

Iris



特別講義

分離論理 Iris の世界

松下 祐介 京都大学 情報学研究科 五十嵐・末永研

2024.9.9 PPLサマースクール 立命館大学大阪いばらきキャンパス

第0部

オープニング

今日 9/9 = 重陽の節句 / 菊の節句

- ◆ 日本では影の薄い節句？
 - ▶ 五節句はこれと $1/7$ = 七草の節句、 $3/3$ = 桃の節句、 $5/5$ = 端午の節句、 $7/7$ = 七夕の節句

- ◆ 何をする日？
 - ▶ 中国 祖先の墓を参拝、菊の花を捧げることも
 - ▶ 日本 菊酒を飲んで長寿を祈る風習
 - 草の戸や 日暮れてくれし 菊の酒 松尾 芭蕉



講師 松下 祐介

京都大学 五十嵐・末永研 学振PD特別研究員

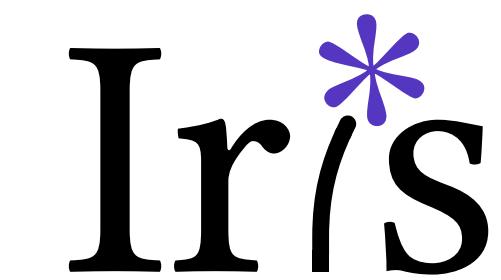


PLDI '22 にて

◆ 可変状態の検証を中心に研究

- ▶ RustHorn [ESOP '20/TOPLAS '21] 
 - Rust プログラムを「預言」で検証
- ▶ RustHornBelt [PLDI '22] 
 - RustHorn の手法を分離論理 Iris で基礎づけ
- ▶ Nola [博論 '24]
 - Iris の高階幽靈状態の later 様相の問題を解決

ソフトウェア開発の実践のための理論



♦ Iris: 現代的な分離論理のフレームワーク

- ▶ 分離論理 [O'Hearn+ '01]: モジュラーな可変状態検証
- ▶ ドイツ MPI-SWS Iris/RustBelt グループ
- ▶ 国際会議で Iris を使う研究が多数採択
 - 特に有名な研究が RustBelt [Jung+ '18]



ボス
Derek
Dreyer



| 15:10 - 16:30 Separation Logic at Turing Lecture Chair(s): Azalea Raad Imperial College London | | | |
|---|-----|---|--|
| 15:10 | 20m | ★ An Iris Instance for Verifying CompCert C Programs Talk | William Mansky University of Illinois Chicago, Ke Du Pre-print |
| 15:30 | 20m | ★ The Logical Essence of Well-Bracketed Control Flow Talk | Amin Timany Aarhus University, Armaël Guéneau Université Paris-Saclay - CNRS - ENS Paris-Saclay - Inria, Lars Birkedal Aarhus University |
| 15:50 | 20m | ★ Asynchronous Probabilistic Couplings in Higher-Order Separation Logic Talk | Simon Oddershede Gregersen Aarhus University, Alejandro Aguirre Aarhus University, Philipp G. Haselwarter Aarhus University, Joseph Tassarotti NYU, Lars Birkedal Aarhus University DOI Pre-print |
| 16:10 | 20m | ★ Thunks and Debits in Separation Logic with Time Credits Talk | François Pottier Inria, Armaël Guéneau Université Paris-Saclay - CNRS - ENS Paris-Saclay - Inria, Jacques-Henri Jourdan CNR, LMF, Glen Mével |

POPL 24

この講義

- ♦ 講師: Iris をガッツリ使う現在唯一の日本人?

- ▶ Iris/RustBelt でインターン、RustHornBelt 
- ▶ 博論 Nola は Iris のコア機能についての研究成果



- ♦ 狙い: 現代の分離論理 Iris を理解する

- ▶ Iris がわかると色々な基礎研究ができるて楽しい
- ▶ 若干ハードコアですがどうぞお付き合いください

全体の流れ

♦ 第1部 1時間 現代的な分離論理の基礎

休憩

♦ 第2部 1時間 Iris の発展的話題

休憩

♦ 第3部 1時間 Iris × Rust 質問歓迎！

第1部

現代的な分離論理の基礎

第1部の流れ

- ◆ 1.1 歴史的背景
- ◆ 1.2 基本のメモリ向け分離論理
- ◆ 1.3 代数による抽象化と拡張

1.1

歷史的背景

大テーマ 困難を分割する

- ◆ 難しい問題は分割したい
 - ▶ 全体を部分へと「分離」 ⇔ 部分についての推論を「合成」



デカルト

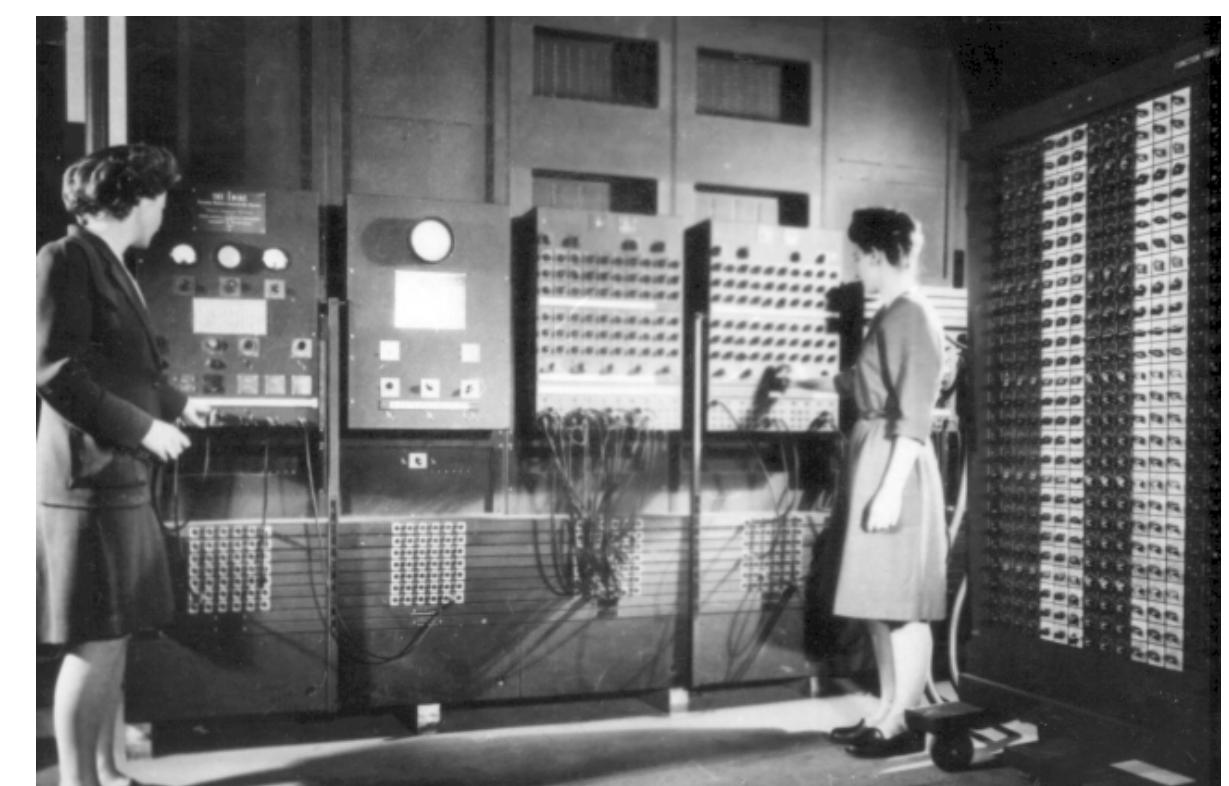
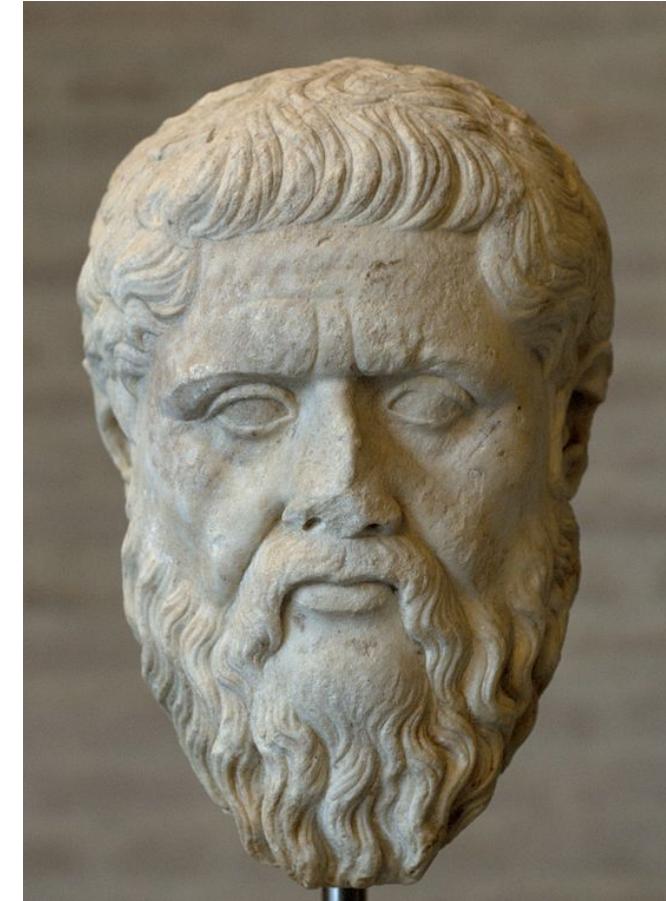
取り組む困難は可能な限り、
より良い解決に要するだけ、
多くの部分に分割すること

方法序説 第二部

計算機における困難 可変状態

- ◆ 数学の「イデア」の世界では全ては不变
 - ▶ とても素直で扱いやすい

- ◆ 計算は有限の「物理」世界で動いている
 - ▶ 本質的に可変状態(メモリ、環境、etc.)を扱う
 - ▶ 可変状態についての推論は難しい
 - 状態変化でそれまでの推論が壊れる可能性
 - 数学ではふつう見ない、計算機科学らしい難しさ



ジラールの線形論理

[Girard '87]

- ♦ 複製できない命題を扱う代表的な論理
 - ▶ 可変状態について論じる、計算機科学への応用も意識

LINEAR LOGIC*

Jean-Yves GIRARD



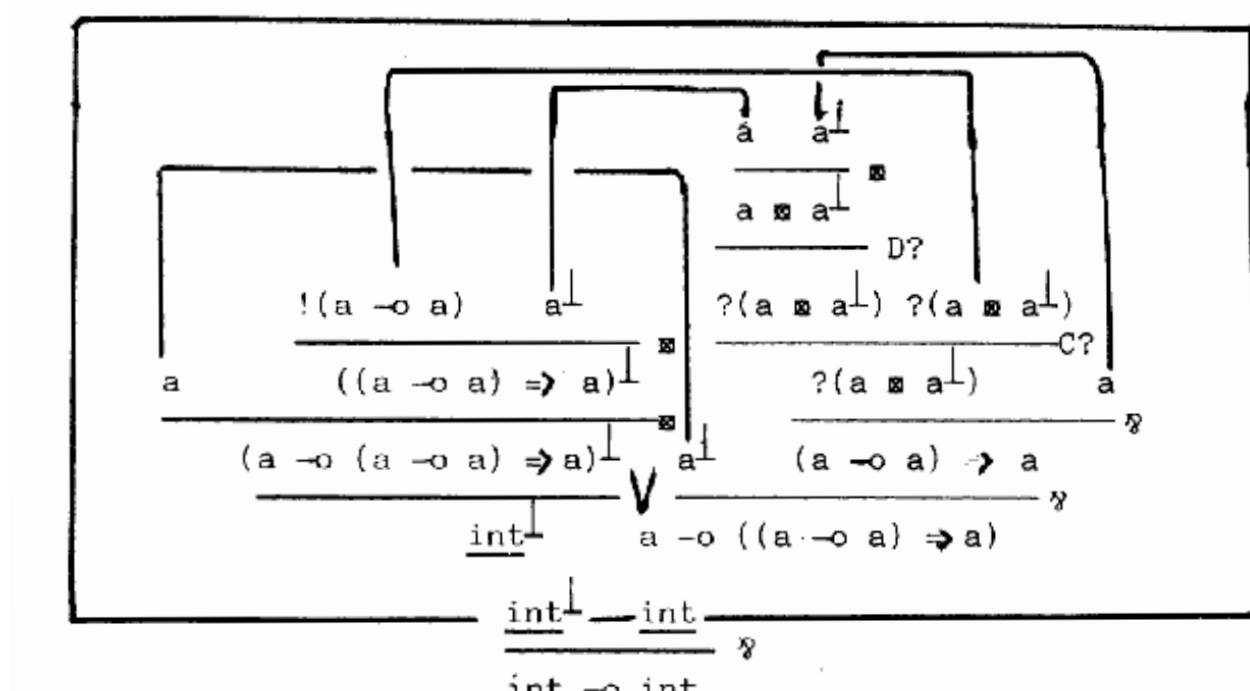
* Because of its length and novelty this paper has not been subjected to the normal process of refereeing. The editor is prepared to share with the author any criticism that eventually will be expressed concerning this work.

“La secrète noirceur du lait...”

(Jacques Audiberti)

$$\frac{\vdash A, C \quad \vdash B, C}{\vdash A \& B, C} \&, \quad \frac{\vdash A, C}{\vdash A \oplus B, C} 1^\oplus \quad \frac{\vdash B, C}{\vdash A \oplus B, C} 2^\oplus$$

$$\frac{\vdash A, C \quad \vdash B, D}{\vdash A \otimes B, C, D} \otimes, \quad \frac{\vdash A, B, C}{\vdash A \wp B, C} \wp.$$



分離論理の前身 BI

[O'Hearn & Pym, '99]

◆ 線形論理を少し変えてより綺麗にした論理

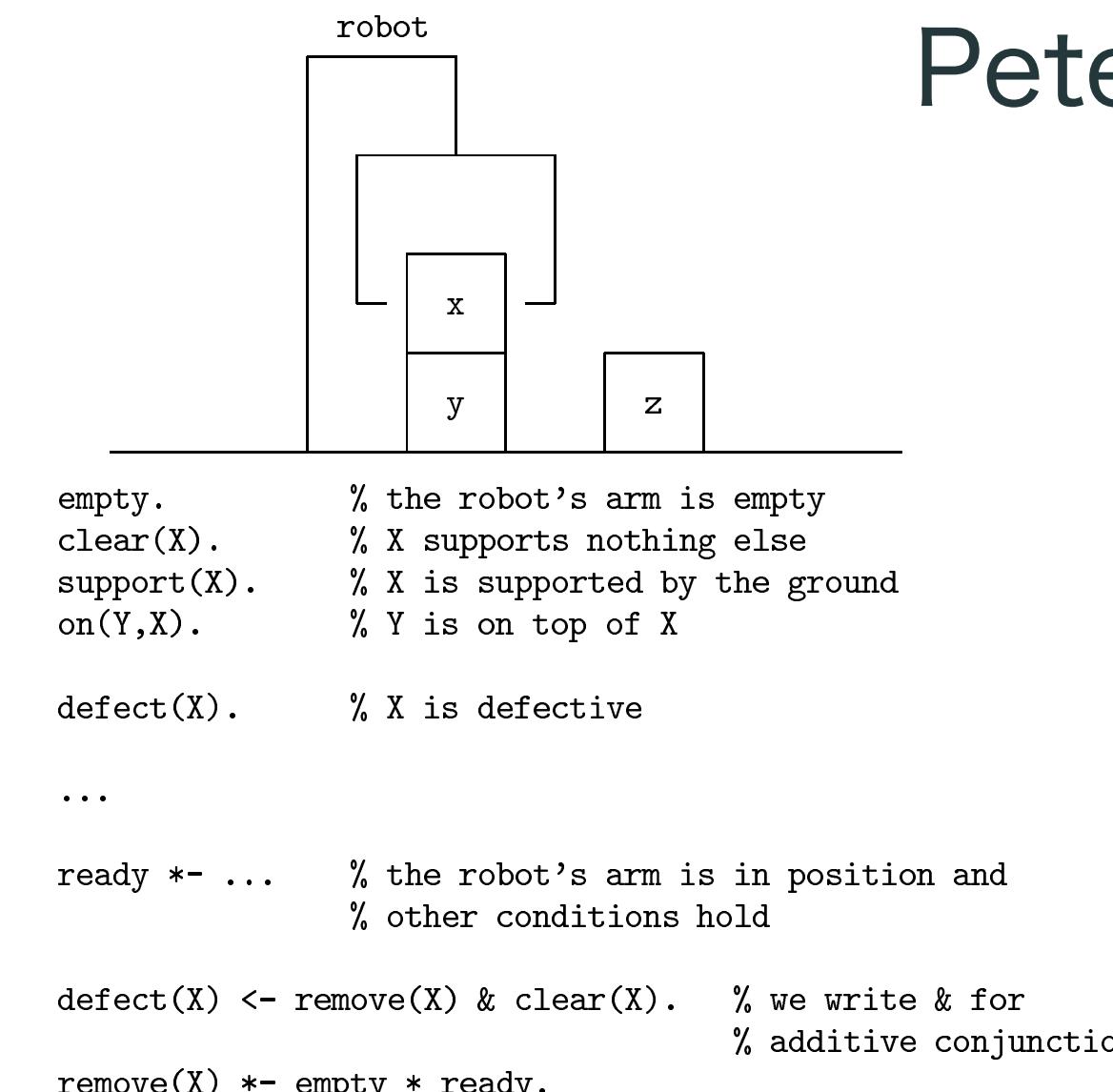
- ▶ “普通” の論理結合子 $\wedge, \rightarrow, \vee$ + “分離” する論理結合子 $*$, --
 - BI = Bunched Implications、含意が2つ $\rightarrow, \text{--}$
- ▶ 計算機科学への応用も意識

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi * \psi} *I$$

$$\frac{\Gamma \vdash \varphi \quad \Delta \vdash \psi}{\Gamma, \Delta \vdash \varphi * \psi} *I$$

$$\frac{\Gamma \vdash \varphi * \psi \quad \Delta \vdash \varphi}{\Gamma, \Delta \vdash \psi} *E$$

$$\frac{\Gamma(\varphi, \psi) \vdash \chi \quad \Delta \vdash \varphi * \psi}{\Gamma(\Delta) \vdash \chi} *E$$



Peter O'Hearn

CC BY-NC 3.0
Royal Society

- ♦ 可変状態をもつプログラムを検証

- ▶ 命題は可変状態のある部分を所有、一時的な知識
- ▶ 並行プログラムもごく自然に検証できる

$$\{ r \mapsto v \} * r = v' \{ r \mapsto v' \}$$

$$\frac{\{P\} e \{Q\} \quad \{P'\} e' \{Q'\}}{\{P * P'\} e \parallel e' \{Q * Q'\}}$$

分離 & 合成



DOI:10.1145/3211968

Separation logic is a key development in formal reasoning about programs, opening up new lines of attack on longstanding problems.

BY PETER O'HEARN

Separation Logic COMMUNICATIONS OF THE ACM

分離論理の近年の盛り上がり

♦ 並行分離論理は '16 ゲーデル賞を受賞

2016 Godel Prize Recognizes Major Advances
in Verification of Concurrent Programs
Stephen Brookes and Peter W. O'Hearn Honored for
Invention of Concurrent Separation Logic

♦ PL・ソフトウェア科学的一大分野として確立

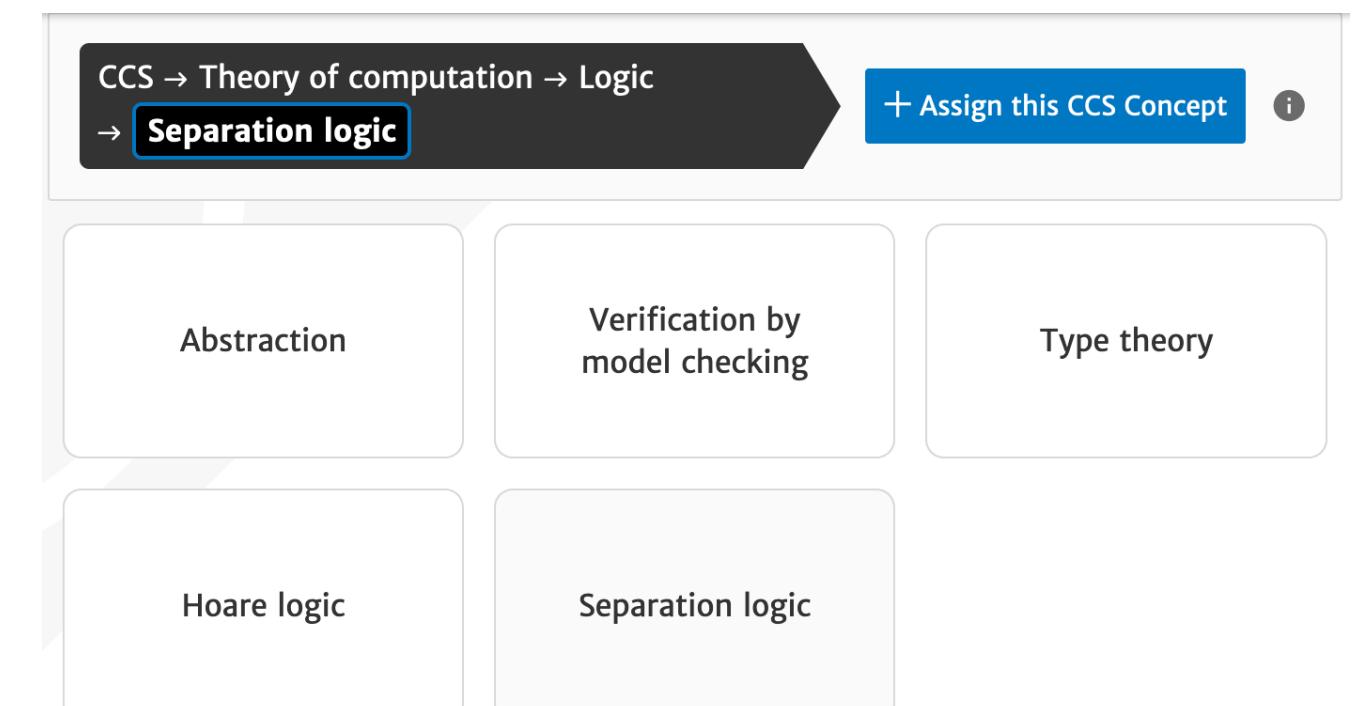
- ▶ ACM CCS '12 ではホーア論理などに並ぶ扱い
- ▶ 分離論理 Iris [Jung+ '15] の登場でさらに加速
 - Church 賞 '23 を受賞



Iris

Thursday, July 13, 2023
Presburger & Church Award Talks
Session Chair: Artur Czumaj

Alonzo Church Award 2023
Less is More: A Brief Retrospektive on Iris
Speakers: Ralf Jung, Robbert Krebbers, Derek Dreyer



分離論理の応用例

- ◆ プログラミング
 - ▶ RustBelt [Jung+ '18] — Rust の安全性保証を Iris で検証
 - ▶ Infer [Calcagno+ '15] — 産業用バグ発見ツール
- ◆ より抽象的なリソース
 - ▶ 確率的分離論理 [Barthe+ '19] — 確率変数を独立性などで分離
 - ▶ 量子分離論理 [Le+ '22] — 量子状態を分離

