# RustHorn: CHC-based Verification for Rust Programs

**ESOP2020**

**Yusuke Matsushita**, Takeshi Tsukada and Naoki Kobayashi

The University of Tokyo, Tokyo, Japan

{yskm24t,tsukada,koba}@is.s.u-tokyo.ac.jp

Talk for PPL2020

# CHCs for Automated Verification

- Reduction to **Constrained Horn Clauses** (CHCs) is a widely studied approach to **automated verification**
  [Grebenshchikov+, 2012] [Bjørner+, 2015]

*Program & Verified Property*

```
int mc91(int n) {
  if (n > 100) return n - 10;
  else return mc91(mc91(n + 11));
}
```

"for any **n** ≦ **101**, **mc91(n) returns 91** *if it terminates*"

*reduction*

*CHC System & Satisfiability*

input output

$$Mc91(n, r) \Longleftarrow n > 100 \ \land\ r = n - 10$$
$$Mc91(n, r) \Longleftarrow n \leq 100 \ \land\ Mc91(n + 11, r') \ \land\ Mc91(r', r)$$
$$r = 91 \Longleftarrow n \leq 101 \ \land\ Mc91(n, r)$$

"this CHC system is **satisfiable**"

A **CHC solver automatically finds out a solution** to the CHC system, e.g.:

$$Mc91(n, r) \ :\Longleftrightarrow\ r = 91 \ \lor\ (n > 100 \ \land\ r = n - 10)$$

# Difficulties with Pointers

- Existing method: **Model the memory as an array** [Gurfinkel+, 2015]
  - **Not very scalable**; *quantified invariants* are involved for even easy programs

*Pointer-Manipulating Program*

```
bool just_rec(int *ma) {
  if (rand() >= 0) return true;
  int old_a = *ma; int b = rand(); just_rec(&b);
  return (old_a == *ma);
}
```

"`just_rec(ma)` **always returns** `true` *if it terminates*"

*reduction by the existing method*

$$JustRec(ma, h, sp, h', sp', r) \impliedby h' = h \land sp' = sp \land r = \text{true}$$

*memory update*

$$JustRec(ma, h, sp, h', sp', r) \impliedby mb = sp'' = sp + 1 \land h'' = h\{mb \leftarrow b\} \land JustRec(mb, h'', sp'', h', sp', r) \land r = (h[ma] == h'[ma])$$

*memory read*

$$r = \text{true} \impliedby JustRec(ma, h, sp, h', sp', r) \land ma \leq sp$$

*address constraint*

Solution: $JustRec(ma, h, sp, h', sp', r) :\iff r = \text{true} \land \forall i \leq sp\,.\, h[i] = h'[i]$

*quantified invariant*

# Our Work

Focusing on programs whose **pointer usages** are managed under **ownership** in the style of the **Rust programming language**,

- We propose a **novel translation** from programs to CHCs **clearing away pointers and heaps**.
  - Also, we formalize and prove its correctness and confirm the effectiveness by experiments.

# Table of Contents

- Ownership and Borrow in Rust
- Our Method
- Formalization and Correctness Proof
- Experiments and Evaluation

