● 松下 祐介 特定助教

Yusuke MATSUSHITA (Program-Specific Assistant Professor)

研究課題： Rust から広がる新時代のソフトウェア開発の探究
（Exploring a New Age of Software Development Springing from Rust）
専門分野： ソフトウェア科学（Software Science）
受入先部局： 情報学研究科（Graduate School of Informatics）
前職の機関名： 京都大学 情報学研究科（Graduate School of Informatics, Kyoto University）

子どもの頃から、数学や言語、音楽に夢中でした。小学生のときプログラミングの楽しさを知りました。そうした好奇心が高じて、私は高校の頃にソフトウェア科学に触れて、今ではこの分野の研究者となりました。

私たちの暮らしが実に多様なソフトウェアに支えられているなかで、理想的なソフトウェアを開発するための方法論を、数学、論理、言語などの方向から探究するのが、ソフトウェア科学です。

1987 年、フランスの論理学者ジラールは、線形論理という不思議な論理を提唱しました。これに刺激を受けたソフトウェア科学の研究をもとに、プログラミング言語「Rust」が 2015 年に誕生し、その安全性の高さから産業で急速に普及しています。

私はこの Rust の面白さに魅了されて、ソフトウェア科学者となりました。白眉プロジェクトで私は、線形論理に由来し Rust で初めて本格的に実用化された「所有権」の考え方を更に発展させて、高度な安全性、性能、機能性を実現できる、新時代のソフトウェア開発技術を探究します。

Since I was a little boy, I was fascinated by math, language and music. I learned the fun of programming when I was in elementary school. Such curiosity led me to software science in high school, and I am now a researcher in this field.

As our lives are supported by a wide variety of software, software science explores sound methodologies for developing ideal software from the perspectives of mathematics, logic, language, and more.

In 1987, French logician Girard proposed a mysterious logic called linear logic. Software science studies inspired by this led to the birth of the Rust programming language in 2015, and Rust is rapidly gaining popularity in the industry for its highly secure nature.

I was enchanted by the fun of Rust, which drove me to become a software scientist. In the Hakubi project, I explore new-age software development technologies that can achieve high levels of security, performance and functionality, by further advancing the idea of ownership, which originated in linear logic and was first put to full practical use by Rust.

ソフトウェア科学とは？

私たちの暮らしは実に多様なソフトウェアに支えられています。パソコンやスマホのアプリも、その裏側で描画、通信、スケジューリングを動かしているのも、家電、銀行の ATM、病院の MRI、信号機、自動車、航空機、宇宙船を制御しているのも、皆ソフトウェアです。

ソフトウェア科学は、人間が理想とするソフトウェアを開発するための方法論を、数学、論理、言語などの方向から探究する学問です。特に、ソフトウェアにおける致命的なバグを防ぐことが核となるゴールで、そのために様々なアプローチが試みられています。例えば、プログラミング言語の設計を工夫すれば、意図した計算をより高度に抽象化して記述できるので、ミスが防げます。また、プログラムが意図した仕様を満たすことを数学・論理的なアプローチで証明する形式検証も、大事な技術です。

Rust とは？

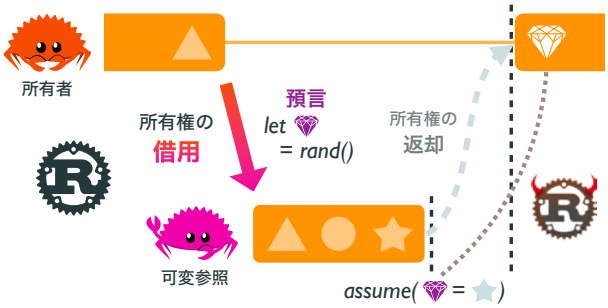
1987 年、フランスの論理学者ジラールは、線形論理という論理を提唱しました。伝統的な論理では命題は永遠に不変な真理を表しますが、線形論理では命題は

複製・破棄が制限された不思議な振る舞いをします。ジラールはこの論理が動的に変化するもの、特にコンピュータ上の計算について論じるのに有用だと考えました。このアイデアに刺激を受けて、ソフトウェア科学でも様々な研究が生まれました。

そうした知見をもとに 2015 年に誕生したのが、プログラミング言語「Rust」です。Rust は、線形論理に由来する「所有権」の考え方にもとづく斬新な型システムにより、高い安全性を実現しています。型システムとは、プログラムで扱うデータに型というラベルを付けて、それらの整合性を自動で実行前に検査して、バグを防ぐ仕組みです。所有権とは、大まかに言って共有されたデータへのアクセス権です。Rust は特に、ハードウェアに近い層で生じる致命的な脆弱性を防ぐことができるため、C や C++ といった従来の言語に変わる存在として、産業で急速に普及し、目覚ましい活躍を遂げています。Rust はソフトウェア科学の新たな可能性を見せてくれる存在だと言えます。

Rust と私

私は東京大学に入学した頃に Rust に出会い、その面白さに魅了されました。論理的に安全性を追求しながら、妥協せずに高い性能の計算を実現する、というのが、私の性格に合ったのかもしれません。私はソフトウェア科学を研究すべく、理学部情報科学科の小林研究室に入り、Rust についての卒業研究をしました。そこで生まれたのが、私の代表作 RustHorn [1] です。これは、Rust プログラムを所有権の力で数学的にモデル化して検証する新手法を提唱し、実証した研究です。Rust には所有権を明示的に渡さずに通信できる「借用」という便利な仕組みがあるのですが、これは理論的にも面白くて、数学的なモデル化が非自明なものでした。この研究では、未来を先取りする「預言」というアイデアでこれを解決しました（図）。この研究は、Creusot という実用的な検証器に取り入れられるなど、大きな広がりを見せています。また私は、ドイツ MPI-SWS RustBelt グループでインターンをして、RustHorn の手法の一般的な論理的基礎を確立する研究をしました [2]。



図：RustHorn の預言による借用のモデル化の概略

Rust から広がる研究

Rust についてはまだまだ大きい研究課題が色々あります。その一つが unsafe コードの検証です。Rust では所有権関連の制約を型で記述できます。通常それらの整合性は自動検査されますが、時には安全性の保証のない「unsafe」コードを書く必要があり、これはバグの温床となるため検証が重要です。私は特に動的検証に着目していて、所有権の制約が実行時にどのように正確に高速に検査できるか、探究しています。

所有権の考え方は、より広い文脈でも応用できます。例えば、量子計算では、複製・破棄が制限される量子状態を扱うため、所有権のアイデアが役に立って、Rust の発想を応用した量子プログラミング言語や、線形論理に由来する量子プログラム論理などが提案されています。確率・統計的な計算や、暗号によるセキュリティなどについても、所有権が使えます。最近私はこうした領域の研究も始めました。

私は、Rust の希望の光を糧に、新時代のソフトウェア開発を切り拓く技術を、柔軟な発想で広く探究していきたいと思います。

参考文献

[1] Yusuke Matsushita, Takeshi Tsukada, and Naoki Kobayashi. 2021. RustHorn: CHC-based Verification for Rust Programs. ACM TOPLAS 43, 4, Article 15. https://doi.org/10.1145/3462205

[2] Yusuke Matsushita, Xavier Denis, Jacques-Henri Jourdan, and Derek Dreyer. 2022. RustHornBelt: A Semantic Foundation for Functional Verification of Rust Programs with Unsafe Code. In Proceedings of ACM PLDI 2022. https://doi.org/10.1145/3519939.3523704