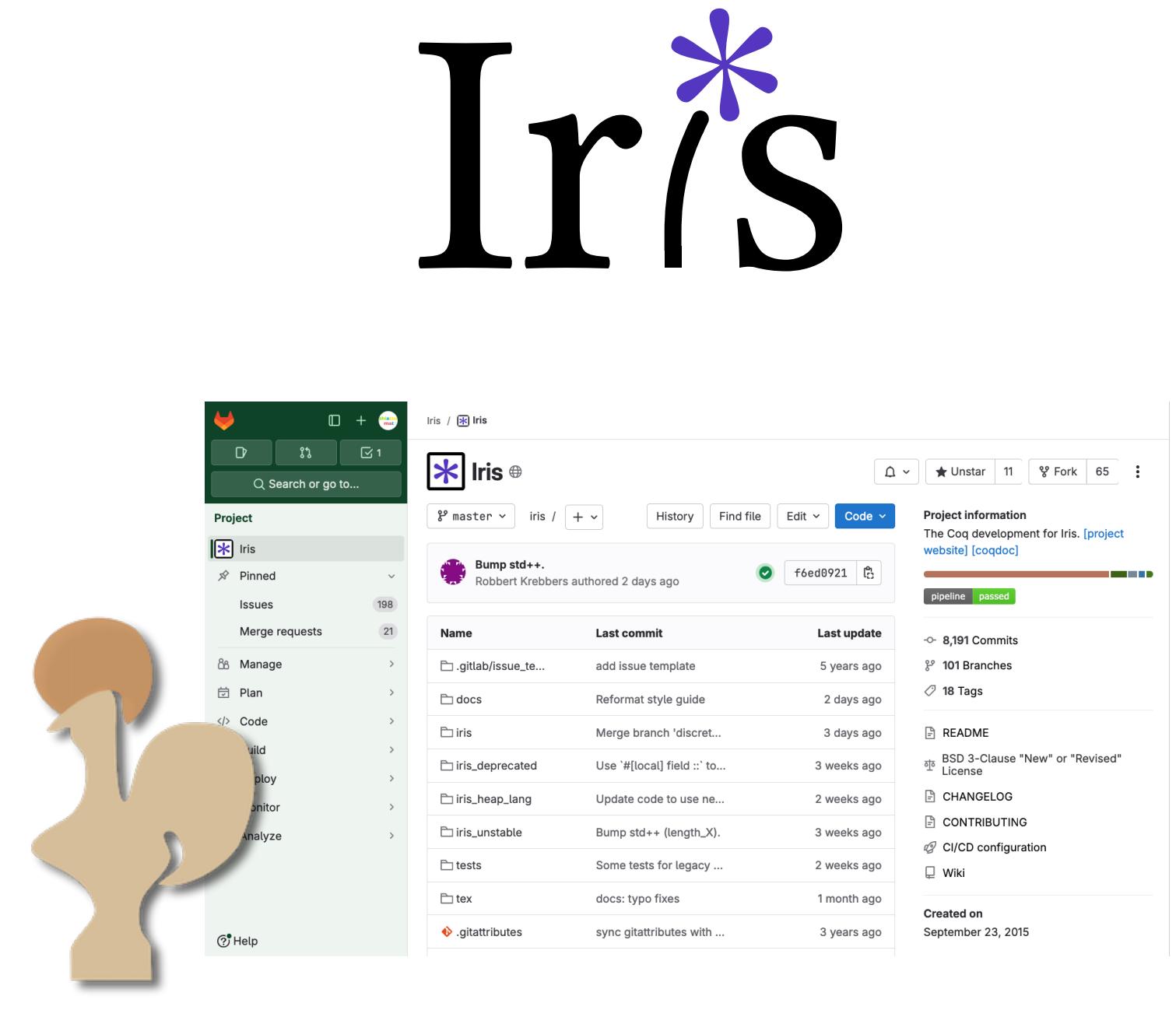


Iris の位置づけ

[Jung+ POPL '15, JFP '18]

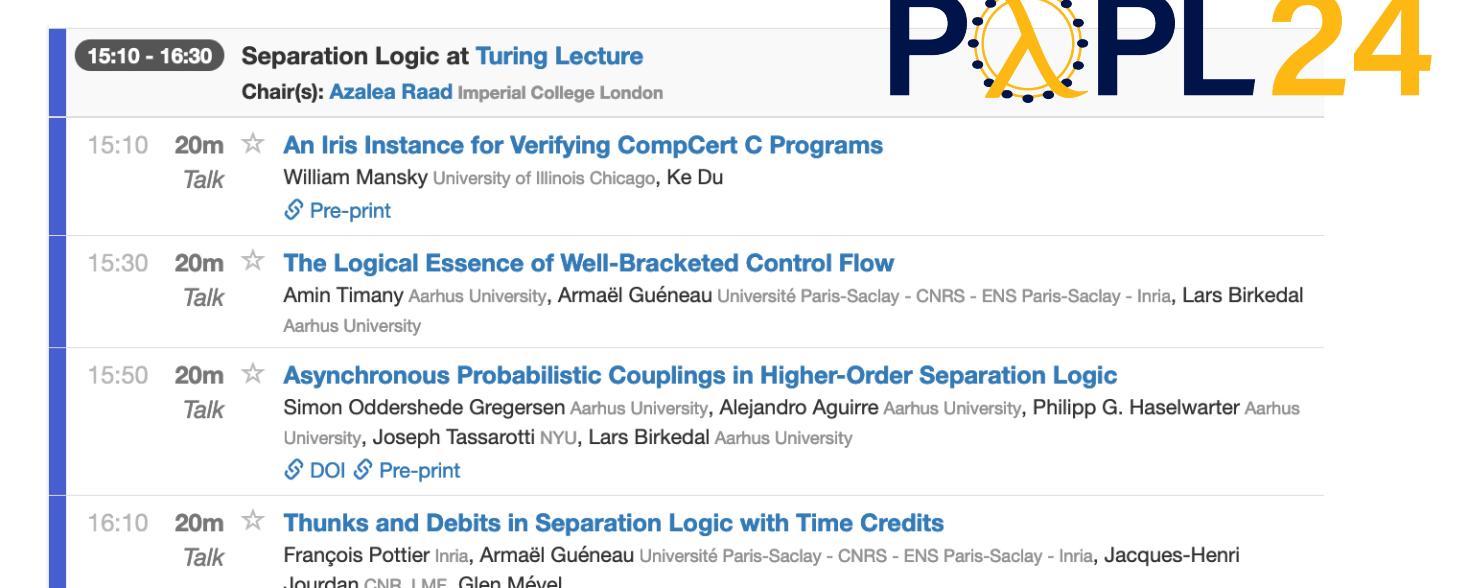
◆ 柔軟性に富む分離論理フレームワーク

- ▶ 可変状態モデルの代数による強力な抽象化
- ▶ 高階幽靈状態による柔軟な表現
- ▶ 成熟した Coq での実装



◆ 様々な応用研究

- ▶ 特に並行プログラムの検証で活躍



Irisについてのおすすめ文献

♦ Iris from the ground up [Jung+ JFP'18]

- ▶ Irisについて基礎から詳しく解説

♦ Understanding and Evolving

the Rust Programming Language [Jung 博論'20]

- ▶ 第1部に Iris のハイレベルな説明、第2部に RustBelt の詳しい解説

UNDERSTANDING AND EVOLVING
THE RUST PROGRAMMING LANGUAGE

Iris from the ground up

A modular foundation for higher-order concurrent separation logic

RALF JUNG

MPI-SWS, Germany

(e-mail: jung@mpi-sws.org)

ROBBERT KREBBERS

Delft University of Technology, The Netherlands

(e-mail: mail@robbertkrebbers.nl)

JACQUES-HENRI JOURDAN

MPI-SWS, Germany

(e-mail: jjourdan@mpi-sws.org)

ALEŠ BIZJAK

Aarhus University, Denmark

(e-mail: abizjak@cs.au.dk)

LARS BIRKEDAL

Aarhus University, Denmark

(e-mail: birkedal@cs.au.dk)

DEREK DREYER

MPI-SWS, Germany

(e-mail: dreyer@mpi-sws.org)

1.2

基本のメモリ向け分離論理

メモリ操作を検証したい

- ◆ メモリ操作は様々なバグを誘発
 - ▶ Use after free、ダングリングポインタ、バッファオーバーラン、アクセス競合、…
- ◆ 現実の深刻な脆弱性の多くがメモリ由来
 - ▶ 有名な Heartbleed もメモリの不正アクセス
 - ▶ ゼロデイ攻撃の約7割がメモリ由来 [Google Zero '19]

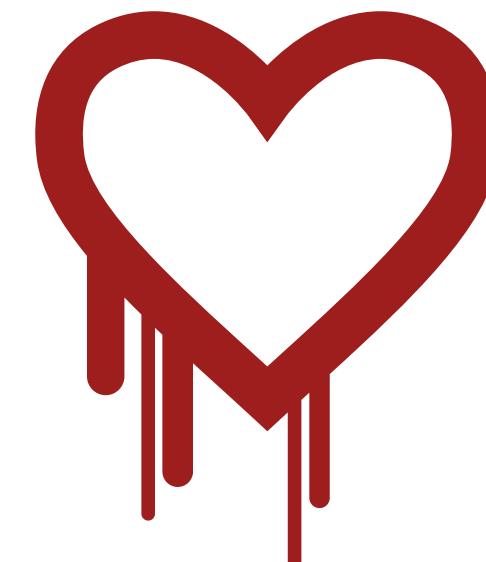
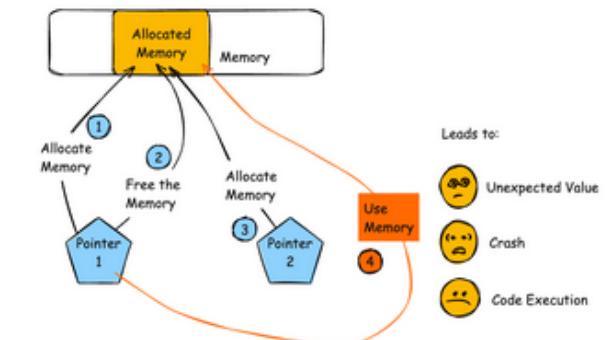
CWE-416: Use After Free

Weakness ID: 416
Vulnerability Mapping: ALLOWED
Abstraction: Variant

View customized information: Conceptual Operational Mapping Friendly Complete Custom

Description

The product reuses or references memory after it has been freed. At some point afterward, the memory may be allocated again and saved in another pointer, while the original pointer references a location somewhere within the new allocation. Any operations using the original pointer are no longer valid because the memory "belongs" to the code that operates on the new pointer.



| CVE | Vendor | Product | Type |
|---------------|-----------|-------------------|-------------------|
| CVE-2019-7286 | Apple | iOS | Memory Corruption |
| CVE-2019-7287 | Apple | iOS | Memory Corruption |
| CVE-2019-0676 | Microsoft | Internet Explorer | Information Leak |
| CVE-2019-5786 | Google | Chrome | Memory Corruption |
| CVE-2019-0808 | Microsoft | Windows | Memory Corruption |
| CVE-2019-0797 | Microsoft | Windows | Memory Corruption |



メモリ操作の検証は難しい

- ♦ 素朴にはエイリアシングについての場合分けが必要

同じアドレスを指す状況



$$\begin{bmatrix} r \mapsto 0 \wedge s \mapsto 0 \\ *r += 1; *s += 2 \end{bmatrix}$$

$$\begin{bmatrix} r \mapsto 1 \wedge s \mapsto 2 \end{bmatrix}$$

r と s はエイリアスしうる
間違い



$$\begin{bmatrix} r = s \wedge \\ r \mapsto 0 \wedge s \mapsto 0 \end{bmatrix}$$

$$*r += 1; *s += 2$$

$$\begin{bmatrix} r \mapsto 3 \wedge s \mapsto 3 \end{bmatrix}$$

予期せぬエイリアス



$$\begin{bmatrix} r \neq s \wedge \\ r \mapsto 0 \wedge s \mapsto 0 \end{bmatrix}$$

$$*r += 1; *s += 2$$

$$\begin{bmatrix} r \mapsto 1 \wedge s \mapsto 2 \end{bmatrix}$$

面倒な場合分け

分離論理による検証

- ♦ 所有権を体系的に分離、モジュラーな検証
 - ▶ 命題は可変状態のある部分を所有、一時的な知識



主な推論規則

- ◆ 単純な規則の組み合わせで検証できる

メモリ操作

$$\begin{array}{l} \{\top\} \text{ ref } v \vee \{\lambda r. r \mapsto v\} \\ \{r \mapsto v\} * r \{\lambda u. u = v * r \mapsto v\} \\ \{r \mapsto v\} * r = v' \{r \mapsto v'\} \end{array}$$

分離論理積

$$\begin{array}{ll} P * \top \equiv P & P * Q \equiv Q * P \\ P * (Q * R) \equiv (P * Q) * R & \\ r \mapsto v * r' \mapsto v' \models r \neq r' & \end{array}$$

ホーア三つ組

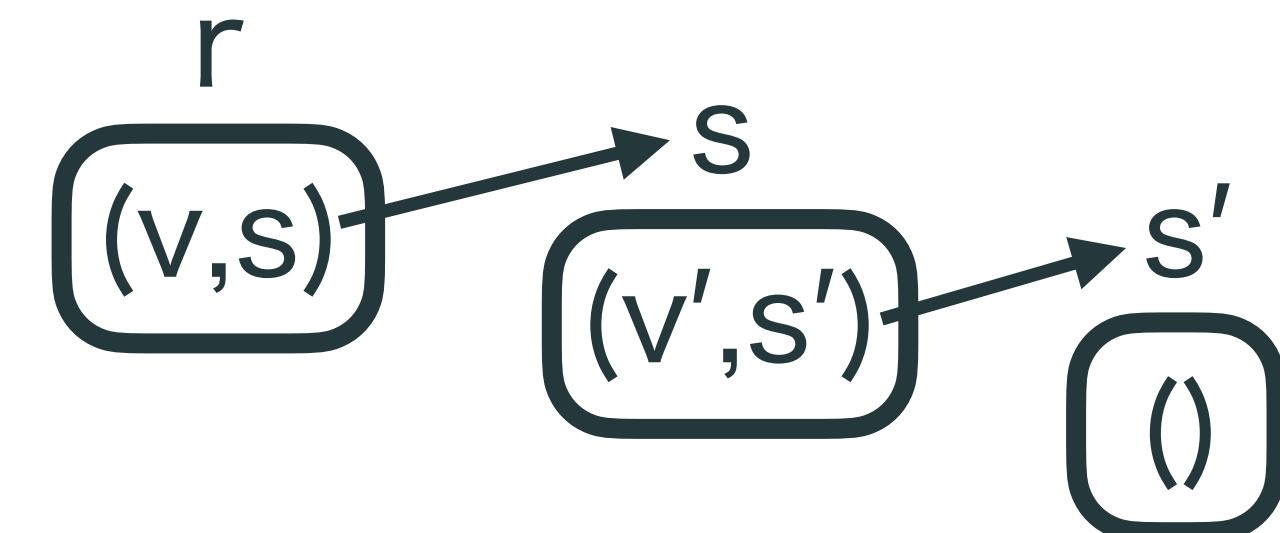
$$\frac{\{P\} e \{\Psi\}}{\{P * R\} e \{\lambda v. \Psi v * R\}}$$

フレーム則

モジュラーな推論の要

例 単連結リストの検証

- ♦ 単連結リストは帰納的に表現できる
 - ▶ 所有権の情報が自然にエンコードされる



$$\text{list } r \triangleq_{\mu} r \mapsto () \vee \exists s, v. r \mapsto (v, s) * \text{list } s$$

```
fn push(r, v) { let v = *r; if v == () { *r = v } else { push(v.1, v) } }
```

{ list r } push(r, v) { list r } \because (余)帰納法

分離論理のモデル

- ♦ 分離論理命題 = ヒープメモリ (の部分領域) 上の述語
 - ▶ $P h$ が成立 $\Leftrightarrow P$ はヒープメモリ h を所有すれば満たされる

$$iProp_{\text{heap}} \triangleq \text{Heap} \xrightarrow{\text{mono}} \text{Prop}$$
$$\text{Heap} \triangleq \text{Addr} \xrightarrow{\text{fin}} \text{Val}$$

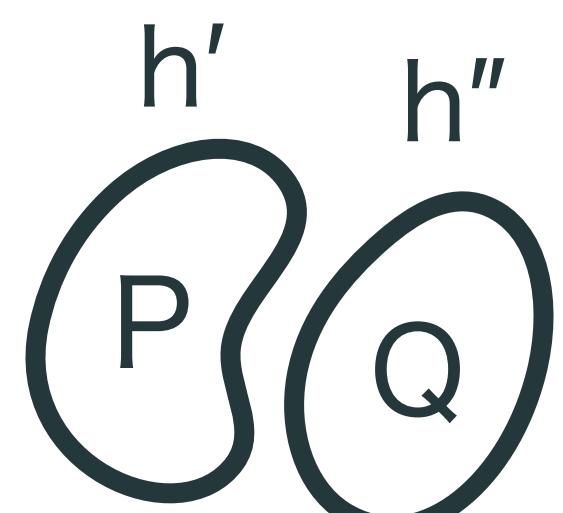
単調性 $h \subseteq h' \rightarrow (Ph \rightarrow Ph')$
アフィン性 $P * Q \models P$
 $P \models Q \triangleq \forall h. Ph \rightarrow Qh$

$$r \mapsto v \triangleq \lambda h. h[r] = v$$

アドレス r 値 v

$$P * Q \triangleq \lambda h. \exists h', h''. h = h' + h'' \wedge Ph' \wedge Qh''$$

非交和



様々な論理結合子

- ♦ ふつうの論理結合子は自然に定義できる
 - ▶ 直観主義論理の推論規則がすべて成立

$$\begin{array}{ll} \lceil \phi \rceil \triangleq \lambda _. \phi \ (\phi \in Prop) & P \wedge Q \triangleq \lambda h. P h \wedge Q h \\ \exists x. P_x \triangleq \lambda h. \exists x. P_x h & P \rightarrow Q \triangleq \lambda h. \forall h' \supseteq h. P h' \rightarrow Q h' \end{array}$$

- ♦ 分離含意 \rightarrow^* もある

$$P \models Q \rightarrow^* R \text{ iff } P * Q \models R$$

- ▶ 分離論理積 $*$ の右隨伴

$$\begin{aligned} P \rightarrow^* Q &\triangleq \lambda h. \forall h' \perp h. \\ &P h' \rightarrow Q (h + h') \end{aligned}$$

ホーア三つ組のモデル

- ♦ ヒープメモリ全体の情報をを利用して定義される

$$\{P\} e \{\Psi\} \triangleq \forall R. P * R \models \text{pwp } e \{\lambda v. \Psi v * R\}$$

フレーム則を満たすための仕掛け

(部分) 最弱事前条件

$$\begin{aligned} \text{pwp } e \{\Psi\} &\triangleq_{\nu} \lambda h. (\exists v = e. \Psi v h) \vee \\ &(\text{red}(e, h) \wedge \forall (e', h') \leftarrow (e, h). \text{pwp } e' \{\Psi\} h') \end{aligned}$$

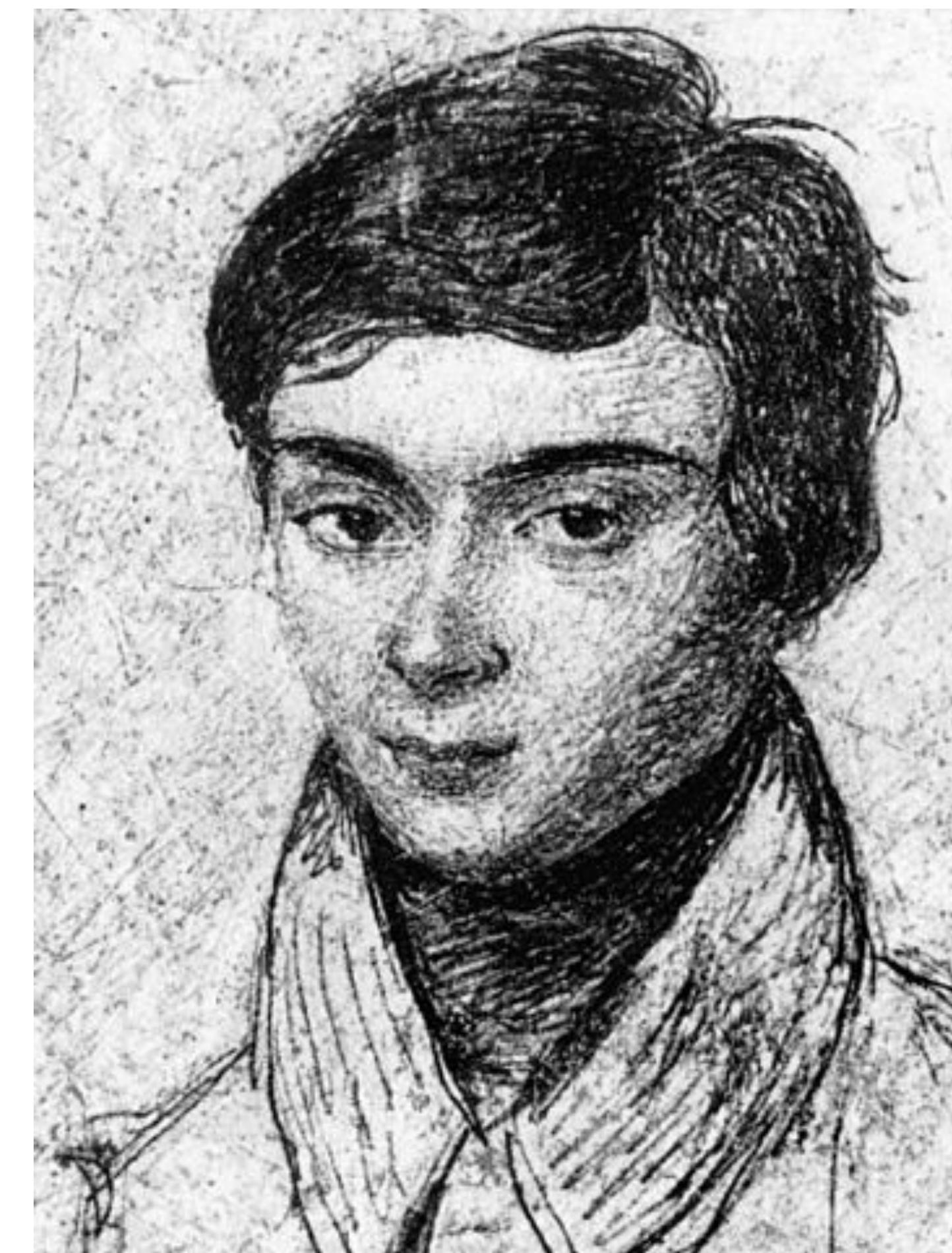
$$\text{red}(e, h) \triangleq \exists (e', h'). (e, h) \hookrightarrow (e', h')$$

後に分離論理の内部で完結する定義に進化

1.3

代数による抽象化と拡張

代数の力



部分可換モノイド PCM

- ◆ 分離 \leftrightarrow 合成 ができる「状態」の数学的モデル
 - ▶ 和の演算が部分関数、合成可能性を自由に定義できる

$\mathcal{A} = (A, + : A^2 \rightarrow A, 0 : A)$ 例: $(Heap, +, \emptyset)$ は PCM
和 = 状態の合成

$$a + 0 = a \quad a + b = b + a \quad \text{cf. 分離論理積の規則}$$

$$a + (b + c) = (a + b) + c \quad P * \top \equiv P \quad P * Q \equiv Q * P$$

定義可能性は両辺で同値

$$P * (Q * R) \equiv (P * Q) * R$$

$$a \leq b \triangleq \exists c. b = a + c \quad a \perp b \triangleq a + b \text{ is defined}$$

PCM による分離論理のモデル化

- ♦ 分離論理は任意の PCM \mathcal{A} の上で展開できる
 - ▶ PCM をリッチにすればより表現力の高い分離論理が得られる

$$iProp \triangleq \mathcal{A}^{\text{mono}} \rightarrow Prop$$

状態の所有 $\boxed{a} \triangleq \lambda b. a \leq b$

$$P * Q \triangleq \lambda a. \exists b, c.$$

$$a = b + c \wedge P b \wedge Q c$$

例 $r \mapsto v \triangleq \boxed{[r := v]}$

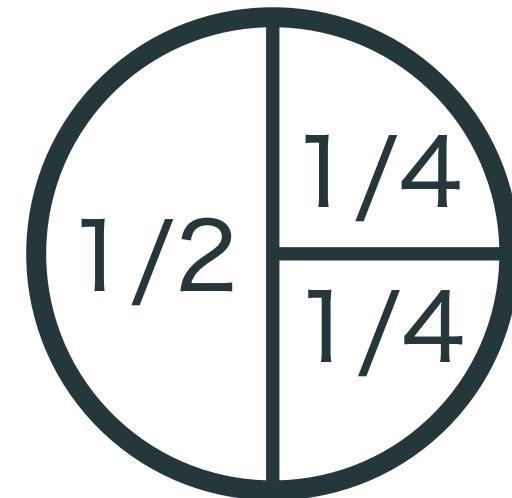
$$\boxed{a} * \boxed{b} \models a \perp b \quad \boxed{a} * \boxed{b} \equiv \boxed{a + b}$$

メモリの拡張例1 分数所有権

[Boyland '03, Bornat+ '05]

- ♦ 分数 points-to トークン
- メモリセルの値を一時的に不变にして共有

所有権を分割



$$\begin{array}{c}
 r \mapsto v \triangleq r \stackrel{1}{\mapsto} v \\
 r \xrightarrow{q+q'} \textcolor{violet}{v} \equiv r \stackrel{q}{\mapsto} v * r \stackrel{q'}{\mapsto} v \quad \{ r \stackrel{q}{\mapsto} v \} * r \{ \lambda u. u = v * r \stackrel{q}{\mapsto} v \}
 \end{array}$$

$$Heap_{\text{fract}} \triangleq Addr \xrightarrow{\text{fin}} (0, 1] \times Val \quad r \stackrel{q}{\mapsto} v \triangleq \boxed{[r := (q, v)]}$$

$$(h + h')[r] \triangleq h[r] + h'[r] \quad (q, v) + (r, v') \triangleq \begin{cases} (q + r, v) & q + r \leq 1, v = v' \\ \text{undefined} & \text{otherwise} \end{cases}$$

メモリの拡張例2 確定による永続共有

- ♦ 永続 points-to トークン

- ▶ メモリセルの値は不变、自由に共有



$$r \Rightarrow v \equiv r \Rightarrow v * r \Rightarrow v$$

$$\{ r \mapsto v \} \{ r \Rightarrow v \}$$

$$\{ r \Rightarrow v \} * r \{ \lambda u. u = v * r \Rightarrow v \}$$

$$Heap_{\text{pers}} \triangleq Addr \xrightarrow{\text{fin}} \mathbb{B} \times Val$$

$$(h + h')[r] \triangleq h[r] + h'[r]$$

$$r \mapsto v \triangleq [r := (\text{tt}, v)]$$

$$r \Rightarrow v \triangleq [r := (\text{ff}, v)]$$

$$(b, v) + (b', v') \triangleq \begin{cases} (ff, v) & b = b' = ff, v = v' \\ \text{undefined} & \text{otherwise} \end{cases}$$

