Extensible Functional-Correctness Verification of Rust Programs by the Technique of Prophecy

預言の技術による Rust プログラムの拡張可能な機能正当性検証

修士論文

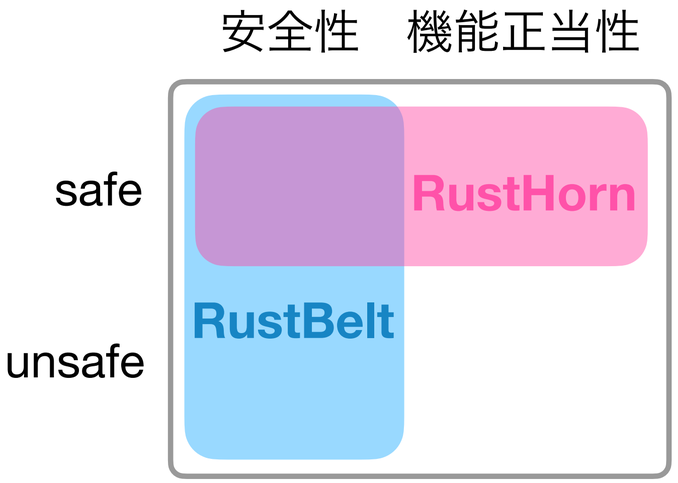
2021年2月1日

情報理工学系研究科コンピュータ科学専攻 小林研究室 松下祐介

背景

- **Rust**: C/C++に代わる言語、所有権に基づく型システムで資源の使用を検査
  - unsafe (= その検査を受けない) コードを一部使って表現力を拡張

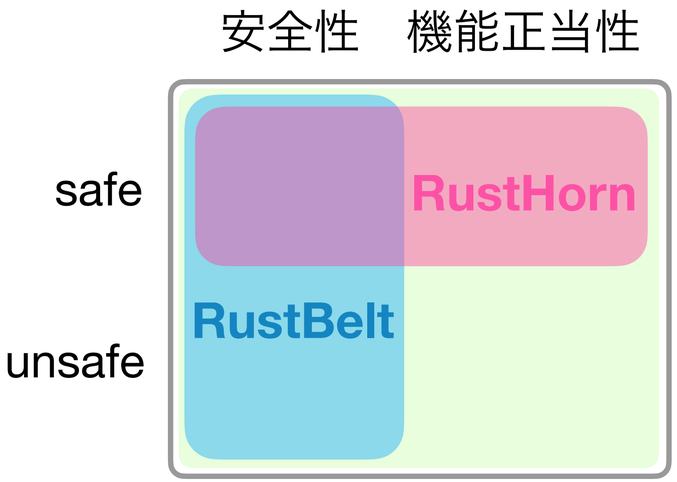
- Rust のプログラム検証の現状
  - **RustBelt** [Jung+ 2018]: Rust の型システムと unsafe コードを持つライブラリの安全性を Coq で検証
  - **RustHorn** [松下+ 2020]: Rust の型の保証を利用、safe な Rust プログラムの機能正当性を自動検証



背景

- **Rust**: C/C++に代わる言語、所有権に基づく型システムで資源の使用を検査
  - unsafe (= その検査を受けない) コードを一部使って表現力を拡張

- Rust のプログラム検証の現状
  - **RustBelt** [Jung+ 2018]: Rust の型システムと unsafe コードを持つライブラリの安全性を Coq で検証
  - **RustHorn** [松下+ 2020]: Rust の型の保証を利用、safe な Rust プログラムの機能正当性を自動検証



全体を扱える強力な
検証体系を作りたい

本研究

- Rust プログラム (unsafe コードを持つ) の機能正当性を型の保証を利用して柔軟に検証できる論理的基盤を築いた
  - Rust プログラムの強力な検証体系のための重要な一歩
  - **拡張可能** (≈ 新しい機能・ライブラリを自由に追加できる) な検証を実現
    - 意味論的型というアイデア、RustBelt の拡張
  - **預言** (≈ 未来の情報を先取りする) の技術を利用
    - Rust の専有参照の表現に使われる、RustHorn の拡張

