ソフトウェアの科学

～バグのない世界を目指して～

講師　松下 祐介　　ファシリテータ　中山 崇

東京大学理学部オープンキャンパス学生講演　2023.8.2

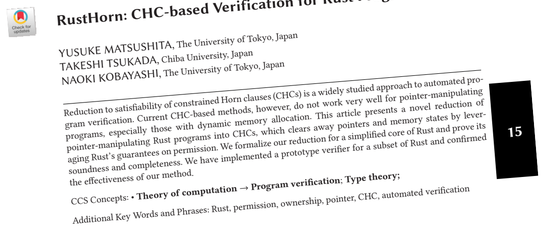
講師

国際会議 PLDI ’22 にて

松下 祐介　小林研 博士3年

灘高等学校　→　東京大学 理科一類
→　東京大学 理学部情報科学科
→　東京大学 院 情報理工コンピュータ科学専攻

ソフトウェア科学
Rustプログラムの検証


RustHorn: CHC-based Verification for Rust Programs

YUSUKE MATSUSHITA, The University of Tokyo, Japan
TAKESHI TSUKADA, Chiba University, Japan
NAOKI KOBAYASHI, The University of Tokyo, Japan

Reduction to satisfiability of constrained Horn clauses (CHCs) is a widely studied approach to automated program verification. Current CHC-based methods, however, do not work very well for pointer-manipulating programs, especially those with dynamic memory allocation. This article presents a novel reduction of pointer-manipulating Rust programs into CHCs, which clears away pointers and memory states by leveraging Rust's guarantees on permission. We formalize our reduction for a simplified core of Rust and prove its soundness and completeness. We have implemented a prototype verifier for a subset of Rust and confirmed the effectiveness of our method.

CCS Concepts: • **Theory of computation → Program verification**; **Type theory**;

Additional Key Words and Phrases: Rust, permission, ownership, pointer, CHC, automated verification