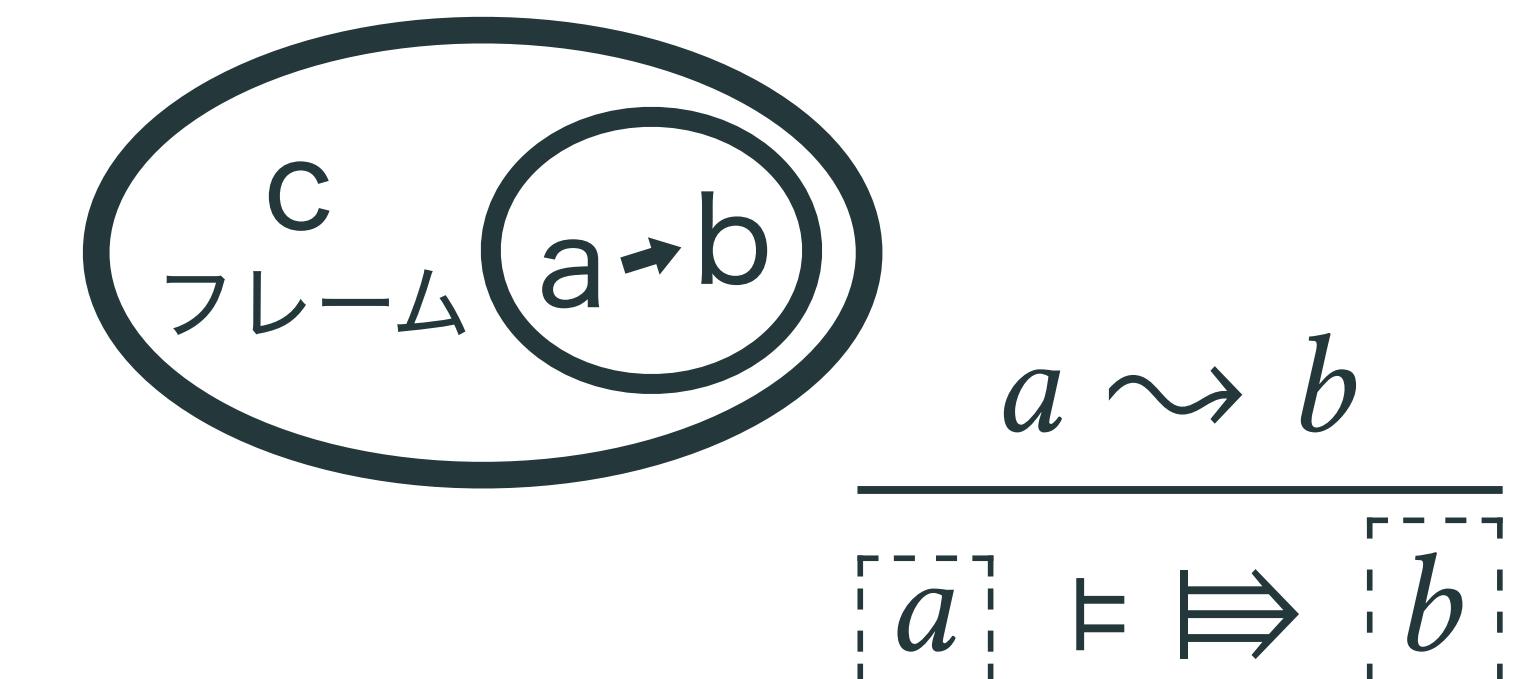
- ◆ 所有する状態を更新したあとを表現

- ▶ 法則 $P \models \Rightarrow P \quad \Rightarrow P \models \Rightarrow \Rightarrow P \quad Q * \Rightarrow P \models \Rightarrow (Q * P)$
 モナド則 フレーム則

- ▶ 不変条件 「すべての状態の和は defined」

- モデル $\Rightarrow P \triangleq \lambda a. \forall c \text{ s.t. } a \perp c. \exists b \perp c. P b$

- PCM 上 $a \rightsquigarrow b \triangleq \forall c \text{ s.t. } a \perp c. b \perp c$



略記 $P \equiv* Q \triangleq P \rightarrow \Rightarrow Q$

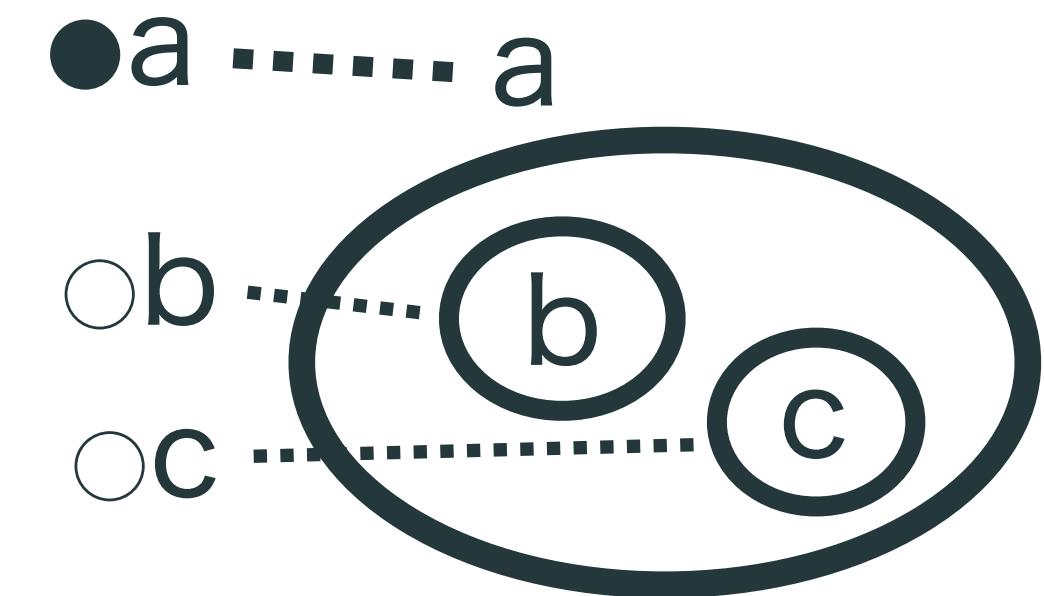
結合が最弱と約束 $\Rightarrow P * Q = \Rightarrow (P * Q)$

Iris では Basic Update
と呼んで $\dot{\Rightarrow}$ と書く

Authoritative PCM

[Dinsdale-Young+ '13]

- ♦ AUTH \mathcal{A} : PCM \mathcal{A} + 全体を把握できる機能
- ▶ 状態全体を知る $\bullet a$ 、部分を所有する $\circ a$



$$\begin{array}{lll} \circ (a + b) = \circ a + \circ b & \frac{\bullet a \perp \circ b}{a \geq b} & \frac{\forall c. a = b + c \rightarrow a' = b' + c}{\bullet a + \circ b \rightsquigarrow \bullet a' + \circ b'} \\ \circ 0 = 0 & & \end{array}$$

例 *Heap* の代わりに *AUTH Heap* を使う

分離論理の内部でメモリ操作を表現

$$r \mapsto v \triangleq \boxed{\circ [r := v]}$$

$$\boxed{\bullet h} * r \mapsto v \models \Rightarrow \boxed{\bullet h\{r := v'\}} * r \mapsto v'$$

ホーア三つ組の新しいモデル

- ♦ Authoritative PCM の力で分離論理の内部で書ける

$$\{P\} e \{\Psi\} \triangleq P \models \text{pwp } e \{\Psi\}$$

$$\begin{aligned} \text{pwp } e \{\Psi\} &\triangleq_{\nu} \lambda h. (\exists v = e. \Rightarrow \Psi v h) \vee \forall h. [\bullet h] \not\models \\ &\quad \text{red}(e, h) \wedge \forall (e', h') \leftarrow (e, h). \Rightarrow [\bullet h'] * \text{pwp } e' \{\Psi\} \end{aligned}$$

並行計算への拡張も容易

$$\begin{aligned} \text{pwp } e \{\Psi\} &\triangleq_{\nu} \lambda h. (\exists v = e. \Rightarrow \Psi v h) \vee \forall h. [\bullet h] \not\models \text{red}(e, h) \wedge \\ &\quad \forall (e', e_{\text{fork}}^?, h') \leftarrow (e, h). \Rightarrow [\bullet h'] * \text{pwp } e' \{\Psi\} * \text{pwp } e_{\text{fork}}^? \{\top\} \end{aligned}$$

分離論理の十全性 Adequacy

- ♦ 分離論理での推論をふつうの世界に持ち出す
 - ▶ (部分) ホーア三つ組の十全性: $\{\top\} \vdash \{\dot{\phi}\}$ ならば
 - e の実行は行き詰まりにならない
 - 任意の実行列 $(e = e_0, h_0) \hookrightarrow (e_1, h_1) \hookrightarrow \dots \hookrightarrow (e_n = v, h_n)$ について $\dot{\phi} \vdash v$

証明

$$\Rightarrow \phi \vdash \phi \quad \frac{[\underline{a}] \models \phi}{\phi} \quad \text{を利用する}$$

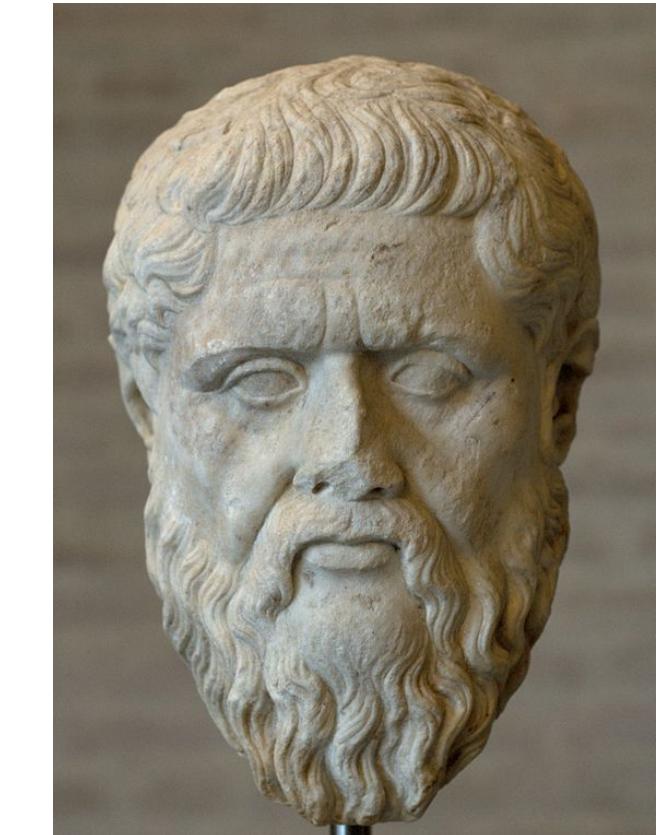
永続の様相 □ P

- ◆ 何も独占しない、自由に共有可能

- ▶ 法則 $\square P \models P$ $\square P \models \square \square P$
- S4 公理 (コモナド則) $Q \wedge \square P \models Q * \square P$
- 分離

► 命題の永続性 P is persistent $\triangleq P \models \Box P$

- $\Box P$ 、純粹な命題、永続的命題の論理積 etc. は永続的
 - 先ほどの $r \Rightarrow v$ も永続的



永続的な H について

$$P \wedge Q \equiv P * Q$$

自由に共有

$$\frac{P \models Q}{P \models \Box Q}$$

$$\frac{Q \models P}{Q \models P * Q}$$

永続周りの略記

- ♦ 永続な命題をふだんから使うと便利

$$P \Rightarrow Q \triangleq \square(P \rightarrow Q) \quad P \Rightarrowtail Q \triangleq \square(P \not\rightarrow Q)$$

$\square(P \not\rightarrow Q)$ と同値

ホーア三つ組は永続な命題 $\{P\} \text{ e } \{\Psi\} \triangleq P \Rightarrow \text{pwp e } \{\Psi\}$

$$\frac{P \Rightarrow Q \quad \{P\} \text{ e } \{\Psi\} \quad \forall v. \Psi v \Rightarrow \Psi' v}{\{P'\} \text{ e } \{\Psi'\}}$$

一般に規則 $\frac{P_1 \cdots P_n}{Q}$ は
 $P_1 \wedge \cdots \wedge P_n \models Q$ の意味

コア付き PCM

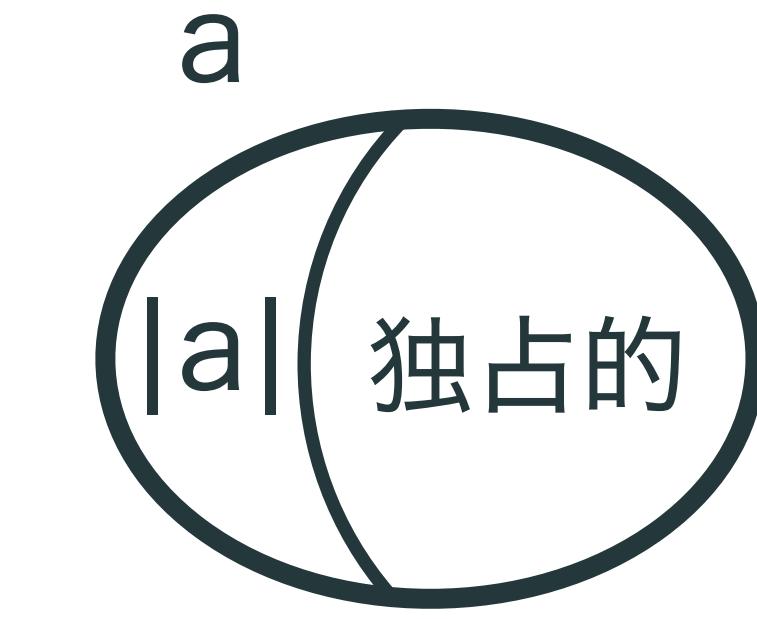
[Dockins+ '09, Jung+ '15]

♦ 永続の様相のためにコア $|\cdot|$ を足す

$$\mathcal{A} = (A, +: A^2 \multimap A, 0: A, |\cdot|: A \rightarrow A)$$

$$|a| + a = a \quad ||a|| = |a| \quad a \leq b \rightarrow |a| \leq |b|$$

特に $|a| = |a| + |a|$ が成立



$$\Box P \triangleq \lambda a. P |a|$$

$[\![a]\!]$ is persistent

例

$Heap_{\text{pers}}$

$$|h|[\textcolor{red}{r}] \triangleq |h[\textcolor{red}{r}]|$$

$$|(b, \textcolor{red}{v})| \triangleq \begin{cases} (\text{ff}, \textcolor{red}{v}) & b = \text{ff} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Iris のリソース代数 RA

[Jung+ '15]

- ♦ コア付き PCM - 0、和は全域 & 妥当性が別

Iris^{*}

$$\mathcal{A} = (A, + : A^2 \rightarrow A, \checkmark : A \rightarrow Prop, |\cdot| : A \rightarrow A^\perp)$$

$$a + b = b + a \quad a + (b + c) = (a + b) + c \quad \checkmark(a + b) \rightarrow \checkmark a$$

$$|a| + a = a \quad ||a|| = |a| \quad a \leq b \rightarrow |a| \leq |b|$$

$$\begin{aligned} A^\perp &\triangleq A \uplus \{\perp\} & a? + \perp &\triangleq \perp + a? \triangleq a? & |\perp| &\triangleq \perp \\ a? \leq b? &\triangleq \exists c?. b? = a? + c? & a \perp b? &\triangleq \checkmark(a + b?) \end{aligned}$$

単位的 RA : 次を満たす 0 をもつ RA $a + 0 = a$ $\checkmark 0$ $|0| = 0$

RA の設計の理由

- ◆ なぜ和が全域で妥当性述語を別に考える?
 - ▶ 直観主義論理 / クリプキ意味論 での定式化がしやすい
 - 和の出力を値の等しさなどの条件で場合分けしたくない
- ◆ なぜ単位元 0 がデフォルトでない?
 - ▶ 排他性が便利 $a \text{ is exclusive} \triangleq \forall b. \neg(a \perp b)$

$$\frac{a \text{ is exclusive}}{a \rightsquigarrow b} \quad a \rightsquigarrow b \triangleq \forall c^? \text{ s.t. } a \perp c^?. b \perp c^?$$

RA 上での分離論理

- ♦ 分離論理の命題は妥当な元の上の述語

$$iProp \triangleq \mathcal{A}^{\checkmark} \xrightarrow{\text{mono}} Prop \quad \mathcal{A}^{\checkmark} \triangleq \{a \in \mathcal{A} \mid \checkmark a\}$$

$$\boxed{a} \triangleq \lambda b. a \leq b$$

$$\begin{aligned} P * Q &\triangleq \lambda a. \exists b, c. \\ &a = b + c \wedge Pb \wedge Qc \end{aligned}$$

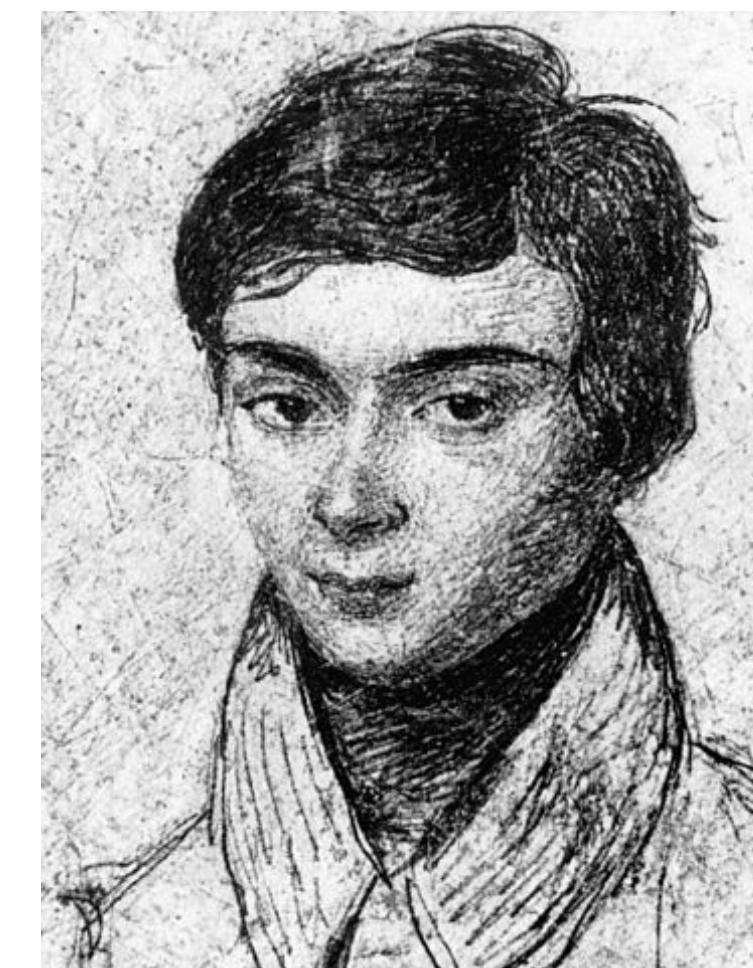
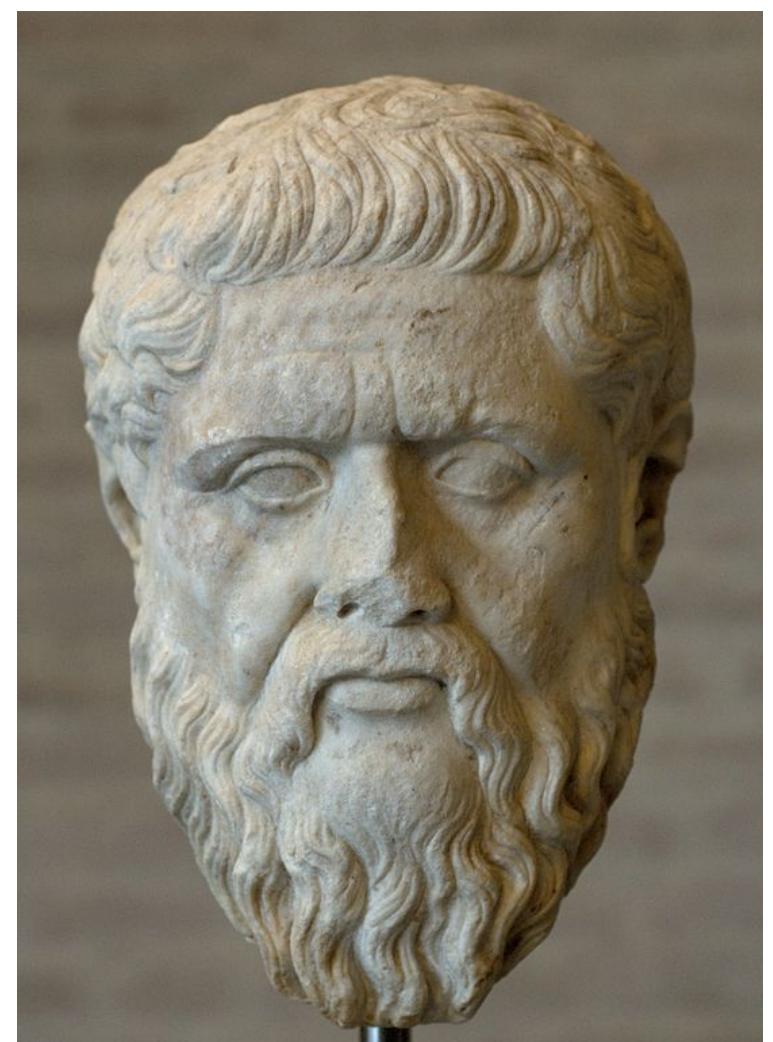
$$\boxed{a} \Rightarrow \checkmark a$$

$$\boxed{a} * \boxed{b} \equiv \boxed{a + b}$$

\boxed{a} is persistent

$$\frac{a \rightsquigarrow b}{\boxed{a} \Rightarrow \boxed{b}}$$

やがて Iris へ



Iris

第2部

Iris の発展的話題

第2部の流れ

- ◆ 2.1 高階幽霊状態と Later 様相
- ◆ 2.2 私の研究 Nola
- ◆ 2.3 Coq での機械化

2.1

高階幽靈状態と Later 様相

Iris は何の頭文字？

- ♦ 公式見解 — 何の頭文字でもない

Iris^{*}

- ♦ Derek “Iris Really Improves Separation logic”
- ♦ Impredicativity, Resources, Invariants, Separation

非可述性

リソース

不变条件

分離

高階幽霊状態

Iris のコア機能 高階幽靈状態

◆ 幽靈状態

- ▶ 検証用の仮想的状態、自由にカスタマイズ可能
 - 例 AUTH *Heap*、物理状態 $h \Leftrightarrow$ 幽靈状態 $\bullet h$

◆ 高階幽靈状態 [Jung+ '16]

Iris



- ▶ 分離論理式に依存、状態共有に便利 代表例: 不変条件・借用
- ▶ 素朴には iProp と状態の間で循環参照が発生、矛盾
 - Iris では矛盾回避のために Later 様相 $\triangleright P$ を使う

- ◆ \boxed{P} : 分離論理の命題 P を常に満たす状態を共有
 - ▶ 永続的、共有可変状態の表現に便利

例 ブール値への共有可変参照 $r : \text{ref bool} \triangleq \boxed{r \mapsto \text{true} \vee r \mapsto \text{false}}$

保証の作成 $\{ \top \} \text{ref true} \{ \lambda r. \boxed{r \mapsto \text{true} \vee r \mapsto \text{false}} \}_{\top}$

保証の遵守 $\{ \boxed{r \mapsto \text{true} \vee r \mapsto \text{false}} \} *r = \text{false} \{ \top \}_{\top}$

保証の利用 $\{ \boxed{r \mapsto \text{true} \vee r \mapsto \text{false}} \} *r \{ \lambda v. v = \text{true} \vee v = \text{false} \}_{\top}$

不变条件 と Later 様相 $\triangleright P$

- ♦ \boxed{P} は正確には $\triangleright P$ を共有する 以下 $\triangleright P$ と書く
 - ▶ Later 様相 $\triangleright P$: 1ステップ先の“世界”で P が成立 [Nakano '00]

証明規則

ae : 原子的な(1ステップの)式

$\triangleright P$ is persistent

$\triangleright P \Rightarrow_{\mathcal{E}} \triangleright P$

$$\frac{\{ \triangleright P * Q \} \text{ ae } \{ \lambda v. \triangleright P * \Psi v \}_{\emptyset}}{\{ \triangleright P * Q \} \text{ ae } \{ \Psi \}_{\top}}$$

