

Detección de Deadlocks en Rust en tiempo de compilación mediante Redes de Petri

Horacio Lisdero Scaffino

Facultad de Ingeniería
Universidad de Buenos Aires

19 de febrero del 2023

Agenda

- 1 Introducción
- 2 Rust
 - ¿Qué es Rust?
 - ¿Por qué Rust?
- 3 Redes de Petri
 - ¿Qué es una red de Petri?
 - Ejemplos
 - ¿Por qué usar redes de Petri?
- 4 Traducción
 - MIR
 - Modelado de hilos de ejecución
 - Modelado de un mutex
 - Modelado de variables de condición
- 5 Bibliografía
- 6 Recursos adicionales

Agenda

1 Introducción

2 Rust

- ¿Qué es Rust?
- ¿Por qué Rust?

3 Redes de Petri

- ¿Qué es una red de Petri?
- Ejemplos
- ¿Por qué usar redes de Petri?

4 Traducción

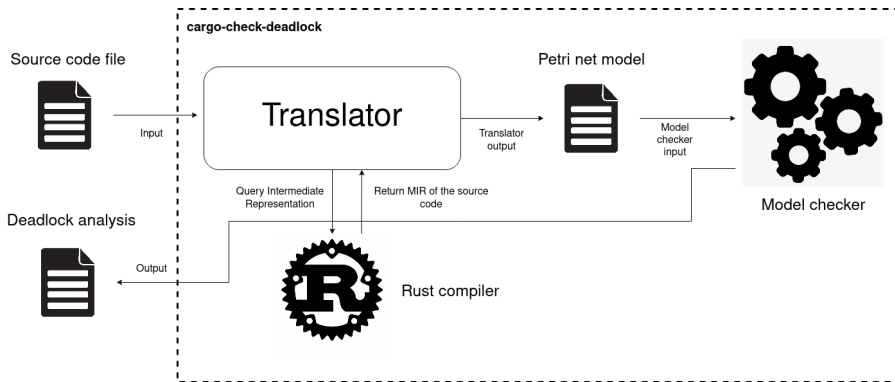
- MIR
- Modelado de hilos de ejecución
- Modelado de un mutex
- Modelado de variables de condición

5 Bibliografía

6 Recursos adicionales

Una vista general de la herramienta

El traductor es el componente principal. El verificador de modelos y el compilador de Rust, *rustc*, son dependencias.



Agenda

1 Introducción

2 Rust

- ¿Qué es Rust?
- ¿Por qué Rust?

3 Redes de Petri

- ¿Qué es una red de Petri?
- Ejemplos
- ¿Por qué usar redes de Petri?

4 Traducción

- MIR
- Modelado de hilos de ejecución
- Modelado de un mutex
- Modelado de variables de condición

5 Bibliografía

6 Recursos adicionales

Agenda

1 Introducción

2 Rust

- ¿Qué es Rust?
- ¿Por qué Rust?

3 Redes de Petri

- ¿Qué es una red de Petri?
- Ejemplos
- ¿Por qué usar redes de Petri?

4 Traducción

- MIR
- Modelado de hilos de ejecución
- Modelado de un mutex
- Modelado de variables de condición

5 Bibliografía

6 Recursos adicionales

¿Qué es Rust?

Rust es un lenguaje de programación multiparadigma de uso general que busca ofrecer a los desarrolladores una forma segura y eficiente de escribir código de bajo nivel.

¿Qué es Rust?

Rust es un lenguaje de programación multiparadigma de uso general que busca ofrecer a los desarrolladores una forma segura y eficiente de escribir código de bajo nivel.

- Uso seguro de la memoria (*Memory safe*)
- Compilado a código máquina, con un *runtime* mínimo
- Expresividad de un lenguaje de alto nivel
- Performance de un lenguaje de bajo nivel (similar a C o C++)

Breve línea de tiempo de Rust

- 2007** Inicio como un proyecto personal de Graydon Hoare, un programador en Mozilla
- 2009** Mozilla comienza a patrocinar oficialmente el proyecto
- 2015** Primera versión estable 1.0
- 2016** Mozilla lanza Servo, un motor de navegador construido con Rust
- 2019** Se estabiliza el soporte de **async/await**
- 2021** AWS, Huawei, Google, Microsoft y Mozilla crean la Rust Foundation.
- 2021** El proyecto Android fomenta el uso de Rust para los componentes del SO por debajo del ART
- 2022** El kernel de Linux añade soporte para Rust junto con C
- 2023** 8 años consecutivos el lenguaje de programación más querido en la Stack Overflow Developer Survey

Uso seguro de la memoria

Logra un uso seguro de la memoria sin recurrir a un recolector de basura o a un contador de referencias. En su lugar, utiliza el concepto de **ownership** (propiedad) y **borrowing** (préstamo).

