卒業研究

所有権型を利用した CHCベースのプログラム検証

小林研究室 松下祐介

概要

- **所有権型**と**CHC**の性質を活かした**新しい検証手法**
  - **CHCベース検証: プログラム**を**論理**に帰着
    - **CHC** = **C**onstrained **H**orn **C**lause 制約付きホーン節
- **所有権型**: **安全**な**メモリ管理**
- **提案手法**は**ポインタ**の表現に特徴
- **実験**で**既存手法**を**上回る性能**を発揮
- **形式化**と**正当性**の予想

目次

背景 CHCベース検証

let rec fact n = **if** n = 0 **then** 1 **else** fact (n - 1)
let main n = **let** r = fact n **in assert** (r ≧ n)

プログラム

↓ 帰着

入力 出力

$$\textbf{fact}(n, r) \Longleftarrow n = 0 \wedge r = 1$$

$$\textbf{fact}(n, r) \Longleftarrow n \neq 0 \wedge \textbf{fact}(n-1, r') \wedge r = n \cdot r'$$

$$r \geq n \Longleftarrow \textbf{fact}(n, r)$$

全称量化

CHC

「fact n が r を**返す**とき、**必ず** r ≧ n」 ⇔ 「**CHC**が**充足可能**」

CHC充足問題にはGPDR [Hoder&Bjørner, 2012] 等の**アルゴリズム**

背景 CHCベース検証×ポインタ

```
let take_max rx ry = if !rx ≧ !ry then rx else ry
let inc_max x y = let rx = ref x in let ry = ref y in
        let rz = take_max rx ry in rz := !rz + 1; (!rx, !ry)
```

↓ 帰着

背景 既存手法でのポインタ

```
let take_max rx ry = if !rx ≧ !ry then rx else ry
let inc_max x y = let rx = ref x in let ry = ref y in
      let rz = take_max rx ry in rz := !rz + 1; (!rx, !ry)
```

↓ 既存手法による帰着

$$\textbf{take-max}(rx, ry, r, a) \iff a[rx] \geq a[ry] \land r = rx$$

$$\textbf{take-max}(rx, ry, r, a) \iff a[rx] < a[ry] \land r = ry$$

一般には与えるのが難しい

恣意的なアドレス

$$\begin{aligned}\textbf{inc-max}(x, y, r) \iff\ & a = a^{\circ}\{0 \leftarrow x\}\{1 \leftarrow y\} \\ & \land\ \textbf{take-max}(0, 1, rz, a) \\ & \land\ a' = a\{rz \leftarrow a[rz] + 1\} \\ & \land\ r = a'[0] \cdot a'[1]\end{aligned}$$

背景 所有権の借用と返却

本研究では **Rust** での**所有権型**をベースに議論

```
let mut x = 5;
{
    let mx = &mut x;
    *mx += 10;
}
x == 15
```

可変参照

x		5
	所有権の借用	
x	**mx**	5
x	**mx**	15
	所有権の返却	
x		15

1つのリソースに対して**所有権**は**1つのエイリアス**のみ持てる

↑ **型システム**で保証

提案手法の概要

ポインタの表現に特徴

既存手法

i

アドレス

a

配列

アドレス ↦ 値
メモリの表現

ポインタ解析や配列理論が必要
状態の分離性が低い
→検証性能の悪化

提案手法

$\langle x, x_* \rangle$

現在の**値**　**未来**の**返却時**の**値**

CHCベース **所有権型**

実験で**既存手法**を上回る**検証性能**を発揮

提案手法の基本例

提案手法
による
帰着

```
P1 let mut x = 5;
   {
P2     let mx = &mut x;
P3     *mx += 10;
P4 }
P5 x == 15
```

$$P_1(r) \impliedby P_2(5, r)$$

未来の返却時の値

$$P_2(x, r) \impliedby P_3(x_*, \langle x, x_* \rangle, r)$$
$$P_3(x_*, \langle x, x_* \rangle, r) \impliedby P_4(x_*, \langle x+10, x_* \rangle, r)$$
$$P_4(x_*, \langle x, x_* \rangle, r) \impliedby x_* = x \wedge P_5(x_*, r)$$
$$P_5(x, r) \impliedby r = x == 15$$

