プログラム言語特論 後半レポート

コンピュータサイエンス専攻 江畑 拓哉 (201920631)

1 Summary of "Constraint Solving for Program Verification Theory and Practice by Example"

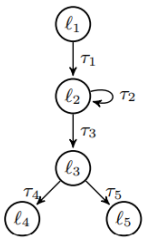
プログラム検証はプログラムの振る舞い (program behavior) の様々な側面から記述される補助アサーション (auxiliary assertions) の構築に依存している。補助アサーションの例としては、帰納的不変式 (inductive invariants)、資源境界 (resource bounds)、到達可能なプログラムの状態を特徴づけるための内挿 (interpolants for characterizing reachable program states)、プログラムの終了までの実行ステップ数を近似するためのランキング関数 (ranking functions for approximating number of execution steps until program termination)、非終了 (non-termination) 性を証明するための再帰集合 (recurrence sets) などが上げられる。昨今の制約ソルバ (constraint solving tools) はプログラム検証を効率的に自動化することを助ける。この論文では、基本的な計算機 (Computer Machinery) として制約ソルバ (constraint solvers) を利用することで上述したような補助アサーションの自動構築のためのアルゴリズムを例とともに示していく。

プログラム検証では一般に制約ベースのアルゴリズムを用いられてきた。このアルゴリズムは 2 つの主要なステップに分かれており、第一ステップは制約を生成するステップで、関心のあるプログラムの特性を制約の集合として定式化し、第二ステップではそれらの制約を解決するステップとなっている。一般的に第二ステップは制約ソルバ手続きの分離 (separation of constraint solving procedure) を用いて実行されている。これによって、問題ごとに専用のプログラム検証ツールを作ることなく、既存の制約ソルバを用いることができる。この論文では、ランキング関数、内挿、不変式、資源境界、再起集合を生成することで制約を用いてプログラムの (非) 終了性、安全性を証明する方法を例を用いて示していく。

この論文で例として用いられているプログラムと、その control-flow グラフ、そして対応遷移関係 (corresponding transition relations) を引用する。

```
main(int x, int y, int z) {
  assume(y >= z);
  while (x < y) {
    x++;
  }
  assert(x >= z);
}
```

(a)



(b)

$$\rho_1 = (y \geq z \wedge x' = x \wedge y' = y \wedge z' = z)$$
$$\rho_2 = (x + 1 \leq y \wedge x' = x + 1 \wedge y' = y \wedge z' = z)$$
$$\rho_3 = (x \geq y \wedge x' = x \wedge y' = y \wedge z' = z)$$
$$\rho_4 = (x \geq z \wedge x' = x \wedge y' = y \wedge z' = z)$$
$$\rho_5 = (x + 1 \leq z \wedge x' = x \wedge y' = y \wedge z' = z)$$

(c)

Fig. 1. An example program (a), its control-flow graph (b), and the corresponding transition relations (c).

Figure 1: Fig. 1. Constraint Solving for Program Verification Theory and Practice by Example より引用

このプログラムにおいては問題の簡潔化のため、プログラム内の整数の自由変数を有理数で近似している。これによって ρ_2 はガード (guard) $x + 1 \leq y$ を持つ。また assert 文の失敗は control location l_5 への到達を以て表現される。

2 線形ランキング関数 (linear ranking functions)

プログラムの終了性を証明するためには、実行ステップ数を過大評価するランキング関数を構築しなければならない。線形ランキング関数は、プログラムの自由変数に対する線形アサーションによってこの近似を行う。

例として Figure 1. の loop 関数の部分を特に注目する。
ループ内には単純な要素のみが含まれているため、制約ベースのランキング関数生成の主要概念を強調することができる。終了を証明するプログラムの自由変数上の線形式 (linear expression) を探索する。プログラム中の自由変数 x, y に関する係数を fx, fy とする (z はこの部分では登場しないため省略する)。このループでステップを踏むことができるすべての状態において、値に下限 δ_0 が設けられており、その値がある一定の正の固定量 δ だけ減少しているならば、これを示す線形式はランキング関数とみなすことができる。以上の値を用いて次の制約を定義できる。

$$\begin{aligned}
&\exists f_x \exists f_y \exists \delta_0 \exists \delta \\
&\forall x \forall y \forall x' \forall y' : \\
&\quad (\rho \geq 1 \wedge \\
&\quad \rho_2 \rightarrow (\quad f_x x + f_y y \geq \delta_0 \wedge \\
&\qquad\qquad f_x x' + f_y y' \geq f_x x + f_y y - \delta \quad)
\end{aligned} \tag{1}$$

つまり $f_x, f_y, \delta_0, \delta$ にある充足可能な割当を行うことで、この loop 関数の線形ランキング関数を決定できる。制約条件 (1) は、x, y, x', y' への普遍的な定量化 $\forall$ を含み、それは既存の制約ソルバを用いた解決を困難にしている。よって $\forall$ を排除するために以下の手続きを踏んでいく。