Evita una gran variedad de clases de errores en tiempo de compilación:

- Double free
- Use after free
- Punteros colgantes (*Dangling pointers*)
- Condiciones de carrera (*Data races*)
- Pasaje de variables no seguras entre hilos

Si se encuentra una violación de las reglas del compilador, el programa simplemente no compilará.

Agenda

1 Introducción

2 Rust

- ¿Qué es Rust?
- ¿Por qué Rust?

3 Redes de Petri

- ¿Qué es una red de Petri?
- Ejemplos
- ¿Por qué usar redes de Petri?

4 Traducción

- MIR
- Modelado de hilos de ejecución
- Modelado de un mutex
- Modelado de variables de condición

5 Bibliografía

6 Recursos adicionales

El uso seguro de la memoria es fundamental para la fiabilidad y la seguridad

Investigaciones empíricas han concluido que alrededor del 70 % de las vulnerabilidades encontradas en grandes bases de código en C/C++ se deben a errores de gestión de memoria. Esta elevada cifra puede observarse en proyectos como:

- Android Open Source Project [1],
- los componentes Bluetooth y multimedia de Android [2],
- el Proyecto Chromium detrás del navegador web Chrome [3],
- el componente CSS de Firefox [4],
- iOS y macOS [5],
- productos de Microsoft [6, 7],
- Ubuntu [8]

La adopción de Rust aumenta velozmente

- El proyecto Android Open Source fomenta el uso de Rust para los componentes del SO por debajo del ART [9].
- El kernel Linux versión 6.1 introduce soporte oficial de las herramientas para programar componentes en Rust [10, 11].
- En Mozilla, el proyecto Oxidation se creó en para aumentar el uso de Rust en Firefox y aplicaciones relacionadas. En marzo de 2023, las líneas de código en Rust representan más del 10 % del total en Firefox Nightly [12].
- En Meta, el uso de Rust como lenguaje de desarrollo *server-side* está aprobado y se fomenta desde julio de 2022 [13].
- En Cloudflare, se construyó desde cero un nuevo proxy HTTP en Rust para superar las limitaciones arquitectónicas de NGINX, reduciendo el uso de CPU en un 70 % y el uso de memoria en un 67 % [14].
- En Discord, reimplementar un servicio crucial en Rust proporcionó grandes beneficios de performance y solucionó un problema ligado a la recolección de basura en Go [15].
- En npm Inc, la empresa detrás del registro npm, Rust permitió escalar los servicios ligados a la CPU a más de 1300 millones de descargas diarias [16].
- Un estudio del código basado en Rust descubrió que se ejecuta tan eficientemente que consume la mitad de electricidad que un programa similar escrito en Java, un lenguaje de uso común en AWS [17].

Las herramientas son modernas y fáciles de usar

- **cargo**, el gestor de paquetes oficial: Formatear, compilar, testear, lint, actualizar y publicar paquetes.
- **Test harness** para pruebas unitarias, pruebas de integración y pruebas en comentarios de documentación (doctests). No se necesitan bibliotecas de terceros.
- Un registro público oficial para los paquetes Rust (llamados “crates”): <https://crates.io/>.
- Generación automática de un sitio web estático a partir de los comentarios en el código del proyecto. Se publica automáticamente en <https://docs.rs/>.
- Un linter oficial incluido en la instalación por defecto que detecta aún más errores y detecta código no idiomático: **clippy**.
- La integración con git, GitHub, VSCode, IntelliJ es excelente y fácil de usar.
- Una nueva versión estable del compilador cada 6 semanas [18].

Agenda

1 Introducción

2 Rust

- ¿Qué es Rust?
- ¿Por qué Rust?

3 Redes de Petri

- ¿Qué es una red de Petri?
- Ejemplos
- ¿Por qué usar redes de Petri?

4 Traducción

- MIR
- Modelado de hilos de ejecución
- Modelado de un mutex
- Modelado de variables de condición

5 Bibliografía

6 Recursos adicionales

Agenda

1 Introducción

2 Rust

- ¿Qué es Rust?
- ¿Por qué Rust?

3 Redes de Petri

- ¿Qué es una red de Petri?
- Ejemplos
- ¿Por qué usar redes de Petri?

4 Traducción

- MIR
- Modelado de hilos de ejecución
- Modelado de un mutex
- Modelado de variables de condición

5 Bibliografía

6 Recursos adicionales

Definición informal

Una red de Petri es una herramienta de modelado matemático utilizada para describir y analizar el comportamiento de sistemas concurrentes. Proporciona una representación gráfica del estado del sistema y sus transiciones, permitiendo el análisis visual y formal de procesos complejos.