$\Rightarrow_{\mathcal{E}}$ は不变条件アクセス用の更新
Iris では Fancy Update という

マスク \top, \emptyset で重複アクセスを阻止
一般には名前空間 \mathcal{N} により細かい制御

Later ▷ の性質

- 命題を弱める、冪等でない

$$P \Rightarrow \triangleright P$$

- Löb 帰納法が使える

► cf. ゲーデルの証明可能性論理 GL

$$\triangleright P \Rightarrow P \quad \triangleright P \Rightarrow \triangleright P$$

$$\frac{\triangleright P \Rightarrow P}{\top \Rightarrow P}$$

- 有用な可換則

$$\triangleright (\forall x. P_x) \equiv \forall x. \triangleright P_x \quad \triangleright \Box P \equiv \Box \triangleright P$$

Transfinite Iris
では不成立

$$\triangleright (P * Q) \equiv \triangleright P * \triangleright Q \quad \triangleright (\exists x. P_x) \equiv \exists x. \triangleright P_x$$

Timeless な命題

- ♦ 実質 Later で弱められない P is timeless $\triangleq \triangleright P \models \diamond P$
 - ▶ Timeless 命題の Later \triangleright は剥がせる (\diamond を吸収する場所なら)
 - Except-0 様相 $\diamond P \triangleq P \vee \triangleright \perp$ はモナディック、特に幂等 $\diamond \diamond P \equiv \diamond P$
 - ▶ 基本のトークン ($r \mapsto v$ 等) やそれらの組み合わせは Timeless
 - ▶ 不変条件 $\boxed{\triangleright P}$ 等は Timeless でない

$\triangleright r \mapsto v \Rightarrow_{\mathcal{E}} r \mapsto v$

$\triangleright \boxed{\triangleright P} \Rightarrow_{\mathcal{E}} \boxed{\triangleright P}$ つらい

不变条件のネスト

- ♦ ネストした参照を表現するのに自然に必要となる

$$r : \text{ref } (\text{ref bool}) \triangleq \boxed{\triangleright \exists s. r \mapsto s * \boxed{\triangleright (s \mapsto \text{true} \vee s \mapsto \text{false})}}$$

- ♦ 中を取り出したときに内側の Later \triangleright が残ってつらい

$$\left\{ \boxed{\triangleright \exists s. r \mapsto s * \boxed{\triangleright (s \mapsto \text{true} \vee s \mapsto \text{false})}} \right\}$$

$* r$

$$\left\{ \lambda s. \triangleright \boxed{\triangleright (s \mapsto \text{true} \vee s \mapsto \text{false})} \right\}_{\top} \text{どうする?}$$

回避策 Step-Indexing

[Nakano '00]

- ♦ プログラムの実行 1ステップ \Leftrightarrow Later \triangleright 1個
 - ▶ プログラムが進行すれば Later が剥がせる

$$\frac{e \hookrightarrow e' \quad \{P\} \ e' \ \{\Psi\}_{\varepsilon}}{\{\triangleright P\} \ e \ \{\Psi\}_{\varepsilon}}$$

1ステップ実行するごとに自己参照を Later で弱める

$$\text{pwp } e \{\Psi\} \approx (\exists v = e. \Psi v) \vee \triangleright \forall e' \leftarrow e. \text{pwp } e' \{\Psi\}$$

Later のガードにより不動点は一意

不变条件の例 ミューテクス

- ♦ 不変条件で自然に表現 & 検証できる
 - ▶ 共有される中身は任意の (Later 付き) 命題で記述できる

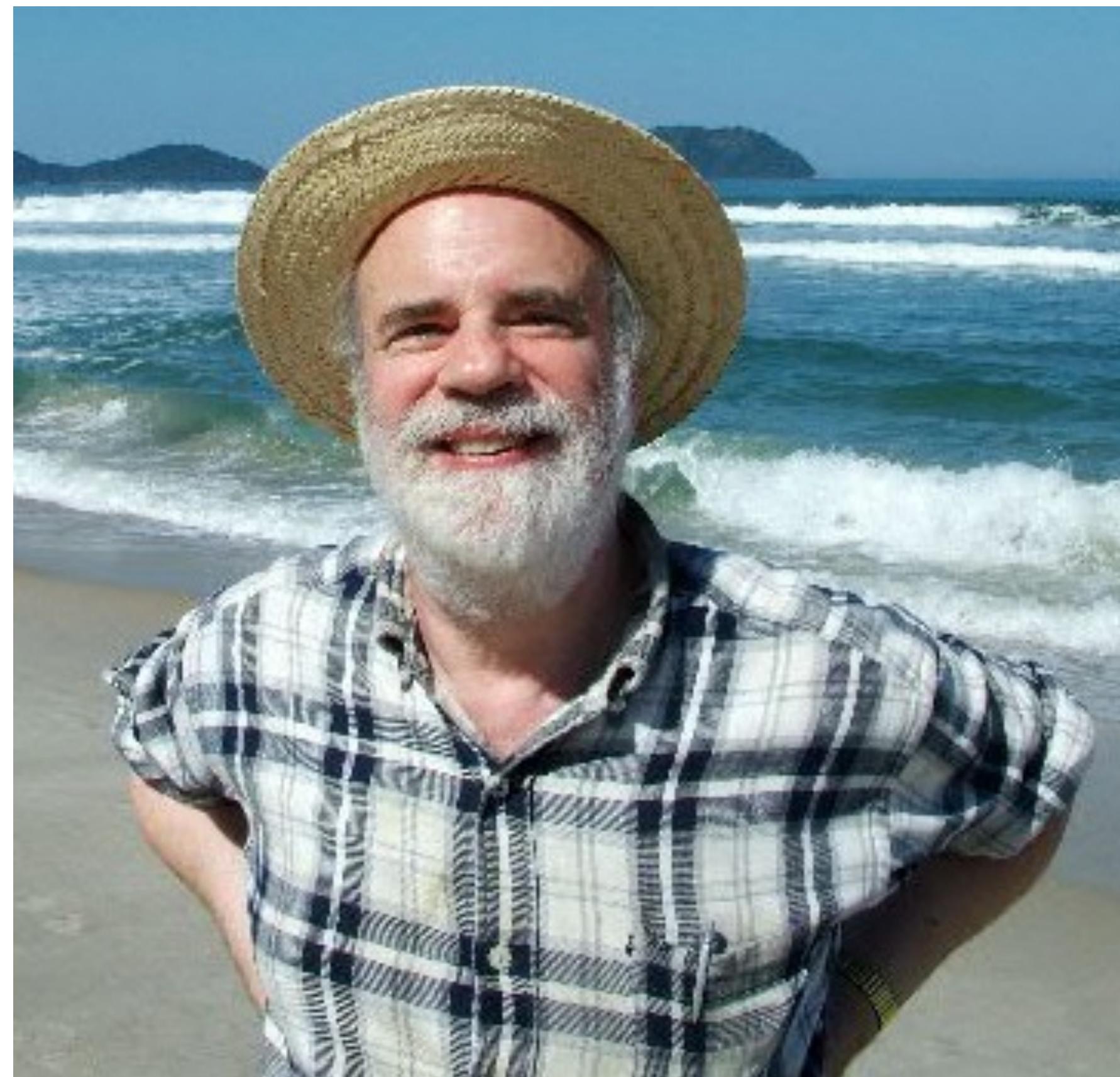
$$\text{mutex } \mathbf{r} \ P \triangleq \boxed{\triangleright ((\mathbf{r} \mapsto \text{false} * P) \vee \mathbf{r} \mapsto \text{true})}$$

$$\mathbf{r} \mapsto \text{false} * \triangleright P \Rightarrow_{\mathcal{E}} \text{mutex } \mathbf{r} \ P$$

$$\left\{ \text{mutex } \mathbf{r} \ P \right\} \text{cas } \mathbf{r} \ \text{false} \ \text{true} \ \left\{ \lambda b. \text{if } b \text{ then } \triangleright P \text{ else } \top \right\}_{\top}$$

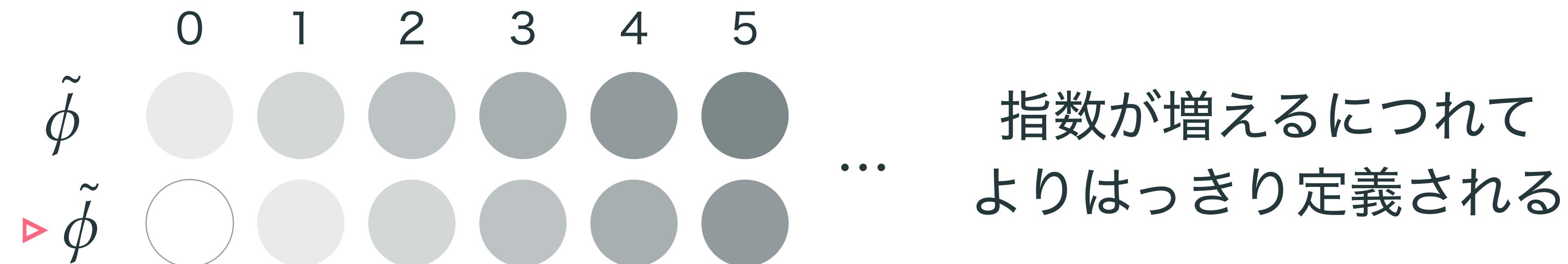
$$\left\{ \text{mutex } \mathbf{r} \ P * \triangleright P \right\} * \mathbf{r} = \text{false} \ \left\{ \top \right\}_{\top}$$

意味論も大事



- ♦ 論理の世界を指数 $0, 1, 2, \dots \in \mathbb{I}$ 上で徐々に構築

- ▶ $\text{Prop}^\sim \triangleq \mathbb{I}^{\text{anti}} \rightarrow \text{Prop}$ を使う、Later 様相 $\blacktriangleright \tilde{\phi} \triangleq \lambda i. \forall j < i. \tilde{\phi}_j$ で遅延



- ▶ 集合 A が指数付き: 指数付き等しさ $\tilde{\equiv}: A^2 \rightarrow \text{Prop}^\sim$ をもつ

- Later 関手 \blacktriangleright で遅延

$$\hat{a}: \blacktriangleright A ::= \text{next}(a: A)$$

$$\text{next } a \tilde{\equiv} \text{next } a' \triangleq \blacktriangleright (a \tilde{\equiv} a')$$

高階幽霊状態と Later

- ♦ iProp と状態の間の依存関係を Later ▶ で断つ

$$\cancel{State \triangleq ? F iProp}$$
$$iProp \triangleq State^\checkmark \rightarrow Prop$$

素朴にやると悪い循環参照

$$State \triangleq F (\triangleright iProp)$$
$$iProp \triangleq \tilde{State}^\checkmark \rightarrow \tilde{Prop}$$

不動点が一意に存在
[America & Rutten '89]

カメラ (指数付き RA) で状態を表現

$$\mathcal{A} = (A \text{ (indexed)}, +: A^2 \rightarrow A, \checkmark: A \rightarrow \tilde{Prop}, |\cdot|: A \rightarrow A^\perp)$$

不变条件のモデル

- ◆ 共有する中身をグローバルに持ち回る

$$\text{LINV} \triangleq \text{AUTH}(\mathbb{N} \xrightarrow{\text{fin}} \text{AG}(\triangleright iProp))$$

$$\boxed{\triangleright P} \triangleq \exists \iota. \boxed{\circ [\iota \leftarrow \text{ag}(\text{next } P)]}_{\text{LINV}}^{\gamma_{\text{LINV}}}$$

$$\begin{aligned} \Rightarrow_{\mathcal{E}} P &\triangleq \text{Wlinv} * \boxed{\mathcal{E}}_{\text{EN}}^{\gamma_{\text{EN}}} \equiv \\ &\diamond (\text{Wlinv} * \boxed{\mathcal{E}}_{\text{EN}}^{\gamma_{\text{EN}}} * P) \end{aligned}$$

Agreement カメラ
 $\text{AG } \mathcal{A}$

$$\begin{aligned} \text{ag } a &= \text{ag } a + \text{ag } a & |\text{ag } a| &= \text{ag } a \\ \checkmark (\text{ag } a + \text{ag } b) &\rightarrow a \tilde{=} b \end{aligned}$$

World Satisfaction: グローバルな不变条件

$$\begin{aligned} \text{Wlinv} &\triangleq \exists I: \mathbb{N} \xrightarrow{\text{fin}} iProp. \boxed{\bullet \text{ ag } I}_{\text{LINV}}^{\gamma_{\text{LINV}}} * \\ &\quad * \bigg(\big(\triangleright I[\iota] * \boxed{\{\iota\}}_{\text{DIS}}^{\gamma_{\text{DIS}}} \big) \vee \boxed{\{\iota\}}_{\text{EN}}^{\gamma_{\text{EN}}} \bigg) \end{aligned}$$

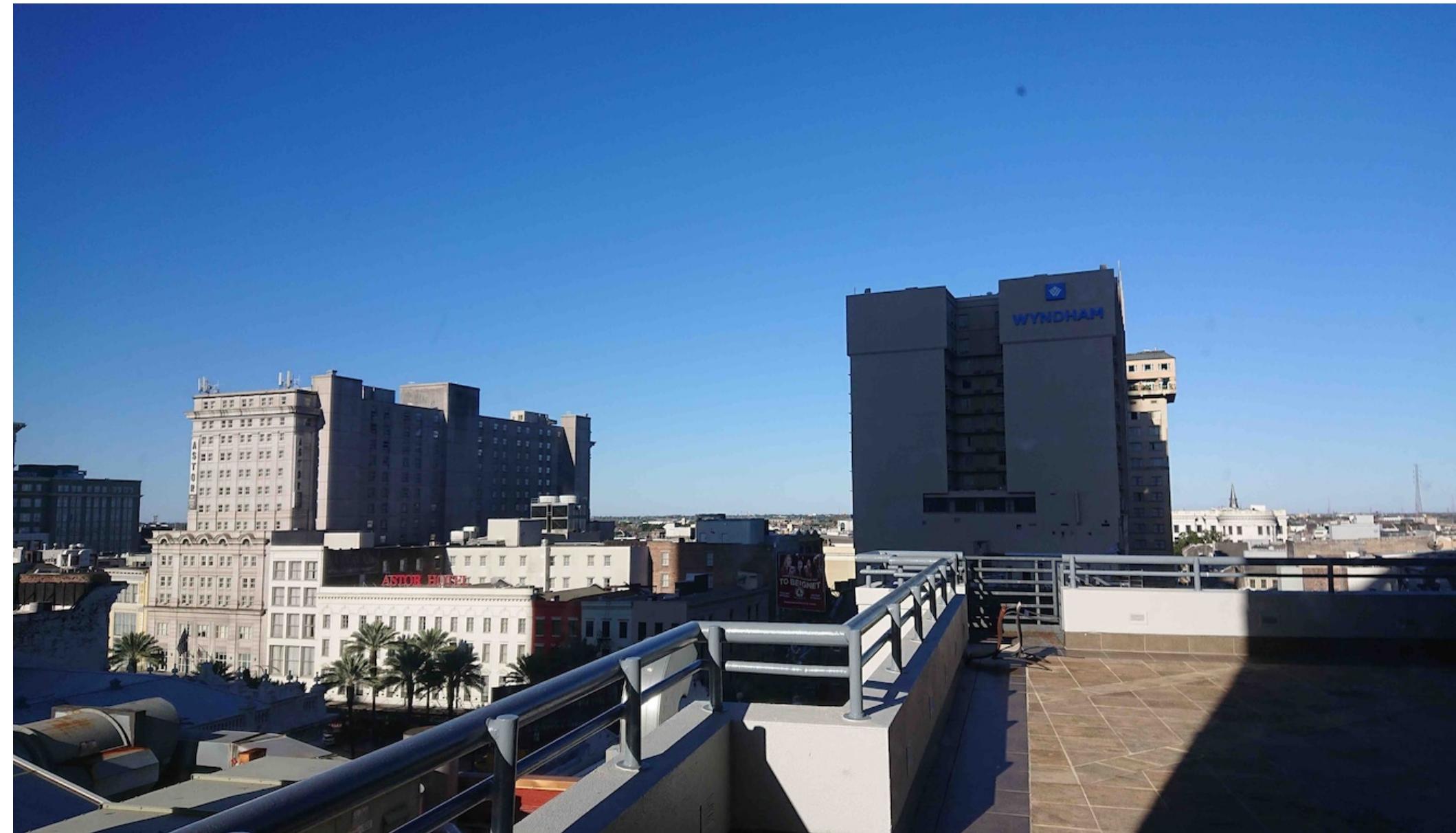
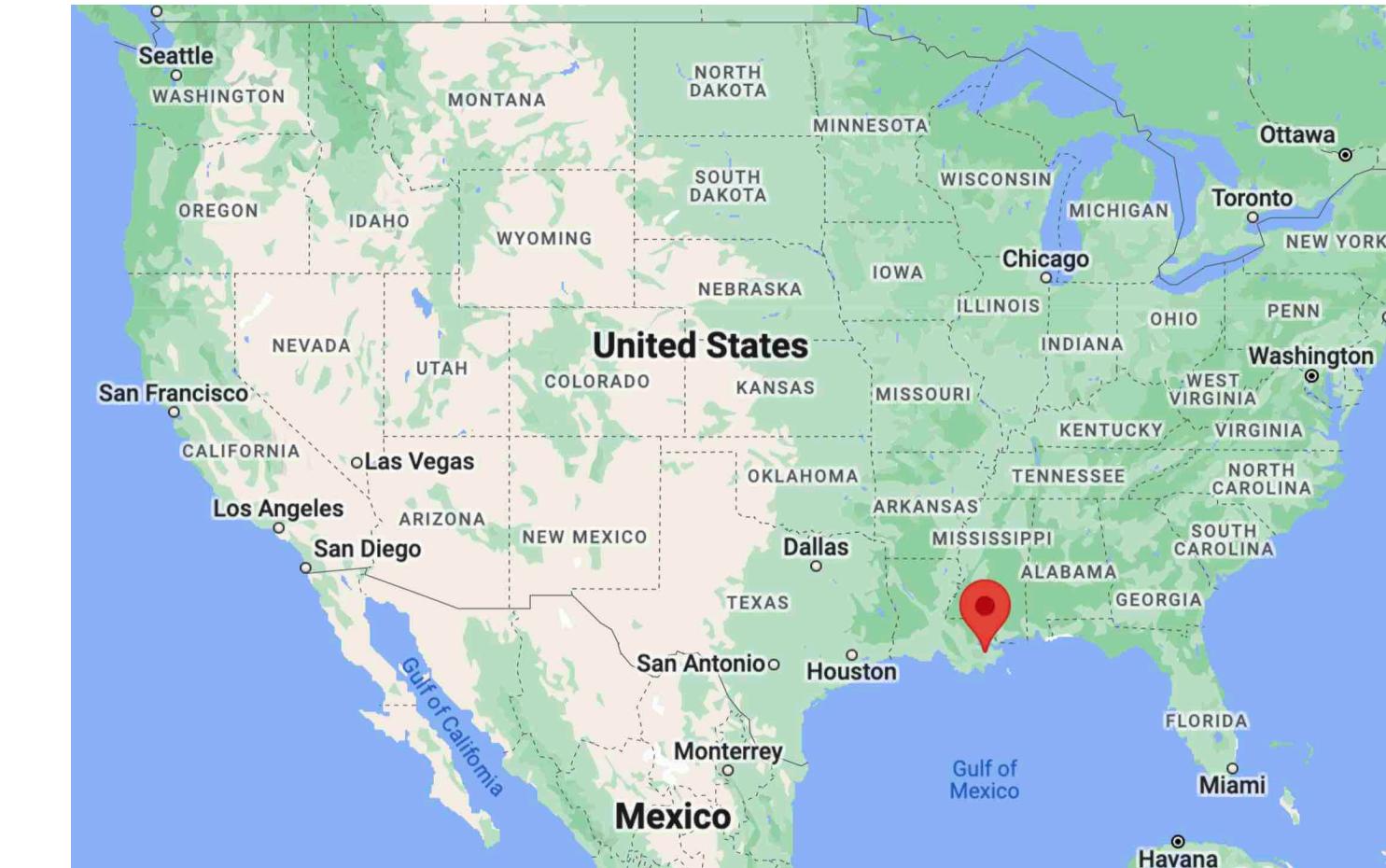
中身

$\text{State} \triangleq \prod_i (\mathbb{N} \xrightarrow{\text{fin}} \mathcal{A}_i) \quad \boxed{a}_{\mathcal{A}_i}^{\gamma} \triangleq \boxed{[i := [\gamma := a]]}$

2.2

私の研究 Nola

NOLA = New Orleans, Louisiana



- ◆ 背景: 高階幽霊状態の Later 様相 ▷ がつらい
 - ▶ 特に Liveness 検証で本質的困難
 - ▶ 素朴に Later を無くすとパラドクス
- ◆ Nola: 高階幽霊状態で Later ▷ を原則無くす
 - ▶ 特に参照をネストしても Later が一切出てこない
 - ▶ 特に 不変条件・借用 にフォーカス

Iris^{*}



分析1 Later を素朴に無くすとパラドクス

- ♦ Landin's knot の停止性が誤って証明できてしまう
 - ▶ Landin's knot: 関数への共有可変参照で自己参照、無限ループ

```
landin ≡ let r = ref fn () {}; *r = fn () { (*r)(); }; (*r)()
```

素朴な不变条件

$$P \Rightarrow_{\top} \boxed{P} \quad \frac{[P * Q] \text{ ae } [\lambda v. P * \Psi v]_{\emptyset}}{[\boxed{P} * Q] \text{ ae } [\Psi]_{\top}}$$

仮に認めると $[\top] \text{ landin } [\top]_{\top}$

証明: $\exists f. r \mapsto f * [\top] f() [\top]_{\top}$ を作る

分析2 Liveness 検証で Later が剥がせない

- ◆ Step-Indexing は Liveness 検証で原則使えない
 - ▶ 高階幽靈状態の Later 様相 ▶ が剥がせない

$$\frac{\text{loop} \hookrightarrow \text{loop} \quad \text{twp loop } [\perp] \Rightarrow \text{twp loop } [\perp]}{\text{twp loop } [\perp] \Rightarrow \text{twp loop } [\perp]}$$

Löb 帰納法 矛盾 !

$$\text{twp } e[\Psi] \approx_{\mu} (\exists v = e. \Psi v) \vee \forall e' \leftarrow e. \text{twp } e'[\Psi]$$

最小不動点

Nola のアイデア パラメタ付き高階幽靈状態

- ♦ 高階幽靈状態を論理式の型 Fml でパラメタ化
 - ▶ 循環参照 ▶ $iProp$ の箇所を一般化、 Later を回避

$$\begin{array}{c} \llbracket P \rrbracket \Rightarrow_{\mathcal{E}}^{\text{Winv}[\]} \boxed{P} \\ \Leftrightarrow_{\mathcal{E}}^W P \triangleq W \not\equiv_{\mathcal{E}} W * P \end{array} \quad \frac{[\llbracket P \rrbracket * Q] \text{ ae } [\lambda v. \llbracket P \rrbracket * \Psi v]_{\emptyset}^{\text{Winv}[\]}}{[\boxed{P} * Q] \text{ ae } [\Psi]_{\top}^{\text{Winv}[\]}}$$

$$State \triangleq F Fml$$

$$P : iProp$$

$$iProp \triangleq State^{\checkmark} \rightarrow Prop^{\sim} \quad \llbracket \rrbracket : Fml \rightarrow iProp$$

Nola での論理式の構築

- ♦ 論理式は構文データとして構築できる
 - ▶ 従来手法を包含: Later \triangleright の下では意味論的な命題が書ける

$$\begin{aligned} P, Q : Fml &::=_{\nu, \mu} P * Q \mid P -* Q \mid \forall_A \Phi \mid \exists_A \Phi \mid \phi \\ &\quad \mid r \mapsto v \mid \boxed{P} \ (P :_{\nu} Fml) \mid \check{\triangleright} \hat{P} \ (\hat{P} : \triangleright iProp) \end{aligned}$$

$$\begin{array}{ll} \llbracket P * Q \rrbracket \triangleq \llbracket P \rrbracket * \llbracket Q \rrbracket & \llbracket \forall_A \Phi \rrbracket \triangleq \forall a : A. \llbracket \Phi a \rrbracket \\ \llbracket r \mapsto v \rrbracket \triangleq r \mapsto v & \llbracket \boxed{P} \rrbracket \triangleq \boxed{P} \quad \llbracket \check{\triangleright} \hat{P} \rrbracket \triangleq \check{\triangleright} \hat{P} \end{array}$$

$$\begin{array}{l} \triangleright P \triangleq \check{\triangleright}(\text{next } P) \\ \check{\triangleright}(\text{next } P) \triangleq \triangleright P \end{array}$$

Nola で Liveness 検証

- ♦ Nola ならネストした参照でも Later ▶ が出ない！

$$\text{list } \Phi \ r \triangleq \boxed{\Phi \ r} * \boxed{\exists s. \ r+1 \mapsto s * \text{list } \Phi \ s}$$

$$[\llbracket \text{list } \Phi \ r \rrbracket] *_{(r+1)} [\lambda s. \llbracket \text{list } \Phi \ s \rrbracket]_{\top}^{\text{Winv}[\]}$$

検証例

$$\frac{\forall r. \ [\boxed{\Phi \ r}] f(r) [\top]_{\top}^{\text{Winv}[\]}}{[\llbracket \text{list } \Phi \ r \rrbracket * c \mapsto n] \ \text{iter}_c(r) [c \mapsto 0]_{\top}^{\text{Winv}[\]}}$$

```
fn iter_c(r) {
  if *c > 0 {
    f(r); *c -= 1;
    iter_c(*(r+1)) } }
```