15


趣味　ピアノ

ファシリテータ

中山 崇　　小林研 修士1年

早稲田高等学校　→　東京大学 理科一類
→　東京大学 理学部情報科学科
→　東京大学 院 情報理工コンピュータ科学専攻

ソフトウェア科学

所有権型を使うプログラム検証

講演内容

1　情報科学って何？

　　東大理情の紹介

2　ソフトウェアの科学

　　Ｑ＆Ａ

Q&A

【学生講演】講師：博士課程3年　松下祐介

ファシリテーター：修士課程1年　中山崇

「ソフトウェアの科学 ～バグのない世界を目指して～」

講師への質問を募集しています

【質問受付時間】2023年8月2日（水）10:10-11:00

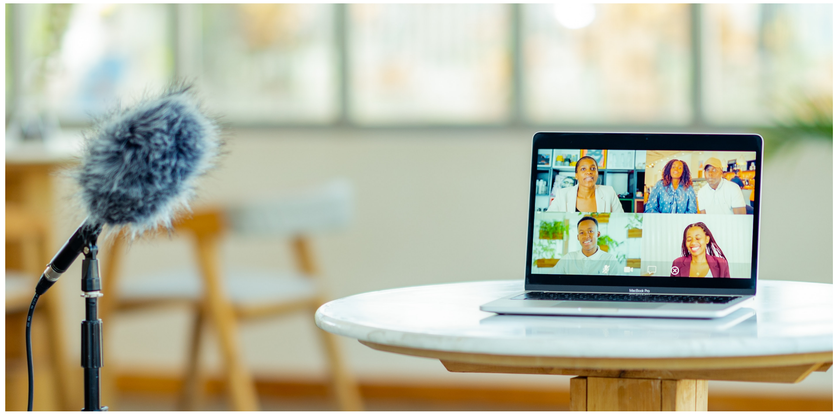
【質問方法は2通り】

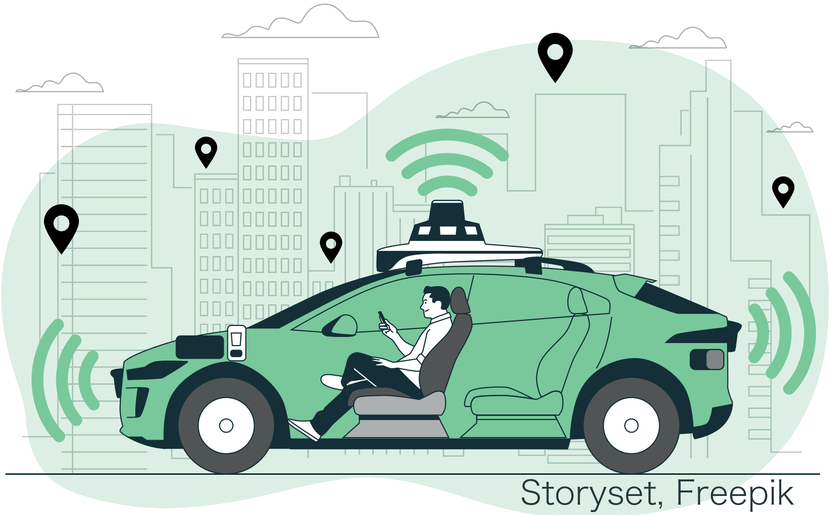
1. **講演画面下のSlidoのURL （https://www.sli.do/jp）をクリックして、アクセスコード＃ 2358739を入力してください。**
2. **右下のQRコードをスマートフォンで読み取り、質問を入力してください。**

