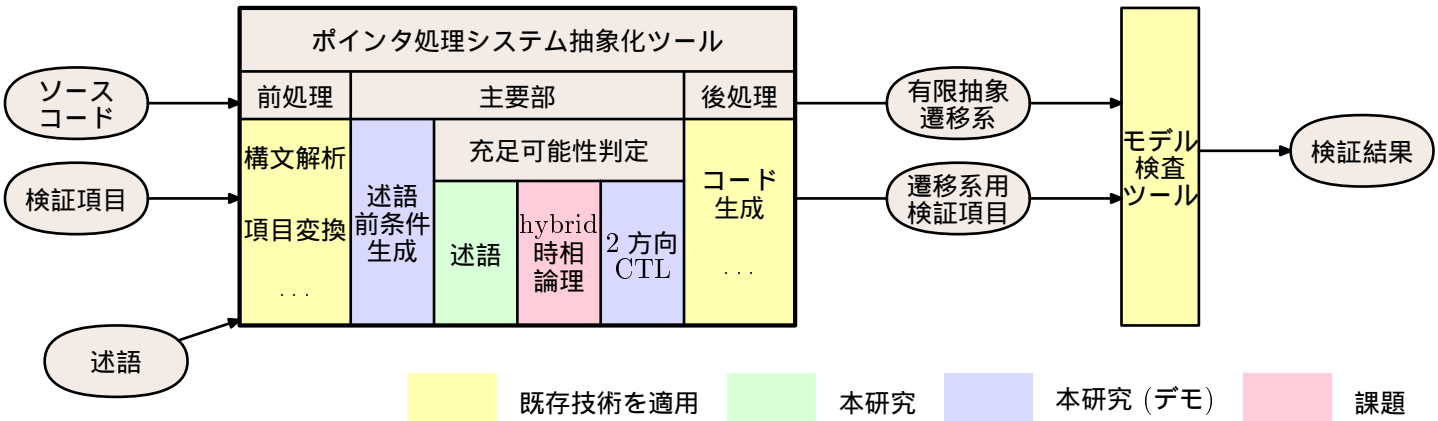
研究課題: 検証における記述量爆発問題の構造変換による解決　　2004 年 12 月 14 日
研究代表: 産業技術総合研究所 システム検証研究センター　木下佳樹

ポインタ処理システム検証のための自動抽象化法

高橋 孝一 †　田辺 良則 ‡,†　高井 利憲 ‡,†　関澤 俊弦 ‡,†
†: 産業技術総合研究所 システム検証研究センター　　‡: 科学技術振興機構

ソフトウェアの検証で鍵になるのは，検証対象の巨大な遷移系を小さくする抽象化である．本研究では，ポインタを処理するシステムを検証するために，時相論理を使用する抽象化手法を開発する．



はじめに

数理的検証技法のひとつであるモデル検査は，設計の検証に適用され，成果をあげてきた．しかし，設計に頻繁な変更がおこるタイプのソフトウェア開発への適用は困難であった．近年，「現代的モデル検査」として，ソースコード自体をモデル検査の対象とする技術が生まれている．ソースコードで表現される遷移系の状態数は莫大であり，抽象化によって少ない状態数の遷移系を作ることが重要である．

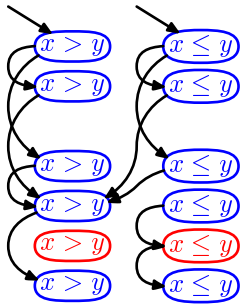
すでに，数値などの単純なデータ型に関する性質についての検証は，述語抽象化の手法で実用化されつつある．しかし，ポインタなどを使って表現される複雑なデータ型は，うまく取り扱えていない．これは，複雑なデータ型に関する性質が，述語抽象化の枠組みに乗る述語で表現できていないからだと考えられる．

本研究では，時相論理によってポインタ構造の性質を表現し，これを述語として述語抽象化を行うことをめざす．普通の検証では，時相論理は，もっぱら性質の時間的な変化を表すのに用いるのに対し，本研究では，これに加えて，時相論理をポインタ構造の空間的性質の表現にも用いる．なお，時相論理を用いたグラフ書換システムの抽象化は，高橋・萩谷によって提案されたもので，すでにセルオートマトンの解析への応用がある (→デモ)．

述語抽象化

次に示すコードをそのまま遷移系とみなすと，変数の値の組み合わせごとに別の状態になるから，サイズが巨大になる．今，「$x > y$」という述語を考え，この述語が成り立つかどうかだけを状態と考えると，状態数は $2\times$(ソース行数) に減り，右の遷移系 (抽象遷移系) が得られる．

```
if (foo(x,y,z) > 0)
  {y := x - 5;}
else
  {x := y + 3;}
if (x - y > 0)
  {ERROR: ;}
```



この遷移系のモデル検査によって，状態 ERROR に到達しないことが検証できる．

抽象遷移系

抽象遷移系は次のように決定する:

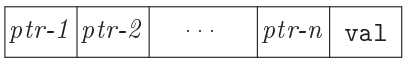
- 述語が，$\{P_0, P_1, P_2\}$ だとする．
- 抽象状態 $=\{(TTT), (TTF), \ldots, (FFF)\}\times$(ソース行の全体)
- 抽象遷移の決定．
  例: (TTF) から (FTF) に抽象遷移があるか?
  - (FTF) の前条件 Q を求める．
    「具体遷移の後で P_0:false, P_1:true, P_2:false になるのならば，具体遷移の前で，Q が成り立つ．」
  - $P_0 \wedge P_1 \wedge \neg P_2 \wedge Q$ の充足可能性を判定する．
    「充足可能なときには，必ず正しく判定する．充足不能なときには，間違えても良い．」
  - Yes なら抽象遷移あり．No なら抽象遷移なし．
- モデル検査
  決定した抽象構造上で，検証したい性質のモデル検査を行う．