目次

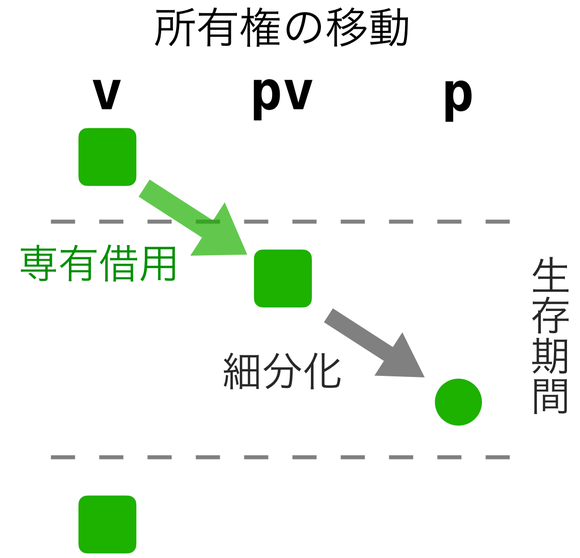
Rust の型システムの概要

- **所有権**: オブジェクトの更新に必要、同時に共有できない
- **専有借用**: あるエイリアスが持つオブジェクトへの所有権を
  事前に決めたある期間 (生存期間) 新しく作ったポインタ (**専有参照**) に貸す
  - 所有権の返却時にコミュニケーションを取らなくて良い

```
let mut v: Vec<int> = vec![0, 1, 2];
let pv: &mut Vec<int> = &mut v;
let p: &mut int = pv.index_mut(0); *p += 7;
print!("{:?}", v);  // [7, 1, 2]
```

専有参照

専有借用



目次

Rust と検証の拡張可能性

- Rust の表現力は unsafe な内部実装 & 所有権に基づくインターフェースを持つライブラリにより柔軟かつ堅牢に拡張できる
  - 例: 配列型 `Vec` への専有参照から要素への専有参照を得るメソッド
    ```
    fn index_mut(&mut Vec<int>, uint) -> &mut int
    ```
- 検証が**拡張可能**である ＝ 新たな機能・ライブラリを追加するときに、それ単体を個別に検証することで、検証済みの既存の機能・ライブラリとの任意の組合せで期待される性質 (安全性・機能正当性) が得られる
  - Rust の検証では unsafe コードに関する拡張可能性が特に重要

意味論的型による Rust の拡張可能な検証

- **意味論的型**: プログラムの型の保証を述語としてモデル化したもの
- 本研究では Rust に対し分離論理 Iris 上で意味論的型を与えることで、型の保証を利用した Rust の機能正当性検証を**拡張可能**な形で実現
  - RustBelt [Jung+ 2018] が Rust の安全性検証に用いた手法を拡張
  - 各機能・ライブラリのための検証規則を個別に Iris 上で導出できる
  - **Iris** [Jung+ 2015]: 非常に表現力の高い汎用の分離論理
    - **分離論理**は資源を扱う論理、Rust の所有権の表現に重要
    - 生存期間に基づく借用を扱ううえで、Iris の表現力を活用

値表現による Rust の機能正当性検証

- Rust プログラムの機能正当性の検証を効率よく行うためには、
  メモリやアドレスを直接表現するのを避けたい

- Rust は所有権に基づく型の保証が強いので、
  多くの場合 Rust のオブジェクトは純粋な値として表現できる
  → 関数型言語と同様の効率のよい機能正当性検証

  - 例: 整数の動的配列 `Vec<int>` は要素の整数のリスト $\mathrm{List}\,\mathbb{Z}$、
    整数への箱ポインタ `Box<int>` は参照先の整数 $\mathbb{Z}$

