ポインタ演算と多次元配列のための所有権篩型システム

藤原　佑輔・松下　祐介・末永　幸平・五十嵐　淳
京都大学情報学研究科

目標: 命令型言語の検証器ConSORT [Toman+, '20] [Tanaka+, '24]の多次元配列拡張

多次元配列版ConSORTの検証例

エイリアシングと破壊的代入を含むプログラムの検証器

- 値の情報を持つ篩型と参照のアクセス権限を表す所有権を組み合わせて検証
- 整数配列の検証は既存研究で可能

参照型 $\Pi x.(\tau\ \mathbf{ref}^{r})$

r:所有権項

- 配列への参照を表す型
- 配列の要素の型τは配列のインデックスxに依存可能

所有権項r

$r \quad ::= \quad \varphi_1 \Rightarrow q_1, \ldots, \varphi_n \Rightarrow q_n, true \Rightarrow q_0$

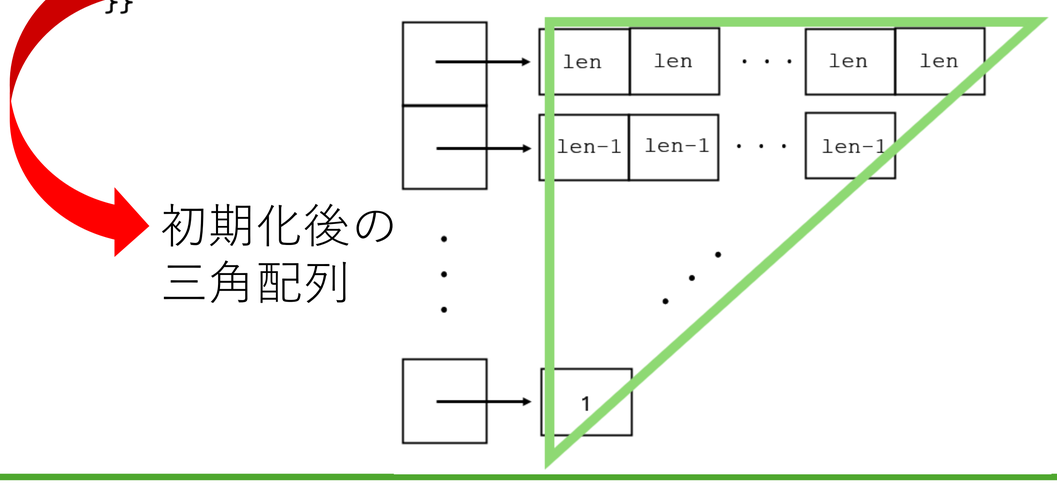
- 配列の各要素の所有権を表し、現在の環境とインデックスでφ_iが満たされるならば所有権はq_i、そうでなければ後続の$\varphi_{i+1} \Rightarrow q_{i+1}$,...により所有権が決定
- φ_i :整数リテラル、整数変数、配列のインデックスを表す変数が現れる量化子のない一階述語論理の命題
- q_iは0以上1以下の有理数
  - q_i =1 : 読み書き可能で、他のエイリアスから読み書き不可
  - 0< q_i <1： 読み出しのみ可能
  - q_i =0：読み書き不可