ポインタ処理システムへの適用

述語抽象化法をポインタ処理システムに適用するために，以下を決定する必要がある．

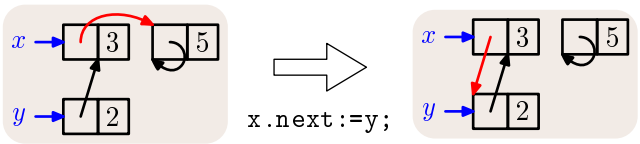
- ソースに記述できる実行文は何か．
- 述語として何を採用するか．
- 各実行文に対して，述語の前条件をどう求めるか．
- 述語の充足可能性をどう判定するか．

ポインタ構造



実行文

`x := y.new();`	生成
`x := y;`	代入
`x := y.`*pointer*`;`	ポインタをたどる
`x.val := y.val;`	値のコピー
`x.`*pointer*`:=y;`	ポインタの張り替え



$(n = 1,\ \mathit{ptr\text{-}1} = \texttt{next})$

述語

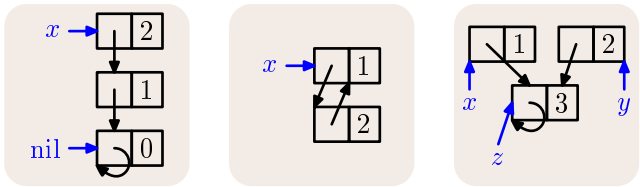
述語は，「時相論理式 φ を成り立たせるノードがある」という形の命題 ($\exists\varphi$ と書く) をブール結合したもの．時相論理にはいろいろな種類があるが，ここでは CTL を採用する．次の論理結合子が使える．

$\mathbf{AX}\varphi$	すべての次の状態で φ が成り立つ．
$\mathbf{EX}\varphi$	次の状態で φ が成り立つものがある．
$\mathbf{AF}\,\varphi$	すべてのパスでいつかは φ が成り立つ．
$\mathbf{EF}\,\varphi$	いつか φ が成り立つパスがある．
$\mathbf{AG}\,\varphi$	すべてのパスでずっと φ が成り立つ．
$\mathbf{EG}\,\varphi$	ずっと φ が成り立つパスがある．

さらに，$\mathbf{A}$ や $\mathbf{E}$ の添字に，ポインタ名を指定して区別できる．また，$\bar{a}$ は，ポインタ a を逆にたどることを意味する (2 方向 CTL)．例を挙げる．

(1) 変数 x によって指され，値は 2 で，ポインタ a をたどると nil に行き着く．
(2) ポインタ a をたどると自分に行き着く．(ポインタ a がループ状になっている．)
(3) z が指すノードが，x と y の指すノードに共通の，a による「次」のノードになっている．

(1) $x \wedge 2 \wedge \mathbf{E}_a\mathbf{F}\ \mathrm{nil}$　(2) $x \wedge \mathbf{E}_a\mathbf{X}\,\mathbf{E}_a\mathbf{F}\,x$　(3) $z \wedge \mathbf{E}_{\bar{a}}\mathbf{X}\,x \wedge \mathbf{E}_{\bar{a}}\mathbf{X}\,y$



前条件

どの実行文についても，具体遷移の後で述語 P が成り立つことと同値になる具体遷移前の述語 Q を，すなわち述語 P の前条件を求めることができる．(→デモ)

充足可能性

BDD を用いて，2 方向 CTL の充足可能性判定器を実装した．(→デモ)

述語の充足判定手続きは，たとえば次のようにすることができる．述語 $Q = \bigwedge_i\{\exists\varphi_i\} \wedge \bigwedge_j\{\forall\psi_j\}$ に対して，

- $\exists\varphi_i$ のうち，$\varphi_i = x_i \wedge \varphi_i'$ の形 (x_i は変数) のものについて，$\mathbf{AG}\,(x_i \rightarrow \varphi_i')$ をつくり，これら全部の連言を χ_1 とする．
- $\exists\varphi_i$ のうち，上の形でないものの 1 つ i_0 を選ぶ．
- $\forall\psi_j$ に対して $\mathbf{AG}\,(\psi_j)$ 全部の連言を χ_2 とする．

として，論理式 $\chi_1 \wedge \chi_2 \wedge \varphi_{i_0}$ に，(2 方向) CTL の充足可能性決定手続きを適用する．

課題: 充足可能性判定の精度

現在の充足判定手続きの精度は，あまり高くなく，意味のある結果が得られないことが多い．最大の問題は，手続きが判定するのは充足する Kripke 構造の有無であり，ポインタ構造の有無との間にギャップがあることである．

これを解決するために，「hybrid 時相論理」と呼ばれる時相論理の充足可能性問題を解く必要があり，現在の研究課題である．

デモ

前条件生成器

ポインタ構造を操作する実行文とポインタ構造に関する述語を入力して，実行文に関する述語の前条件たる述語を求める．

2CTL 充足可能性判定器

2 方向 CTL 論理式の入力に対し，充足可能性を判定する．充足可能である場合には，さらに，その論理式のモデルを表示する．

セルオートマトンの解析

1 次元セルオートマトンの書き換えルールと初期配置の入力に対し，抽象状態遷移を計算して表示する．シミュレーションも行う．

メール: {k.takahashi,tanabe.yoshinori,t-takai,sekizawa-t}@aist.go.jp