表現力とパラドクス

- ♦ Nola 用のホーア三つ組は Fml に直接入れられない
 - ▶ 解釈が循環参照をもち ill-defined になるため → パラドクス回避

$$\llbracket [P] e [\Psi]_{\mathcal{E}} \rrbracket \triangleq \llbracket [P] \rrbracket e \llbracket [\Psi] \rrbracket_{\mathcal{E}}^{\text{Winv}} \llbracket$$

- ▶ 階層化などの工夫をすれば入れられる

$$\llbracket [P] e [\Psi]^0_{\mathcal{E}} \rrbracket_1 \triangleq \llbracket [P]_1 \rrbracket e \llbracket [\Psi]_1 \rrbracket_{\mathcal{E}}^{\text{Winv}} \llbracket \rrbracket_0 \quad Fml_0, Fml_1 \\ \llbracket \rrbracket_i : Fml_i \rightarrow iProp$$

発展的話題 意味論的書き換え

- ♦ 高階幽靈状態の論理式を意味論的に書き換えたい
 - ▶ Nola では論理式 Fml と解釈 $\llbracket \cdot \rrbracket$ が別なので難しい

$$\llbracket \boxed{P * Q} \rrbracket \Leftrightarrow \llbracket \boxed{Q * P} \rrbracket$$

$$\llbracket \boxed{P * Q} \rrbracket \Leftrightarrow \llbracket \boxed{Q * P} \rrbracket$$

$$\frac{\alpha \sqsubseteq \beta \quad \beta \sqsubseteq \alpha}{\llbracket \&^\alpha P \rrbracket \Leftrightarrow \llbracket \&^\beta P \rrbracket}$$

~~$$\llbracket \boxed{P} \rrbracket = \exists Q \text{ s.t. } \llbracket P \rrbracket \Leftrightarrow \llbracket Q \rrbracket. \boxed{Q}$$~~

循環参照

発見 擬似最大不動点 psg

- ◆ 任意の写像 $f: \mathcal{L} \rightarrow \mathcal{L}$ (\mathcal{L} : 完備束) に対して定義可能
 - ▶ $\text{Psgoid } f : f$ と (無限でもよい) 交わりで閉じている最小の集合
 - ▶ 擬似最大不動点 $\text{psg } f : \text{Psgoid } f$ の最小元 ($= \wedge \text{Psgoid } f$)

psg は後不動点

$$\text{psg } f \leq f(\text{psg } f)$$

Psgoid の元は分解可能

$$\frac{a \in \text{Psgoid } f}{a = \wedge \{ f b \mid b \in \text{Psgoid } f, a \leq f b \}}$$

f が単調のとき $\text{Psgoid } f = \{ f^\alpha \top \mid \alpha \}$ $\text{psg } f = \text{gfp } f$

psg-導出可能性による意味論的書き換え

◆ 導出可能性 der を psg で構築

判断 $J: \text{Judg} ::= P \Leftrightarrow Q$

$d: \text{Judg} \rightarrow i\text{Prop}$

解釈を d で
パラメタ化

$$\llbracket \boxed{P} \rrbracket_d \triangleq \exists Q \text{ s.t. } d(P \Leftrightarrow Q). \boxed{Q}$$

$$\llbracket P \Leftrightarrow Q \rrbracket_d^+ \triangleq \llbracket P \rrbracket_d \Leftrightarrow \llbracket Q \rrbracket_d$$

$$\llbracket P * Q \rrbracket_d \triangleq \llbracket P \rrbracket_d * \llbracket Q \rrbracket_d \quad \llbracket r \mapsto v \rrbracket_d \triangleq r \mapsto v$$

導出可能性 $\text{der} \triangleq \text{psg} \llbracket \rrbracket^+$ $\text{Deriv} \triangleq \text{Psgoid} \llbracket \rrbracket^+$ $\llbracket \rrbracket \triangleq \llbracket \rrbracket_{\text{der}}$

意味論的
書き換え
$$\frac{d \in \text{Deriv} \quad \forall d' \in \text{Deriv}. \llbracket P \rrbracket_{d'} \Leftrightarrow \llbracket Q \rrbracket_{d'}}{\llbracket \boxed{P} \rrbracket_d \Leftrightarrow \llbracket \boxed{Q} \rrbracket_d}$$

ちゃんと使える

$$\frac{[\llbracket \boxed{P} \rrbracket * Q] \text{ ae } [\lambda v. \llbracket \boxed{P} \rrbracket * \Psi v]^\text{Winv}_\emptyset}{[\llbracket \boxed{P} \rrbracket * Q] \text{ ae } [\Psi]^\text{Winv}_\top}$$

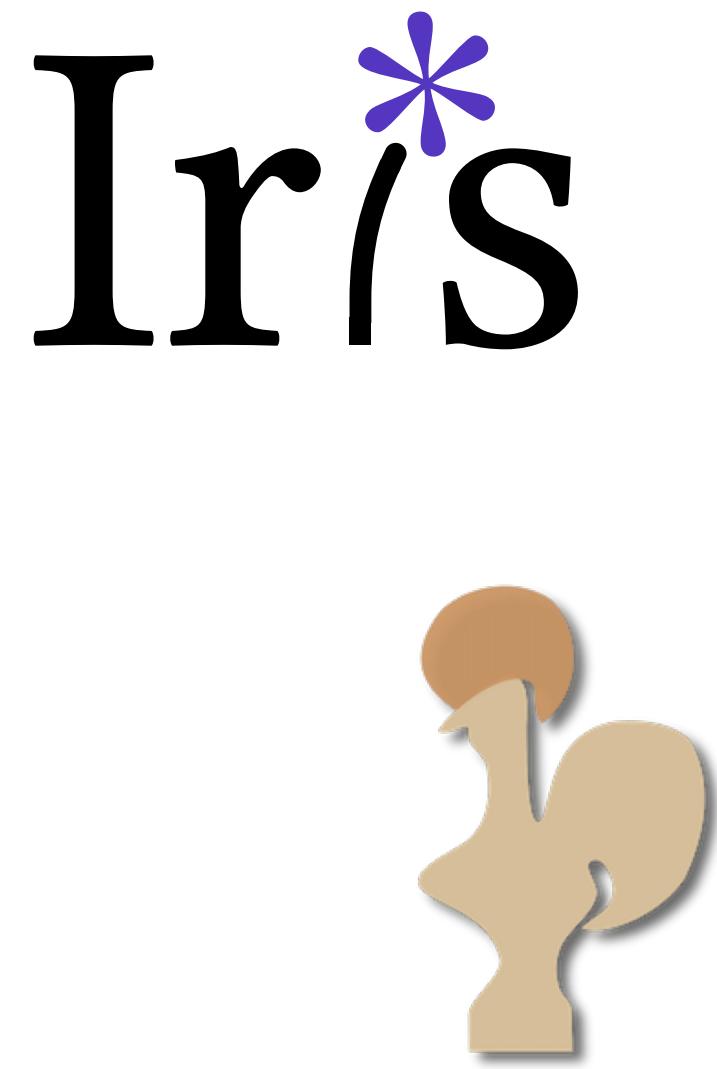
Takeout 自己参照との戦い

- ◆ 自己参照は上手く行くときはとても便利
 - ▶ 調子に乗ると矛盾を引き起こす
- ◆ 論理の設計はしばしば自己参照との戦い
 - ▶ パラメタ化するのは悪い自己参照を避ける常套手段

2.3

Coq での機械化

Iris Coq



Iris / Iris

Iris ●

Project information

The Coq development for Iris. [project website] [coqdoc]

pipeline passed

-o 8,191 Commits

89 101 Branches

18 Tags

README

BSD 3-Clause "New" or "Revised" License

CHANGELOG

CONTRIBUTING

CI/CD configuration

Wiki

Created on

September 23, 2015

Code

master / iris / +

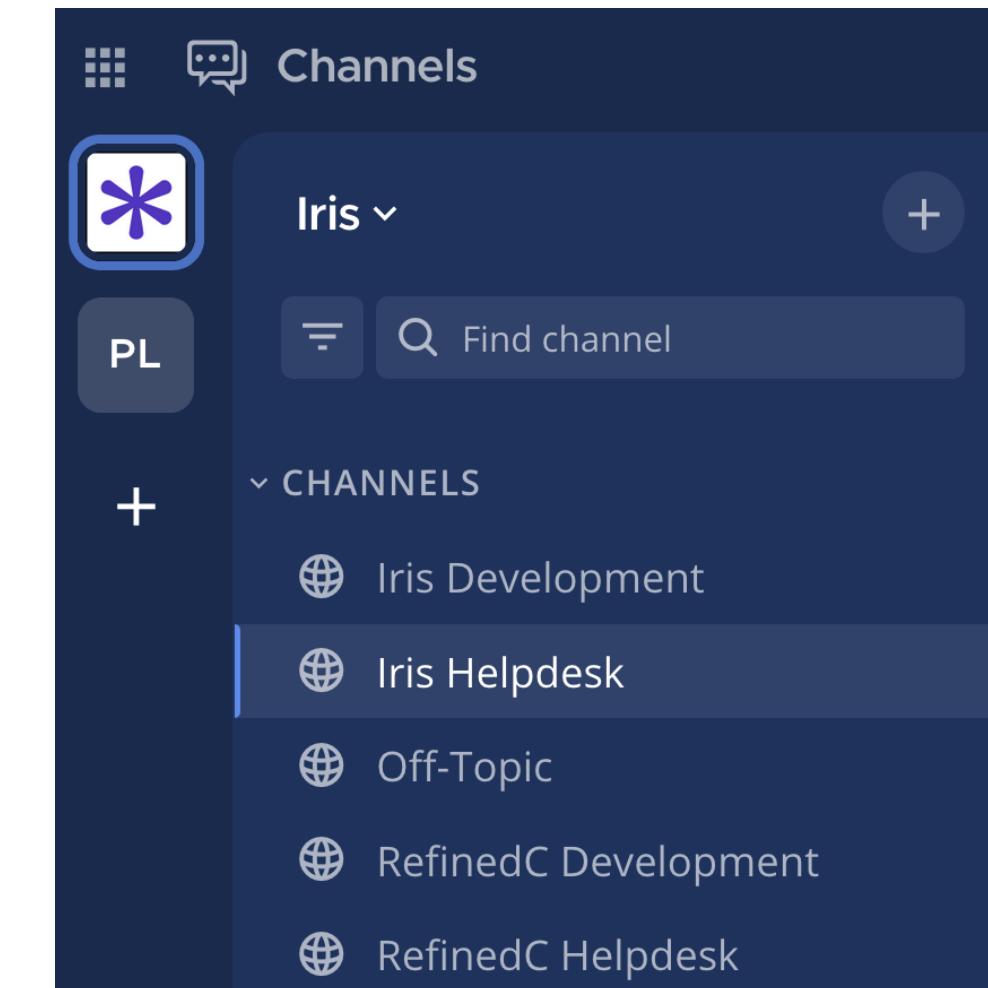
Bump std++. Robbert Krebbers authored 2 days ago f6ed0921

| Name | Last commit | Last update |
|---------------------|-------------------------------|-------------|
| .gitlab/issue_te... | add issue template | 5 years ago |
| docs | Reformat style guide | 2 days ago |
| iris | Merge branch 'discret... | 3 days ago |
| iris_deprecated | Use `#[local] field ::` to... | 3 weeks ago |
| iris_heap_lang | Update code to use ne... | 2 weeks ago |
| iris_unstable | Bump std++ (length_X). | 3 weeks ago |
| tests | Some tests for legacy ... | 2 weeks ago |
| tex | docs: typo fixes | 1 month ago |
| .gitattributes | sync gitattributes with ... | 3 years ago |

Iris の活発なコミュニティ

♦ Iris Mattermost

- Iris Helpdesk で活発な質疑
- 他のチャンネルでも面白い議論

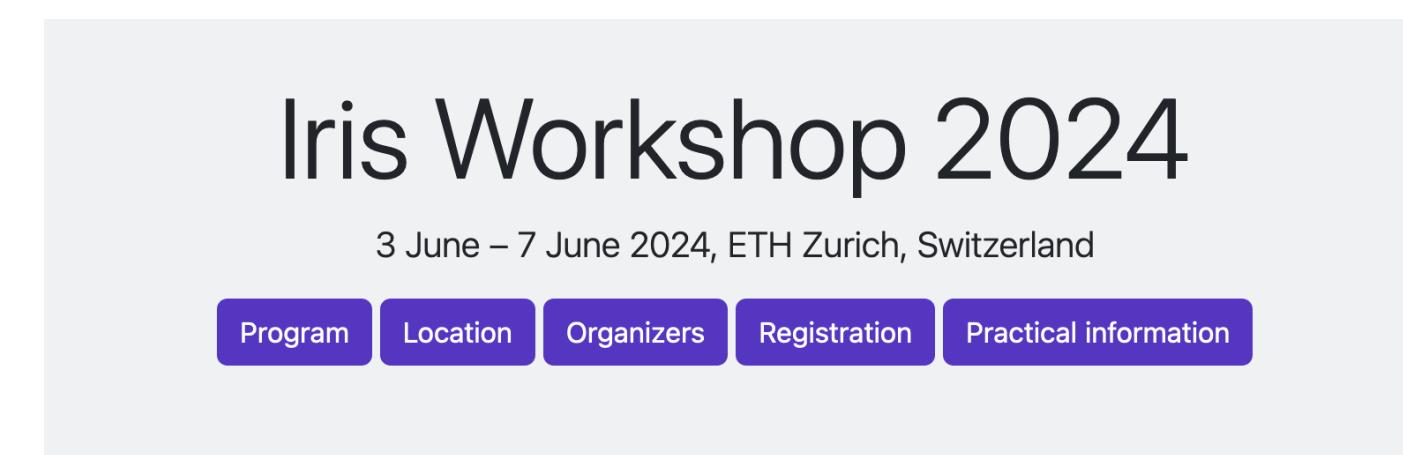


 **Mattermost**

A screenshot of the Iris Helpdesk channel in Mattermost. The channel has 279 members and 1 message. A conversation is shown between three users: Robbert Krebbers, François Pottier, and Simon Spies. Robbert Krebbers asks a question about Iris-related names. François Pottier replies quickly, and Simon Spies responds with a detailed explanation about fair scheduling and termination proofs.

♦ Iris Workshop

- 毎年ヨーロッパで開催
- 3日間の充実したトーク



The fourth edition of the Iris workshop will be held in Zurich and it will be focused on Iris-related research to allow for in-depth talks and discussions. There will be a three-day long program with talks from Monday 3 June until Wednesday 5 June. Participants are encouraged to stay 1-2 days longer if they can, for more informal discussions and interaction.



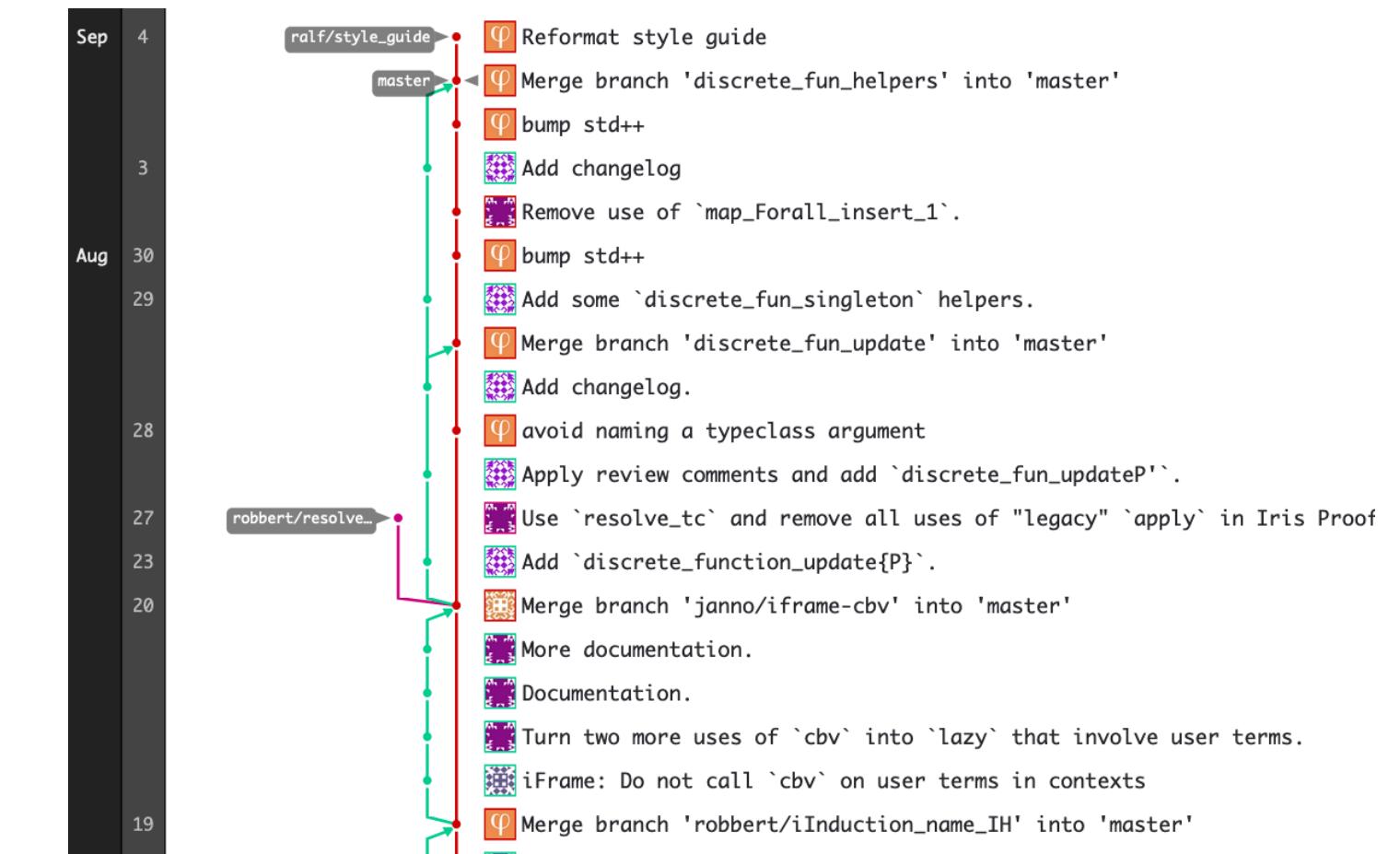
The Pulse Programming Language

4th Iris Workshop
Aseem Rastogi, Gabriel Ebner, Guido Martínez, Nik Swamy, Tahina Ramananandro
Megan Frisella, Thibault Dardinier, and soon Jonas Fiala
June 2024

Iris Coq 開発状況

♦ 活発な開発

- ▶ 2014年から開発、現在 ver. 4.2
- ▶ 日々活発なコミット、累計 $\geq 8k$



♦ 強固な実装

- ▶ CI/CD 整備、opam 対応
- ▶ 充実したドキュメント
- ▶ 最新の Coq 機能を活用

| Status | Pipeline | Created by | Stages |
|--|--|------------|--------|
| Passed ⌚ 00:06:13 ⌚ 1 hour ago | Merge branch 'discrete_fun_helpers' into 'mast... #106535 ⌚ master -> a92765d3 ⓘ scheduled latest | | |
| Passed ⌚ 00:04:59 ⌚ 3 hours ago | Reformat style guide #106523 ⌚ 1072 -> a658b538 ⓘ latest merge request | | |
| Passed ⌚ 00:04:06 ⌚ 6 hours ago | Merge branch 'discrete_fun_helpers' into 'mast... #106514 ⌚ master -> a92765d3 ⓘ latest | | |
| Passed ⌚ 00:04:53 ⌚ 8 hours ago | bump std++ #106505 ⌚ master -> 0653ba6c ⓘ | | |

proof_guide.md 7.93 KiB

Blame Edit Replace Delete

Iris Proof Guide

This work-in-progress document serves to explain how Iris proofs are typically carried out in Coq: what are the common patterns and conventions, what are the common pitfalls. This complements the tactic documentation for the [proof mode](#) and [HeapLang](#).

Order of Requires

In Coq, declarations in modules imported later may override the previous definition. Therefore, in order to make sure the most relevant declarations and notations always take priority, we recommend importing dependencies from the furthest to the closest.

Coq std++

- ♦ Iris チームが開発、Iris Coq の基盤のライブラリ
 - ▶ 標準ライブラリを拡張、公理を避けつつ様々な便利機能
 - 決定アルゴリズムによる場合分け、正規化による等式の成立

The screenshot shows the GitHub interface for the stdpp repository. At the top, there's a header with a star icon, a fork icon, and a 'mat' profile picture. Below it is a search bar and a sidebar with project navigation links like 'Project', 'Issues', 'Merge requests', 'Code', 'Build', 'Deploy', 'Monitor', and 'Help'. The main area displays a list of commits. A prominent commit by Robbert Krebbers is shown: 'Merge branch 'robbert/sorted_unique' into 'master''. Below the commit list is a table showing recent activity across various files and branches.

| Name | Last commit | Last update |
|-----------------|-----------------------------------|--------------|
| docs | docs: fix typo | 2 months ago |
| stdpp | Add variants of `Sorted_u... | 1 day ago |
| stdpp_bitvector | Rename `X_length` into `le... | 3 weeks ago |
| stdpp_unstable | Allow compiling the packa... | 2 months ago |
| tests | Test cases for string notat... | 1 day ago |
| .gitattributes | try to fix shell script line e... | 3 years ago |

Coq-std++ [coqdoc](https://plv.mpi-sws.org/coqdoc/stdpp/)(<https://plv.mpi-sws.org/coqdoc/stdpp/>)

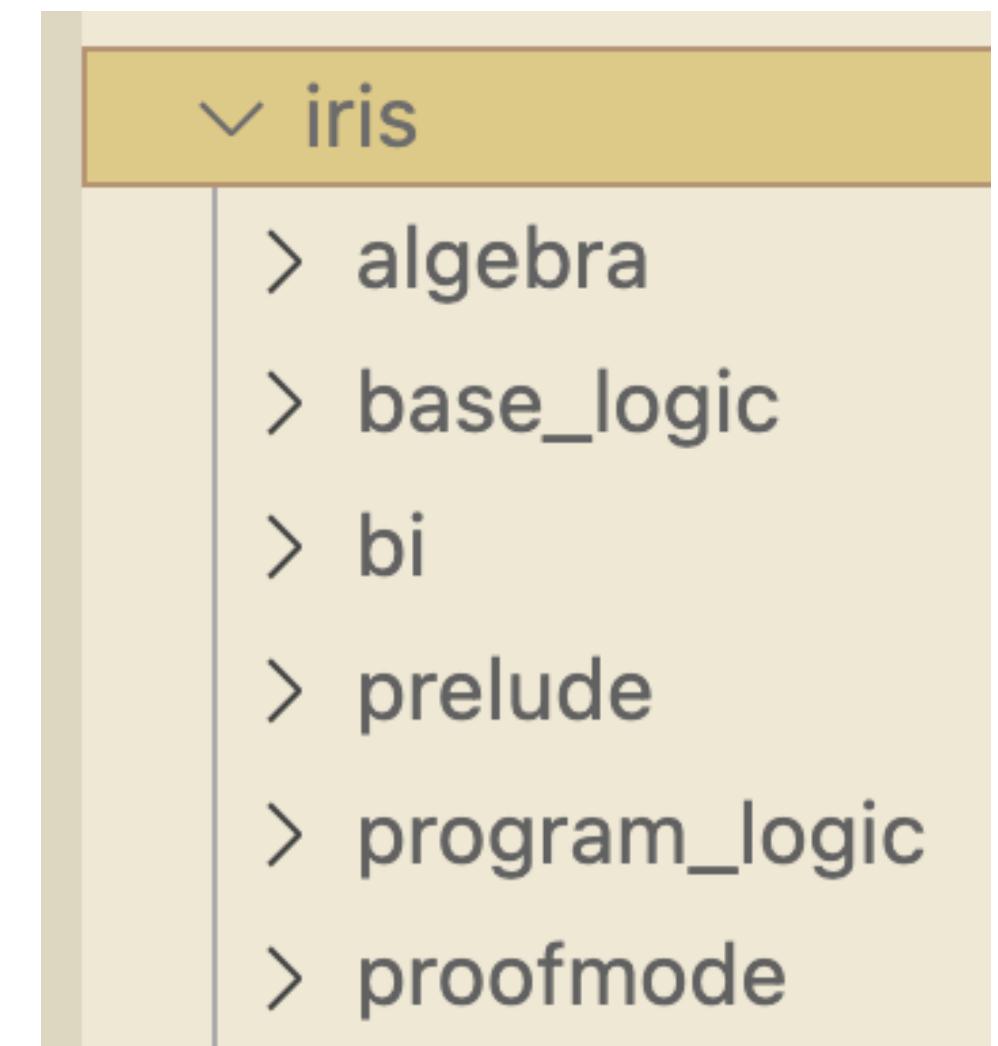
This project contains an extended "Standard Library" for Coq called coq-std++. The key features of this library are as follows:

- It provides a great number of definitions and lemmas for common data structures such as lists, finite maps, finite sets, and finite multisets.
- It uses type classes for common notations (like \emptyset , \mathbb{U} , and Haskell-style monad notations) so that these can be overloaded for different data structures.
- It uses type classes to keep track of common properties of types, like it having decidable equality or being countable or finite.
- Most data structures are represented in canonical ways so that Leibniz equality can be used as much as possible (for example, for maps we have $m1 = m2 \text{ iff } \forall i, m1 !! i = m2 !! i$). On top of that, the library provides setoid instances for most types and operations.
- It provides various tactics for common tasks, like an ssreflect inspired `done` tactic for finishing trivial goals, a simple breadth-first solver `naive_solver`, an equality simplifier `simplify_eq`, a solver `solve_proper` for proving compatibility of functions with respect to relations, and a solver `set_solver` for goals involving set operations.
- It is entirely dependency- and axiom-free.

