所有権型を利用した CHC ベースのプログラム検証

松下 祐介　　小林 直樹　　塚田 武志

（東京大学理学部情報科学科）

概要

- 所有権型とCHCの性質を活かした新しいプログラム検証手法を提案
  - ポインタの表現に特徴
  - 実験で既存手法を上回る検証性能を確認

背景 CHCベース検証

```
fn fact(n: i32) {
  if n == 0 { 1 } else { n * fact(n - 1) }
}
fn test(n: i32) {
  let r = fact(n); assert!(r ≥ n);
}
```

プログラム

帰着 →

$$\mathbf{fact}(n, r) \Longleftarrow n = 0 \ \wedge \ r = 1$$
$$\mathbf{fact}(n, r) \Longleftarrow n \neq 0 \ \wedge \ \mathbf{fact}(n-1, r') \ \wedge \ r = n \cdot r'$$
（入力 出力：n, r；関数呼び出し：$\mathbf{fact}(n-1, r')$）
$$r \geq n \Longleftarrow \mathbf{fact}(n, r)$$

CHC

- プログラムから論理への帰着による検証
- CHC充足問題には GPDR [Hoder&Bjørner, 2012] などの効率の良いアルゴリズム

背景 所有権型

```
let x : i32 = 5;
{
  let mx : &mut i32 = &mut x;
  *mx += 10;
}
assert!(x == 15)
```

可変参照　可変借用

x → 所有権の借用 → mx
x ← mx
x ← 所有権の返却 ← mx

Rust

- 1つのエイリアスだけが完全な所有権を持てる
  - 独特の型システムによって保証される
  - メモリ操作の安全性をもたらす
- 近年 Rust というプログラミング言語が採用
  - C/C++のような高速なメモリ操作と所有権による安全性
  - ライフタイムを用いた柔軟な所有権型システム
  - さらに内部可変性を用いた機能拡張
  - Redux OS や Servo ブラウザなどの大規模な実用例

既存手法

px → i　　a

ポインタ　アドレス　　配列（アドレス ↦ 値）

- メモリの状態が配列にまとめられる
  - 状態の分離性が低い
  - ポインタ解析 & 配列理論が必要

提案手法

mx → $\langle x, x_* \rangle$

可変参照　現在の値 未来の返却時の値

- アドレスを考える必要がない
- メモリ状態が分離されている

↓

よりよい検証性能を発揮

```
fn take_max<'a>
(mx: &'a mut i32, my: &'a mut i32)
→ &'a mut i32 {
  if *mx ≥ *my { mx }
  else { my }
}
fn inc_max(mut x: i32, mut y: i32)
→ (i32, i32) {
  {
    let mx = &mut x; let my = &mut y;
    let mz = take_max(mx, my); *mz += 1;
  }
  (x, y)
}
```

→

$$\mathbf{take\text{-}max}(\langle x, x_* \rangle, \langle y, y_* \rangle, r) \Longleftarrow x \geq y \ \wedge \ y_* = y \ \wedge \ r = \langle x, x_* \rangle$$
$$\mathbf{take\text{-}max}(\langle x, x_* \rangle, \langle y, y_* \rangle, r) \Longleftarrow x < y \ \wedge \ x_* = x \ \wedge \ r = \langle y, y_* \rangle$$
$$\mathbf{inc\text{-}max}(x, y, r) \Longleftarrow \mathbf{take\text{-}max}(\langle x, x_* \rangle, \langle y, y_* \rangle, \langle z, z_* \rangle) \ \wedge \ z_* = z + 1 \ \wedge \ r = (x_*, y_*)$$