こちらで参加：
slido.com
#2358 739

情報科学って何？

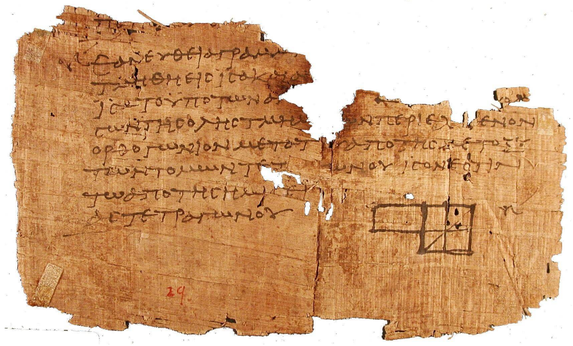
21世紀 = 情報の世紀





Storyset, Freepik

計算・情報処理の歴史



紀元前3世紀　ユークリッドの互除法

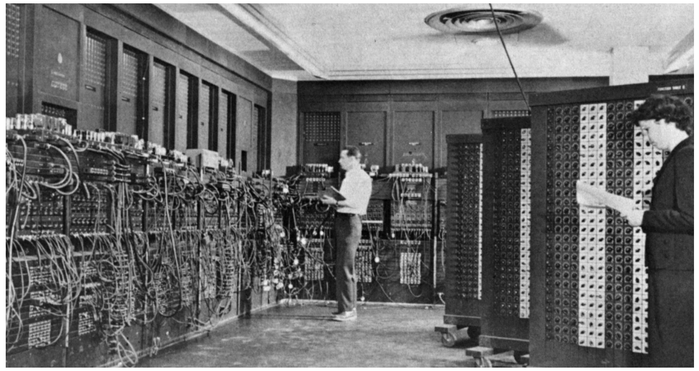
アルゴリズム

紀元前2世紀　天体の動きの予測

アナログコンピュータ

1804年　ジャカード織機

プログラミング

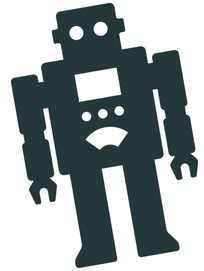


1945年　ENIAC

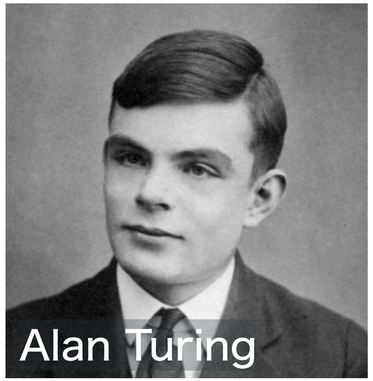
デジタルコンピュータ

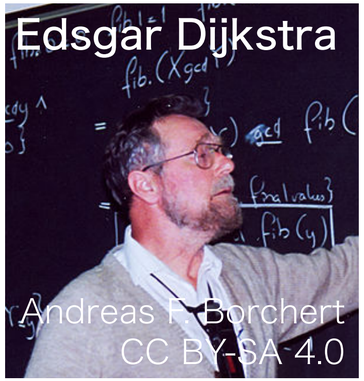
情報科学 = コンピュータ科学

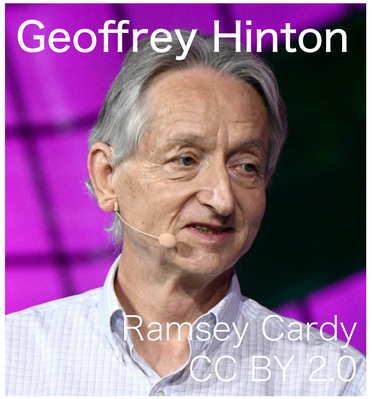
コンピュータによる



計算と情報処理の理論と実践

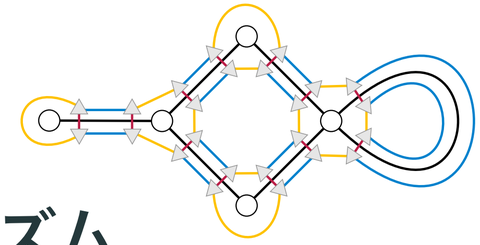






情報科学の対象

理論

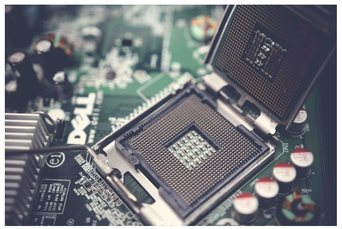


アルゴリズム

暗号

量子コンピュータ

ソフトウェア

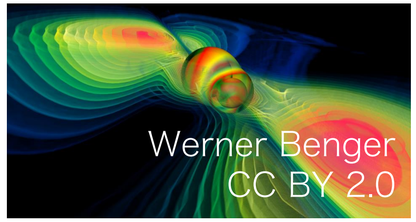


ハードウェア

OS

応用

数値計算



スパコン

UI

自動運転

人工知能

情報科学と近い分野

幾何

統計

論理

倫理

哲学

人工知能

アルゴリズム

言語

数学

計算

情報科学

情報

量子

グラフ

電磁気

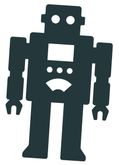
代数

整数

物理学

1+2=?

