抽象化検証ツールTLATの構築に向けて [*1]

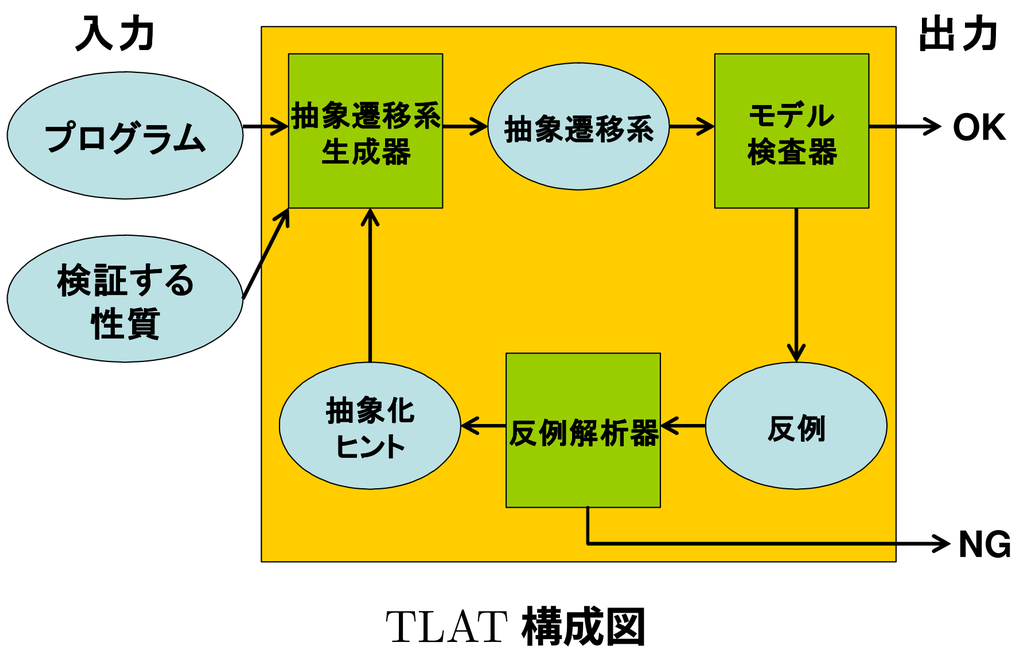
田辺 良則 [†]　関澤 俊弦 [†]　湯浅 能史 [†,‡]　高橋 孝一 [†]

†: 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

ヒープを扱うプログラムの性質を検証する抽象化検証ツール TLAT (Temporal Logic Abstraction Tool) を紹介する．現在構築中のこのツールでは，述語抽象化の技法によるソフトウェアモデル検査を行う．すなわち，利用者が指定する「述語」に基づいて，プログラムの抽象構造を作成し，これをモデル検査することによって，与えられた性質がもとのプログラムにおいて成り立つかどうかを検査する．TLAT において特徴的なのは，ヒープを Kripke 構造とみなすことにより，述語に様相・時相論理を用いることである．様相論理の充足可能性判定手続きを利用することで，抽象構造の自動的な生成を可能にしている．しかし，この素朴なアイディアに基づく抽象構造では，粒度が粗すぎて，十分な検証ができないことが多い．そこで，論理式を充足する Kripke 構造に制限を付けた判定手続きを開発した．これを用いることで，実用的な検証を可能とする粒度を持つ抽象構造を生成することができる．