# Table of Contents

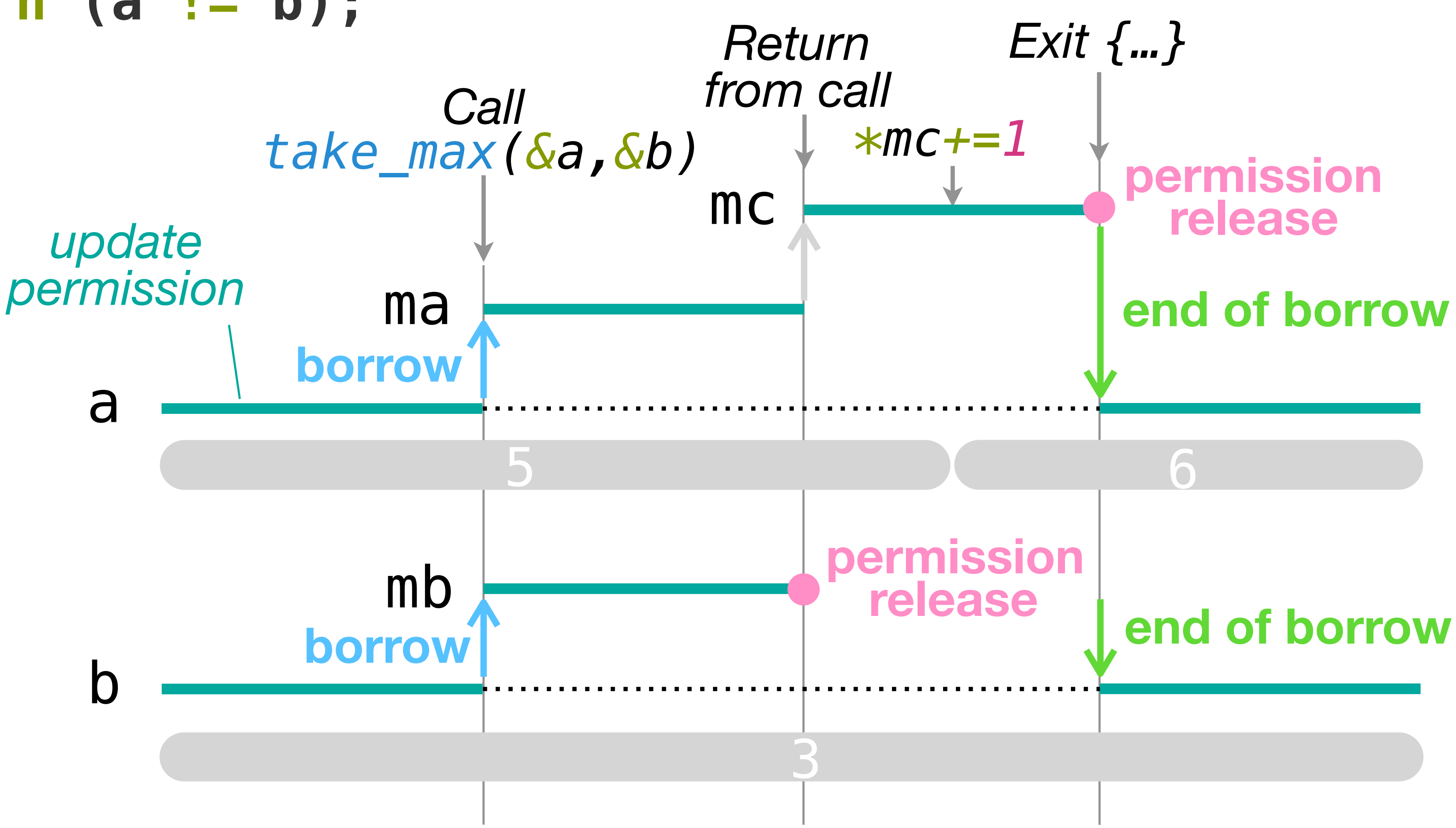
- **Ownership and Borrow in Rust**
- Our Method
- Formalization and Correctness Proof
- Experiments and Evaluation

# Ownership and Borrow in Rust

- The ownership system guarantees that:
  - For each memory cell and **at each moment**, we have
    (i) **only one alias** with the **update permission** to the cell; or
    (ii) **some aliases** with the **read permission** to the cell
  - "*If an alias can read data, any other alias cannot update it*"
- **Borrow**: *a temporary transfer of a permission*
  - This makes Rust really interesting!
  - The end of borrow is statically managed by *lifetimes*

# Example of Borrow

```
int *take_max(int *ma, int *mb) {
  if (*ma >= *mb) return ma; else return mb;
}
bool inc_max(int a, int b) {
  { int *mc = take_max(&a, &b); // borrow a & b
    *mc += 1; } // end of borrow
  return (a != b);
}
```




When `inc_max(5,3)` is called

# Table of Contents

- Ownership and Borrow in Rust
- **Our Method**
- Formalization and Correctness Proof
- Experiments and Evaluation

# Our Method

**mutable reference** (i.e. pointer with a **borrowed** update permission)

- Model a **pointer `ma`** simply as a **pair of values** $\langle a, a_\circ \rangle$
  - the **current value** $a$ & the **value $a_\circ$ at the end of borrow**
  - Access to the **future information** is related to **prophecy variables** [Abadi & Lamport, 1991] [Jung+, 2020]

# Our Method

**Our model of pointer `ma`**

$$\langle a, a_{\circ} \rangle$$

How can this work?