- Lugares: Representan estados en el sistema (*circles*)
- Transiciones: Representan eventos o acciones que ocurren en el sistema (*rectángulos*).
- Tokens: Marcas dentro de lugares que son creadas y consumidas por las transiciones (*puntos dentro de lugares*)

Definición matemática

Una red de Petri es una 5-tupla, $PN = (P, T, F, W, M_0)$ donde:

$P = \{p_1, p_2, \dots, p_m\}$ es un conjunto finito de lugares,

$T = \{t_1, t_2, \dots, t_n\}$ es un conjunto finito de transiciones,

$F \subseteq (P \times T) \cup (T \times P)$ es un conjunto de arcos (relación de flujo),

$W : F \rightarrow \{1, 2, 3, \dots\}$ es una función de peso para los arcos,

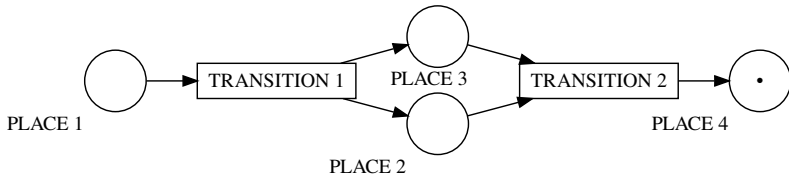
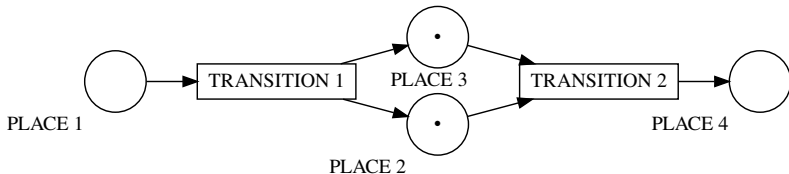
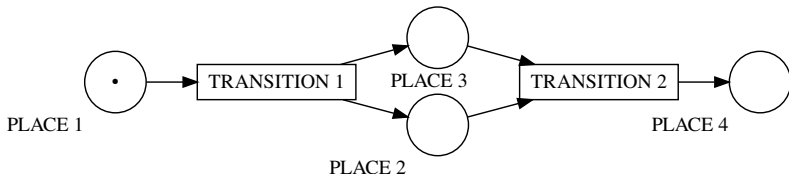
$M_0 : P \rightarrow \{0, 1, 2, 3, \dots\}$ es el marcado inicial,

$P \cap T = \emptyset$ y $P \cup T \neq \emptyset$

El grafo es por definición *bipartito*. Solo puede haber aristas:

- de lugares a transiciones
- de transiciones a lugares

Regla de disparo de transición



Agenda

1 Introducción

2 Rust

- ¿Qué es Rust?
- ¿Por qué Rust?

3 Redes de Petri

- ¿Qué es una red de Petri?
- **Ejemplos**
- ¿Por qué usar redes de Petri?

4 Traducción

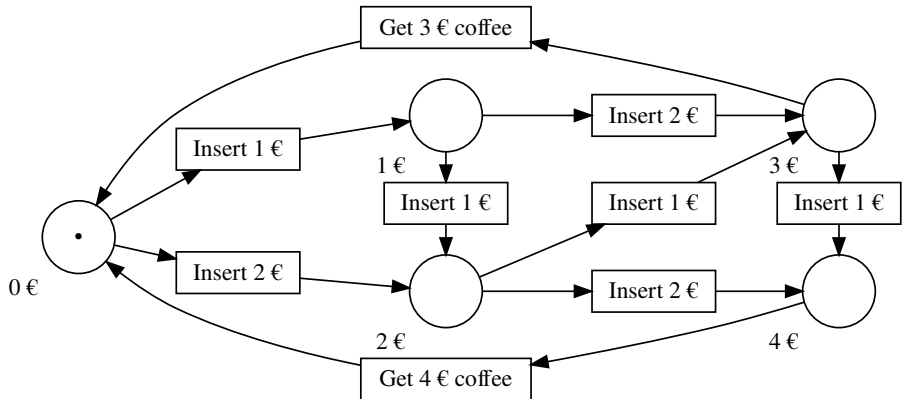
- MIR
- Modelado de hilos de ejecución
- Modelado de un mutex
- Modelado de variables de condición

5 Bibliografía

6 Recursos adicionales

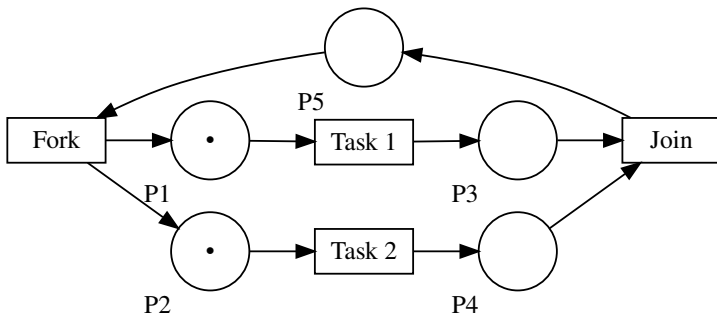
Máquina expendedora

Esta es una máquina de estados finitos (*Finite-state machine*), una subclase de redes de Petri.