返却による値の確定

提案手法の発展例 1/2

```
fn take_max ( mx: &mut i32, my: &mut i32 ) → &mut i32 {
    if ( *mx ≧ *my ) { mx } else { my }
}
```

↓ 提案手法による帰着

捨てる可変参照の返却時の値を確定

take-max$(\langle x, x_* \rangle, \langle y, y_* \rangle, r)$

$$\Longleftarrow \quad x \geq y \ \wedge \ y_* = y \ \wedge \ r = \langle x, x_* \rangle$$

take-max$(\langle x, x_* \rangle, \langle y, y_* \rangle, r)$

$$\Longleftarrow \quad x < y \ \wedge \ x_* = x \ \wedge \ r = \langle y, y_* \rangle$$

提案手法の発展例 2/2

```
fn inc_max ( mut x: i32, mut y: i32 ) → (i32, i32) {
    {
        let mz = take_max ( &mut x, &mut y ); *mz += 1;
    }
    (x, y)
}
```

↓ 提案手法による帰着

inc-max(x, y, r)

未来の返却時の値

$\Longleftarrow$ **take-max**$(\langle x, x_* \rangle, \langle y, y_* \rangle, \langle z, z_* \rangle)$

$\wedge\ z_* = z + 1\ \wedge\ r = x_* \cdot y_*$

形式化と正当性の予想

- **所有権型**を用いる**対象言語 (Rust)** を**形式化**
  - RustBelt [Jung+, 2018] の λ_{Rust} を大幅に**整理・単純化**
  - スタックとヒープによる簡潔な**操作的意味論**
- **CHCへの変換**を**形式的**に記述
- **生成されるCHC**の**正当性**について**予想**(部分的証明)
  - **操作的意味論**から**各関数**に対応する**自然な述語**を**構成**
    - **可変参照**の扱いが技術的に難しい
  - **生成したCHC**の与える述語 と **自然**な述語 の**等価性**
    - **CHC** ≧ **自然** の側は容易、**CHC** ≦ **自然** の側が**未解決**

実験概要

- **単連結リスト**を**可変参照**で更新する**再帰関数**を交えた **Rust**プログラムに関して11の**検証問題**を用意
- **提案手法**の**CHC** + **Spacer** **CHCソルバ** [Komuravelli+, 2013]

  vs. **SeaHorn 検証システム** [Gurfinkel+, 2015]
  - **SeaHorn**: 対象は C/C++、**ポインタ**の扱いに**既存手法**
  - **提案手法**の**CHC**: **Rust** プログラムをもとに手で書いた
  - **SeaHorn**への入力: **Rust** を手で**C**に書き換えたもの
- まだ**本格的な実験ではない**が、十分な**差**が見られた

実験結果

提案手法が全問題で SeaHorn に勝る

問題	提案手法	SeaHorn
calc-1	**0.03s**	1.55
calc-2	**0.07**	timeout
calc-3	**0.15**	timeout
calc-4	**0.27**	timeout
calc-5	**0.62**	timeout
back	**0.14**	timeout
find	**0.40**	timeout
size	**0.05**	timeout
single	**0.24**	timeout
double-1	**0.70**	timeout
double-2	**1.03**	timeout

関連研究

- **所有権型**なしに**ポインタ**を扱う**CHC**ベース検証
  - **JayHorn** [Kahsai+, 2016]：**Java Bytecode** を対象
  - **SeaHorn** [Gurfinkel+, 2015]：**C/C++→LLVM IR** を対象
- **所有権型**を利用する**半自動検証**
  - **Rust2Viper** [Hahn, 2016]：“**許可**”を扱う**中間言語**への帰着
  - **Electrolysis** [Ullrich, 2016]：**Rust** の不完全な**純粋関数化**
- **Rust** の**形式化**
  - **Patina** [Reed, 2015]：**右辺値**等も扱うが、**不完全**な形式化
  - **λ_{Rust}** [Jung+, 2018]：**Unsafe** な**ライブラリ**も扱える

結論

- 所有権型とCHCの性質を活かした新しい検証手法
  - 可変参照→現在の値と未来の返却時の値の組
  - 実験で既存手法を上回る性能を発揮
  - 形式化と正当性の予想
- 今後の課題
  - 正当性の完全な証明
  - Rustのための検証ツールの開発・本格的な実験