# Example I: `take_max`/`inc_max`

```
int *take_max(int *ma, int *mb) {
  if (*ma >= *mb) return ma; else return mb;
}
bool inc_max(int a, int b) {
  { int *mc = take_max(&a, &b); // borrow a & b
    *mc += 1; } // end of borrow
  return (a != b);
}
```

"for any **a** & **b**, **`inc_max(a,b)`** **returns** `true`"

*reduction by our method*

ma mb

*permission release*

$$TakeMax(\langle a, a_\circ \rangle, \langle b, b_\circ \rangle, r) \iff a \geq b \ \wedge \ b_\circ = b \ \wedge \ r = \langle a, a_\circ \rangle$$

$$TakeMax(\langle a, a_\circ \rangle, \langle b, b_\circ \rangle, r) \iff a < b \ \wedge \ a_\circ = a \ \wedge \ r = \langle b, b_\circ \rangle$$

mc

$$IncMax(a, b, r) \iff TakeMax(\langle a, a_\circ \rangle, \langle b, b_\circ \rangle, \langle c, c_\circ \rangle) \ \wedge \ c' = c + 1 \ \wedge$$

$$c_\circ = c' \ \wedge \ r = (a_\circ \mathrel{!=} b_\circ)$$

*borrow*

*permission release*

$$r = \text{true} \iff IncMax(a, b, r)$$

**Key idea: *set $x_\circ = x$ when the permission of $mx = \langle x, x_\circ \rangle$ is released***

# Example II: `take_some/inc_some`

- **Our method works well with recursive data types!**
  - This makes our method strong and interesting

```
enum List { Cons(i32, Box<List>), Nil } use List::*;

fn take_some(mxs: &mut List) -> &mut i32 {
  match mxs {
    Cons(mx, mxs2) => if rand() { mx } else { take_some(mxs2) }
    Nil => take_some(mxs)
  }
}
```

`take_some(mxs)` takes a ***mutable reference*** to a ***randomly chosen element*** of `*mxs`

```
fn sum(xs:&List)->i32 { match xs {Cons(x,xs2)=>x+sum(xs2),Nil=>0} }

fn inc_some(mut xs: List) -> bool {
  let n = sum(&xs); let my = take_some(&mut xs);
  *my += 1; sum(&xs) == n + 1
}
```

`inc_some(xs)` ***increments some element*** of `xs` and ***checks*** that the ***sum*** has increased by *1*

# Example II: `take_some/inc_some`

```
fn take_some(mxs: &mut List) -> &mut i32 {
  match mxs {
    Cons(mx, mxs2) => if rand() { mx } else { take_some(mxs2) }
    Nil => take_some(mxs)
  }
}
fn inc_some(mut xs: List) -> bool {
  let n = sum(&xs); let my = take_some(&mut xs);
  *my += 1; sum(&xs) == n + 1
}
```

"for any **`xs`**, **`inc_some(xs)`** **always returns** **`true`** *if it terminates*"

*reduction by our method*

$$TakeSome(\langle [x\,|\,xs'], xs_\circ\rangle, r) \;\Longleftarrow\; xs_\circ = [x_\circ\,|\,xs'_\circ] \;\wedge\; xs'_\circ = xs' \;\wedge\; r = \langle x, x_\circ\rangle$$

$$TakeSome(\langle [x\,|\,xs'], xs_\circ\rangle, r) \;\Longleftarrow\; xs_\circ = [x_\circ\,|\,xs'_\circ] \;\wedge\; x_\circ = x \;\wedge\; TakeSome(\langle xs', xs'_\circ\rangle, r)$$

$$TakeSome(\langle [\,], xs_\circ\rangle, r) \;\Longleftarrow\; TakeSome(\langle [\,], xs_\circ\rangle, r)$$

$$IncSome(xs, r) \;\Longleftarrow\; TakeSome(\langle xs, xs_\circ\rangle, \langle y, y_\circ\rangle) \;\wedge\; y_\circ = y + 1 \;\wedge\; r = (\mathsf{sum}(xs_\circ) == \mathsf{sum}(xs) + 1)$$

$$r = \mathsf{true} \;\Longleftarrow\; IncSome(xs, r)$$

It has a **simple solution**:

$$TakeSome(\langle xs, xs_\circ\rangle, \langle y, y_\circ\rangle) \;:\Longleftrightarrow\; y_\circ - y = \mathsf{sum}(xs_\circ) - \mathsf{sum}(xs)$$

$$IncSome(xs, r) \;:\Longleftrightarrow\; r = \mathsf{true}$$

# Going Beyond CHCs

- Our method can be extended into **reduction from a Rust program to a functional program**
  - By this view we can **apply various verification techniques to Rust** (e.g. model checking, Boogie, Coq)

```
int *take_max(int *ma, int *mb) {
  if (*ma >= *mb) return ma; else return mb;
}

bool inc_max(int a, int b) {
  { int *mc = take_max(&a, &b); // borrow a & b
    *mc += 1; } // end of borrow
  return (a != b);
}
```

⬇ *reduction to a functional program*

*permission release*

```
let take_max (a, a') (b, b') =
  if a >= b then (assume (b' = b); (a, a'))
            else (assume (a' = a); (b, b'))
```

*borrow*

```
let inc_max a b =
  let a' = rand () in let b' = rand () in
  let (c, c') = take_max (a, a') (b, b') in
  assume (c' = c + 1); a' <> b'
```

*permission release*

# Advanced Features

- **Closure**
  - **Permission release on the enclosed data** is the twist
  - **`FnMut`** can be modeled as a closure that **generates a newer version of itself** after it is called
- **`RefCell<T>`** etc.
  - We need to deal with **real sharing of data**
  - Simple remedy: pass around a global array for `RefCell` values
    - **At the very time** a **mutable reference** $\langle a, a_\circ \rangle$ is taken from a `RefCell`, the data at the array **is updated into** $a_\circ$

# Table of Contents

- Ownership and Borrow in Rust
- Our Method
- **Formalization and Correctness Proof**
- Experiments and Evaluation

# Formalization and Correctness Proof

- **Formalized the core of Rust**
  - Similar to $\boldsymbol{\lambda}_{\textbf{Rust}}$ of RustBelt [Jung+, 2018] but ours is simpler; **permission releases** and **lifetimes** are made explicit
- **Proved the correctness of the translation**
  - By techniques based on **bisimulations**
  - A concept strongly related to **prophecy variables** is used

# Formalization and Correctness Proof

$$\frac{S_{\Pi,f,L} = \mathsf{let}\ y = \mathsf{mutbor}_\alpha\ x;\ \mathsf{goto}\ L' \quad \boldsymbol{x}_\circ \text{ is fresh}}{[f,L]_\Theta\, \mathcal{F}+\{(x,\langle\hat{v}_*\rangle)\};\ \mathcal{S}\mid_{\mathbf{A}} \to_\Pi [f,L']_\Theta\, \mathcal{F}+\{(y,\langle\hat{v}_*,\boldsymbol{x}_\circ\rangle),(x,\langle\boldsymbol{x}_\circ\rangle)\};\ \mathcal{S}\mid_{\mathbf{A}}}$$

$$\frac{S_{\Pi,f,L} = \mathsf{let}\ y = \mathsf{mutbor}_\alpha\ x;\ \mathsf{goto}\ L' \quad \boldsymbol{x}_\circ \text{ is fresh}}{[f,L]_\Theta\, \mathcal{F}+\{(x,\langle\hat{v}_*,\boldsymbol{x}'_\circ\rangle)\};\ \mathcal{S}\mid_{\mathbf{A}} \to_\Pi [f,L']_\Theta\, \mathcal{F}+\{(y,\langle\hat{v}_*,\boldsymbol{x}_\circ\rangle),(x,\langle\boldsymbol{x}_\circ,\boldsymbol{x}'_\circ\rangle)\};\ \mathcal{S}\mid_{\mathbf{A}}}$$

$$\frac{S_{\Pi,f,L} = \mathsf{drop}\ x;\ \mathsf{goto}\ L' \quad \mathrm{Ty}_{\Pi,f,L}(x) = \check{P}\,T}{[f,L]_\Theta\, \mathcal{F}+\{(x,\hat{v})\};\ \mathcal{S}\mid_{\mathbf{A}} \to_\Pi [f,L']_\Theta\, \mathcal{F};\ \mathcal{S}\mid_{\mathbf{A}}}$$