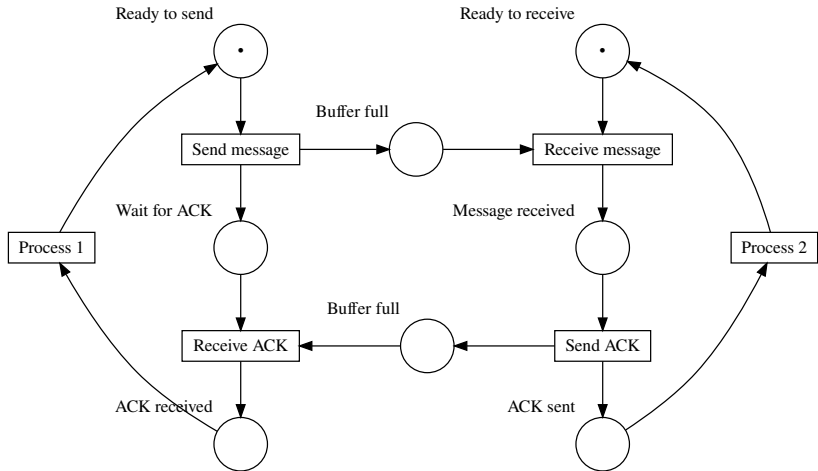
Actividades en paralelo: Fork/Join

Esto es un grafo marcado (*Marked graph*), una subclase de redes de Petri. Nótese la concurrencia entre la Tarea 1 y la Tarea 2. Esto no puede modelar con una única máquina de estados finitos.



Protocolos de comunicación: Send with ACK

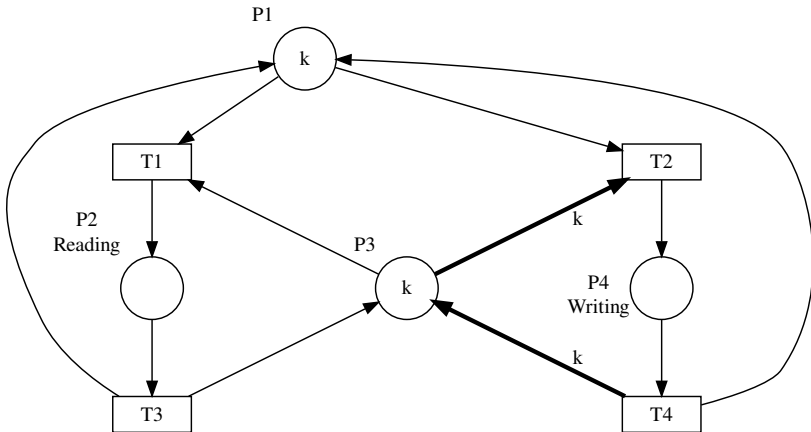
Un protocolo simple en el que el Proceso 1 envía mensajes al Proceso 2 y espera a recibir un acuse de recibo (*acknowledgment*) antes de continuar. Para simplificar, no se ha incluido ningún mecanismo de *timeout*.



Sincronización: Lectores y escritores

Un sistema de redes de Petri con k procesos que leen o escriben un valor compartido.

- Si un proceso escribe, entonces ningún proceso puede leer.
- Si un proceso está leyendo, entonces ningún proceso puede escribir.
- Sólo puede haber cero o un proceso escribiendo en cualquier momento dado.



Agenda

1 Introducción

2 Rust

- ¿Qué es Rust?
- ¿Por qué Rust?

3 Redes de Petri

- ¿Qué es una red de Petri?
- Ejemplos
- ¿Por qué usar redes de Petri?

4 Traducción

- MIR
- Modelado de hilos de ejecución
- Modelado de un mutex
- Modelado de variables de condición

5 Bibliografía

6 Recursos adicionales

Análisis de alcance

Las redes de Petri pueden analizarse utilizando métodos formales para concluir si la red puede alcanzar un deadlock o no. Existe una noción de *liveness* análoga a la que se encuentra en los sistemas informáticos.

Análisis de alcance

Las redes de Petri pueden analizarse utilizando métodos formales para concluir si la red puede alcanzar un deadlock o no. Existe una noción de *liveness* análoga a la que se encuentra en los sistemas informáticos.

Existen varios verificadores de modelos de última generación e incluso existe un Model Checking Contest que se celebra todos los años. Las herramientas más avanzadas pueden manejar modelos de redes de Petri con más de **70 000 transiciones** y **un millón de lugares**.