Iris Coq 実装の構造

◆ 色々な汎用的コンポーネント

- ▶ `iris.bi` : 分離論理一般のライブラリ
- ▶ `iris.proofmode` : Iris Proof Mode
 - 分離論理一般でインタラクティブな証明
- ▶ `iris.algebra` : RA / Camera のライブラリ
- ▶ `iris.base_logic` : いわゆる Iris の分離論理
- ▶ `iris.program_logic` : 言語に依らない汎用プログラム論理



- ♦ 分離論理における自然なインタラクティブな証明

- ▶ 分離論理の仮定に名前をつけて操作できる
 - 空間的な仮定は使うと消費される

```
"Hlock" : is_lock γ lk R
"Hlocked" : locked γ
"HR" : R
"ΗΦ" : ▷ (True -* Φ #())
-----*
WP release lk {{ v, Φ v }}
```

```
"Hinv" : inv N (lock_inv γ l R)
-----
"Ηlocked" : locked γ
"ΗR" : R
"ΗΦ" : ▷ (True -* Φ #())
-----*
WP release #l {{ v, Φ v }}
```

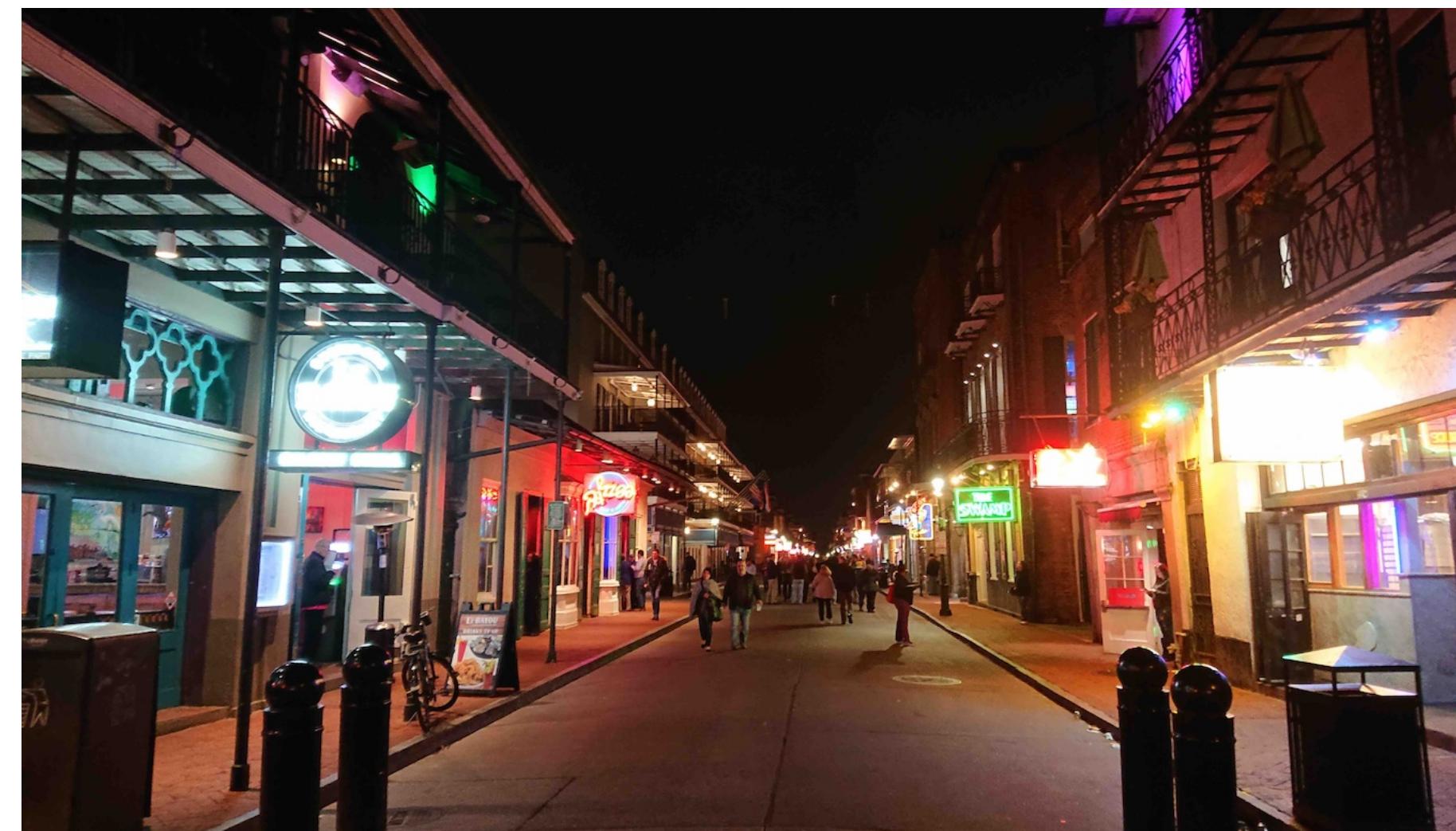
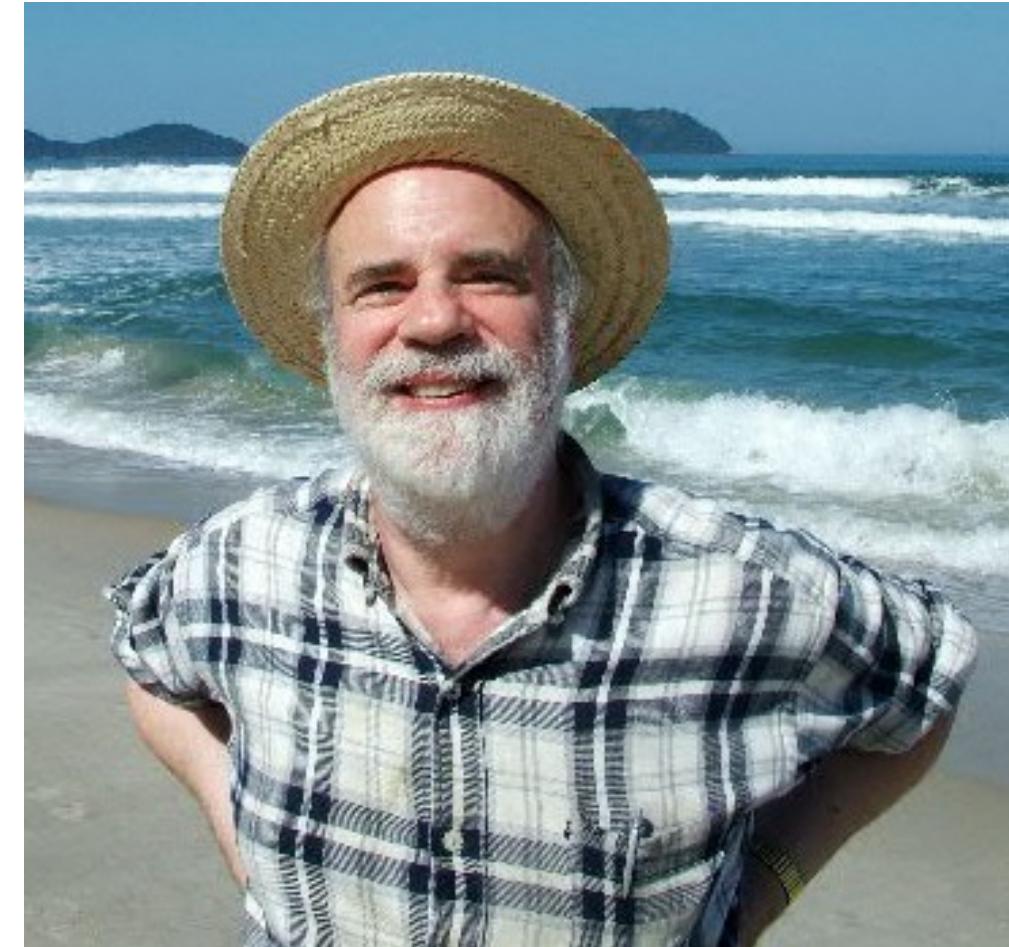
iDestruct "Hlock" as (l ->) "#Hinv".

Iris Coq デモ

♦ Iris Coq で遊んでみる

Iris の世界を楽しもう

Iris *



第3部

Iris × Rust

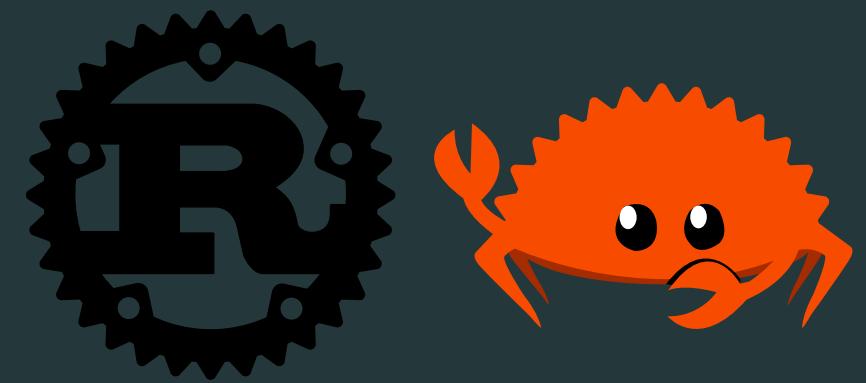
第3部の流れ

- ◆ 3.1 Rust プログラミング言語
- ◆ 3.2 RustBelt
- ◆ 3.3 私の研究 RustHorn+Belt

3.1

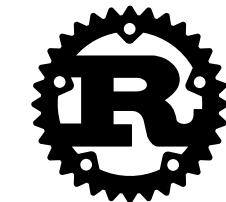
Rust プログラミング言語

Rust が今熱い



♦ 高性能 × 安全性 × 開発効率 の言語、'15 誕生

- ▶ C ('72~)・C++ ('85~) をよりモダンに、安全に
- ▶ 借用 [Grossman+ '02] にもとづく所有権型の力

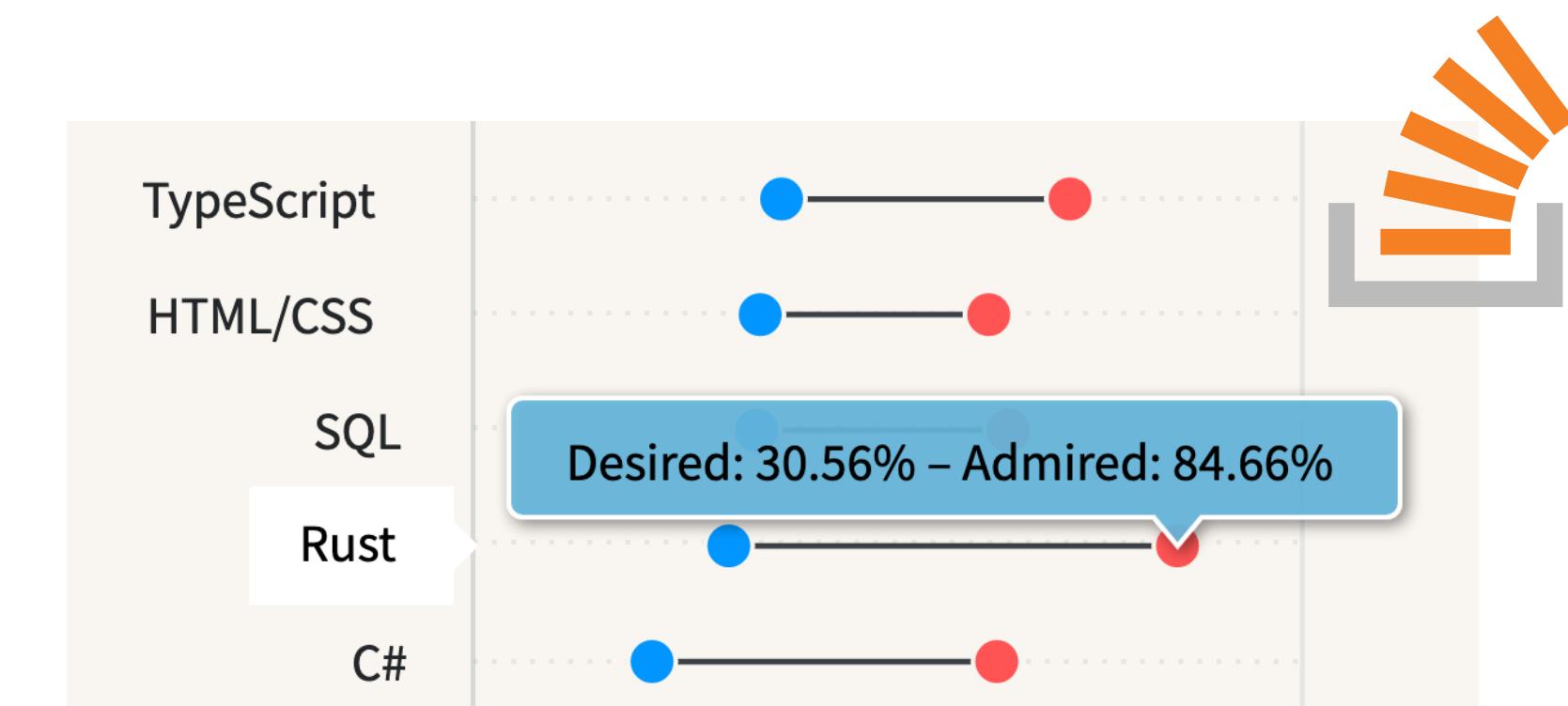
Rust 

効率的で信頼できるソフトウェアを誰もがつくれる言語

なぜRustか？

| パフォーマンス | 信頼性 | 生産性 |
|--|---|--|
| Rustは非常に高速でメモリ効率が高くランタイムやガベージコレクタがないため、パフォーマンス重視のサービスを実装できます。組込み機器上で実行したり他の言語との調和も簡単にできます。 | Rustの豊かな型システムと所有権モデルによりメモリ安全性とスレッド安全性が保証されます。さらに様々な種類のバグをコンパイル時に排除することができるです。 | Rustには優れたドキュメント、有用なエラーメッセージを備えた使いやすいコンパイラ、および統合されたパッケージマネージャとビルドツール、多数のエディタに対応するスマートな自動補完と型検査機能、自動フォーマッタといった一流のツール群が数多く揃っています。 |

はじめる [バージョン 1.78.0](#)



StackOverflow 調査では
8年連続 最も Admired な言語

世界で活躍する Rust

♦ 現実のソフトウェアで Rust が活躍

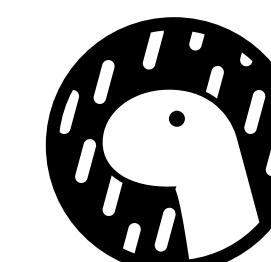
- ▶ Android は Rust を採用 → 脆弱性減
- ▶ Firecracker VM・ChromeOS VM
- ▶ Firefox の描画エンジン Servo
- ▶ Deno・Wasmer 等の言語ランタイム



Firecracker



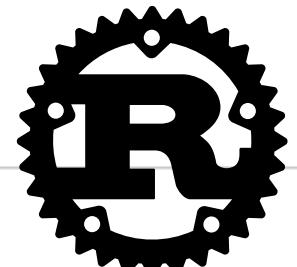
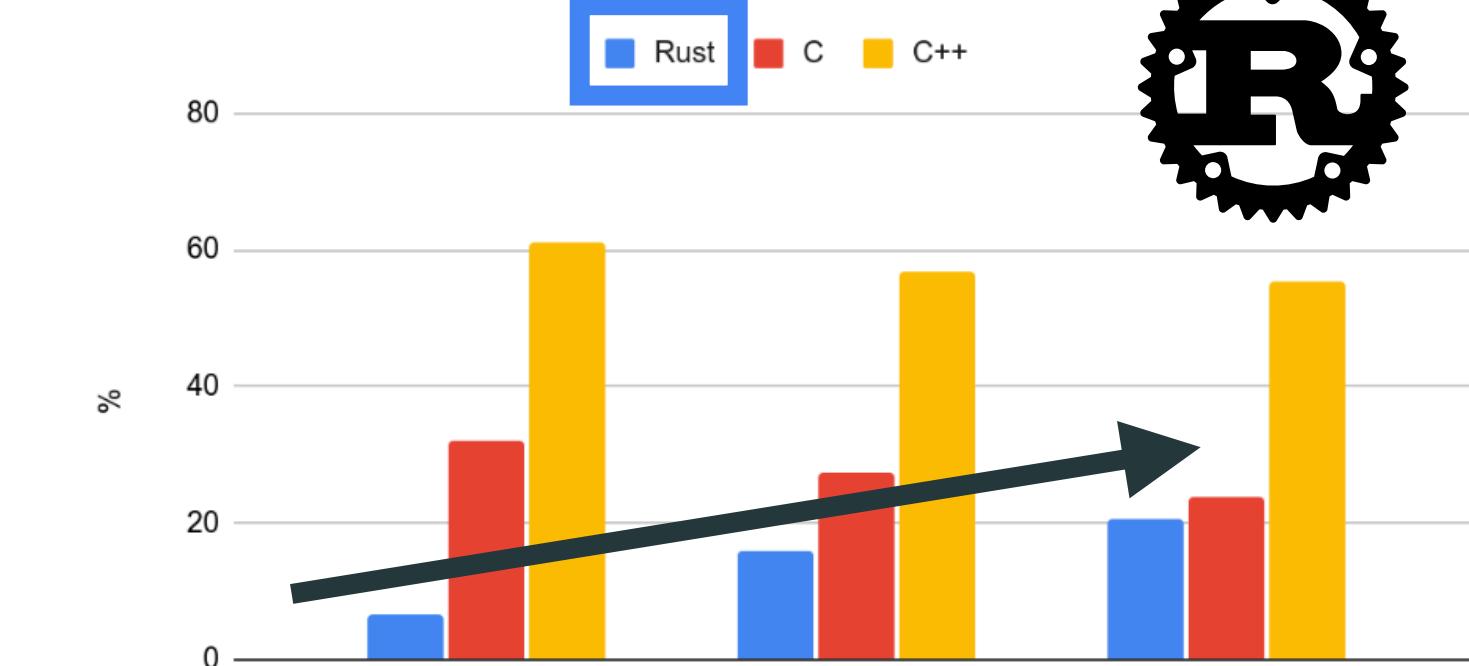
Firefox Browser



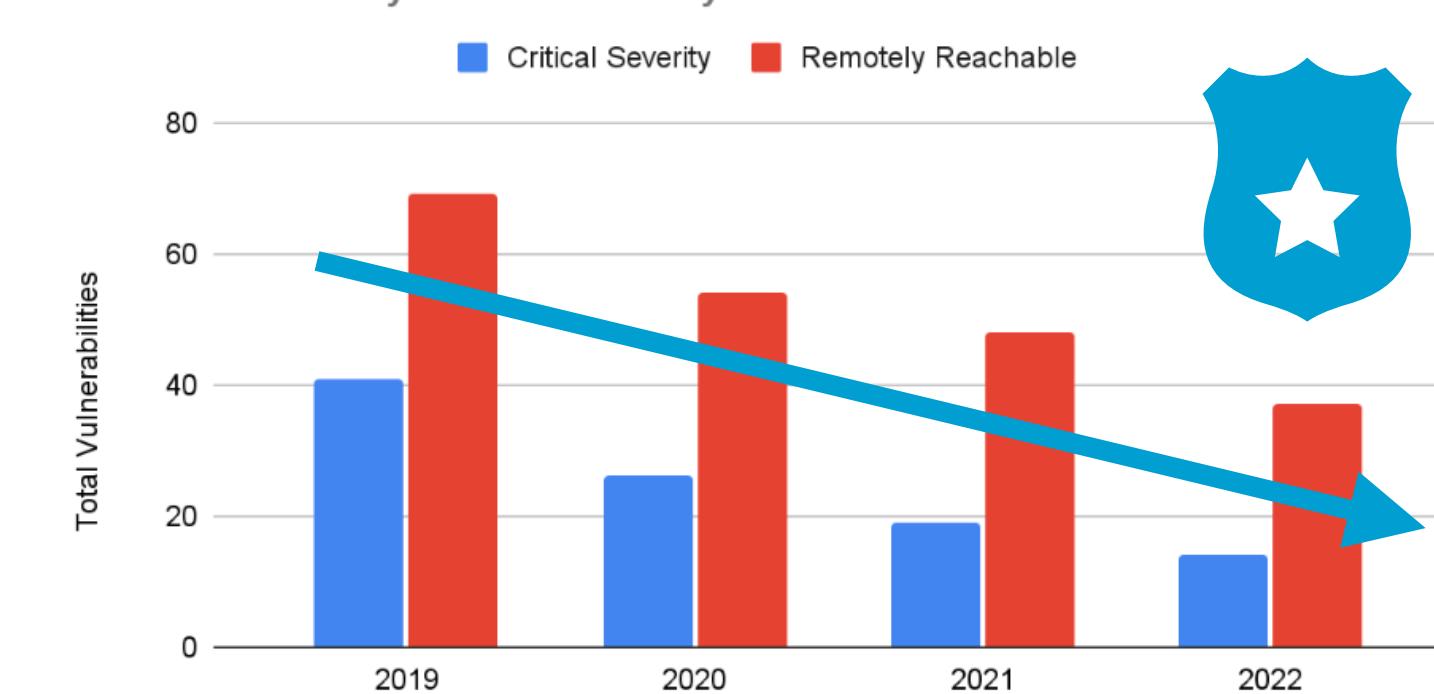
Deno



New Native Code



Critical-Severity and Remotely Reachable Vulnerabilities



Rust 財団

- ♦ Rust 財団が Mozilla から独立し '21 誕生
 - ▶ AWS・Google・Huawei・Meta・Microsoft が出資



**Google Contributes \$1M to
Rust Foundation to Support
C++/Rust “Interop Initiative”**

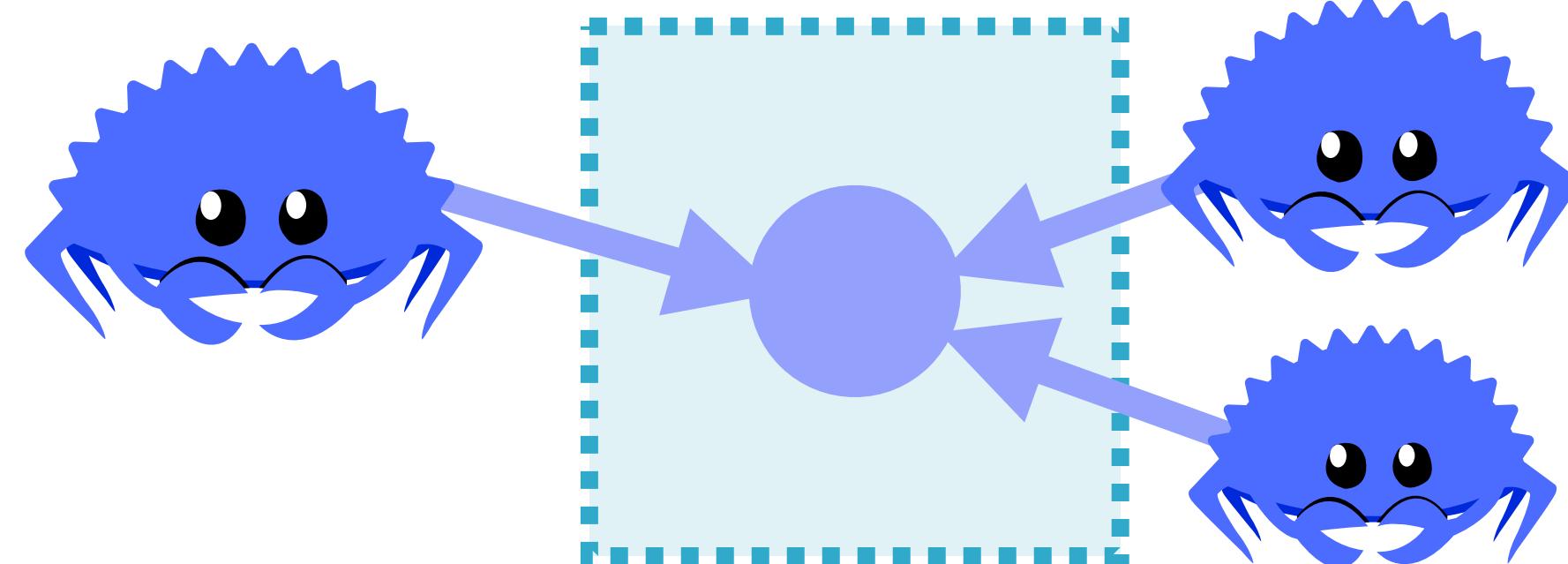
Rust の特長

- ◆ 高性能 — C/C++ と同等
 - ▶ 直接のメモリ・スレッド操作、GC がデフォルトでない
- ◆ 安全性 — 過去に類をみないレベル
 - ▶ 高度な所有権型、借用・unsafe・内部可変性で柔軟に
- ◆ 開発効率 — 現代的な機能とツール
 - ▶ 型推論、パターンマッチ、トレイト、衛生的マクロ、…
 - ▶ 包括的パッケージマネージャ、LSP での VS Code 連携、…