まず制約生成のために遷移関係を行列形式に落とし込む。尚このために等式を不等式の形に変形している。

$$\begin{aligned}
\rho_2 &= (x + 1 \leq x' = x + 1 \wedge y' = y) \\
&= (x - y \leq -1 \wedge -x + x' \leq 1 \wedge \\
&\quad x - x' \leq -1 \wedge -y + y' \leq 0 y - y' \leq 0) \\
&= \begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \\ x' \\ y' \end{pmatrix} \leq \begin{pmatrix} -1 \\ 1 \\ -1 \\ 0 \\ 0 \end{pmatrix}
\end{aligned}$$

また (1) の下界と減少に関する条件は以下の行列形式に落とし込める。

$$f_x x + f_y y \geq \delta_0 = \begin{pmatrix} -f_x & -f_y & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ x' \\ y' \end{pmatrix} \leq -\delta_0$$

$$f_x x' + f_y y' \leq f_x x + f_y y - \delta = \begin{pmatrix} -f_x & -f_y & f_x & f_y \end{pmatrix} \begin{pmatrix} x \\ y \\ x' \\ y' \end{pmatrix} \leq -\delta$$

これに以下で示される Farkas の補題 (Farkas' Lemma) を適用して $\forall$ を排除していく。

$$\exists x : Ax \leq b \rightarrow ((\forall x : Ax \leq b \rightarrow cx \leq \gamma) \leftrightarrow (\exists \lambda : \lambda \geq 0 \wedge \lambda A = c \wedge \lambda b \leq \gamma))$$

Farkas の補題より、ある充足可能な x についての式 (左辺) から以下の実質等値な関係 (右辺) を導き出すことができ ($c = 0, \gamma = -1$)、これを (1) に適用することで (2) のような $\forall$ を排除した制約を得ることができる。

$$(\forall x : \neg(Ax \leq b)) \leftrightarrow (\exists \lambda : \lambda \leq 0 \wedge \lambda A = 0 \wedge \lambda b \leq -1)$$

$$\begin{aligned}
&\exists f_x \exists f_y \exists \delta_0 \exists \delta \\
&\quad \exists \lambda \exists \mu : \\
&\quad\quad \delta \geq 1 \quad \wedge \\
&\quad\quad \lambda \geq 0 \quad \wedge \\
&\quad\quad \mu \geq 0 \quad \wedge \\
&\lambda \begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{pmatrix} = \begin{pmatrix} -f_x & -f_y & 0 & 0 \end{pmatrix} \wedge \lambda \begin{pmatrix} -1 \\ 1 \\ -1 \\ 0 \\ 0 \end{pmatrix} \leq -\delta_0 \wedge \\
&\mu \begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{pmatrix} = \begin{pmatrix} -f_x & -f_y & f_x & f_y \end{pmatrix} \wedge \mu \begin{pmatrix} -1 \\ 1 \\ -1 \\ 0 \\ 0 \end{pmatrix} \leq -\delta
\end{aligned} \tag{2}$$

この制約では $\exists$ の有利変数のみを含み線形 (不) 等式からなりたっている。したがって、これは既存のツールで解決できる有理数上の線形計画法の問題として扱

うことができるようになった。

これを後述するアルゴリズムを用いて解くと以下の結果が得られる。

$$\begin{aligned}\lambda &= \begin{pmatrix}1 & 0 & 0 & 0 & 0\end{pmatrix}\\ \mu &= \begin{pmatrix}0 & 0 & 1 & 1 & 0\end{pmatrix}\\ f_x &= -1\\ f_y &= 1\\ \delta_0 &= 1\\ \delta &= 1\end{aligned}$$

これを解釈すると、各イテレーションで $-x+y$ が少なくとも 1 ずつ減少すること、ループガード (loop guard) を満たすすべてのイテレーションの状態で $-x+y$ は 1 以上であることが示されている。

尚 v をプログラムの変数に対する線形不等式の集合として、状態遷移を $\rho(v,v')$ として表すと以下の式が成り立つ。

$$\rho(v,v') = R\begin{pmatrix}v\\ v'\end{pmatrix} \le r$$

そして v の係数 f のベクトルが線形ランキング関数を定義する条件は以下の制約によって表される。

$$\exists f \exists \delta_0 \exists \delta \forall v \forall v' : \delta \ge 1 \wedge \rho(v,v') \to (fv \ge \delta_0 \wedge fv' \le -\delta) \tag{3}$$

(3) へ Farkas の補題を適用するとヨのみの式 ((2) と参照) として再構築することができる。

$$\begin{aligned}&\exists f \exists \delta_0 \exists \delta\\ &\exists \lambda \exists \mu :\\ &\quad \delta \ge 1 \wedge\\ &\quad \lambda \ge 0 \wedge \mu \ge 0 \wedge\\ &\quad \lambda R = \begin{pmatrix}-f & 0\end{pmatrix} \wedge \lambda r \le -\delta_0\\ &\quad \mu R = \begin{pmatrix}-f & f\end{pmatrix} \wedge \mu r \le -\delta\end{aligned} \tag{4}$$