Análisis de alcance

Las redes de Petri pueden analizarse utilizando métodos formales para concluir si la red puede alcanzar un deadlock o no. Existe una noción de *liveness* análoga a la que se encuentra en los sistemas informáticos.

Existen varios verificadores de modelos de última generación e incluso existe un Model Checking Contest que se celebra todos los años. Las herramientas más avanzadas pueden manejar modelos de redes de Petri con más de **70 000 transiciones** y **un millón de lugares**.

Traducir el código fuente a una red de Petri se ha hecho antes para otros lenguajes de programación [19, 20] y también para Rust [21, 22]. La dificultad reside en soportar más primitivas de sincronización que simples mutexes y traducir código de aplicaciones del mundo real.

Agenda

1 Introducción

2 Rust

- ¿Qué es Rust?
- ¿Por qué Rust?

3 Redes de Petri

- ¿Qué es una red de Petri?
- Ejemplos
- ¿Por qué usar redes de Petri?

4 Traducción

- MIR
- Modelado de hilos de ejecución
- Modelado de un mutex
- Modelado de variables de condición

5 Bibliografía

6 Recursos adicionales

Compilation stages in *rustc*

- Lexing: The source text is turned into a stream of atomic source code units known as tokens.

Compilation stages in *rustc*

- Lexing: The source text is turned into a stream of atomic source code units known as tokens.
- Parsing: The stream of tokens is converted into an **Abstract Syntax Tree**.

Compilation stages in *rustc*

- Lexing: The source text is turned into a stream of atomic source code units known as tokens.
- Parsing: The stream of tokens is converted into an **Abstract Syntax Tree**.
- High-level Intermediate Representation (HIR):
 - Desugar loops: **while** and **for** to simple **loop**.
 - Type inference: The automatic detection of a type of an expression.
 - Trait solving: Ensuring that each implementation block (**impl**) points to a valid trait.
 - Type checking.

Compilation stages in *rustc*

- Lexing: The source text is turned into a stream of atomic source code units known as tokens.
- Parsing: The stream of tokens is converted into an **Abstract Syntax Tree**.
- High-level Intermediate Representation (HIR):
 - Desugar loops: **while** and **for** to simple **loop**.
 - Type inference: The automatic detection of a type of an expression.
 - Trait solving: Ensuring that each implementation block (**impl**) points to a valid trait.
 - Type checking.
- Mid-level Intermediate Representation (MIR):
 - Checking of exhaustiveness of pattern matching.
 - Desugar method calls to function calls
(`x.method(y)` becomes `Type::method(&x, y)`).
 - Add implicit dereferencing operations.
 - Borrow checking.

Compilation stages in *rustc*

- Lexing: The source text is turned into a stream of atomic source code units known as tokens.
- Parsing: The stream of tokens is converted into an **Abstract Syntax Tree**.
- High-level Intermediate Representation (HIR):
 - Desugar loops: **while** and **for** to simple **loop**.
 - Type inference: The automatic detection of a type of an expression.
 - Trait solving: Ensuring that each implementation block (**impl**) points to a valid trait.
 - Type checking.
- Mid-level Intermediate Representation (MIR):
 - Checking of exhaustiveness of pattern matching.
 - Desugar method calls to function calls
(`x.method(y)` becomes `Type::method(&x, y)`).
 - Add implicit dereferencing operations.
 - Borrow checking.
- Code generation:
 - *rustc* relies on LLVM as a backend.
 - It leverages many optimizations of the LLVM intermediate representation.
 - LLVM takes over from this point on.
 - At the end, object files are linked to create an executable.

Agenda

1 Introducción

2 Rust

- ¿Qué es Rust?
- ¿Por qué Rust?

3 Redes de Petri

- ¿Qué es una red de Petri?
- Ejemplos
- ¿Por qué usar redes de Petri?

4 Traducción

- **MIR**
- Modelado de hilos de ejecución
- Modelado de un mutex
- Modelado de variables de condición

5 Bibliografía

6 Recursos adicionales

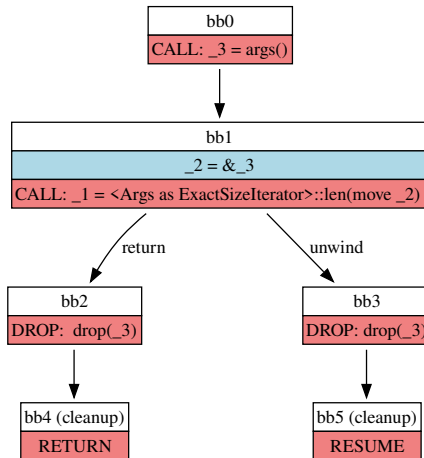
Hello World in MIR

BB means “basic block”. Each one is formed by statements and one terminator statement. The terminator statement is the only place where the control flow can jump to another basic block.

```
1  fn main() -> () {
2      let mut _0: ();
3      let _1: ();
4      let mut _2: std::fmt::Arguments<'_>;
5      let mut _3: &[&str];
6      let mut _4: &[&str; 1];
7
8      bb0: {
9          _4 = const _;
10         _3 = _4 as &[&str] (Pointer(Unsize));
11         _2 = Arguments::<'_>::new_const(move _3) -> bb1;
12     }
13
14     bb1: {
15         _1 = _print(move _2) -> bb2;
16     }
17
18     bb2: {
19         return;
20     }
21 }
```

MIR como un grafo que muestra el flujo de ejecución

The MIR is a form of control flow graph (CFG) used in compilers. In this form, the translation to a Petri net becomes evident.