基本的な変換例

高度な変換例

```
enum List<T> { Nil, Cons(T, Box<List<T>>) }
fn split_mut_list<'a>(mlx: &'a mut List<i32>) → List<&'a mut i32> {
  match mlx {
    List::Nil ⇒ List::Nil,
    List::Cons(mx, mlx2) ⇒ List::Cons(mx, box split_mut_list(mlx2))
  }
}
fn sort_list<'a>(lmx: List<&'a mut i32>) → List<&'a mut i32> {
  …
}
fn carve_list<'a>(i: i32, lmx: List<&'a mut i32>) {
  match lmx {
    List::Nil ⇒ {}
    List::Cons(mx, box lmx2) ⇒ { *mx -= i; carve_list(i+1, lmx2) }
  }
}
fn sort_carve_list(mut lx: List<i32>) → List<i32> {
  carve_list(0, sort_list(split_mut_list(&mut lx)));
  lx
}
// 例）sort_carve_list(list![3,5,1]) == list![2,3,1]
// リストの i 番目（0-indexed）に小さい要素を i 減らす関数
// 各要素への可変参照からなるリストを作って値でソートし、それを利用して値を更新
```

提案手法 →

$$\mathbf{split\text{-}mut\text{-}list}(\langle [], lx_* \rangle, lmr) \Longleftarrow lx_* = [] \ \wedge \ lmr = []$$
$$\mathbf{split\text{-}mut\text{-}list}(\langle [x \mid lx'], lx_* \rangle, lmr) \Longleftarrow lx_* = [x_* \mid lx'_*] \ \wedge \ \mathbf{split\text{-}mut\text{-}list}(\langle lx', lx'_* \rangle, lmr') \ \wedge \ lmr = [\langle x, x_* \rangle \mid lmr']$$
$$\mathbf{sort\text{-}list}(lmx, lmr) \Longleftarrow \cdots$$
$$\mathbf{carve\text{-}list}(i, []) \Longleftarrow \top$$
$$\mathbf{carve\text{-}list}(i, [\langle x, x_* \rangle \mid lmx']) \Longleftarrow x_* = x - i \ \wedge \ \mathbf{carve\text{-}list}(i+1, lmx')$$
$$\mathbf{sort\text{-}carve\text{-}list}(lx, lr) \Longleftarrow \mathbf{split\text{-}mut\text{-}list}(\langle lx, lx_* \rangle, lmr) \ \wedge \ \mathbf{sort\text{-}list}(lmr, lmr') \ \wedge \ \mathbf{carve\text{-}list}(0, lmr') \ \wedge \ lr = lx_*$$

実験補足

calc-2	$\mathbf{sort\text{-}carve\text{-}list}([3,1], [2,1])$
find	$\exists x, y \,.\, \mathbf{sort\text{-}carve\text{-}list}([x,3,4], [y,3,2])$
double-2	$\forall x_1, x_2, y_1, y_2 \,.\, x_1 > x_2 \ \wedge \ \mathbf{sort\text{-}carve\text{-}list}([x_1, x_2], [y_1, y_2]) \implies x_2 = y_2$

実験

- このRustプログラムについて、11の検証問題を用意
- 提案手法のCHC + Spacer CHCソルバ [Komuravelli+, 2013] vs. SeaHorn [Gurfinkel+, 2015]
  - SeaHorn: 対象はC/C++、前述の既存手法を利用
  - 提案手法のCHC: 手で与えた
  - SeaHornへの入力: Rustコードを手でCに書き換え
- 提案手法が全問題で SeaHorn に優る
  - まだ本格的な実験ではないが十分な差

問題	提案手法	SeaHorn
calc-1	0.03s	1.55s
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

これまでの成果

- 実験で 提案手法 + Spacer CHCソルバ が SeaHorn を上回る検証性能を発揮することを確認
- Rust の基本サブセットを簡潔に形式化（λ_{Rust} [Jung+,2018] の整理・単純化）
- 提案手法を形式言語からCHCへの変換として形式的に記述し、変換の正当性を部分的に証明

→

これからの目標

- Rustのための実用的な検証器を目指す
  - 内部可変性は既存手法との連携で対処
  - 抽象解釈などの手法も組み合わせる