3 (制約付き) 線形内挿の計算手法 (how to compute (Constrained) linear interpolant

s)
内挿 (Interpolants) はある望ましい性質を持つプログラムの状態とその性質に違反するプログラムの状態と区別することができるプログラムの状態に関する論理的なアサーションである。

内挿はプログラムの状態の集合を自動的に抽象化する際に重要な役割を担っており、プログラム検証ツールにとって非常に重要な構成要素である。以下に線形内挿の計算手法についてのアルゴリズムを Figure 1. の例を用いて示す。この特徴として、追加の制約を用いることで結果にバイアスをかけられるという点を挙げることができる。

プログラム検証において内挿は、プログラムのパスから抽出された式、言い換えるとプログラムの control flow グラフに従うプログラムの状態のシーケンスから計算される。

ループに入らずに assert の状態を失敗するプログラムの実行に対応するパス τ_1, τ_3, τ_5 について考えたとき、この場合の自由変数の値は変更されず、一連の条件 $y \ge z \wedge x \ge y \wedge x+1 \le z$ が課されている。このシーケンスは望ましいものではなく、プログラム検証では、プログラムの状態を τ_3 を取ったあとの状態を分離するための内挿クエリを発行する。
形式的には、内挿と呼ばれる $i_x x + i_y y + i_z z \le i_0$ という不等式について考える。

$$\begin{aligned}&\exists i_x \exists i_y \exists i_z \exists i_0\\ &\forall x \forall y \forall z :\\ &((y \ge z \wedge x \ge y) \to i_x x + i_y y + i_z z \le i_0) \wedge\\ &((i_x x + i_y y + i_z z \le i_0 \wedge x+1 \le z) \to 0 \le -1)\end{aligned} \tag{5}$$

$i_x x + i_y y + i_z z \le i_0$ は $y \ge z \wedge x \ge y$ と $x+1 \le z$ の両方に登場する自由変数を参照している必要があり、すなわち i_y は 0 である必要があると推論される。これは上記の制約のみで保証することができる。

まず一連の条件より以下の行列形式が求められる。

$$\begin{aligned}(y \ge z \wedge x \ge y \wedge x+1 \le z) &=\\ (-y+z \le 0 \wedge -x+y \le 0 \wedge x-z \le -1) &=\\ \begin{pmatrix}0 & -1 & 1\\ -1 & 1 & 0\\ 1 & 0 & -1\end{pmatrix}\begin{pmatrix}x\\ y\\ z\end{pmatrix} &\le \begin{pmatrix}0\\ 0\\ -1\end{pmatrix}\end{aligned}$$

2 と同様に (5) の $\forall$ をなくすため Farkas の補題を適用すると以下の形になる。

$$\begin{aligned}&\exists i_x \exists i_y \exists i_z \exists i_0\\ &\exists \lambda \exists \mu :\\ &\lambda \ge 0 \wedge \mu \ge 0 \quad \wedge\\ &\begin{pmatrix}\lambda & \mu\end{pmatrix}\begin{pmatrix}0 & -1 & 1\\ -1 & 1 & 0\\ 1 & 0 & -1\end{pmatrix} = 0 \quad \wedge \quad \begin{pmatrix}\lambda & \mu\end{pmatrix}\begin{pmatrix}0\\ 0\\ -1\end{pmatrix} \le -1 \wedge\\ &\begin{pmatrix}i_x & i_y & i_z\end{pmatrix} = \lambda\begin{pmatrix}0 & -1 & 1\\ -1 & 1 & 0\end{pmatrix} \wedge i_0 = \lambda\begin{pmatrix}0\\ 0\end{pmatrix}\end{aligned} \tag{6}$$

ただし λ と μ は線型結合を表しており、充足不可能な不等式 $0 \le -1$ を導出するための線形結合を表している。 λ は上から 2 つの不等式に追従しており、 μ は 3 つ目の不等式に追従している。

これを後述のアルゴリズムを用いて解くと以下のようになる。

$$\begin{aligned}\lambda &= \begin{pmatrix}1 & 1\end{pmatrix}\\ \mu &= 1\\ i_x &= -1\\ i_y &= 0\\ i_z &= 1\\ i_0 &= 0\end{aligned}$$

これによって内挿は $-x+z \le 0$ であることがわかる。

本アルゴリズムでは、2 つの線形不等式、 $Av \le a$ と $Bv \le b$ の集合を引数に取る。これらの不等式は互いに充塞可能になることがなく、以下の制約を充足可能にする内挿 $iv \le i_0$ を計算する。

$$\begin{aligned}&\exists i \exists i_0\\ &\forall v :\\ &\quad (Av \le a \to iv \le i_0) \wedge\\ &\quad ((iv \le i_0 \wedge Bv \le b) \to 0 \le -1)\end{aligned} \tag{7}$$