Agenda

1 Introducción

2 Rust

- ¿Qué es Rust?
- ¿Por qué Rust?

3 Redes de Petri

- ¿Qué es una red de Petri?
- Ejemplos
- ¿Por qué usar redes de Petri?

4 Traducción

- MIR
- **Modelado de hilos de ejecución**
- Modelado de un mutex
- Modelado de variables de condición

5 Bibliografía

6 Recursos adicionales

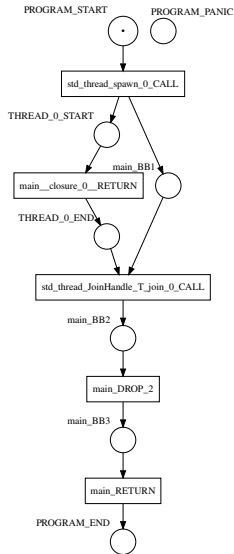
Programa de ejemplo

Let's consider a trivial program that spawns a thread that does nothing and immediately joins it.

```
1  fn main() {  
2      let thread_join_handle = std::thread::spawn(move || {  
3          // some work here  
4      });  
5      // some work here  
6      let _res = thread_join_handle.join();  
7  }
```

- `std::thread::spawn` should create an additional token that models the program counter of the second thread.
- The joining thread should wait until the spawned thread finishes.

Modelo de red de Petri de un thread



Agenda

1 Introducción

2 Rust

- ¿Qué es Rust?
- ¿Por qué Rust?

3 Redes de Petri

- ¿Qué es una red de Petri?
- Ejemplos
- ¿Por qué usar redes de Petri?

4 Traducción

- MIR
- Modelado de hilos de ejecución
- **Modelado de un mutex**
- Modelado de variables de condición

5 Bibliografía

6 Recursos adicionales

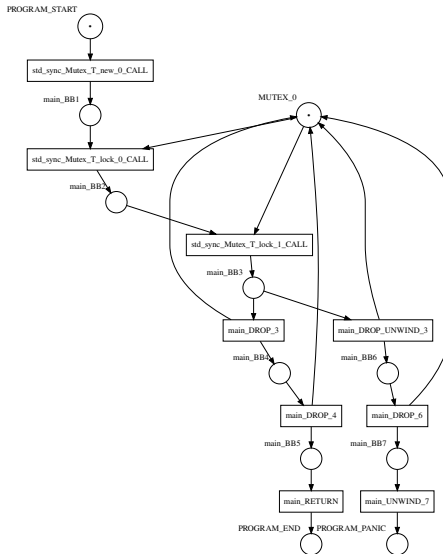
Programa de ejemplo

Consider a simple program that locks a mutex twice. The second lock operation will deadlock because the lock handle returned by the first call to `std::sync::Mutex::lock` is not dropped until it falls out of scope.

```
1  fn main() {  
2      let data = std::sync::Mutex::new(0);  
3      let _d1 = data.lock();  
4      let _d2 = data.lock(); // cannot lock, since d1 is still active  
5  }
```

- There should be a single place that models the mutex.
- Locking the mutex is taking the token from the mutex place.
- Unlocking the mutex is setting the token back in the mutex place.

Modelo de red de Petri de un mutex



Agenda

1 Introducción

2 Rust

- ¿Qué es Rust?
- ¿Por qué Rust?

3 Redes de Petri

- ¿Qué es una red de Petri?
- Ejemplos
- ¿Por qué usar redes de Petri?

