時相論理式を用いた抽象化法のツール化に向けて*

田辺 良則† 高橋 孝一‡ 高井 利憲†

† 科学技術振興機構, CREST
‡ 産業技術総合研究所

{tanabe.yoshinori,k.takahashi,t-takai}@aist.go.jp

グラフ書換えに関する性質のモデル検査による検証を可能にする手法に, 時相論理を用いるものがある. この手法に述語抽象化法を組み合わせることによって検証ツールとするために必要な事項を検討する.

1 はじめに

近年, 述語抽象化法によるモデル検査は, ソースコード検証へ応用されており, 検証を行うツールも開発されている. しかし, これらのツールでは, ポインタを使用したデータ構造に関する性質を検証することは困難である.

一方, グラフ書換に関する性質の検証のために, 時相論理式 CTL*を使用した抽象化によるモデル検査の理論 [4] がある. この理論に, 述語抽象化 [3] を組み合わせることによって, ポインタを使用したデータ構造に関する性質をモデル検査法 [2] で検証するツールの作成を計画している. これを行うために必要な事項を, 以下のように検討する.

2 節では, 対象とするデータ構造およびその操作言語を定義し, 3 節では, 性質記述言語を定める. この操作言語によるソースコードに対して, 述語抽象化によるモデル検査法で性質を検証するツールの作成が目標となる. 述語抽象化法を適用させる方針を 4 節で述べ, 5 節でこれを例証する. 最後に, 6 節でツール化に向けての課題について述べる.

2 データ構造

次のようなデータ構造 (セル) を考察の対象とする. セルには一つの整数と, 一つの「ポインタ」が格納できる. 「ポインタ」の値は, 他のセルへの参照である. また, nil という名前の特別なセルが定められている. 任意の数の「ポインタ型の変数」が使用できる. `nil` という名前の特別な変数が用意されており, セル nil を参照している. 「アクセス可能な」セルは, 変数からポインタをたどって行けるものだけである. データ構造に対する操作は, 図 1 に示す BNF ふうに記述された言語で表現されるものとする. この言語の意味論の詳細は省略するが, `new()` は新規のセルの作成, `.next` はセルのポインタ, `.val` はセルの整数を表している. したがって, 以下のような操作が可能になっている.

- 新規のセルを作成して変数やセルのポインタに代入する.
- 変数に, セルへの参照を代入する.
- セルのポインタや整数に, 値を代入する.

3 性質記述

検証する性質は, 図 2 の`<pred>`として記述する.

意味論は次のように定める. セルからなるデータ構造は, ポインタを遷移関係と考えることで, Kripke 構造とみなすことができる. ただし, 原子論理式の集合は`<atomForm>`全体であって, ラベル付け関数は, セル x と整数 m, 変数 v に対して,

- $x \models m \iff x$ に格納されている整数が m である.
- $x \models v \iff v$ が x を指している.

となるように定める. `<tForm>`は, `<atomForm>`から作られる LTL 論理式であるが, **F** ,**G** , **U** 演算子のオペランドは, **X** 演算子がたかだか 1 つのものに制限されている. 変数 v と`<tForm>` f に対し, 「v`|=`f」が成り立つことを, v が指すセルを x として, $x \models f$ であることで定義する.

*本研究は, 科学技術振興機構戦略的想像研究推進事業 (CREST) 研究領域「情報社会を支える新しい高性能情報処理技術」研究課題「検証における記述量爆発問題の構造変換による解決」の一部として遂行された.

```
<identifiler> ::= (略)
<integer constant> ::= (略)
<var> ::= <identifier>
<cellLExp> ::= <var> | <cellLExp>.next
<cellExp> ::= new() | <cellLExp>
<intExp> ::= <integer constant>
             | <cellExp>.val
<intLExp> ::= <cellLExp>.val
<assign> ::= <cellLExp>:=<cellExp>
             | <intLExp>:=<intExp>
<basicPred> ::= <intExp>==<intExp>
             | <cellExp>==<cellExp>
<boolExp> ::= <basicPred>にブール演算を
              施して得られるもの
<statement> ::= <assign>
   | if(<boolExp>){<statement>*}
   | while(<boolExp>){<statement>*}
```

図 1: 操作記述言語

```
<atomForm> ::= <integer constant> | <var>
<tForm1> ::= <atomForm> | X <atomForm>
       | ! <tForm1> | <tForm1> && <tForm1>
<tForm> ::= <tForm1> U <tForm1>
       | F <tForm1> | G <tForm1>
       | ! <tForm> | <tForm> && <tForm>
<tPred> ::= <var> |= <tForm>
<pred> ::= <tPred>
       | ! <pred> | <pred> && <pred>
```

図 2: 性質記述

4 述語抽象化

述語抽象化に用いる述語も, <pred>で表現する.

抽象遷移は, 次の各代入文に関する遷移によって定まる.

1. `x := y;`
2. `x := new();`
3. `x := y.next;`
4. `x.val := c;`
5. `x.val := y.val;`
6. `x.next := y;`

ただし, x, y は変数, c は整数定数である.