$$\frac{S_{\Pi,f,L} = \mathsf{drop}\ x;\ \mathsf{goto}\ L' \quad \mathrm{Ty}_{\Pi,f,L}(x) = \mathsf{mut}_\alpha\,T}{[f,L]_\Theta\, \mathcal{F}+\{(x,\langle\hat{v}_*,\boldsymbol{x}_\circ\rangle)\};\ \mathcal{S}\mid_{\mathbf{A}} \to_\Pi \big([f,L']_\Theta\, \mathcal{F};\ \mathcal{S}\mid_{\mathbf{A}}\big)[\hat{v}_*/\boldsymbol{x}_\circ]}$$

$$\frac{S_{\Pi,f,L} = \mathsf{immut}\ x;\ \mathsf{goto}\ L'}{[f,L]_\Theta\, \mathcal{F}+\{(x,\langle\hat{v}_*,\boldsymbol{x}_\circ\rangle)\};\ \mathcal{S}\mid_{\mathbf{A}} \to_\Pi \big([f,L']_\Theta\, \mathcal{F}+\{(x,\langle\hat{v}_*\rangle)\};\ \mathcal{S}\mid_{\mathbf{A}}\big)[\hat{v}_*/\boldsymbol{x}_\circ]}$$

$$\frac{S_{\Pi,f,L} = \mathsf{swap}(*x,*y);\ \mathsf{goto}\ L' \quad \mathrm{Ty}_{\Pi,f,L}(y) = \mathsf{own}\,T}{\begin{array}{l}[f,L]_\Theta\, \mathcal{F}+\{(x,\langle\hat{v}_*,\boldsymbol{x}_\circ\rangle),(y,\langle\hat{w}_*\rangle)\};\ \mathcal{S}\mid_{\mathbf{A}}\\ \qquad \to_\Pi [f,L']_\Theta\, \mathcal{F}+\{(x,\langle\hat{w}_*,\boldsymbol{x}_\circ\rangle),(y,\langle\hat{v}_*\rangle)\};\ \mathcal{S}\mid_{\mathbf{A}}\end{array}}$$

$$\frac{S_{\Pi,f,L} = \mathsf{swap}(*x,*y);\ \mathsf{goto}\ L' \quad \mathrm{Ty}_{\Pi,f,L}(y) = \mathsf{mut}_\alpha\,T}{\begin{array}{l}[f,L]_\Theta\, \mathcal{F}+\{(x,\langle\hat{v}_*,\boldsymbol{x}_\circ\rangle),(y,\langle\hat{w}_*,\boldsymbol{y}_\circ\rangle)\};\ \mathcal{S}\mid_{\mathbf{A}}\\ \qquad \to_\Pi [f,L']_\Theta\, \mathcal{F}+\{(x,\langle\hat{w}_*,\boldsymbol{x}_\circ\rangle),(y,\langle\hat{v}_*,\boldsymbol{y}_\circ\rangle)\};\ \mathcal{S}\mid_{\mathbf{A}}\end{array}}$$

$$(\!|L\colon \mathsf{let}\ y = \mathsf{mutbor}_\alpha\ x;\ \mathsf{goto}\ L'|\!)_{\Pi,f} := \begin{cases} \left\{\begin{array}{l}\forall(\boldsymbol{\Delta}_{\Pi,f,L}+\{(x_\circ,(\!|T|\!))\}).\\ \ \check{\varphi}_{\Pi,f,L} \Longleftarrow \check{\varphi}_{\Pi,f,L'}[\langle *x, x_\circ\rangle/y, \langle x_\circ\rangle/x]\end{array}\right\} & (\mathrm{Ty}_{\Pi,f,L}(x) = \mathsf{own}\,T)\\ \left\{\begin{array}{l}\forall(\boldsymbol{\Delta}_{\Pi,f,L}+\{(x_\circ,(\!|T|\!))\}).\\ \ \check{\varphi}_{\Pi,f,L} \Longleftarrow \check{\varphi}_{\Pi,f,L'}[\langle *x, x_\circ\rangle/y, \langle x_\circ, \circ x\rangle/x]\end{array}\right\} & (\mathrm{Ty}_{\Pi,f,L}(x) = \mathsf{mut}_\alpha\,T)\end{cases}$$