この式へ Farkas の補題を適用することでヨの自由変数のみの式へ変形する。

$$\begin{aligned}&\exists i \exists i_0\\ &\exists \lambda \exists \mu :\\ &\quad \lambda \ge 0 \wedge \mu \ge 0 \wedge\\ &\quad \begin{pmatrix}\lambda & \mu\end{pmatrix}\begin{pmatrix}A\\ B\end{pmatrix} = 0 \wedge \begin{pmatrix}\lambda & \mu\end{pmatrix}\begin{pmatrix}a\\ b\end{pmatrix} \le -1 \wedge\\ &\quad i = \lambda A \wedge i_0 = \lambda a\end{aligned} \tag{8}$$

内挿を計算するための制約ベースのアプローチは、追加の制約を使って結果の内挿をバイアスするためのユニークな機会が得られる。実際 (6) はバイアス条件を符号化する追加の制約 $C(i,i_0) \le c$ で拡張することができた。

4 線形不変式生成 (linear invariants generation) とプログラムテストケースを活用する最適化手法

不変式はプログラムの実行中にその値が変わらない自由変数のアサーションである。プログラム検証では、不変式はプログラムの到達可能な状態を記述するために用いられ、プログラムの正確さについて推論するために必要なツールである。以降では不変式がプログラム中のエラー箇所の非到達性を向上させることが制約ベースの技術を使って計算することができることや、制約生成タスクの結果を簡略化させるためのテストベースのアプローチが示される。更に不変式と境界生成の密接な関わりについても簡潔に示す。

Figure 1. を用いて不変式生成の例を示し、エラー箇所を示す ℓ_5 への非到達性を証明する不変式の制約を行う。

目標として、ℓ_2 と ℓ_3 それぞれついての 2 つの線形不等式 $p_x x + p_y y + p_z z \le p_0$ と $q_x x + q_y y + q_z z \le q_0$ を解くことを設定する。これらの不等式は、(1) それぞれの箇所でのすべてのプログラムの到達可能な状態を示せる、(2) (1) を証明するための帰納仮説を提供できる、(3) すべてのプログラムの実行がエラー箇所である ℓ_5 へ到達しないことを示せるように求められます。まず未知の不変式の係数を用いて (1-3) の制約を表すと以下のようになる。

$$\begin{aligned}&\exists p_x \exists p_y \exists p_z \exists p_0 \exists q_x \exists q_z y \exists q_z \exists q_0\\ &\forall x \forall y \forall z \forall x' \forall y' \forall z' :\\ &(\rho_1 \to p_x x' + p_y y' + p_z z' \le p_0) \wedge\\ &((p_x x + p_y y + p_z z \le p_0 \wedge \rho_2) \to p_x x' + p_y y' + p_z z' \le p_0) \wedge\\ &((p_x x + p_y y + p_z z \le p_0 \wedge \rho_3) \to q_x x' + q_y y' + q_z z' \le q_0) \wedge\\ &((q_x x + q_y y + q_z z \le p_0 \wedge \rho_4) \to 0 \le 0) \wedge\\ &((q_x x + q_y y + q_z z \le p_0 \wedge \rho_5) \to 0 \le -1)\end{aligned} \tag{9}$$

上式にはそれぞれのプログラムでの遷移ごとに、この制約に対応する連言が含まれている。例えば 1 つ目の連言は、任意の状態へ τ_1 が適用されたならば必ず $p_x x + p_y y + p_z z \leq p_0$ で表される状態へ向かうことを保証している。

それぞれの遷移関係 $\rho_1, \ldots, \rho_5$ を行列形式 $R_1 \begin{pmatrix} v \\ v' \end{pmatrix} \leq r_1, \ldots, R_5 \begin{pmatrix} v \\ v' \end{pmatrix} \leq r_5$ として表し、v, v' はプログラムの自由変数 x, y, z と x', y', z' のベクトルを表す。(9) へ Farkas の補題を適用して $\forall$ を除去すると、以下の制約を得る。

$$
\begin{aligned}
&\exists p_x \exists p_y \exists p_z \exists p_0 \exists q_x \exists q_y \exists q_z \exists q_0 \\
&\quad \exists \lambda_1 \exists \lambda_2 \exists \lambda_3 \exists \lambda_4 \exists \lambda_5 : \\
&\quad \lambda_1 \geq 0 \wedge \cdots \wedge \lambda_5 \geq 0 \quad \wedge \\
&\quad\quad \lambda_1 R_1 &&= \begin{pmatrix} 0 & p_x & p_y & p_z \end{pmatrix} \wedge \lambda_1 r_1 \leq p_0 \wedge \\
&\quad\quad \lambda_2 \begin{pmatrix} p_x & p_y & p_z & 0 \\ & & R_2 & \end{pmatrix} &&= \begin{pmatrix} 0 & p_x & p_y & p_z \end{pmatrix} \wedge \lambda_2 \begin{pmatrix} p_0 \\ r_2 \end{pmatrix} \leq p_0 \wedge \\
&\quad\quad \lambda_3 \begin{pmatrix} p_x & p_y & p_z & 0 \\ & & R_3 & \end{pmatrix} &&= \begin{pmatrix} 0 & p_x & p_y & p_z \end{pmatrix} \wedge \lambda_3 \begin{pmatrix} p_0 \\ r_3 \end{pmatrix} \leq q_0 \wedge \\
&\quad\quad \lambda_4 \begin{pmatrix} q_x & q_y & q_z & 0 \\ & & R_4 & \end{pmatrix} &&= 0 \wedge \lambda_4 \begin{pmatrix} q_0 \\ r_4 \end{pmatrix} \leq 0 \wedge \\
&\quad\quad \lambda_5 \begin{pmatrix} q_x & q_y & q_z & 0 \\ & & R_5 & \end{pmatrix} &&= 0 \wedge \lambda_5 \begin{pmatrix} p_0 \\ r_5 \end{pmatrix} \leq -1 \wedge
\end{aligned} \tag{10}
$$