機械



東大理情の紹介

東京大学　理学部情報科学科

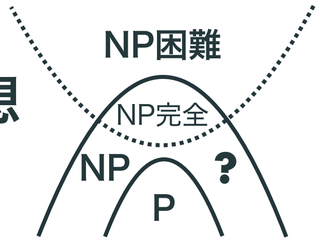
DEPARTMENT OF INFORMATION SCIENCE

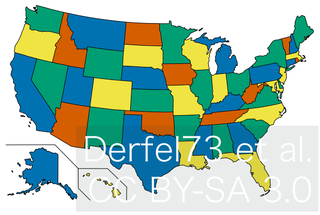
情報技術の原理・理論

理情の紹介

離散数学

P≠NP 予想

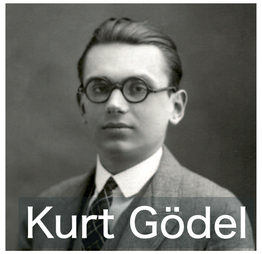




四色定理

論理学

不完全性定理



計算可能性

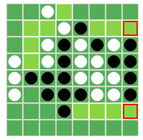
機械学習

統計

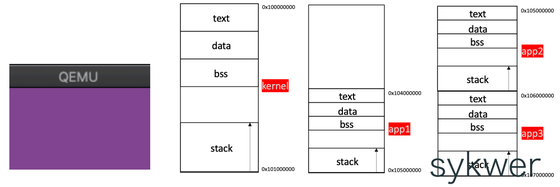
凸最適化

プログラミング演習

OCaml

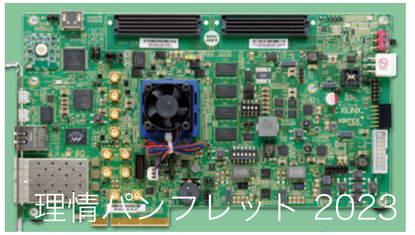


```
let rec fix f x
  = f (fix f) x
```