$$(\!|L\colon \mathsf{drop}\ x;\ \mathsf{goto}\ L'|\!)_{\Pi,f} := \begin{cases} \{\forall(\boldsymbol{\Delta}_{\Pi,f,L}).\ \check{\varphi}_{\Pi,f,L} \Longleftarrow \check{\varphi}_{\Pi,f,L'}\} & (\mathrm{Ty}_{\Pi,f,L}(x) = \check{P}\,T)\\ \left\{\begin{array}{l}\forall(\boldsymbol{\Delta}_{\Pi,f,L}-\{(x,\mathsf{mut}\,(\!|T|\!))\}+\{(x_*,(\!|T|\!))\}).\\ \ \check{\varphi}_{\Pi,f,L}[\langle x_*, x_*\rangle/x] \Longleftarrow \check{\varphi}_{\Pi,f,L'}\end{array}\right\} & (\mathrm{Ty}_{\Pi,f,L}(x) = \mathsf{mut}_\alpha\,T)\end{cases}$$

$$(\!|L\colon \mathsf{immut}\ x;\ \mathsf{goto}\ L'|\!)_{\Pi,f} := \left\{\begin{array}{l}\forall(\boldsymbol{\Delta}_{\Pi,f,L}-\{x,\mathsf{mut}\,(\!|T|\!)\}+\{x_*,(\!|T|\!)\}).\\ \ \check{\varphi}_{\Pi,f,L}[\langle x_*, x_*\rangle/x] \Longleftarrow \check{\varphi}_{\Pi,f,L'}[\langle x_*\rangle/x]\end{array}\right\} \quad (\mathrm{Ty}_{\Pi,f,L}(x) = \mathsf{mut}_\alpha\,T)$$

$$(\!|L\colon \mathsf{swap}(*x,*y);\ \mathsf{goto}\ L'|\!)_{\Pi,f} := \begin{cases} \{\forall(\boldsymbol{\Delta}_{\Pi,f,L}).\ \check{\varphi}_{\Pi,f,L} \Longleftarrow \check{\varphi}_{\Pi,f,L'}[\langle *y, \circ x\rangle/x, \langle *x\rangle/y]\} & (\mathrm{Ty}_{\Pi,f,L}(y) = \mathsf{own}\,T)\\ \{\forall(\boldsymbol{\Delta}_{\Pi,f,L}).\ \check{\varphi}_{\Pi,f,L} \Longleftarrow \check{\varphi}_{\Pi,f,L'}[\langle *y, \circ x\rangle/x, \langle *x, \circ y\rangle/y]\} & (\mathrm{Ty}_{\Pi,f,L}(y) = \mathsf{mut}_\alpha\,T)\end{cases}$$

$$(\!|L\colon \mathsf{let}\ *y = x;\ \mathsf{goto}\ L'|\!)_{\Pi,f} := \{\forall(\boldsymbol{\Delta}_{\Pi,f,L}).\ \check{\varphi}_{\Pi,f,L} \Longleftarrow \check{\varphi}_{\Pi,f,L'}[\langle x\rangle/y]\}$$

