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、帰納的不変量 (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、不変量、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。

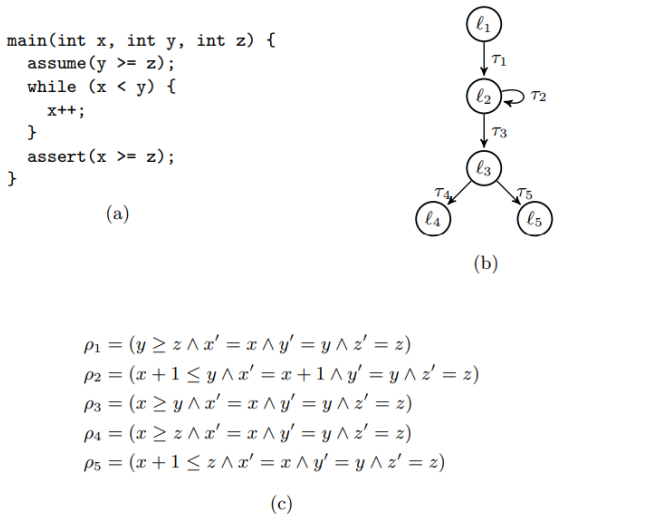

Fig. 1. An example program (a), its control-flow graph (b), and the corresponding transition relations (c).

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 $f_x, f_y, \delta_0, \delta$ にある満足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、ある満足した 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。

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

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

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。

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

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

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。

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

$i_x x + i_y y + i_z z \le i_0$ 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。

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

?? 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。

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

ただし λ と μ は線型結合を表しており、満足しない不等式 $0 \le -1$ 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満足することがなく、以下の制約を満足する内挿 $iv \le i_0$ を計算する。

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

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。

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

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 線形不変量生成 (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

不変量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、不変料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、不変量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。更に不変量と境界生成の密接な関わりについても簡潔に示す。

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

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。まず未知の不変量係数で (1-3) の制約を表すと以下のようになる。

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

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

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