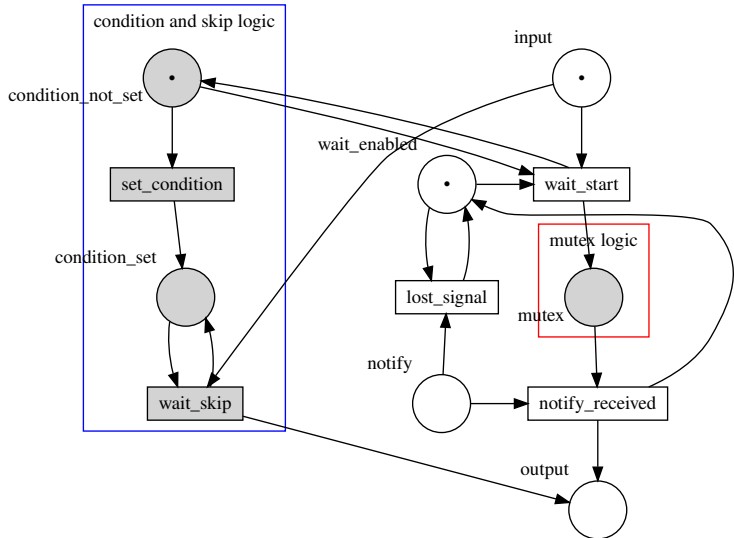
4 Traducción

- MIR
- Modelado de hilos de ejecución
- Modelado de un mutex
- **Modelado de variables de condición**

5 Bibliografía

6 Recursos adicionales

Cómo modelar una *condition variable*



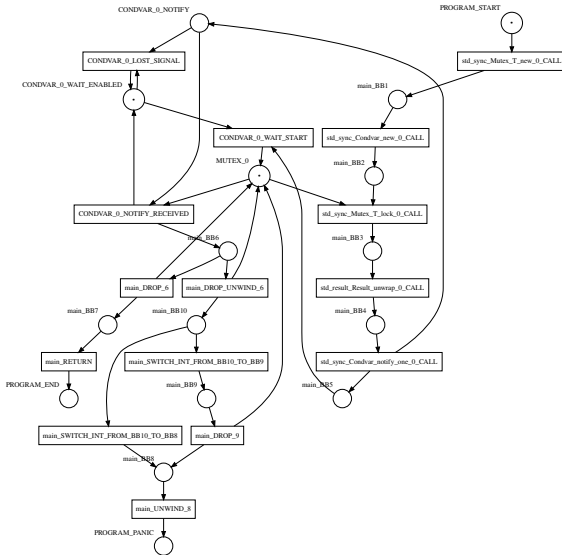
Programa de ejemplo

We have to use a very simple example program to keep the net small. In this case, the thread is trying to notify itself, which leads to a lost signal.

```
1  fn main() {  
2      let mutex = std::sync::Mutex::new(false);  
3      let cvar = std::sync::Condvar::new();  
4      let mutex_guard = mutex.lock().unwrap();  
5      cvar.notify_one();  
6      let _result = cvar.wait(mutex_guard);  
7  }
```

- The model for the condition variable should appear in the Petri net.
- The notify place should be set.
- But the signal gets consumed because `std::sync::Condvar::wait` was not called.

Modelo de red de Petri para el programa de ejemplo



Agenda

1 Introducción

2 Rust

- ¿Qué es Rust?
- ¿Por qué Rust?

3 Redes de Petri

- ¿Qué es una red de Petri?
- Ejemplos
- ¿Por qué usar redes de Petri?

4 Traducción

- MIR
- Modelado de hilos de ejecución
- Modelado de un mutex
- Modelado de variables de condición

5 Bibliografía

6 Recursos adicionales

Bibliografía I



E. Stepanov, "Detecting Memory Corruption Bugs With HWSan." <https://android-developers.googleblog.com/2020/02/detecting-memory-corruption-bugs-with-hwasan.html>, 2020.
Accessed on 2023-02-24.



J. V. Stoep and C. Zhang, "Queue the Hardening Enhancements." <https://android-developers.googleblog.com/2020/02/detecting-memory-corruption-bugs-with-hwasan.html>, 2020.
Accessed on 2023-02-24.



The Chromium Projects, "Memory safety."
<https://www.chromium.org/Home/chromium-security/memory-safety/>, 2015.
Accessed on 2023-02-24.



D. Hosfelt, "Implications of Rewriting a Browser Component in Rust."
<https://hacks.mozilla.org/2019/02/rewriting-a-browser-component-in-rust/>, 2019.
Accessed on 2023-02-24.



P. Kehrer, "Memory Unsafety in Apple's Operating Systems."
<https://langui.sh/2019/07/23/apple-memory-safety/>, 2019.
Accessed on 2023-02-24.



M. Miller, "Trends, Challenges, and Strategic Shifts in the Software Vulnerability Mitigation Landscape."
<https://www.youtube.com/watch?v=PjbG0jjnBZQ>, 2019.
Accessed on 2023-02-24.



S. Fernandez, "A proactive approach to more secure code."
<https://msrc.microsoft.com/blog/2019/07/a-proactive-approach-to-more-secure-code/>, 2019.
Accessed on 2023-02-24.

Bibliografía II



A. Gaynor, "What science can tell us about C and C++'s security."

<https://alexgaynor.net/2020/may/27/science-on-memory-unsafety-and-security/>, 2020.

Accessed on 2023-02-24.