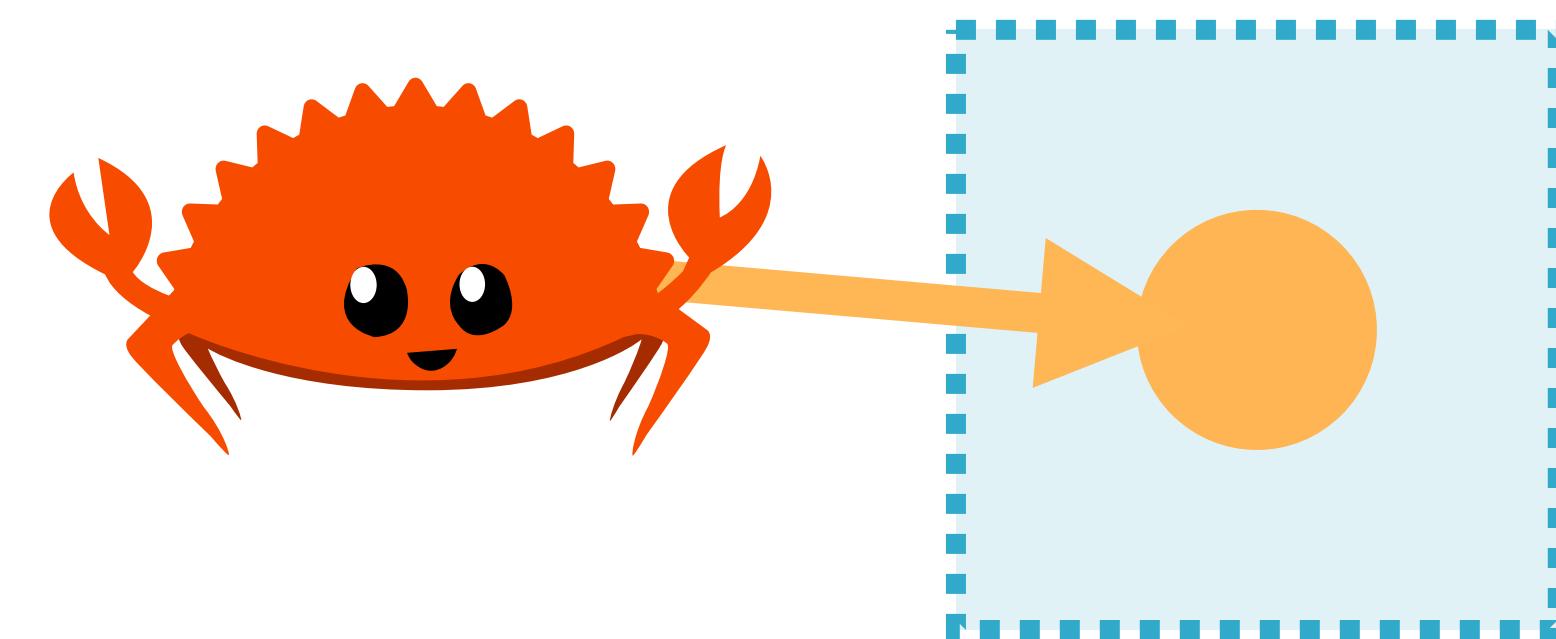
Rust の基本思想 共有 xor 可変

- ♦ 各領域は 共有 か 可変 のいずれか
 - ▶ アクセス競合がなくなり安全になる

共有かつ不变



独占かつ可変



Rust の所有権型

◆ 型に所有権情報 共有 xor 可変を静的に検証

- ▶ 保証: アクセス競合がない、メモリ・スレッド安全
- ▶ プログラマに優しい自動で軽量の形式検証
- ▶ 成功の3つの鍵 — 借用・unsafe・内部可変性

アカデミックな研究

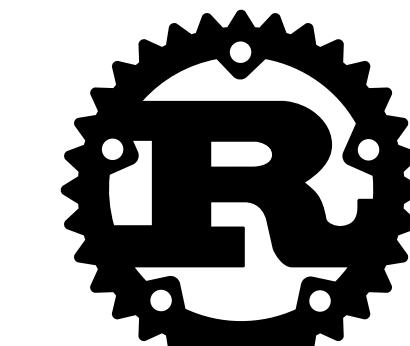
リージョン型 [Tofte&Talpin '97]

所有権型 [Clarke+ '98]

Cyclone [Grossman+ '02]



初めて本格的に実用



Rust

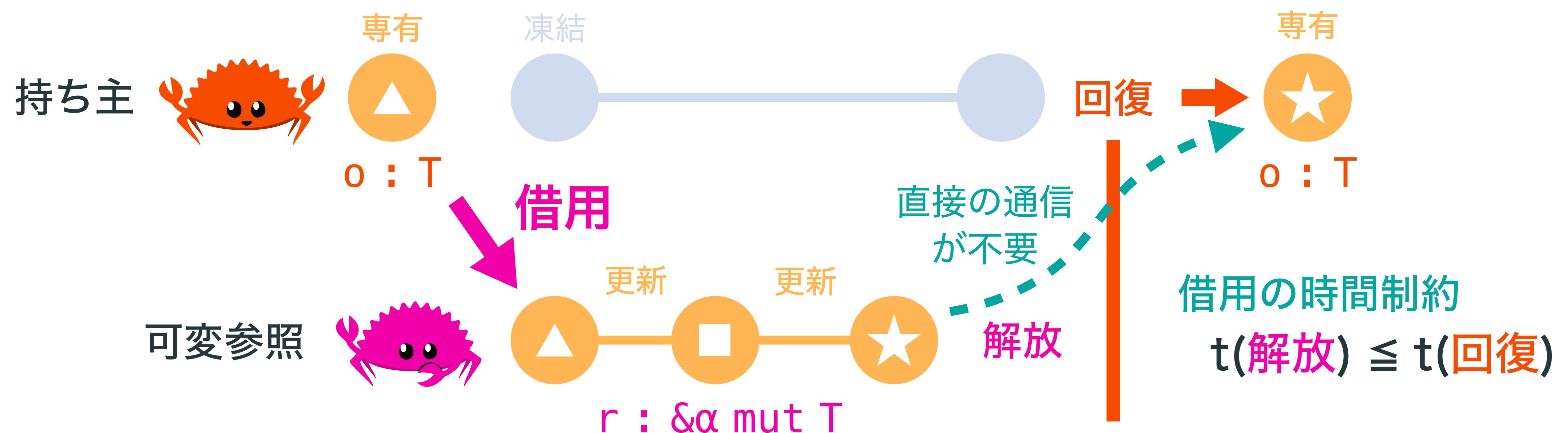
'15~

ソフトウェア科学の画期

Rust の鍵#1 借用

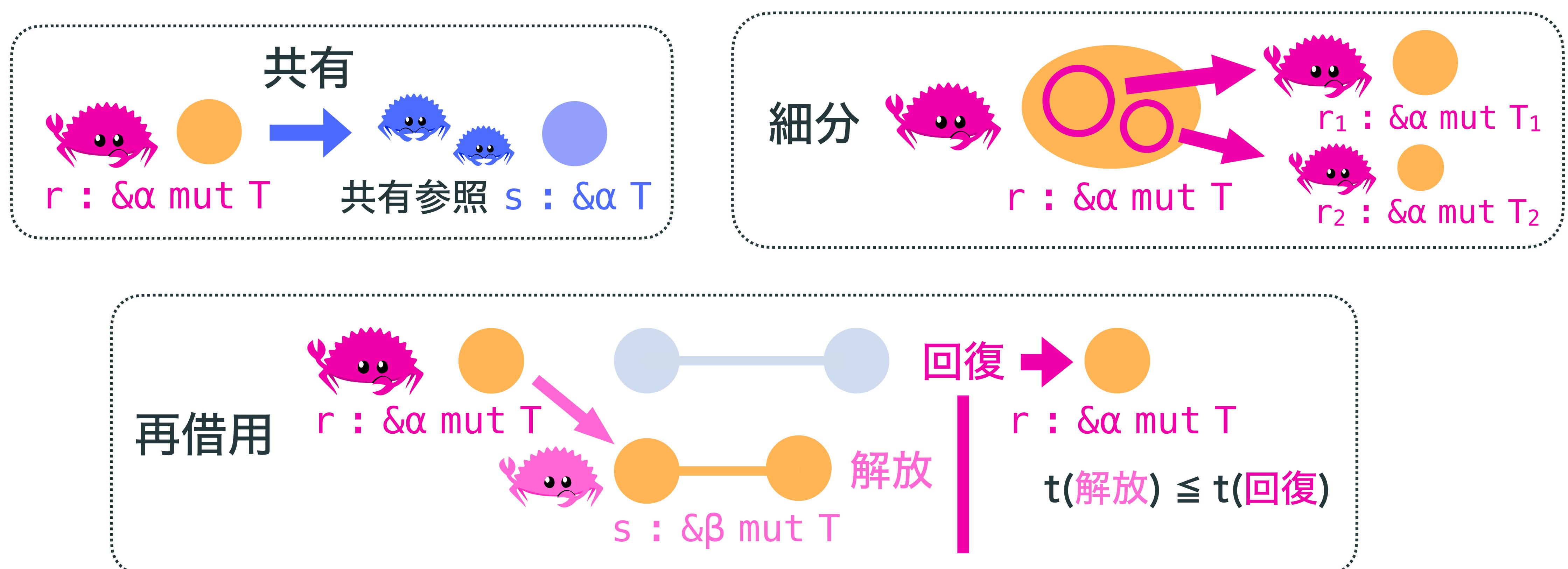
[Grossman+ '02]

- ♦ 所有権を一時的に借りる
 - ▶ 所有権を解放するときに直接の通信が不要



Rust の鍵#1 借用への操作

- ♦ 借用について高レベルな操作が提供されている



Rust の鍵#1 借用検査

♦ 借用の制約の自動静的検証

- ▶ 特に時間制約 $t(\text{解放}) \leq t(\text{回復})$ を検査

前世代

スコープ基準

$t(\text{解放}) = \text{スコープ終了}$

既存研究に直接由来

Cyclone [Grossman+ '02]

現世代

非字句的ライフタイム

$t(\text{解放})$ を生存解析で推論

ライフタイム = 連続する
プログラムポイントの区間



Niko Matsakis

次世代？

制御可能な借用検査

可変参照型で借用元を管理

自己借用への対応

```
fn new_widget(&self, name: String) -> Widget {
    let name_suffix: &'name str = &name[3..];
        // --- borrowed from "name"
    let model_prefix: &'self.model str = &self.model[..2];
        // ----- borrowed from "self.model"
}
```

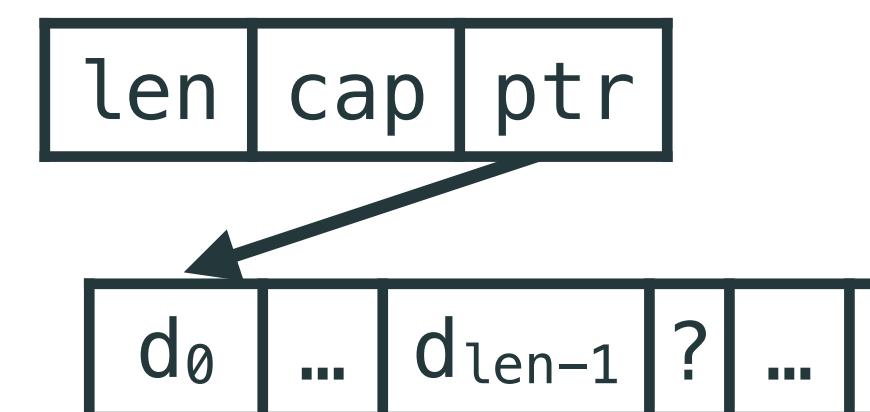
Rust の鍵#2 unsafe

- ♦ unsafe = 限界のある静的型検査に対するバックドア
 - ▶ 特に所有権検査を受けない生ポインタ *mut T の操作
 - ▶ 重要パターン: unsafe 実装を safe な型でカプセル化

例

動的配列型

```
struct Vec<T> {  
    len : uint,  
    cap : uint,  
    ptr : *mut T  
}
```

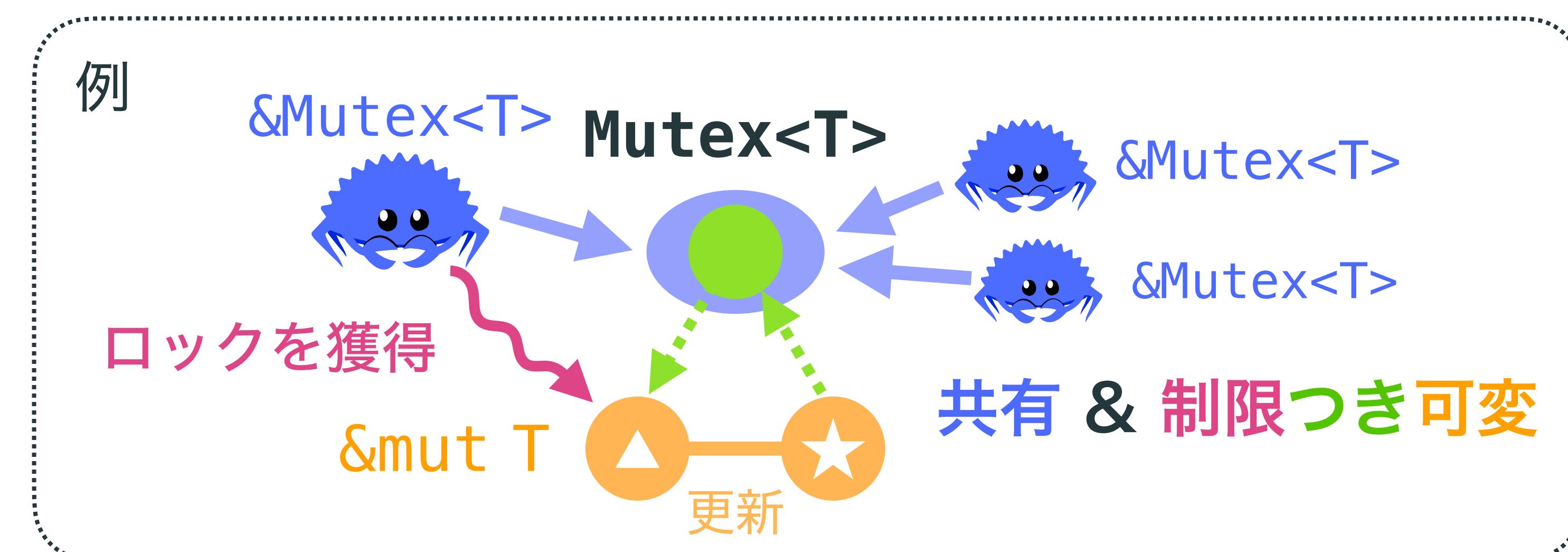


要素への可変参照を取得

```
fn index_mut<&mut T>(v : &mut Vec<T>,  
                      i : uint) -> &mut T {  
    assert!(i < v.len); 境界検査  
    unsafe { v.ptr.offset(i) }   
}
```

Rust の鍵#3 内部可変性

- ◆ 共有借用中に制限つきで許される可変性
 - ▶ ふつうの型 (int, String, …) : 共有中はデータが完全に不变
 - ▶ 特定の型のみ内部可変性: Mutex, RwLock, Arc, Cell, …



3.2

RustBelt

Rust の安全性保証は非自明

- ◆ Rust: 低レベル操作に高度な安全性保証
 - ▶ 共有 xor 可変、アクセス競合がない → メモリ・スレッド安全
- ◆ 以下の2つがその鍵
 - ▶ 高度な静的所有権型システム
 - 特に借用について柔軟な操作を提供
 - ▶ 注意深く設計された safe Rust API
 - unsafe 実装を適切にカプセル化、場合によっては内部可変性も

Rust の安全性保証は壊れうる

♦ 型システム・API でそれぞれ脆弱性の危険



| Author | | Label | | Projects | | Mile |
|---|---|---|--|----------|--|------|
| 5 Open | ✓ 24 Closed | | | | | |
| alias bound candidates for opaques allow cyclic reasoning | A-impl-trait A-typesystem C-bug | F-type_alias_impl_trait I-unsound requires-nightly T-compiler T-types | #109387 opened on Mar 20, 2023 by lcnr | | | |
| HRTB on subtrait unsoundly provides HTRB on supertrait with weaker implied bounds | A-lifetimes A-traits A-typesystem C-bug I-unsound P-high S-bug-has-test T-compiler T-types | #84591 opened on Apr 27, 2021 by steffahn | | | | |
| Functions, closures, and HRTB-trait-objects can implement traits such that validity of associated types is never checked. | A-associated-items A-closures A-dst A-lifetimes A-traits A-typesystem C-bug I-unsound P-medium S-bug-has-test T-compiler T-types | #84533 opened on Apr 25, 2021 by steffahn | | | | |
| 'static closures/FnDefs/futures with non-'static return type are unsound | A-associated-items A-async-await A-closures A-lifetimes A-trait A-typesystem AsyncAwait-Triaged C-bug I-unsound P-high S-bug-has-test T-compiler T-lang T-types | #84366 opened on Apr 20, 2021 by steffahn | | | | |
| Implied bounds on nested references + variance = soundness hole | A-typesystem A-variance C-bug I-unsound P-medium S-bug-has-test T-types | #25860 opened on May 29, 2015 by aturon | | | | |

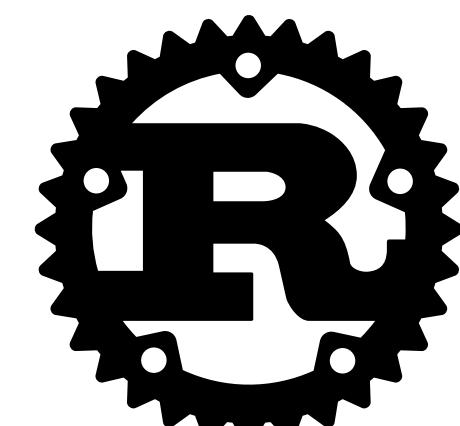
| Author | | Label | | Projects | | Mile |
|---|--|-------------------|---|----------|--|------|
| 1 Open | ✓ 42 Closed | | | | | |
| thread::park causes undefined behaviour on panic | A-atomic I-unsound P-medium | T-libs T-libs-api | #102398 opened on Oct 11, 2022 by joboet was closed on Oct 11, 2022 | | | |
| #95295 causes unsoundness in multiple existing crates | I-unsound regression-from-stable-to-beta T-compiler T-libs-api | | #101899 opened on Nov 10, 2022 by LegionMammal978 was closed on Nov 10, 2022 ⇨ 1.65.0 | | | |
| Missing A: Sync bound for Allocator for vec::IntoIter | A-allocators C-bug I-unsound T-libs-api | | #92633 opened on Jan 15, 2022 by steffahn was closed on Jan 15, 2022 | | | |
| iter::Fuse is unsound with how specialization currently behaves around HRTB fn pointers | A-iterators A-lifetimes A-specialization A-trait A-typesystem C-bug I-unsound P-critical T-compiler T-lang T-libs T-libs-api | | #85863 opened on Jul 15, 2021 by steffahn was closed on Jul 15, 2021 | | | |
| A Pin unsoundness involving an impl DerefMut for Pin<&dyn LocalTrait> | A-pin C-bug I-unsound P-high S-bug-has-test T-compiler T-lang T-libs-api T-types | | #85099 opened on May 9, 2021 by steffahn | | | |

Rust の安全性保証を証明したい

- ◆ Rust の安全性保証はとても非自明なので証明したい
 - ▶ 特に複数の Rust API を組み合わせたときに一般にどうなる？
- ◆ さらに証明を モジュラー / 拡張可能にしたい
 - ▶ 新しい Rust API を追加するときに、証明全体をやり直さず、その API についてのみ検証すれば済むようにしたい



- ◆ Rust の安全性を分離論理 Iris で検証、Coq で機械化
 - ▶ Rust の所有権型・型付け判断に Iris 上で意味論（論理関係）
 - モジュラー / 拡張可能な証明、Mutex・Arc 等の Rust API をカバー
 - ▶ 理論的貢献 ライフタイム論理 — Rust の借用を高階幽霊状態で表現



Ir/^{*}S

意味論による Rust の安全性証明

- ♦ Rust の所有権型に分離論理の述語の意味論を与える
 - ▶ 型付け判断は分離論理のホーア三つ組として解釈、検証
 - 証明したい主張（安全性）を意味論にしてしまう
 - ▶ 構文的な「進行と保存」 [Wright & Felleisen '94] より柔軟で強力
 - 実行の途中は構文的に型がつかなくて構わない

$$\llbracket T \vdash e \mid v. T' \rrbracket \triangleq \{ \llbracket T \rrbracket \} \ e \ \{ \lambda v. \llbracket T' \rrbracket \} \quad \llbracket \overline{v: T} \rrbracket \triangleq * \overline{\llbracket T \rrbracket(v)}$$

$$\llbracket \text{int} \rrbracket(v) \triangleq v \in \mathbb{Z} \quad \llbracket \text{Box}\langle T \rangle \rrbracket(r) \triangleq \exists \bar{v}. \ r \mapsto \bar{v} * \text{free } r * \triangleright \llbracket T \rrbracket(\bar{v})$$

理論的貢献 ライフタイム論理

- ♦ 借用を分離論理 Iris で抽象化、モジュラーな検証を実現
 - ▶ 各ライフタイム α の状態をトークン $[\alpha]_q, \dagger\alpha$ で管理
 - ▶ フル借用 $\&_{full}^{\alpha} \triangleright P$: ライフタイム α のもとで命題 P : iProp を借用
 - 不変状態 $\Box P$ と同様の高階幽霊状態、Later \triangleright の問題

| 継続中 | 終了済 | 借り手 | 貸し手 |
|---|--|-----|-----|
| $\top \Rightarrow_{\top} \exists \alpha. [\alpha]_1 * ([\alpha]_1 \Rightarrow_{\top} \dagger\alpha)$ | $\triangleright P \Rightarrow_{\top} \&_{full}^{\alpha} \triangleright P * (\dagger\alpha \not\equiv_{\top} \triangleright P)$ | | |
| 中身 | | | |
| アクセス $[\alpha]_q * \&_{full}^{\alpha} \triangleright P \Rightarrow_{\top} \triangleright P * (\triangleright P \not\equiv_{\top} [\alpha]_q * \&_{full}^{\alpha} \triangleright P)$ | | | |

Rust の参照の意味論

- ♦ 可変参照: フル借用

$$[\![\&\alpha \text{ mut } T]\!](r) \triangleq \&_{\text{full}}^\alpha \triangleright (\exists \bar{v}. r \mapsto \bar{v} * [\![T]\!](\bar{v}))$$

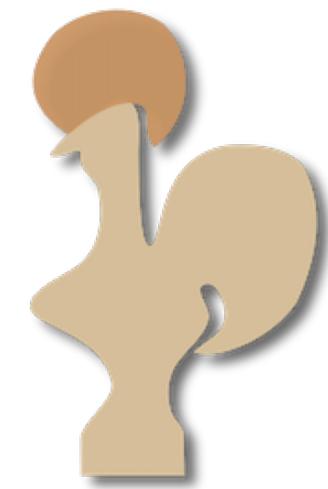
- ♦ 共有参照: 不変条件に入れた借用

$$[\![\&\alpha \text{ Mutex}\langle T \rangle]\!](r) \triangleq \exists \alpha' \sqsupseteq \alpha.$$

$$\&_{\text{at}}^\alpha \triangleright (r \mapsto \text{true} \vee (r \mapsto \text{false} * [\![\&\alpha' \text{ mut } T]\!](r + 1)))$$

$\&_{\text{at}}^\alpha \triangleright P$: $\boxed{\&_{\text{full}}^\alpha \triangleright P}$ に相当、内側のフル借用に \triangleright が付かない

RustBelt Coq / lambda-rust



Iris / lambda-rust

lambda-rust

Project information
The Coq development of LambdaRust

1,260 Commits
37 Branches
3 Tags
516.9 MiB Project Storage
1 Release

README
BSD 3-Clause "New" or "Revised" License
CI/CD configuration
+ Add CHANGELOG
+ Add CONTRIBUTING
+ Add Kubernetes cluster
+ Add Wiki
+ Configure Integrations