しかしこの制約は未知の要素 $\lambda_1, \ldots, \lambda_5$ と $p_x, p_y, p_z, p_0, q_x, q_y, q_z, q_0$ の乗算を含んでいるので非線形なものとなっている。このため追加の非線形項を減らし、残っている未知係数についての分析を行い制約をわかりやすくするステップを導入する。

(10) を解くことで以下の結果を得る。

$$
\begin{aligned}
\lambda_1 &= \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix} \\
\lambda_2 &= \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \end{pmatrix} \\
\lambda_3 &= \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \end{pmatrix} \\
\lambda_4 &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
\lambda_5 &= \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \end{pmatrix} \\
p_x = 0 \, p_y &= -1 \, p_z = 1 \, p_0 = 0 \\
q_x = -1 \, q_y &= 0 \, q_z = 1 \, q_0 = 1
\end{aligned}
$$

これによって、ℓ_2 における不変式 $-y + x \leq 0$ と ℓ_3 における不変式 $-x + z \leq 0$ を得る。

まず 1 つ目の入力として、データ変数 v と プログラムカウンター pc、遷移に関する無限集合 $\mathcal{T}$、開始位置 $\ell_{\mathcal{I}}$、エラー箇所 $\ell_{\mathcal{E}} \in \mathcal{L}$ を含む入力プログラム $P = (v, pc, \mathcal{L}, \mathcal{T}, \ell_{\mathcal{I}}, \ell_{\mathcal{E}})$ を定義する。それぞれの遷移を表す $(l, \rho(v, v'), l') \in \mathcal{T}$ には開始位置 l と 遷移関係 $\rho(v, v')$、目的先 l' が含まれている。2 つ目の入力は、未知係数 I_ℓ と i_ℓ を含むプログラムの自由変数についての線形不等式 $I_\ell v \leq i_\ell$ の集合をそれぞれの control location l へ割り当てるテンプレートマップである。これらを用いて次の制約を満たすような係数を見つけるための式へ書き換える。

$$
\begin{aligned}
&\exists I_{\ell \in \mathcal{L}} \exists i_{\ell \in \mathcal{L}} \\
&\quad \forall v \forall v' : \\
&\quad (I_{\ell_{\mathcal{I}}} = 0 \wedge i_{\ell_{\mathcal{I}}} = 0) \quad \wedge \quad (I_{\ell_{\mathcal{E}}} \wedge i_{\ell_{\mathcal{E}}} = -1) \wedge \\
&\quad (\forall (\ell, \rho(v, v'), l') \in \mathcal{T} : \\
&\quad\quad (I_l v \leq i_\ell \wedge \rho(v, v')) \quad \rightarrow \quad I_{l'} v' \leq i_{l'})
\end{aligned} \tag{11}
$$

まずこの制約によってテンプレート $I_{\ell_{\mathcal{I}}} v \leq i_{\ell_{\mathcal{I}}}$ よって課される開始位置 ℓ_0 に制限がないことが保証されている。そして制約はどの実行もエラー箇所に到達しないことが必要とされる。例えば対応するテンプレート $I_{\ell_{\mathcal{E}}} v \leq i_{\ell_{\mathcal{E}}}$ は満たされない不等式の集合を生み出す。それぞれのプログラムの遷移については、制約はこの遷移を取ることによって到達可能な状態の集合がそれぞれの不等式の集合の元にあることが要求される。

Farkas の補題を用いて $\forall$ を除去すると以下に変形できる。

$$
\begin{aligned}
&\exists I_{l \in \mathcal{L}} \exists i_{l \in \mathcal{L}} \\
&\quad \exists \Lambda_{\tau \in \mathcal{T}} : \\
&(I_{\ell_{\mathcal{I}}} = 0 \wedge i_{\ell_{\mathcal{I}}} = 0) \quad \wedge \quad (I_{\ell_{\mathcal{E}}} = 0 \wedge i_{\ell_{\mathcal{E}}} = -1) \wedge \\
&(\forall \tau = (\ell, R \begin{pmatrix} v \\ v' \end{pmatrix} \leq r, \ell') \quad \in \quad \mathcal{T} : \\
&\qquad\qquad \Lambda_\tau \geq 0 \wedge \\
&\qquad\qquad \Lambda \begin{pmatrix} I_\ell & 0 \\ R & \end{pmatrix} = I_{\ell'} \wedge \Lambda_\tau \begin{pmatrix} i_\ell \\ r \end{pmatrix} \leq i_{\ell'})
\end{aligned} \tag{12}
$$