sykwer
utokyo_syspro_baremetal

CPU実験

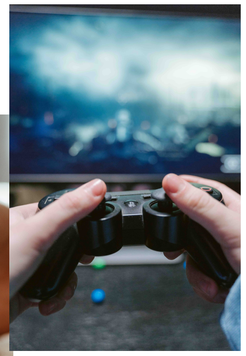


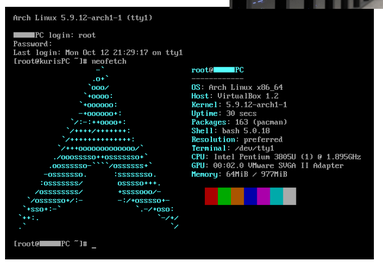
…ほか色々

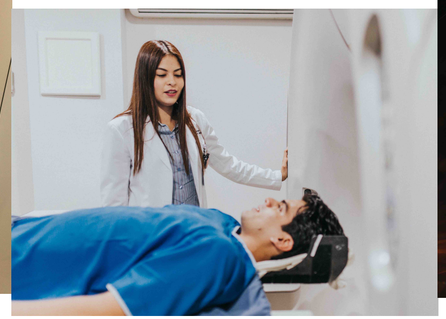
ソフトウェアの科学

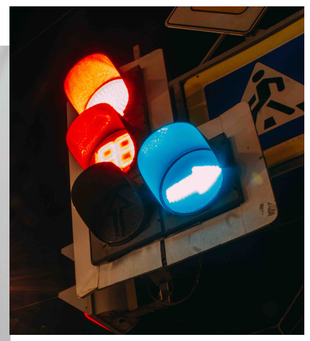
ソフトウェア

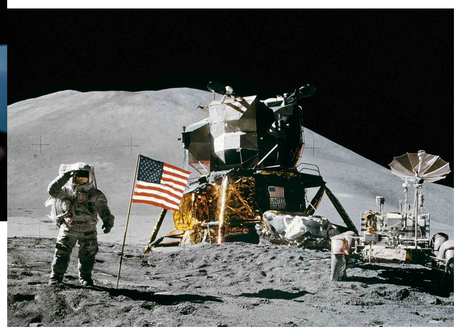
コンピュータに指令して仕事をさせるもの









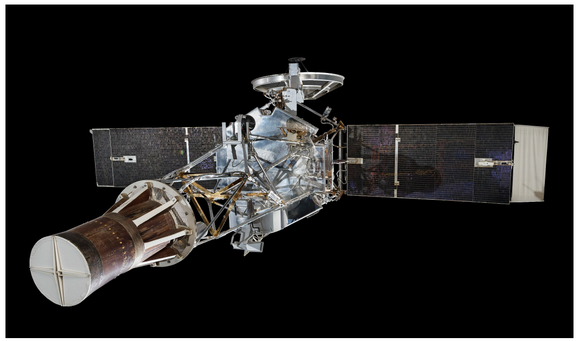


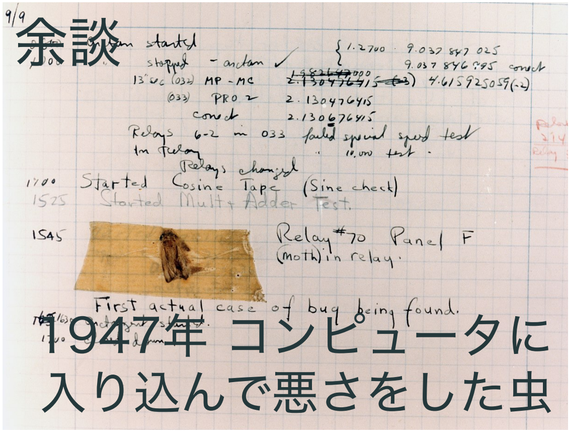
ソフトウェア・バグ

予期しない動作を起こすソフトウェアの誤り

1962年　NASA
金星探査機 マリナー1号 墜落

約2億ドルの損害　　バグ　$\overline{\dot{R}} \rightarrow \dot{R}$



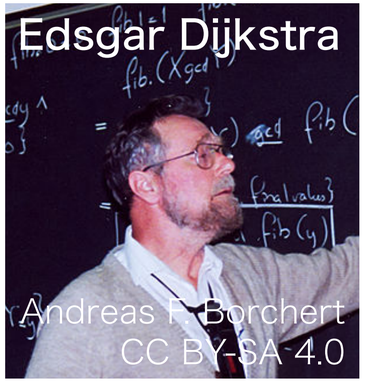


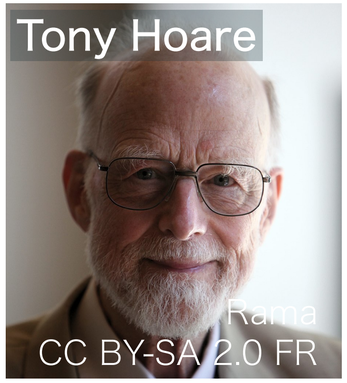
1947年 コンピュータに
入り込んで悪さをした虫

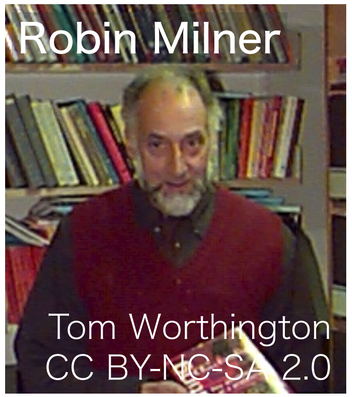
ソフトウェア科学 & 工学

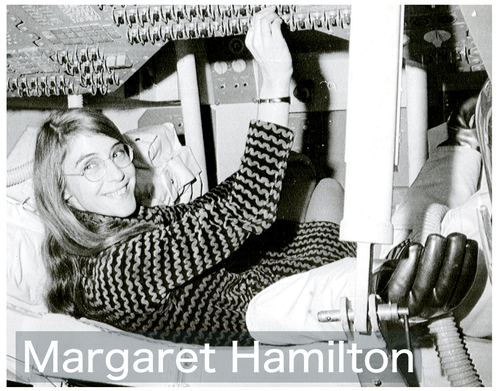
ソフトウェア開発の方法論