預言による Rust の専有参照の表現

- 専有参照 `&mut T` を単に参照先の現在の値として表現すると、
  借用終了時に専有参照による更新の結果を借用元に伝えられない

- 本研究では、**専有参照**を現在の値 v と借用終了時の値 $v_{\text{。}}$ の組として表現
  - 専有借用開始時に $v_{\text{。}}$ を**預言** (≈ 未来の情報を先取り)
  - 専有参照 $(v, v_{\text{。}})$ を解放する時に預言 $v_{\text{。}}$ の値を v に確定させる
  - RustHorn [松下+ 2020] が提案、専有参照への様々な操作を表現できる

預言の新しい定式化

- 預言は様々な文脈で使われてきた ([Abadi & Lamport 1988] 等) が、
  Rust の表現には特に柔軟な預言の扱いが必要 (例: 専有参照への専有参照)

- 本研究では分離論理 Iris 上で預言の新しい定式化を与えた
  - 預言に関する可能性の集合を絞り込んでいく、預言に関する情報は分けて持つ

預言の定式化の補題群 (参考)

新しい預言変数の取得

$$\frac{\exists \_ : T.\ \mathsf{True}}{\mathsf{True} \Rrightarrow_{\varnothing} \exists x\ \text{s.t.}\ x.\mathsf{type} = T.\ [x]_1}$$

預言変数の値の確定・預言観測の取得

$$\frac{\mathrm{Dep}(\hat{v}, Y)}{[x]_1 * [Y]_q \Rrightarrow_{\mathcal{N}_{\mathsf{proph}}} \langle \pi.\ \pi\, x = \hat{v}\, \pi \rangle * [Y]_q}$$

預言観測の合成

$$\frac{\langle \pi.\ \phi_\pi \rangle \qquad \langle \pi.\ \psi_\pi \rangle}{\langle \pi.\ \phi_\pi \wedge \psi_\pi \rangle}$$

預言観測の弱化

$$\frac{\forall\, \pi.\ \phi_\pi \Rightarrow \psi_\pi \qquad \langle \pi.\ \phi_\pi \rangle}{\langle \pi.\ \psi_\pi \rangle}$$

預言観測の利用

$$\frac{\langle \pi.\ \hat{\phi}\, \pi \rangle}{\exists\, \pi_*.\ \hat{\phi}\, \pi_*}$$

目次

- 準備
- **本研究**: 基本アイデア / **実際の検証**
- 関連研究
- 結論

型の保証を利用した検証の形式化

- Rust の型システムに機能正当性の情報を加えた篩型システムを構築
  - 篩型判断: 型判断 + 式の挙動を表す述語変換子 (事後条件 ↦ 事前条件)
  - 篩型判断は意味論的型を利用して Iris 上の述語としてモデル化される
    → 各機能・ライブラリのための検証規則を個別に Iris 上で導出できる

篩型判断　$\alpha \mid \Delta; \overrightarrow{\mathfrak{a} :^{act} \tau'} \vdash e : \tau \mid \Delta'; \overrightarrow{\mathfrak{b} :^{act'} \tau''} \mid pre$

述語変換子
入力オブジェクト　式　結果の型　出力オブジェクト

対象言語

$$e : \mathrm{Expr} ::= a \mid \mathfrak{a} \mid e(\vec{e'}) \mid \mathrm{alloc}\, e \mid \mathrm{free}\, e \mid *e \mid e \leftarrow e' \mid e.e'$$
$$\mid \mathrm{case}\, e \,\mathrm{of}\, \{0 \to e', 1 \to e''\} \mid e \parallel e' \mid e\ iop\ e' \mid e\ irel\ e' \mid \mathrm{ndint}$$

導出される検証規則の例 (基本操作)