Λ と I_ℓ、Λ と i_ℓ の乗算に非線形性が見られ、理論上では (10) の非線形制約は有理数/実数上の Quantifier elimination の手続きによって解決することができるとされているが、実用上ではこの直接的なアプローチは容易に困難になりやすい。このため様々にある非線形項の量を減らす技術の一つである、プログラムテストを用いた手法を適用する。

以下の一連のプログラムの状態について議論する。これらは到達可能な状態であり、すべてのプログラムの不変式はこの状態を含んでいることが推論できる。

$$
\begin{aligned}
s_1 &= (\ell_1, x = 1, y = 2, z = 1) \\
s_2 &= (\ell_2, x = 2, y = 2, x = 1) \\
s_3 &= (\ell_2, x = 2, y = 2, z = 1) \\
s_4 &= (\ell_3, x = 2, y = 2, z = 1) \\
s_5 &= (\ell_4, x = 2, y = 2, z = 1)
\end{aligned}
$$

ℓ_2 と ℓ_3 に関する不変式のテンプレートについてそれぞれ考えると、以下の制約が得られる。

$$
\begin{aligned}
\varphi_1 &= (p_x 1 + p_y 2 + p_z 1 \leq p_0) \\
\varphi_2 &= (p_x 2 + p_y 2 + p_z 1 \leq p_0) \\
\varphi_3 &= (q_x 2 + q_y 2 + q_z 1 \leq q_0)
\end{aligned}
$$

これらの連言、つまり $\varphi_1 \wedge \varphi_2 \wedge \varphi_3$ を取ると次の式が得られる。これを解集合をそのままに (10) へ追加の強化として加えると、非線形部の単純化を行うことができ、制約解決効率を向上させることができる。

$$
p_x 1 + p_y 2 + p_z 1 \leq p_0 \wedge p_x 2 + p_y 2 + p_z 1 \leq p_0 \wedge q_x 2 + q_y 2 + q_z 1 \leq q_0
$$

S をプログラムの到達可能な状態を表す無限集合であるとして、次の制約を解集合をそのままに (12) へ加える。

次に未知アサーションとしての境界生成を行う。

プログラム実行によってメモリや実行時間などの様々な資源を消費する。資源境界は資源消費量を見積もるために有用な論理的なアサーションであり、特に限られたリソースの可用性しかないプログラム実行環境においてこの自動生成は重要である。またこの資源境界を求めることとプログラム状態の到達可能性を求めることには強いつながりがある。言い換えると与えられた境界内ですべてのプログラムの実行が成立するのかのチェックを、資源消費を追跡するプログラムの補助自由変数に対するアサーションとみなすことができる。以降の例では、x を用いて実行時間の経過を追跡し、z の値でこれの下限を求める。

未知の資源境界は、生成された制約にわすかな修正を加えた後、上述の不変式生成アルゴリズムを使用して合成することができる。
以降に制約 (9) と (10) を、遷移関係 ρ_4 ρ_5 によって表されるアサーションの状態の仮定の元で loop 回数の境界を特定することで修正する方法を示す。

まず未知境界のアサーションを以下の不等式で示す。この式はその境界範囲の妥当性を証明する不変式とともに係数 b_y, b_z, b_0 の値を求めることを目的とする。

$$
x \leq b_y y + b_z z + b_0
$$

次に (9) の最後の 2 つの連言を置き換えることで、この目的を制約に落とし込む。この式は loop の出口を出た後の時点のプログラムの不変式が境界の妥当性を含意していることを必要とする。

$$
q_x x + q_y y + q_z z \leq q_0 \rightarrow x \leq b_y y + b_z z + b_0
$$

修正された制約から $\forall$ を除去し、これを解くことで x に対する境界が存在していないことがわかる。よって次の仮定の状態を加えた修正されたプログラムを提案する。

```
assume(z >= x);
```

この修正されたプログラムを用いることで以下の境界を得ることができる。

$$
x \leq y
$$

5 プログラムの非終了性判定 (とそのための再帰集合の構築) 法

既存のプログラムの終了性を証明するためのツールにある固有の制限は、非決定的な結果が報告されるケースを引き起こす。終了することができる引数がツールによって見つからなかったからと言って特定の入力に対して終了しないと言い切れるわけではないので、プログラムの非終了性を判定するための手法が必要となる。以降では非終了なプログラムの実行の存在を証明することができる再帰集合の概念を用いてこれを解決する。

ここでは Figure 2. を例にとる。以下に論文中のそれを引用する。尚このプログラムには値のオーバーフローが存在しないものとする。

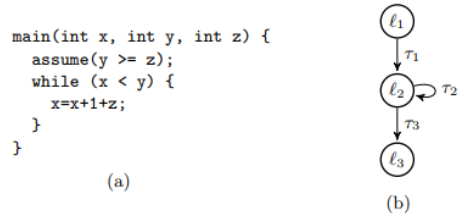


Fig. 2. A non-terminating example program (a), its control-flow graph (b), and the corresponding transition relations (c).