J. V. Stoep and S. Hines, "Rust in the Android platform."

<https://security.googleblog.com/2021/04/rust-in-android-platform.html>, 2021.

Accessed on 2023-02-22.



J. Corbet, "The 6.1 kernel is out." <https://lwn.net/Articles/917504/>, 2022.

Accessed on 2023-02-24.



S. D. Simone, "Linux 6.1 Officially Adds Support for Rust in the Kernel."

<https://www.infoq.com/news/2022/12/linux-6-1-rust/>, 2022.

Accessed on 2023-02-24.



Mozilla Wiki, "Oxidation Project." <https://wiki.mozilla.org/Oxidation>, 2015.

Accessed on 2023-03-20.



E. Garcia, "Programming languages endorsed for server-side use at Meta." <https://engineering.fb.com/2022/07/27/developer-tools/programming-languages-endorsed-for-server-side-use-at-meta/>, 2022.

Accessed on 2023-02-24.



Y. Wu and A. Hauck, "How we built Pingora, the proxy that connects Cloudflare to the Internet." <https://blog.cloudflare.com/how-we-built-pingora-the-proxy-that-connects-cloudflare-to-the-internet/>,

2022.

Accessed on 2023-03-20.

Bibliografía III



J. Howarth, "Why Discord is switching from Go to Rust."

<https://discord.com/blog/why-discord-is-switching-from-go-to-rust>, 2020.
Accessed on 2023-03-20.



The Rust Project Developers, "Rust case study: Community makes rust an easy choice for npm," 2019.

<https://www.rust-lang.org/static/pdfs/Rust-npm-Whitepaper.pdf>.



R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. P. Fernandes, and J. Saraiva, "Energy efficiency across programming languages: how do energy, time, and memory relate?," in *Proceedings of the 10th ACM SIGPLAN international conference on software language engineering*, pp. 256–267, 2017.



P. Albini, "RustFest Barcelona - Shipping a stable compiler every six weeks."

<https://www.youtube.com/watch?v=As1gXp5kX1M>, 2019.
Accessed on 2023-02-24.



K. M. Kavi, A. Moshtaghi, and D.-J. Chen, "Modeling multithreaded applications using petri nets," *International Journal of Parallel Programming*, vol. 30, pp. 353–371, 2002.



A. Moshtaghi, "Modeling Multithreaded Applications Using Petri Nets," Master's thesis, The University of Alabama in Huntsville, 2001.



T. Meyer, "A Petri Net semantics for Rust," Master's thesis, Universität Rostock — Fakultät für Informatik und Elektrotechnik, 2020.

<https://github.com/Skasselbard/Granite/blob/master/doc/MasterThesis/main.pdf>.



K. Zhang and G. Liua, "Automatically transform rust source to petri nets for checking deadlocks," *arXiv preprint arXiv:2212.02754*, 2022.

Agenda

1 Introducción

2 Rust

- ¿Qué es Rust?
- ¿Por qué Rust?

3 Redes de Petri

- ¿Qué es una red de Petri?
- Ejemplos
- ¿Por qué usar redes de Petri?

4 Traducción

- MIR
- Modelado de hilos de ejecución
- Modelado de un mutex
- Modelado de variables de condición

5 Bibliografía

6 Recursos adicionales

Recursos para aprender Rust

- [The Rust Book](#): Available online and locally with the default Rust installation.
- [Rust by Example](#): Another official book with a more practical approach.
- [Rustlings](#): Small exercises to get you used to reading and writing Rust code!
- [Comprehensive Rust](#): A three-day Rust course developed by the Android team.
- [Take your first steps with Rust](#): A simple course on Microsoft Learn.
- [Rust Programming Course for Beginners](#) by freeCodeCamp.org.
- [No Boilerplate](#): A Youtube channel mainly dedicated to topics connected with Rust. Some ideas were used for this presentation.

Simuladores en línea de redes de Petri

- A simple simulator by Igor Kim can be found on <https://petri.hp102.ru/>. A tutorial video on Youtube and example nets are included in the tool.
- A complement to this is a series of interactive tutorials by Prof. Wil van der Aalst at the University of Hamburg. These tutorials are Adobe Flash Player files (with extension `.swf`) that modern web browsers cannot execute. Luckily, an online Flash emulator like the one found on https://flashplayer.fullstacks.net/?kind=Flash_Emulator can be used to upload the files and execute them.
- Another online Petri net editor and simulator is <http://www.biregal.com/>. The user can draw the net, add the tokens, and then manually fire transitions.

¿Preguntas?

Links

Tesis <https://github.com/hlisdero/thesis>

Herramienta <https://github.com/hlisdero/cargo-check-deadlock>

Presentación https://github.com/hlisdero/thesis/tree/main/presentation_es

Crate publicado <https://crates.io/crates/cargo-check-deadlock>