SLAM[1] などの述語抽象化ツールでは, 実行文 A に対する遷移は, 次の原理で決定されている. 述語の集合を $\{P_1, \ldots, P_n\}$ とし, 抽象構造の点 $(b_1, \ldots, b_n)$ (各 b_i は true または false) をとる. $\varphi = \bigwedge_i Q_i$ (ただし, Q_i は, b_i が true のとき P_i, b_i が false のとき $\neg P_i$) とする. 述語 P_j に対する A についての最弱前条件を ψ_j とするとき, $B_j \subseteq \{\text{true}, \text{false}\}$ を次で定める.

```
1  // (1)
2  y := nil;
3  while (not (x == nil)) {
4    t := y;
5    y := x;
6    x := x.next;
7    y.next := t;
8  }
9  // (2)
```

図 3: サンプルコード

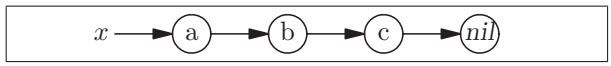

図 4: 入力データ構造

- $\text{true} \in B_j \iff (\psi_j \wedge \varphi)$ が充足可能
- $\text{false} \in B_j \iff (\neg\psi_j \wedge \varphi)$ が充足可能

このとき, 抽象構造において $(b_1, \ldots, b_n)$ から A によって遷移可能な点の全体は, $\prod_j B_j$ である.

我々の構造について, 抽象遷移に関して同じ定義を採用することは望ましくない. 代入文 4,5 のように, リンク構造が変化する場合には, 最弱前条件を代入前の構造で表現することができないため, 煩雑となるからである. 述語抽象化法の正当性は, 抽象遷移が (リレーションの集合として) 増大しても保たれる. そこで, 各述語 P と代入文 A に対して, その最弱前条件よりも強く, 表現しやすい条件 ψ_P を与えることにする. そうすれば,

- $\text{true} \in B_j \iff (\neg\psi_{\neg P} \wedge \varphi)$ が充足可能
- $\text{false} \in B_j \iff (\neg\psi_P \wedge \varphi)$ が充足可能

として, 抽象遷移を定義することができる.

例えば, 代入文 5 について, 述語 $P =$ "$z \models \mathbf{G}\,\varphi$" に関しては, ψ_P として, 次のものをとる: $(z \models \neg\mathbf{F}\,x \wedge z \models \mathbf{G}\,\varphi) \vee (z \models \mathbf{F}\,x \wedge z \models \mathbf{G}\,\varphi \wedge \langle x, y\rangle \models \varphi \wedge t \models \mathbf{G}\,\varphi)$. ここで, $\langle x, y\rangle \models \varphi$ は, x の次が y であるような構造の x において φ が成立することを意味するものとする.

5 サンプルコードによる例証

操作記述言語を使用したサンプルコードを図 3 に示す. このコード片は, (1) において x が図 4 のようなデータ構造を指していると想定して, 変数 x が指しているセル構造の「ポインタを逆向きに」して, その先頭を変数 y に格納するように記述したものである.

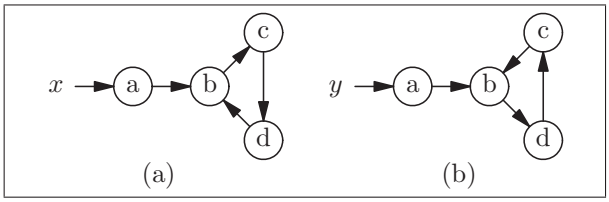

図 5: 反例となるデータ構造

そこで, 例えば φ を $\mathbf{G}\neg(1 \wedge \mathbf{X}2)$ (「その位置から先はずっと, 1 のセルの次には 2 のセルは現れない」), ψ を $\mathbf{G}\neg(2 \wedge \mathbf{X}1)$ (「その位置から先はずっと, 2 のセルの次には 1 のセルは現れない」) とした場合に, (1) において性質 x$\models \varphi$ を仮定すれば, (2) において性質 y$\models \psi$ が成り立つことが期待される.

次の述語を使用して, 検証を試みる: $v \models \varphi$, $v \models \psi$, $v \models 1$, $v \models 2$. ただし, v は x, y, t のすべてを使用する.

しかし, この検証は失敗する. 反例を検討すると, ループ不変式であると考えていた x$\models \varphi$ が, 7 行目の直後で不成立となることが原因であることがわかる. これは, y が x から到達可能であり得ることから生じる. 実際, x が図 5(a) に示すデータ構造を指していた場合にこのコードを実行すると, y が図 5(b) のデータ構造を指して停止する.

そこで, (1) における仮定として, $x \models \mathbf{F}\,\mathrm{nil}$ を追加する. これによって x$\models \varphi$, y$\models \psi$ の両方がループ不変式となり, 検証は成功する.

6 ツール化への課題

以上述べた内容に従ってツール化を実現するために, 次の課題を解決する必要がある.

(1) 述語に関する前条件の確定

4 節の最後に例示したように, 各代入文について, 述語と代入文の組み合わせに対して, 採用する前条件を決定する必要がある. 最弱前条件が常に使用できるわけではないので, 条件の強さと計算量とのバランスをとって決めることになる.

(2) 決定手続きのアルゴリズム

4 節に述べたように, 遷移関係の決定には, 述語の充足可能性を決定する手続きが必要であるので, この具体的なアルゴリズムを定める必要がある.

(3) モデル検査器の決定

抽象構造に対してはモデル検査を行うことになる. 既存のモデル検査器を利用するためのインタフェースを定める必要がある.

参考文献

[1] T. Ball, R. Majumdar, T. D. Millstein, and S. K. Rajamani. Automatic predicate abstraction of C programs. In *SIGPLAN Conference on Programming Language Design and Implementation*, pp. 203–213, 2001.

[2] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[3] S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In O. Grumberg ed., *Proc. 9th International Conference on Computer Aided Verification (CAV'97)*, pp. 72–83. Springer Verlag, 1997.

[4] K. Takahashi and M. Hagiya. Abstraction of graph transformation using temporal formulas. In *Supplemental Volume of the 2003 International Conference on Dependable Systems and Networks (DNS-2003)*, pp. W–65 to W–66, 2003.