->tr != NULL &&
15              rm->tr != t)
16         rm = rm->tr;
17       if (rm->tr == NULL) {
18         rm->tr = t; // link thread
19         t = t->tl;
20       }
21       else { // rm has a thread to t
22         visit_node(t);
23         rm->tr = NULL; // unlink thread
24         t = t->tr;
25       }
26     }
27   }
28 }
```

プログラム1: Morrisのアルゴリズムの Cによる実装

2　検証方針の概要

プログラム1の部分正当性を，Hubertらによる Schorr-Waiteの二分木走査アルゴリズムの実装の検証[3]と同様の方針に従って検証した．まず検証すべき性質を事前・事後条件および不変条件として形式化し，ソースコード中にアノテーションとして記述する．ソースコードはCプログラム用の検証支援ツールCaduceusで処理される．CaduceusはWhyプラットフォーム[2]

*"On Verifying C Source Code using Proof Assistant Tools: A Case Study with Morris' Tree Traversal Algorithm", Kazuhiro YAMADA, Takuo WATANABE, Sosuke MORIGUCHI and Shin-ya NISHIZAKI, Department of Computer Sciece, Tokyo Institute of Technology

```
/*@ requires
  @ (\forall bintree x; x != \null &&
  @  reachable(t, x) => \valid(x)) &&
  @ (\forall bintree x; x != \null &&
  @  reachable(t, x) => !x->m && !x->th)
  ...
  @ ensures
  @ (\forall bintree x;
  @  \old(x->tr) == x->tr) &&
  @ (\forall bintree x: x != \null &&
  @  reacheable(\old(t), x) =>
  @  x->m && !x->th) &&
  ...
 */
void traverse(bintree t) { ... }
```

図 1: 事前・事後条件の一部

```
/*@ invariant
  @ (InvI_1 :: \forall bintree x;
  @  reachable_to_right(rm, x)
  @  => reachable_to_right(\old(rm), x))
  @ &&
  @ (InvI_2 :: rm != \null)
 */
while (rm->rt != NULL && rm->rt != t)
```

図 2: 内側ループの不変条件

に含まれるツールの一つであり[1]，アノテーション付のソースコードから Why への入力を生成する．Why はその結果を用いてバックエンドとなる自動証明系および証明支援系用の検証条件を生成する．今回我々は，生成された検証条件の証明に自動証明系 Simplify[2] および対話的証明支援系 Coq[1] を使用した．

実際の検証にあたり，節点の構造体 `bintree_node` に走査済みを示すフラグ `m` と，`tr` が糸であることを示すフラグ `th` を追加し，これらのフラグを適切に設定するよう関数 `traverse` を修正した．証明すべき性質は [3] と同様のものとした．具体的には，入力が正しい二分木であり，各節点のフラグが 0 で初期化されていることが事前条件であり，事後条件は，任意の節点についてその右の子は実行前と等しく，根から到達可能な全ての節点について `m` が 1 かつ `th` が 0 であること等である．

実際にソースコード中のアノテーションとして記述した事前・事後条件の一部を図 1 に，検証に利用したループ不変条件 (内側ループのみ) を図 2 に示す．実際に全ての事前・事後条件および不変条件を記述したソースコードは約 100 行になる．ここで用いられている `reachable` および `reachable_to_right` などの述語は，生成された検証条件を証明する際に証明支援系において定義する必要がある．

[1] 現在 Caduceus および Why は，C のための仕様記述・検証統合環境である Frama-C の一部となっている．

[2] Why プラットフォームに含まれる．

3 検証結果と考察

アノテーション付ソースコードから生成された検証条件は全部で 11 個であった．そのうち 2 個は自動証明系 Simplify によって証明できたので，残り 9 個を Coq で証明している．証明を行うにあたって，3 個の公理と 6 個の述語を定義した．

ポインタ操作を伴うプログラムを扱っているので，適切なメモリモデルを仮定する必要がある．ここでは [3] と同様に，component-as-array と呼ばれるモデルを採用した．これは構造体の配列を各フィールドを要素とする複数の配列で代用するもので，スカラ型を要素とする配列のみを扱えばよくなるため検証が容易になる．共用体や構造体内部でのポインタ演算は扱えないが，今回の目的には十分である．

Coq によって生成された検証条件を証明するにあたり，64 個の補題とその証明を行った．生成された検証条件とそれらの補題，およびその証明は全部で 1800 行ほどであり，証明には約 1ヶ月[3] を要している．現時点では外側のループ不変条件の維持に関する検証条件に一部未完のものがあるが，他は証明済みである．

実際の検証作業では，アノテーション (特にループ不変条件) を修正しつつ行うことが多く，修正後に再度検証条件を生成し，それらを証明支援系で証明するといった作業の繰り返しになる．一般に修正前の証明は再利用できないため，作業にかかる時間は大きくなる．また現時点では Morris のアルゴリズムにおける糸の性質を十分利用できていない．今後それらを用いることで，停止性を含めたより完全な検証結果を得られることが期待できる．

参考文献

[1] Coq. http://coq.inria.fr/.

[2] Why. http://why.lri.fr.

[3] T. Hubert and C. Marché. A case study of C source code verification: the Schorr-Waite algorithm. In *Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05)*, pp. 190–199. IEEE Computer Society, 2005.

[4] J. M. Morris. Traversing binary trees simply and cheaply. *Information Processing Letters*, 9(5):197–200, 1979.

[3] 第 1 著者が各種ツールを学習しながらの時間である．