整数加算

事後条件　入力　　論理式 (ある種のCPS変換)

$$\mathfrak{a}\colon \mathsf{int},\ \mathfrak{b}\colon \mathsf{int} \vdash \mathfrak{a} + \mathfrak{b} : \mathsf{int} \mid\mid \lambda post, (m, n).\ post\,(m + n)$$

メモリ割り当て

$$\vdash \mathsf{alloc}\, n :\ \mathsf{box}\, \lightning_n \mid\mid \lambda post, ().\ post\,(())$$

let 束縛

$$\frac{\alpha \mid \Delta; \Gamma \vdash e\colon \tau \mid \Delta'; \Gamma' \mid pre \qquad \forall\, \mathfrak{a}.\ \alpha \mid \Delta'; \mathfrak{a}\colon \tau, \Gamma' \vdash e'[\mathfrak{a}/a]\colon \tau' \mid \Delta''; \Gamma'' \mid pre'}{\alpha \mid \Delta; \Gamma \vdash \mathsf{let}\, a = e\ \mathsf{in}\ e' :\ \tau' \mid \Delta''; \Gamma'' \mid pre \circ pre'}$$

配列への共有参照からの要素アクセス (unsafe コード)

$$\alpha \mid \mathfrak{a}\colon \&^{\alpha}_{\mathsf{shr}}\, \mathsf{vec}\,\mathsf{int},\ \mathfrak{b}\colon \mathsf{int} \vdash (*\mathfrak{a}).\mathfrak{b} : \&^{\alpha}_{\mathsf{shr}}\,\mathsf{int} \mid\mid \lambda post, (v, i).\ 0 \le i < \mathsf{len}\, v\ \wedge\ post\,(v[i])$$

導出される検証規則の例 (専有参照の操作)

専有借用

$$\mathfrak{a}\colon \mathrm{box}\,\tau \;\vdash\; \mathrm{skip} \mid \mathfrak{a}\colon \&^{\alpha}_{\mathrm{unq}}\,\tau,\ \mathfrak{a}:{}^{\dagger\alpha}\,\mathrm{box}\,\tau \mid \lambda\, post, (v).\ \forall\, v_\circ.\ post\,((v, v_\circ), v_\circ)$$

預言　専有参照　借用元

専有参照の解放

$$\frac{\beta \sqsubseteq \&^{\alpha}_{\mathrm{unq}}\,\tau}{\beta \mid \mathfrak{a}\colon \&^{\alpha}_{\mathrm{unq}}\,\tau \;\vdash\; \mathrm{skip} \mid \lambda\, post, ((v, v_\circ)).\ v_\circ = v \;\Rightarrow\; post\,()}$$

預言を確定

配列への専有借用からの要素アクセス (Vec の `index_mut` に相当、unsafe コード)

$$\alpha \mid \mathfrak{a}\colon \&^{\alpha}_{\mathrm{unq}}\,\mathrm{vec}\,\mathrm{int},\ \mathfrak{b}\colon \mathrm{int} \;\vdash\; (*\mathfrak{a}).\mathfrak{b} : \&^{\alpha}_{\mathrm{unq}}\,\mathrm{int} \mid\mid$$
$$\lambda\, post, ((v, v_\circ), i).\ 0 \le i < \mathrm{len}\, v \ \wedge\ \forall\, n_\circ.\ v_\circ = v\{i \leftarrow n_\circ\} \Rightarrow post\,((v[i], n_\circ))$$

配列の預言 $v_\circ$ を第 i 要素以外で確定

複雑なプログラムの検証

- 単純な検証規則の組合せで複雑なプログラムの挙動も検証できる
  - 健全性は分離論理 Iris の力で一般に保証される

例

リストへの専有参照を分割し、リストの各要素への専有参照のリストを返す関数

$$\begin{aligned}\mathrm{split}^{\tau}_{\mathrm{list}} \;:=\; & \mathrm{fn}\, \mathit{self}\,(a)\, \{\; \mathrm{let}\; b = \mathrm{alloc}\, 3\; \mathrm{in}\;\; \mathrm{case}\; {*a}\; \mathrm{of}\; \{ \\ & \quad 0 \to\;\; b \leftarrow 0;\; b, \\ & \quad 1 \to\;\; \mathrm{let}\; a' = a.1\; \mathrm{in}\; \left(b \leftarrow 1;\; b.2 \leftarrow \mathit{self}\,(*(a'.|\tau|));\; b.1 \leftarrow a'\right);\; b \\ & \}\,\}\end{aligned}$$