特に　バグを防ぐ







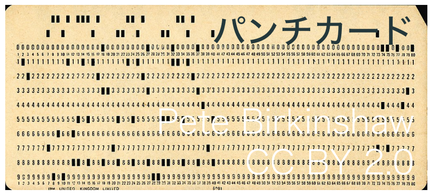


ソフトウェア記述の進化

機械語

最初期～

ビット列 …000110…



逐語訳

アセンブリ

1947年～

命令の列

```
ADD  EAX, 7
MOV  EBX, 3
JMP  300
```

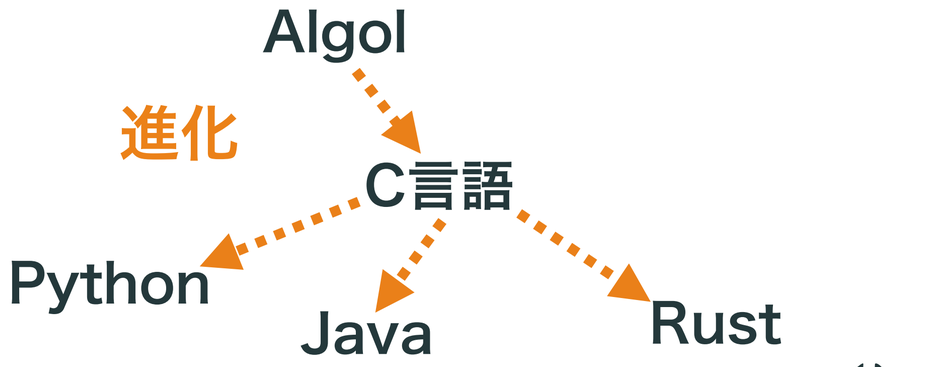
抽象化

プログラミング言語 1952年～

人間の直感に近い

プログラム

```
r ← readLn
print $ pi * r * r
```

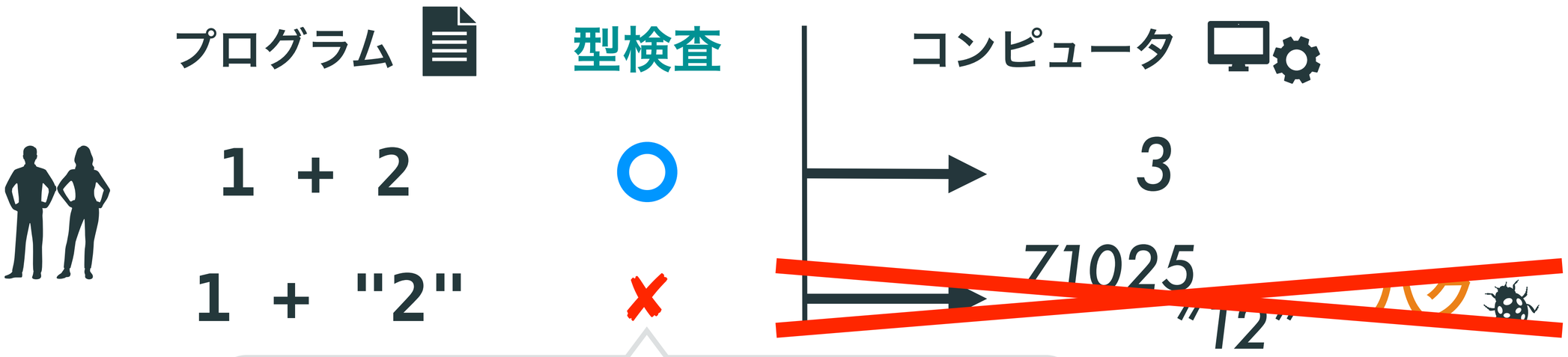


ソフトウェア科学・工学

型によるバグ防止

型　＝　データの種類・属性

123 ： 整数　　"abc" ： 文字列

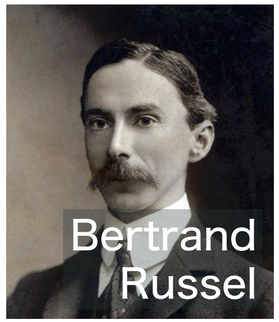


型エラー　整数 1 + 文字列 "2" は不正です

型の発展

1903年　数学における型

$$3 : \mathbb{N} \qquad \sin : \mathbb{R} \to \mathbb{R}$$



1958年　プログラミング言語の型

1. *Type declarations*

Type declarations serve to declare certain variables, or functions, to given class, such as the class of integers or class of Boolean values.