構成図

TLAT は，**ヒープを扱うプログラムの性質を検証**するツールである．右図に TLAT の構成図を示す．



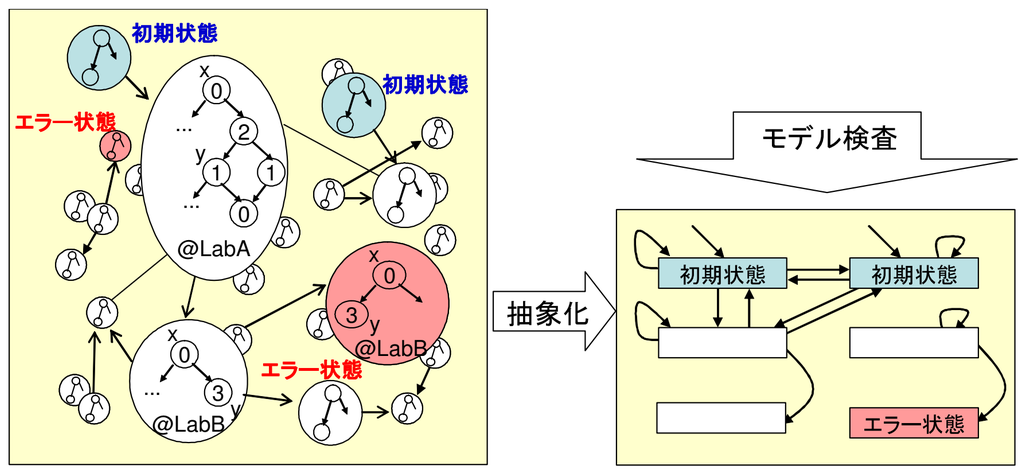
TLAT 構成図

動作原理: 述語抽象化

プログラムはヒープを扱うので，その状態は無限個である．TLAT は，この無限状態遷移系 (具体遷移系) を，右図に示すように**抽象化**によって有限状態の遷移系 (**抽象遷移系**) に変換する．変換された有限遷移系を，**モデル検査**することによって，与えられた性質を検証する．

抽象化は，**述語抽象化**の手法による:

- 有限個の述語 $P_1, \ldots, P_n$ を選ぶ．
- 抽象遷移系の状態は，プログラム中に適切に設定されたラベル位置ごとに，T または F を n 個並べたもの．前から j 番目が T であるのは，P_j が真，F であるのは P_j が偽であることに相当する．
- 抽象遷移の決定: 例えば $n = 2$ として，ラベル A からラベル B の間のプログラム実行文 (の列) が s の場合，状態 (TF)@A と状態 (FT)@B 間に遷移があるかどうかは，$P_1 \wedge \neg P_2 \wedge \mathrm{wp}(s, \neg P_1 \wedge P_2)$ が**充足可能**であるかどうかで決定する．wp は，**最弱前条件**を表す．右図参照．



抽象化

述語抽象化

---

*1 本研究は，科学技術振興機構戦略的創造研究推進事業 (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」の一部として行われた．

プログラミング言語 PML

変数 (以下では v_1 と v_2) はすべてポインタ型で，「ノード」を指す．各ノードは，ポインタ型のフィールド (以下では f) をいくつか持つ．各ノードは，列挙型の「値」(以下では n) を持つ．

代入文

- *v1* := *v2*; 変数への代入
- *v1* := *v2*.*f*; 変数への代入
- *v1* := new(); 新規ノードの作成
- *v1*.val := *n*; 値の更新
- *v1*.*f* := *v2*; フィールドの更新

複合文

- if *cond* then *stmt1* else *stmt2* fi
- while *cond* do *stmt* od
- *stmt1* *stmt2*

ヒープ状態の記述

2 方向 CTL 式をベースにした p-論理式というものを用いる．p-論理式は，v をプログラム変数，φ を 2 方向 CTL 式として，$v \supset \varphi$ と書かれる．意味は，「ヒープを Kripke 構造とみなしたとき，v が指しているノードで，φ が成立する」．

2 方向 CTL の原子論理式は次のいずれか:

変数 v	v によって指されているノードで成立
値 n	ノードが値 n を持っているとき成立

2 方向 CTL の論理記号の例:

$\mathbf{E}_f\mathbf{X}\,\varphi$	ポインタ f に関する 1 つ先で，φ が成立．
$\mathbf{E}_f\mathbf{F}\,\varphi$	ポインタ f でたどっていけるノードで，φ が成立しているところがある．
$\mathbf{E}_{\overline{f}}\mathbf{F}\,\varphi$	ポインタ f を逆にたどっていけるノードで，φ が成立しているところがある．
$\mathbf{A}_{\overline{f}}\mathbf{G}\,\varphi$	ポインタ f を逆にたどっていけるすべてのノードで，φ が成立．

検証する性質の記述

LTL 式を用いる．

LTL の時相論理記号は次の 2 つ:

$\Box\varphi$	ずっと φ が成り立つ．
$\Diamond\varphi$	いつかは φ が成り立つ．

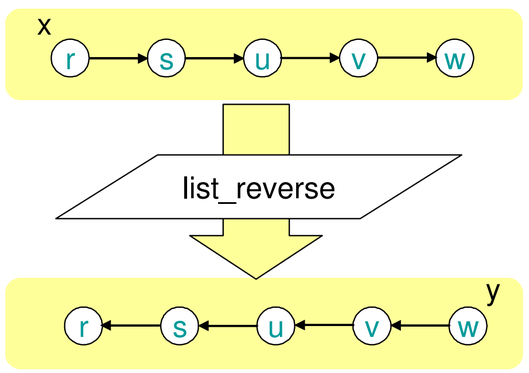
原子論理式は，次のいずれか:

@*label*	ラベル *label* の文を実行しようとしていることを示す．
p-論理式	ヒープの状態を表す．
abort	(ヌルポインタ参照などの) エラー状態を示す．

検証例

プログラム (list_reverse)

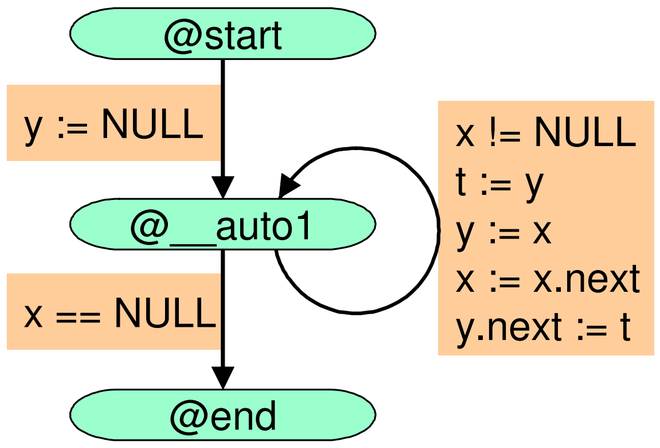
```
start:
  y := NULL;
__auto1:
  while (x != NULL) do
    t := y;  y := x;
    x := x.next;  y.next := t;
  od
end:
```



検証したい性質の例:

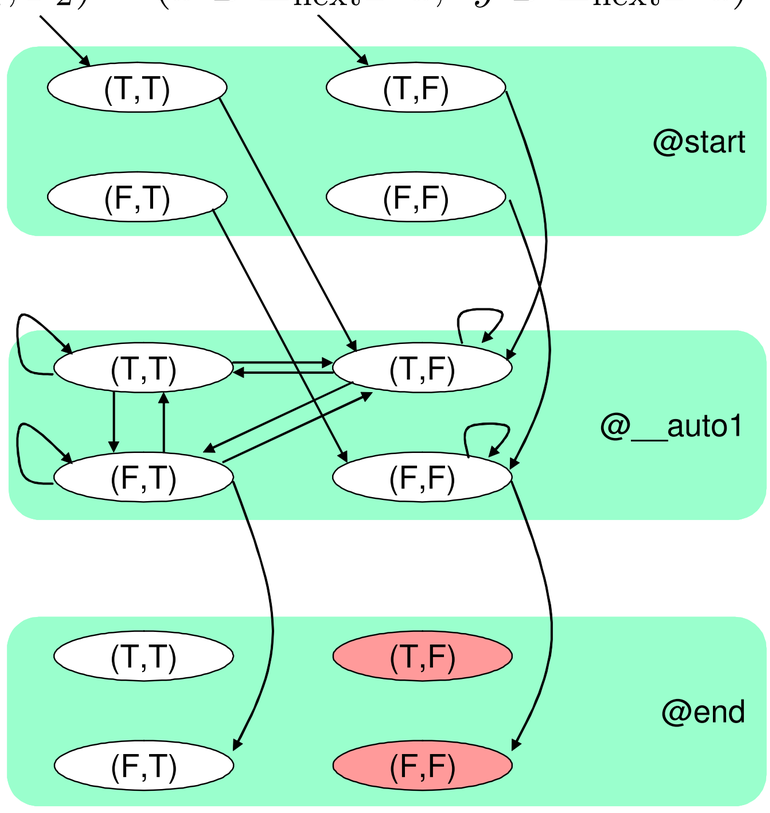
	性質	LTL 式での表現
1	実行中に，ヌルポインタは参照されない．	$\Box\neg$ abort
2	入力でノード u がノード x から到達可能なら，出力では，ノード y から到達可能．	$\Box(@\text{start} \wedge x \supset \mathbf{E}_{\text{next}}\mathbf{F}\,u \Rightarrow \Box(@\text{end} \Rightarrow y \supset \mathbf{E}_{\text{next}}\mathbf{F}\,u))$
3	入力で $u \to v$ の順で並んでいたノードは，出力では，$v \to u$ の順に並ぶ．	$\Box(@\text{start} \wedge x \supset \mathbf{E}_{\text{next}}\mathbf{F}\,(u \wedge \mathbf{E}_{\text{next}}\mathbf{X}\,v) \Rightarrow \Box(@\text{end} \Rightarrow v \supset \mathbf{E}_{\text{next}}\mathbf{X}\,u))$