に対して、各機能のための単純な検証規則の組合せで、以下の仕様を検証できる

$$\begin{aligned}&\mathrm{split}^{\tau}_{\mathrm{list}} : \forall\, \alpha.\; \mathrm{fn}(\&^{\alpha}_{\mathrm{unq}}\, \mathrm{list}\, \tau) \to \mathrm{box}\, \mathrm{list}\, \&^{\alpha}_{\mathrm{unq}}\, \tau \;\mid \\ &\qquad \lambda \mathit{post}, ((v, v_\circ)).\;\; \mathrm{len}\, v_\circ = \mathrm{len}\, v \;\Longrightarrow\; \mathit{post}\,(\mathrm{zip}\, v\, v_\circ)\end{aligned}$$

目次

- 準備
- 本研究
- **関連研究**
- 結論

関連研究

- RustBelt [Jung+ 2018]: 内部可変性 (= 共有できる権限を通して内部データを更新できる) を持つ型 (例: `Mutex`) の安全性を広く検証
- RustHorn [松下+ 2020]: 預言を用いた帰着を提案、構文論的な健全性・完全性証明を与えたが、新機能・ライブラリへの拡張は非自明
- Future is Ours [Jung+ 2020]: Iris 上で預言変数を扱う手法を提案、預言の表現力が Rust の専有借用のモデル化に不十分
- Prusti [Astrauskas+ 2019]: Iris より弱い分離論理で Rust プログラムを半自動検証、専有参照の分割等の操作が上手く表現できない

目次

- 準備
- 本研究
- 関連研究
- **結論**

結論

- Rust プログラム (unsafe コードを持つ) の機能正当性を型の保証を利用して柔軟に検証できる論理的基盤を築いた
  - 意味論的型により分離論理 Iris 上で拡張可能な検証を実現
  - 専有参照を預言により表現、新しい預言の定式化を開発

- 今後の課題
  - 本研究の結果の Coq における機械的証明
  - 大きい Rust ライブラリにもスケールするプログラム論理の構築
  - 幅広い機能 (特に内部可変性) を上手く扱う、より強力な検証手法の開発

博士課程の研究計画

- 大テーマ：**システムプログラム**の効率の良い検証
  “変化する状態の大域的な共有” (生のメモリ操作・並行処理)
  - **Rust** プログラムの機能正当性を効率良く検証できる
    強力なプラットフォームを作る (修論の方向性の更なる発展)
    - スケーラブルな検証を実現できるプログラム論理の構築
    - Rely-Guarantee 等の技術と融合、より多くの Rust プログラムを検証
  - **C** や **LLVM** のシステムプログラムを効率よく検証する手法の開発
    - CHC ソルバやモデル検査器等の技術と分離論理の融合

実績

- 論文
  - Yusuke Matsushita, Takeshi Tsukada and Naoki Kobayashi. 2020. RustHorn: CHC-based Verification for Rust Programs. ESOP 2020. (トップジャーナル TOPLAS Special Issue に招待され、投稿中)
- 口頭発表
  - 松下祐介, 塚田武志, 小林直樹. RustHorn: CHC-based Verification for Rust Programs. 日本ソフトウェア科学会 (JSSST) 大会 2020, トップカンファレンス特別講演.
- 研究インターンシップ
  - Max Planck Institute for Software Systems の RustBelt チームでのインターンシップ. 2020年9-12月. 現在も共同研究は継続中.