Figure 2: Fig. 2. Constraint Solving for Program Verification Theory and Practice by Example より引用

非終了性を証明するために loop の開始地点へ到達し、次の loop へ向かう可能性があるプログラムの状態で構成される再帰集合を計算する。理想的な再帰集合は、プログラムの自由変数 x, y, z からなるベクトル v と未知係数 p, p_0, q, q_0 からなる 2 つの不等式の連言 $pv \leq p_0 \wedge qv \leq q_0 = Sv \leq s$ によって表現されると仮定すると以下の制約が書ける。

$$\begin{aligned} &\exists S \exists s : \\ &(\exists v \exists v' \quad : \rho_1(v, v') \wedge Sv' \leq s) \wedge \\ &(\forall v \exists v' \quad : Sv \leq s \rightarrow (\rho_2(v, v') \wedge Sv' \leq s)) \end{aligned} \tag{13}$$

1 つ目の連言は、再帰集合が空集合でないことと、再帰集合が遷移 τ_1 によって到達可能な状態を少なくとも 1 つは持っていることを保証する。2 つ目の連言は再帰集合内のすべての状態が遷移 τ_2 に従うことと再帰が再帰集合内で閉じていることを保証する。またこれらの性質が保証されると同時に、再帰集合の要素によって構築される無限回実行されるプログラムの実行があることが保証されている。

(13) 式より $\forall$ を除去する前に、遷移関係 $\rho_1(v, v')$ と $\rho_2(v, v')$ の展開を行う。これによって $'$ を除去することができる。

$$\begin{aligned} &\exists S \exists s : \\ &(\exists x \exists y \exists z \quad : y \leq z \wedge S \begin{pmatrix} x \\ y \\ z \end{pmatrix} \leq s) \wedge \\ &(\forall x \forall y \forall z \quad : S \begin{pmatrix} x \\ y \\ z \end{pmatrix} \leq s \rightarrow (x + 1 \leq y \wedge S \begin{pmatrix} x + 1 + z \\ y \\ z \end{pmatrix} \leq s)) \end{aligned}$$

次に Farkas の補題を用いて $\forall$ を除去する。 尚 S_x, S_y, S_z はそれぞれ S の 1 列目、2 列目、3 列目を表す。

$$\begin{aligned} &\exists S \exists s : \\ &(\exists x \exists y \exists z \quad : y \geq z \wedge S \begin{pmatrix} x \\ y \\ z \end{pmatrix} \leq s) \wedge \\ &(\exists \lambda \quad : \lambda \geq 0 \wedge \lambda S = \begin{pmatrix} 1 & -1 & 0 \end{pmatrix} \wedge \lambda s \leq -1) \wedge \\ &(\exists \Lambda \quad : \Lambda \geq 0 \wedge \Lambda S = \begin{pmatrix} S_x & S_y & S_z + S_x \end{pmatrix} \wedge \Lambda s \leq (s - S_x)) \end{aligned} \tag{14}$$

(14) にも非線形な制約が存在しているため、不変式生成で用いた技術を同様に適用すると、結果として以下の解が得られる。

$$\begin{aligned} x &= -2 \\ y &= -1 \\ z &= -1 \\ \lambda &= \begin{pmatrix} 1 & 0 \end{pmatrix} \\ \Lambda &= \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \\ p &= \begin{pmatrix} 1 & -1 & 0 \end{pmatrix} \\ p_0 &= -1 \\ q &= \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \\ q_0 &= -1 \end{aligned}$$

上記の解によって以下の再帰集合が得られる。

$$x - y \leq -1 \wedge z \leq -1$$

またプログラムが終了しない自由変数の割当が $x = -2, y = -1, z = -1$ であることもわかる。

6 線形アサーション合成のための制約ベースのアルゴリズムの、線形算術関数と未解釈関数の組み合わせを処理するために拡張する方法

本章では例えば複雑すぎる関数に代表されるような未解釈関数を線形演算と組み合わせて補助アサーションを生成するための手法を示す。この拡張の基礎は論理理論の組み合わせによる階層的なアプローチである。以下小規模な例を用いて線形演算と関数シンボルの内挿アルゴリズムを示す。

互いに満たされることのない 2 つのアサーション φ と ψ について内挿アルゴリズムを考える。

$$\begin{aligned} \varphi &= (x \leq a \wedge a \leq y \wedge f(a) \leq 0) \\ \psi &= (y \leq b \wedge b \leq x \wedge 1 \leq f(b)) \end{aligned}$$