Form: $\Delta \sim$ *type* (I, I, ⌣⌣⌣, I, I[ ], ⌣⌣⌣, I[,], ⌣⌣⌣, I[,,], ⌣⌣⌣)

representative of some [illegible] or Boolean

Throughout the program, the variables, or functions named by the iden

ソフトウェア科学

より高度な型でバグを防ぐ

```
IO ()
```

```
{ v:int | v>0 }
```

より精密に検査

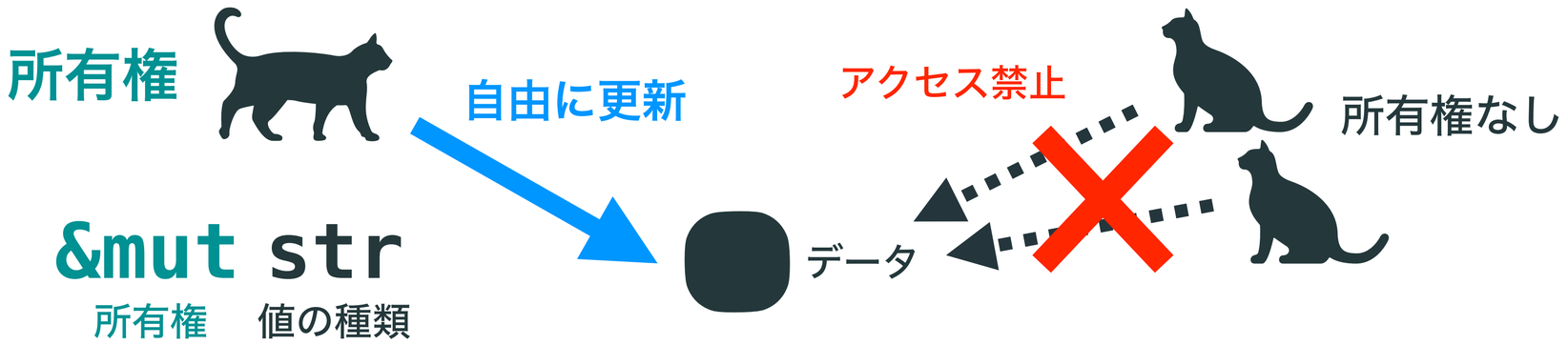
```
[int] ⊸ int
```

```
&mut str
```

Rust プログラミング言語

1972 C言語 ⋯▶ 1983 C++ ⋯▶ 2015 Rust

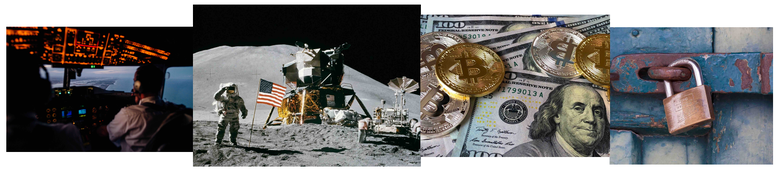
所有権型 1998年～ で競合のバグを防ぐ



採用例　Google Android　深刻なバグが激減

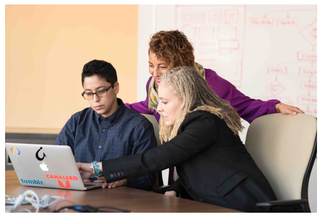
ソフトウェア科学 & 工学 のトピック

ソフトウェア検証



ソフトウェア最適化

プログラム生成



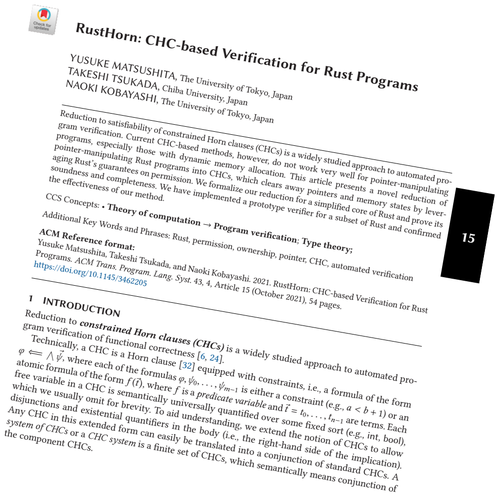
開発の実態の分析

研究紹介　RustHorn

松下・塚田・小林　2020

Rust プログラムの自動検証

Rustプログラム　→（所有権型／未来の先取り）→　論理的モデル



RustHorn: CHC-based Verification for Rust Programs

YUSUKE MATSUSHITA, The University of Tokyo, Japan
TAKESHI TSUKADA, Chiba University, Japan
NAOKI KOBAYASHI, The University of Tokyo, Japan

15

Reduction to satisfiability of constrained Horn clauses (CHCs) is a widely studied approach to automated program verification. Current CHC-based methods, however, do not work very well for pointer-manipulating programs, especially those with dynamic memory allocation. This article presents a novel reduction of pointer-manipulating Rust programs into CHCs, which clears away pointers and memory states by leveraging Rust's guarantees on permission. We formalize our reduction for a simplified core of Rust and prove its soundness and completeness. We have implemented a prototype verifier for a subset of Rust and confirmed the effectiveness of our method.

CCS Concepts: • **Theory of computation → Program verification**; **Type theory**;

Additional Key Words and Phrases: Rust, permission, ownership, pointer, CHC, automated verification