Created on
May 04, 2016

| Name | Last commit | Last update |
|----------------------|--------------------------------|---------------|
| λ lambda-rust | Bump std++ (length_X). | 3 weeks ago |
| ↳ lifetime | Add lemmas used by R... | 1 week ago |
| ↳ .gitattributes | Coq syntax. | 4 years ago |
| ↳ .gitignore | update build system | 3 years ago |
| ↳ .gitlab-ci.yml | publish lifetime logic de... | 6 months ago |
| ↳ LICENSE | fix trailing empty line | 4 years ago |
| ↳ Makefile | update build system | 3 years ago |
| ↳ Makefile.coq.lo... | avoid deprecated fgrep | 1 year ago |
| ↳ Missing.md | Mutex might become c... | 2 years ago |
| ↳ README.md | update to Coq 8.18 | 11 months ago |
| ↳ _CoqProject | split the lifetime logic in... | 6 months ago |

RustBelt の後継研究

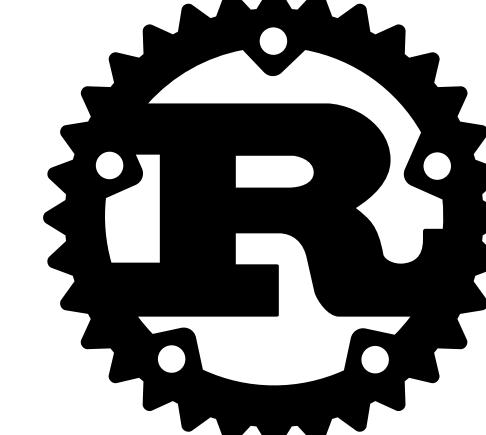
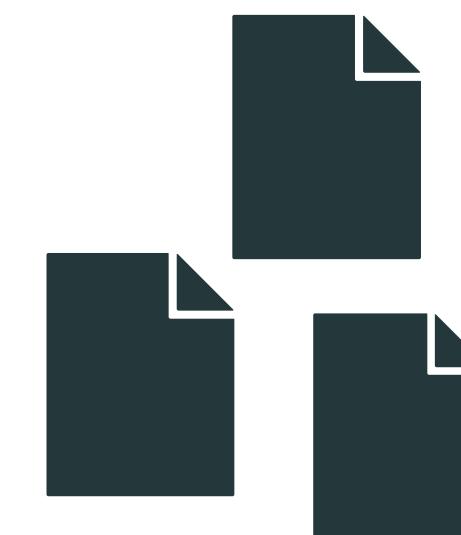
- ◆ RustBelt × 弱メモリ [Dang+ '20]
 - ▶ RustBelt を拡張、RC11 弱メモリモデルでの安全性を保証
 - ▶ Arc ライ브ライの難しい同期バグを発見 ([Issue #51780](#))
- ◆ RustHornBelt [松下+ '22] — 後述
- ◆ RefinedRust [Gäher+ '24]
 - ▶ RustBelt を篩型で自動化、RefinedC [Sammller+ '21] がベース

3.3

私の研究 RustHorn+Belt

Rust プログラム検証

- ♦ Rust プログラムの動作の正しさを検証したい
 - ▶ Rust の所有権型システムだけでは値の内容までは検証できない
 - ▶ 所有権の情報を利用して高レベルに・モジュラーに検証したい



forall
exists

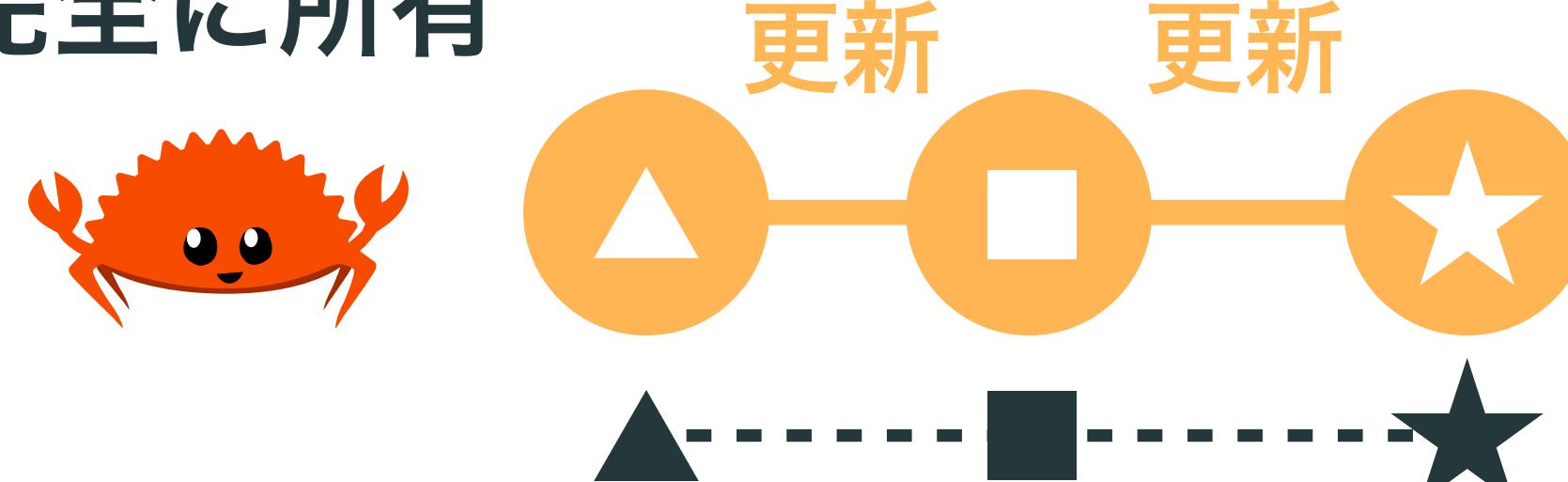
QED

基本アイデア 所有権で“関数型”にする

- ◆ Rust の所有権を使って関数型に帰着する
 - ▶ メモリのないステートレスなモデルで効率的な検証を実現

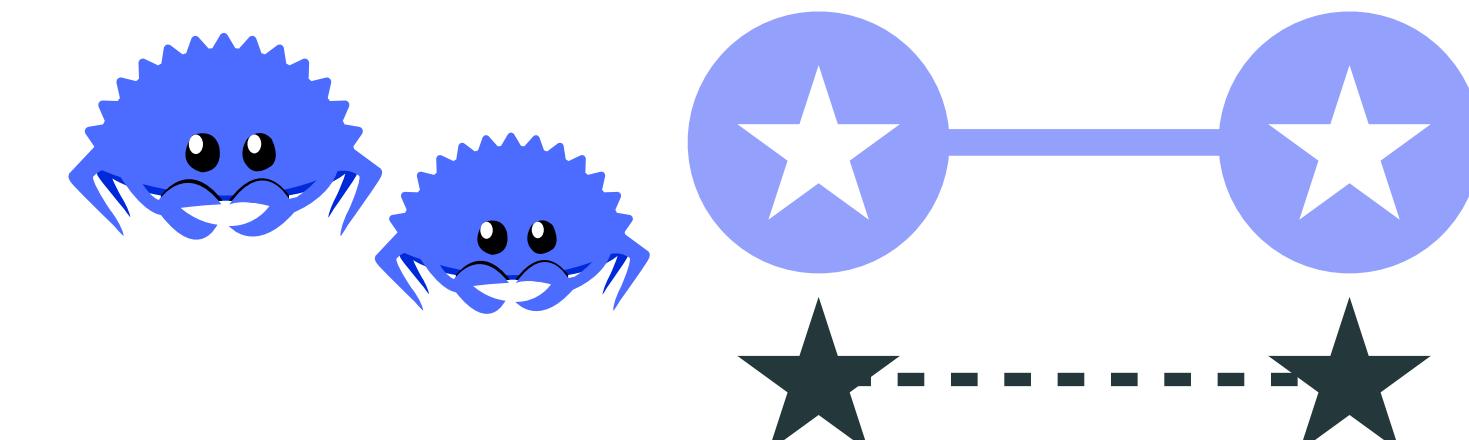
Rust プログラム → 関数型プログラム

完全に所有



値の変化をトラック可能

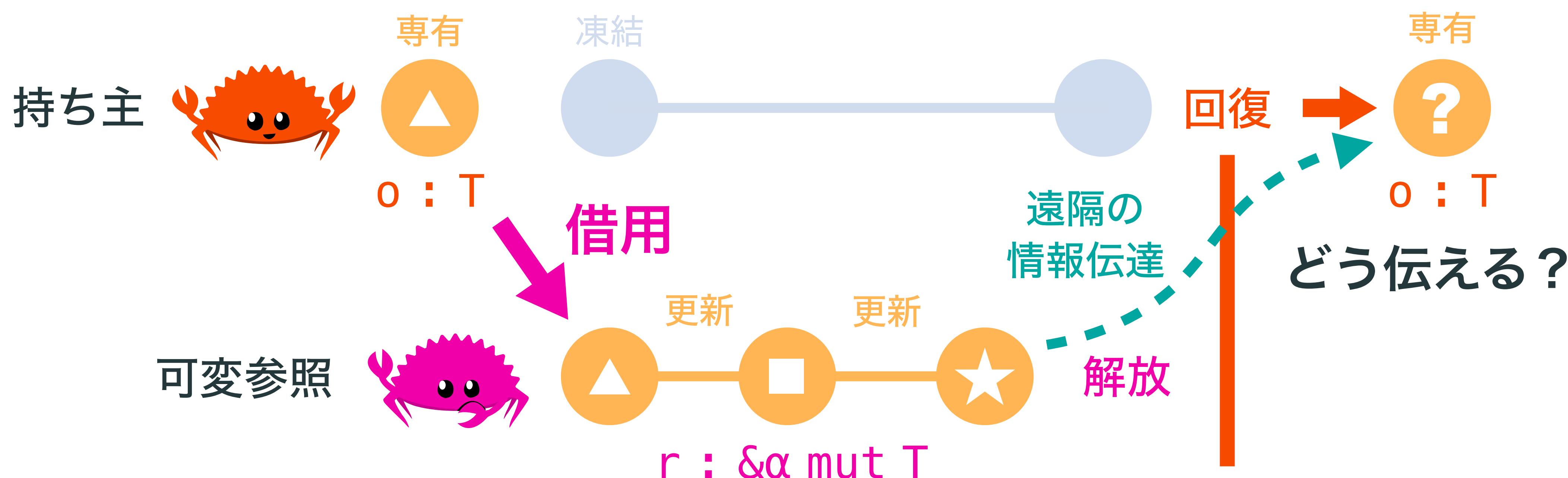
不变で共有



単に不变な値として表現

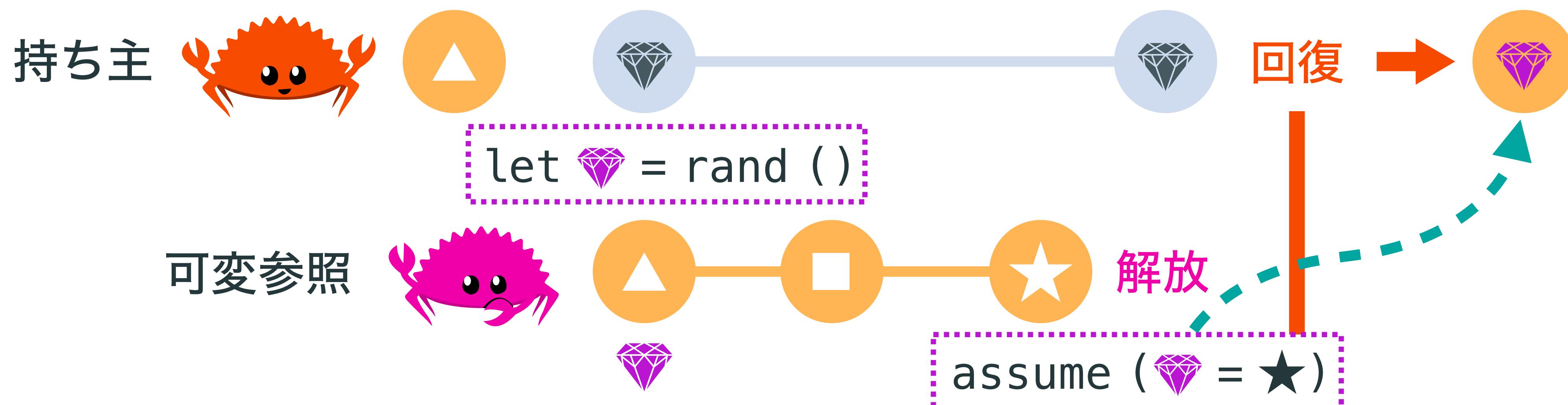
挑戦 借用のモデル化

- ♦ 借用は遠隔の情報伝達を許すから難しい
 - ▶ 可変参照による更新の結果を持ち主にどう伝える？





- ◆ 更新の最終結果の値を「預言」 [Abadi & Lamport '88]
- ▶ 借用開始時に預言 を作る、持ち主と可変参照で共有
- ▶ 可変参照は所有権解放時に預言を確定



RustHorn 方式の預言の力

- ♦ 借用への様々な操作が預言で自然にモデル化できる
 - ▶ 借用の細分は預言の部分的解決でモデル化できる

例 `{ true } fn push(v : &mut Vec<T>, a : T) { ^v = *v ++ [a] }`
 `{ i < v.len }`

 預言 

細分 `fn index_mut(v : &mut Vec<T>, i : uint) -> &mut T`

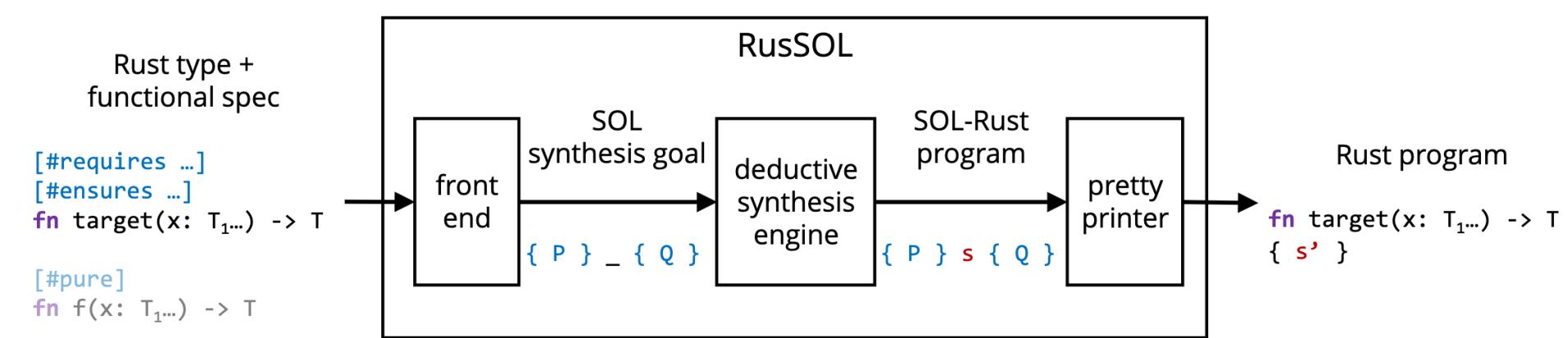
`{ *res = *v[i] }  get` `{ ^v = *v { i ← ^res } }  set`

 預言  小預言 

 預言を部分的解決

RustHorn と後継研究

- ◆ RustHorn [松下+ '20/'21] — 全自動検証 
 - ▶ Rust コードを預言で CHC (不動点付き論理式) に変換・検証
 - 特に再帰的データ型周りで面白い例、CHC の研究にも影響 [V.K.+ '22]
- ◆ Creusot [Denis+ '22] — 半自動検証 
 - ▶ 注釈付き Rust コードを預言で変換し Why3 で検証
- ◆ RusSOL [Fiala+ '23] — 自動合成
 - ▶ 預言による仕様から Rust コードを合成



RustHorn の預言の手法は 便利 & 非自明

- ◆ 借用の細分、再借用、参照のネストがあっても自然に動作
 - ▶ 可変参照が預言のおかげで第一級の値として扱える
- ◆ どうやって正当性を証明する?
 - ▶ 預言を未来の借用終了時の値と結びつけられる?
 - ▶ 可変参照への可変参照だと預言への預言が生まれる?



- ◆ RustHorn の預言の手法の正しさを Iris & Coq で検証
 - ▶ RustBelt [Jung+ '18] を拡張、型の意味論で値の情報ももつ
 - ▶ 所有権型により生成される預言的な関数型モデルの正当性を証明
 - モジュラー / 拡張可能な証明、Vec · IterMut 等の Rust API をカバー
 - ▶ 理論的貢献 パラメトリック預言 — 分離論理上での預言の表現



RustHorn の形式化型仕様システム

- ◆ Rust の型付け判断に閾数型仕様を加える
 - ▶ 型情報にもとづく RustHorn の閾数型モデルの導出を自然に表現

型仕様判断 $T \vdash e \mid v.T' \rightsquigarrow \phi$

述語変換子 $\dot{\phi} : ([T'] \rightarrow Prop) \rightarrow ([T] \rightarrow Prop)$

意味論の拡張 基本アイデア

- ♦ 型の意味論に値についてのパラメータを加える
 - ▶ 型仕様判断で値についての事前・事後条件をチェック

$$\llbracket T \vdash e \mid v. T' \rightsquigarrow \dot{\phi} \rrbracket \triangleq \forall \psi.$$

$$\{ \exists \bar{a} \text{ s.t. } \dot{\phi} \psi \bar{a}. \llbracket T \rrbracket(\bar{a}) \} \in \{ \lambda v. \exists \bar{b} \text{ s.t. } \psi \bar{b}. \llbracket T' \rrbracket(\bar{b}) \}$$

$$\llbracket \mathbf{int} \rrbracket(n, n') \triangleq n' = n$$

$$\llbracket \mathbf{Box} < T > \rrbracket(r, a) \triangleq \exists \bar{v}. r \mapsto \bar{v} * \text{free } r * \triangleright \llbracket T \rrbracket(\bar{v}, a)$$

理論的貢献 パラメトリック預言

♦ 預言についての全可能性 $\pi: PrAsn$ の上でパラメタ化

- ▶ π は全預言変数への値の割当 $PrAsn \triangleq \forall A. PrVar_A \rightarrow A$
 - 透視モナド $Clair A \triangleq PrAsn \rightarrow A$ の値 \hat{a} の上で色々考える
- ▶ 預言観察 $\langle \hat{\phi} \rangle$: 「今妥当な全 π で $\hat{\phi} \pi$ が成立」、永続的

$$\top \Rightarrow \exists \textcolor{violet}{x}. [\textcolor{violet}{x}]_1 \quad [\textcolor{violet}{x}]_1 \Rightarrow \langle \lambda \pi. \pi \textcolor{violet}{x} = v \rangle$$

$$\langle \hat{\phi} \rangle \text{ is persistent } \quad \langle \hat{\phi} \rangle \Rightarrow \exists \pi. \phi \pi$$

意味論の拡張 預言対応版

- ♦ 値を透視モナドに包む

- ▶ 事前・事後条件は預言観察の中に入れる

$$\llbracket T \vdash e \mid v. T' \rightsquigarrow \dot{\phi} \rrbracket \triangleq \forall \hat{\psi}.$$

$$\left\{ \exists \bar{a} \text{ s.t. } \langle \lambda \pi. \dot{\phi}(\hat{\psi} \pi) (\bar{a} \pi) \rangle. \llbracket T \rrbracket(\bar{a}) \right\} \mathbf{e} \left\{ \lambda v. \exists \bar{b} \text{ s.t. } \langle \lambda \pi. (\hat{\psi} \pi)(\bar{b} \pi) \rangle. \llbracket T' \rrbracket(\bar{b}) \right\}$$

$$\llbracket \mathbf{int} \rrbracket(n, \hat{n}') \triangleq \hat{n}' = \text{const } n$$

$$\llbracket \mathbf{Box} < T > \rrbracket(r, \hat{a}) \triangleq \exists \bar{v}. r \mapsto \bar{v} * \text{free } r * \triangleright \llbracket T \rrbracket(\bar{v}, \hat{a})$$

Rust の可変参照の預言的意味論

- ♦ ライフタイム論理のフル借用を利用
 - ▶ 預言に関する仕組みを後から追加

$$\llbracket \&\alpha \text{ mut } T \rrbracket(r, \hat{p}) \triangleq \exists \hat{a}, x \text{ s.t. } \hat{p} = \lambda \pi. (\hat{a} \pi, \pi x).$$

$$VO_x(\hat{a}) * \&_{\text{full}}^{\alpha} \triangleright (\exists \hat{a}', \bar{v}. r \mapsto \bar{v} * \llbracket T \rrbracket(\bar{v}, \hat{a}') * PC_x(\hat{a}'))$$

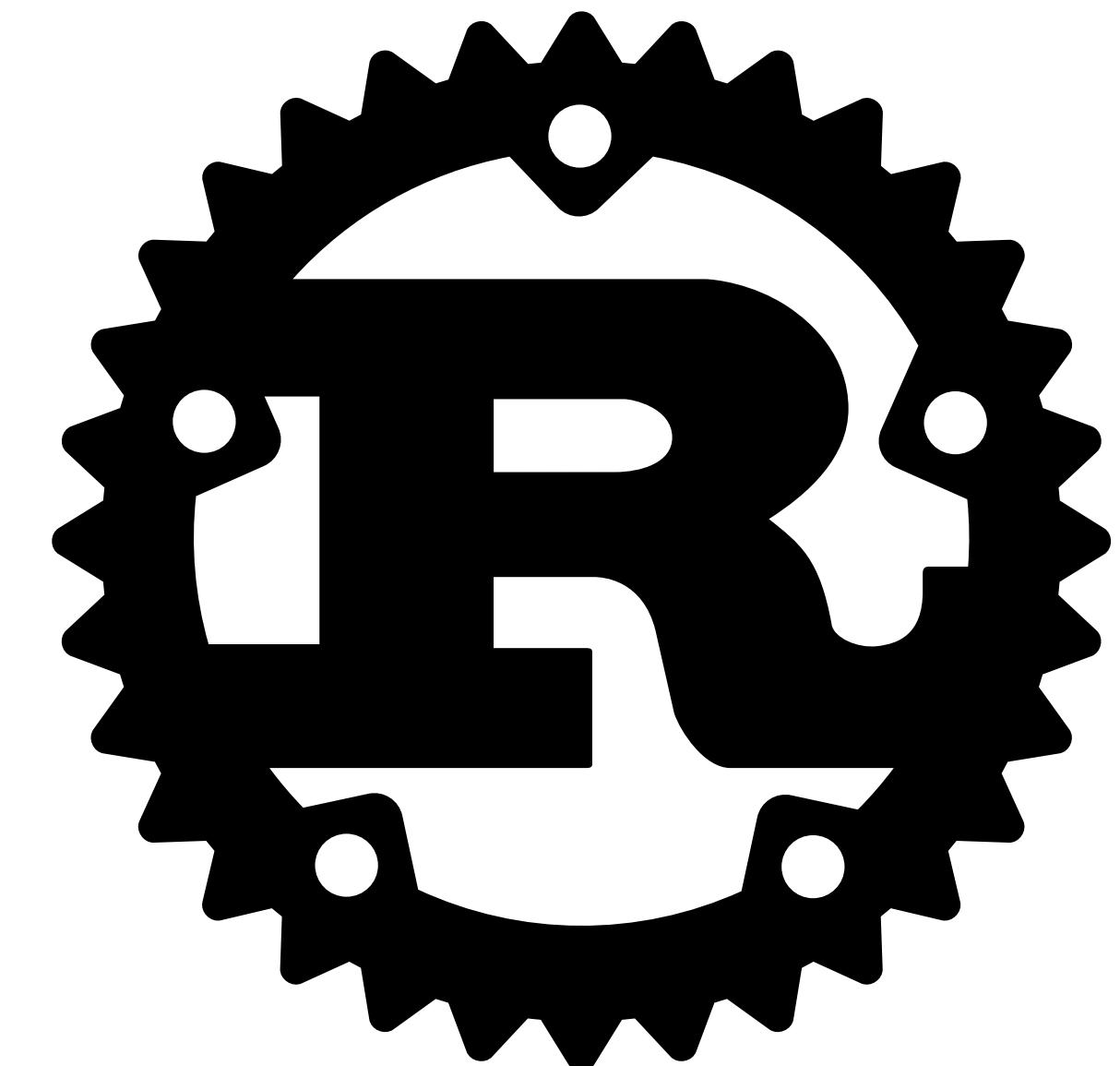
$$VO_x(\hat{a}) * PC_x(\hat{a}') \Rightarrow \hat{a} = \hat{a}' \quad VO_x(\hat{a}) * PC_x(\hat{a}) \Rightarrow VO_x(\hat{a}') * PC_x(\hat{a}')$$

$$VO_x(a) * PC_x(a) \Rightarrow \langle \lambda \pi. \pi x = a \rangle * VO_x(a)$$

これからの展望

- ◆ Liveness 検証もしたい
 - ▶ 最近 Nola [松下 博論'24] で借用を Later-free に定式化、可能に
- ◆ 現実の Rust プログラムをもっと検証したい
 - ▶ RustHornBelt での unsafe な世界の検証は低レベルで大変
 - ▶ unsafe な世界についても高レベルな検証ができる?
 - Rust 検証器 Verus [Lattuada+ '23] は期待株、預言を入れる動きも
 - どういう推論規則が作れるか基礎研究がまだ必要

Rust を Iris で開拓しよう





ご清聴ありがとうございました