拡張のポイント

所有権項中のφに配列の外側のインデックスの出現を許容することで、三角配列のような柔軟な型の表現が可能

三角配列を初期化する関数の型付け例

```
// nested_array_init:
// ⟨arr : Πx2.(Πx1.(int ref) ref^{0≤x2≤len−1 ⇒ 1}), len : int⟩ →
// ⟨arr : Πx2.(Πx1.({ν : int | ν = len − x2} ref^{r1}) ref^{r2}), len : int⟩
// r1 = 0 ≤ x1 ≤ len − x2 − 1 ⇒ 1, r2 = 0 ≤ x2 ≤ len − 1 ⇒ 1
nested_array_init(arr, len){
  if len > 0 then {
    let sub_arr = alloc len : int ref in
    int_array_init(sub_arr, len, len);
    // Fill len elements of sub_arr with len
    arr := sub_arr;
    // arr : Πx2.(Πx1.({ν : int | x2 = 0 ⟹ ν = len} ref^{r1}) ref^{r2})
    // r1 = (x2 = 0 ∧ 0 ≤ x1 ≤ len − 1 ⇒ 1), r2 = 0 ≤ x2 ≤ len − 1 ⇒ 1
    assert(**arr = len); let next_arr = arr + 1 in
    // arr : Πx2.(Πx1.({ν = len} ref^{r1}) ref^{r2})
    // r1 = 0 ≤ x1 ≤ len − 1 ⇒ 1, r2 = (x2 = 0 ⇒ 1)
    // next_arr : Πx2.(Πx1.(int ref) ref^{0≤x2≤len−2 ⇒ 1})
    nested_array_init(next_arr, len-1);
    // next_arr : Πx2.(Πx1.({ν : int | ν = len − x2 − 1} ref^{r1}) ref^{r2})
    // r1 = 0 ≤ x1 ≤ len − x2 − 2 ⇒ 1, r2 = 0 ≤ x2 ≤ len − 2 ⇒ 1
    alias(next_arr = arr + 1);
    // ⟨arr : Πx2.(Πx1.({ν : int | ν = len − x2} ref^{r1}) ref^{r2}), len : int⟩
    // r1 = 0 ≤ x1 ≤ len − x2 − 1 ⇒ 1, r2 = 0 ≤ x2 ≤ len − 1 ⇒ 1
  }}
```



初期化後の三角配列

性能評価実験(GitHub repository)

検証器を実装し、多次元配列を含む19のベンチマークの検証にかかる時間を測定(以下抜粋)

Name	Total time (ownership+refinement)
Indexed-Matrix(2D)	36.895 (0.433+36.462)
Indexed-Matrix(3D)	17.232 (13.311+3.920)
Indexed-Matrix(4D)	350.784 (114.746+236.037)
Add-Matrix	208.473 (191.750+16.722)
Trace-Matrix	54.533 (10.845+43.688)
Trans-Matrix	163.251 (79.337+83.913)
Eta-Equ-Trace	18.331 (16.563+1.767)
Compare-Element	16.550 (2.110+14.439)
Share-Add-Matrix	376.867 (29.328+347.538)

最も長い検証例でも6分程度であり、現実的な時間で検証可能

検証の高速化のための工夫

- 所有権項の形式を限定することで型推論時の制約を単純化
- 型$\Pi x.(\tau\ \mathsf{ref}^{\mathsf{r}})$に対してrは以下のいずれか
  1. $x = 0 \Rightarrow o_1, x \in [1, h] \Rightarrow o_2$
  2. $y = 0 \wedge x \in [l_1, h_1] \Rightarrow \mathrm{o}_1, \mathrm{y} \in [1, h_2] \wedge x \in [l_3, h_3] \Rightarrow o_2$
  3. $x \in [l, h] \Rightarrow o$
- *1*と*2*は配列の先頭のみ所有権を特別扱いする層とその内側の所有権項
- *3*は*1,2*より表現力は低いが、制約の量を減少に寄与
- 形式を限定した所有権項でも非自明な内容の検証が行えることを左記の実験で確認

検証器verusとの比較

- verusを用いると、すべて2秒未満で検証可能
- verusの場合、検証を通過させるために多くの型注釈をプログラマに要求
  - 2DのIndexed-Matrixの場合、本検証器では4行/40行の型注釈に対し、verusでは15行/54行の型注釈

現在の状況

- プログラムからownership termと型の条件に関わる制約を生成する型推論アルゴリズム を構築済
- 多次元配列を検証可能な型検証器を実装済
- プログラムに型がつくならばassert文が失敗せず、配列の範囲外アクセスやnullへのアクセスが存在しないことの証明済
- 所有権項の他のデータ構造への応用を考案中
- 理論的には必要のない型注釈を検証時に要する場合があり、少ない型注釈で検証できる方法を検討中