ACM Reference format:
Yusuke Matsushita, Takeshi Tsukada, and Naoki Kobayashi. 2021. RustHorn: CHC-based Verification for Rust Programs. *ACM Trans. Program. Lang. Syst.* 43, 4, Article 15 (October 2021), 54 pages.
https://doi.org/10.1145/3462205

1 INTRODUCTION

Reduction to ***constrained Horn clauses (CHCs)*** is a widely studied approach to automated program verification of functional correctness [6, 24].

Technically, a CHC is a Horn clause [32] equipped with constraints, i.e., a formula of the form $\varphi \Leftarrow \bigwedge \tilde{\psi}$, where each of the formulas $\varphi, \psi_0, \ldots, \psi_{m-1}$ is either a constraint (e.g., $a < b + 1$) or an atomic formula of the form $f(\tilde{t})$, where f is a *predicate variable* and $\tilde{t} = t_0, \ldots, t_{n-1}$ are terms. Each free variable in a CHC is semantically universally quantified over some fixed sort (e.g., int, bool), which we usually omit for brevity. To aid understanding, we extend the notion of CHCs to allow disjunctions and existential quantifiers in the body (i.e., the right-hand side of the implication). Any CHC in this extended form can easily be translated into a conjunction of standard CHCs. A *system of CHCs* or a *CHC system* is a finite set of CHCs, which semantically means conjunction of the component CHCs.

貸　借

a

未来　a'　(a, a')

$a' = \cdots$

a'

To be continued...

Q & A

Q&A

【学生講演】講師：博士課程3年　松下祐介

ファシリテーター：修士課程1年　中山崇

「ソフトウェアの科学 ～バグのない世界を目指して～」

講師への質問を募集しています

【質問受付時間】2023年8月2日（水）10:10-11:00

【質問方法は2通り】

1. **講演画面下のSlidoのURL （https://www.sli.do/jp）をクリックして、アクセスコード＃ 2358739を入力してください。**
2. **右下のQRコードをスマートフォンで読み取り、質問を入力してください。**

こちらで参加：
slido.com
#2358 739