充足不可能性の証明には、線形演算の関数 (LI, linear arithmetic function) と未解釈の関数 (UIF, uninterpreted function) についての推論が必要であり、それらは論理的帰結関係 $\vDash_{LI+UIF}$ によって示される。

$$\varphi \wedge \psi \vDash_{LI+UIF} \bot$$

この内挿アルゴリズムの目的は次のようなアサーション $\mathcal{X}$ を求めることになる。

$$\begin{aligned} &\varphi \vDash_{LI+UIF} \mathcal{X} \\ &\mathcal{X} \wedge \psi \vDash_{LI+UIF} \bot \\ &\mathcal{X} \textit{is expressed over common symbols of } \varphi \textit{ and } \psi \end{aligned} \tag{15}$$

まず以下のように関数の適用から算術制約を分離する生成ステップを行う。

$$\begin{aligned} \varphi_{LI} &= (x \leq a \wedge a \leq y \wedge c \leq 0) \\ \psi_{LI} &= (y \leq b \wedge b \leq x \wedge 1 \leq d) \\ D &= \{c \mapsto f(a), d \mapsto f(b)\} \\ X &= \{a = b \rightarrow c = d\} \end{aligned}$$

不等式 φ と ψ においては、関数シンボルを自由変数で置き換えており、その置換を行うために射の集合である D を用いている。集合 X は、関数適用の結果に得られるすべてのペアに対して作成された機能公理インスタンス (functionality axiom instances) を示している。これらの例は線形演算で表現することができる。尚この例では、このようなインスタンスは 1 つのみとなっている。

階層的な推論アプローチは、以下の式で示されるように、 X に収集されたインスタンスがアサーション φ_{LI} と ψ_{LI} の相互充足不能性について証明するのに充足していることを保証している。

$$\varphi_{LI} \wedge \psi_{LI} \wedge \bigwedge X \vDash_{LI} \bot$$

X の公理インスタンスには φ_{LI} と ψ_{LI} の両方にある自由変数が含まれているため、(15) の 3 行目の条件に違反するような内挿の結果を得てしまう。 (Unfortunately we cannot apply an algorithm for interpolation in linear arithmetic on the unsatisfiable conjunction presented above since the axiom instance in X contains variables that appear both in φ_{LI} and ψ_{LI}, which will lead to an interpolation result that violates the third condition in (15).)

代わりに case-based な推論を行う。まず純粋な (X を考慮しない) 内挿について考えることで内挿を計算することを試みる。しかしこれについては以下のように相互充足性を持ってしまうために成功しない。

$$\varphi_{LI} \wedge \psi_{LI} \nvDash \bot$$

純粋なアサーションの連言は、 X からの機能公理インスタンスを適用するための前提条件を求められる。

$$\varphi_{LI} \wedge \psi_{LI} \vDash a = b$$

これによって φ_{LI} と ψ_{LI} の自由変数上で表現できる中間項を導き出すことができる。

$$\begin{aligned} \varphi_{LI} \wedge \psi_{LI} &\vDash a \leq y \wedge y \leq b \\ \varphi_{LI} \wedge \psi_{LI} &\vDash a \geq x \wedge x \geq b \end{aligned}$$

これらの結果を入れ替えることで次の結論を導き出すことができる。

$$\begin{aligned} \varphi_{LI} \wedge \psi_{LI} &\models x \leq a \wedge a \leq y \\ \varphi_{LI} \wedge \psi_{LI} &\models y \leq b \wedge b \leq x \end{aligned}$$

上式は内挿アルゴリズムによって適切な case の推論を導き出すために用いられる。更に対応する新しい自由変数 e とともに追加の関数適用 $f(y)$ を導入し、集合 D に追加する。

$$D = \{c \mapsto f(a), d \mapsto f(b), e \mapsto f(y)\}$$

case の推論を行う第一段階として、以下の充足不可能な連言を計算する。

$$(\varphi_{LI} \wedge a = e) \wedge (\psi_{LI} \wedge e = b) \models_{LI} \bot$$

3 で用いたアルゴリズムを適用することで、以下の式のような部分内挿 $e \leq 0$ を得ることができる。

$$\begin{aligned} \varphi_{LI} \wedge a = e &\models_{LI} e \leq 0 \\ e \leq 0 \wedge \psi_{LI} \wedge e = b &\models_{LI} \bot \end{aligned}$$

さらに以下のように case 推論の情報を使って部分内挿を完成させることができる。

$$\mathcal{X}_{LI} = (x \neq y \vee (x = y \wedge e \leq 0))$$

対応する関数適用を新たな自由変数に置き換えた後、元の入力である φ と ψ の内挿 $\mathcal{X}$ を得る。

$$\begin{aligned} \mathcal{X} &= (x \neq y \vee (x = y \wedge e \leq 0))[f(q)/e] \\ &= x \neq y \vee (x = y \wedge f(q) \leq 0) \end{aligned}$$