制御流れグラフ



性質 2 の検証のための抽象構造

述語 $(P_1, P_2) = (x \supset \mathbf{E}_{\mathrm{next}}\mathbf{F}\, u,\ \ y \supset \mathbf{E}_{\mathrm{next}}\mathbf{F}\, u)$



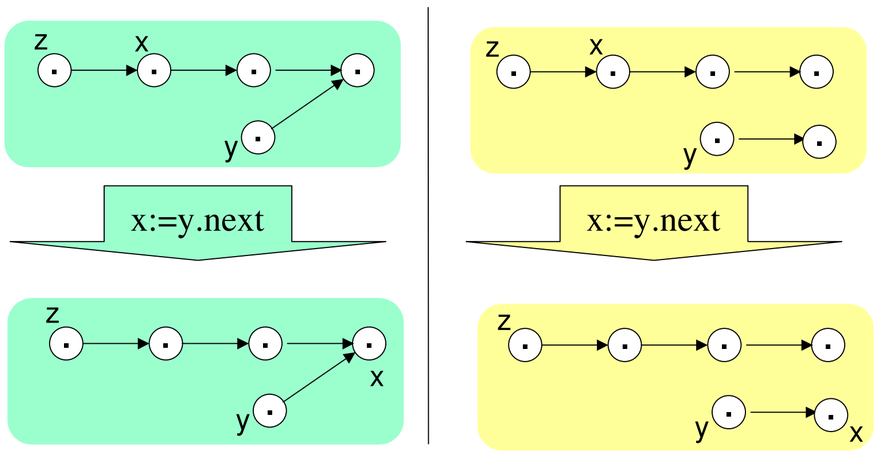
最弱前条件

各プログラム実行文 s と，p-論理式 Q に対して，TLAT は内部的に最弱前条件 $\mathrm{wp}(s, Q)$ を計算する．

例: $\mathrm{wp}(\mathtt{x:=y.next},\ z \supset \mathbf{E}_{\mathrm{next}}\mathbf{F}\, x)$

$$= \quad z \supset \mathbf{E}_{\mathrm{next}}\mathbf{F}\, \mathbf{E}_{\overline{\mathrm{next}}}\mathbf{X}\, y.$$

右図で，左側は $z \supset \mathbf{E}_{\mathrm{next}}\mathbf{F}\, x$ が成立する例，右側は成立しない例である．



充足可能性決定手続き

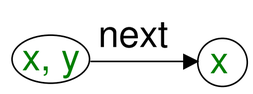
p-論理式のベースにある 2 方向 CTL の充足可能性について，著者らは，BDD を使用した効率的な健全かつ完全な決定手続きを提案している．TLAT で使用する p-論理式の充足可能性決定手続きは，これをベースとしている．しかし，右に示すように，Kripke 構造としての充足可能性とヒープ構造の意味での充足可能性には次の事実に基づくギャップがある:

- **プログラム変数が指すノードは 1 つだけ**
- **各フィールドが指すノードは 1 つだけ**

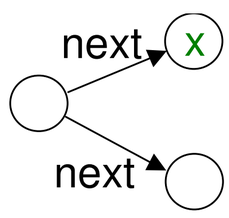
TLAT では，このギャップを埋めるように充足可能性判定手続きを修正して使用している．現在のところ，この手続きは健全ではあるが，完全ではない．

ヒープ構造では充足されないが，Kripke 構造では充足される論理式:

(1) $x \wedge y \wedge \mathbf{E}_{\mathrm{next}}\mathbf{X}\,(x \wedge \neg y)$



(2) $(\mathbf{E}_{\mathrm{next}}\mathbf{X}\, x) \wedge (\mathbf{E}_{\mathrm{next}}\mathbf{X}\, \neg x)$



プロトタイプ v1

- **抽象遷移系生成部分を実装．**
- **モデル検査は，既存のツール (SPIN) を使用．**
- **利用者が抽象化のヒントを与える．**
- **反例解析は，利用者が行う．**
- **現在コーディング中．(2005 年 12 月動作予定)**