$$(\!|L\colon \mathsf{let}\ y = *x;\ \mathsf{goto}\ L'|\!)_{\Pi,f} := \begin{cases} \{\forall(\boldsymbol{\Delta}_{\Pi,f,L}).\ \check{\varphi}_{\Pi,f,L} \Longleftarrow \check{\varphi}_{\Pi,f,L'}[*x/y]\} & (\mathrm{Ty}_{\Pi,f,L}(x) = \mathsf{own}\,P\,T)\\ \{\forall(\boldsymbol{\Delta}_{\Pi,f,L}).\ \check{\varphi}_{\Pi,f,L} \Longleftarrow \check{\varphi}_{\Pi,f,L'}[\langle **x\rangle/y]\} & (\mathrm{Ty}_{\Pi,f,L}(x) = \mathsf{immut}_\alpha\,P\,T)\\ \{\forall(\boldsymbol{\Delta}_{\Pi,f,L}).\ \check{\varphi}_{\Pi,f,L} \Longleftarrow \check{\varphi}_{\Pi,f,L'}[\langle **x, *\circ x\rangle/y]\} & (\mathrm{Ty}_{\Pi,f,L}(x) = \mathsf{mut}_\alpha\,\mathsf{own}\,T)\\ \left\{\begin{array}{l}\forall(\boldsymbol{\Delta}_{\Pi,f,L}-\{(x,\mathsf{mut\,box}\,(\!|T|\!))\}+\{(x_*,\mathsf{box}\,(\!|T|\!))\}).\\ \ \check{\varphi}_{\Pi,f,L}[\langle x_*, x_*\rangle/x] \Longleftarrow \check{\varphi}_{\Pi,f,L'}[x_*/y]\end{array}\right\} & (\mathrm{Ty}_{\Pi,f,L}(x) = \mathsf{mut}_\alpha\,\mathsf{immut}_\beta\,T)\\ \left\{\begin{array}{l}\forall(\boldsymbol{\Delta}_{\Pi,f,L}-\{(x,\mathsf{mut\,mut}\,(\!|T|\!))\}\\ \quad +\{(x_*,\mathsf{mut}\,(\!|T|\!)),(x_{*\circ},(\!|T|\!))\}).\\ \ \check{\varphi}_{\Pi,f,L}[\langle x_*, \langle x_{*\circ}, \circ x_*\rangle\rangle/x]\\ \qquad \Longleftarrow \check{\varphi}_{\Pi,f,L'}[\langle *x_*, x_{*\circ}\rangle/y]\end{array}\right\} & (\mathrm{Ty}_{\Pi,f,L}(x) = \mathsf{mut}_\alpha\,\mathsf{mut}_\beta\,T)\end{cases}$$

$$\frac{\mathrm{readout}^{\mathbf{a}}_{\mathbf{H},\hat{D}\circ\check{P}}(*a :: T \mid \hat{v};\ \hat{\mathcal{X}},\hat{\mathcal{M}})}{\mathrm{readout}^{\mathbf{a}}_{\mathbf{H},\hat{D}}(a :: \check{P}\,T \mid \langle\hat{v}\rangle;\ \hat{\mathcal{X}},\hat{\mathcal{M}})}$$

$$\hat{D}\circ\mathsf{own} := \hat{D} \quad \mathrm{hot}\circ\mathsf{immut}_\beta := \mathrm{cold}_\beta \quad \mathrm{cold}_\alpha\circ\mathsf{immut}_\beta := \mathrm{cold}_\alpha$$

$$\frac{\mathrm{readout}^{\mathbf{a}}_{\mathbf{H},\mathrm{hot}}(*a :: T \mid \hat{v};\ \hat{\mathcal{X}},\hat{\mathcal{M}})}{\mathrm{readout}^{\mathbf{a}}_{\mathbf{H},\mathrm{hot}}(a :: \mathsf{mut}_\beta\,T \mid \langle\hat{v},\boldsymbol{x}\rangle;\ \hat{\mathcal{X}}\oplus\{\!|\mathrm{give}_\beta(*a;\boldsymbol{x} :: T)|\!\},\ \hat{\mathcal{M}})}$$

$$\frac{\mathrm{readout}^{\mathbf{a}}_{\mathbf{H},\mathrm{cold}_\beta}(*a :: T \mid \hat{v};\ \hat{\mathcal{X}},\hat{\mathcal{M}})}{\mathrm{readout}^{\mathbf{a}}_{\mathbf{H},\mathrm{cold}_\beta}(a :: \mathsf{mut}_{\beta'}\,T \mid \langle\hat{v},\boldsymbol{x}\rangle;\ \hat{\mathcal{X}},\hat{\mathcal{M}})}$$

$$\mathrm{readout}^{\dagger\alpha}_{\mathbf{H},\hat{D}}(*a :: T \mid x;\ \{\!|\mathrm{take}^{\dagger\alpha}(*a;\boldsymbol{x} :: T)|\!\},\varnothing)$$

$$\frac{\mathbf{H}(a) = a' \quad \mathrm{readout}^{\mathbf{a}}_{\mathbf{H},\hat{D}}(a' :: P\,T \mid \hat{v};\ \hat{\mathcal{X}},\hat{\mathcal{M}})}{\mathrm{readout}^{\mathbf{a}}_{\mathbf{H},\hat{D}}(*a :: P\,T \mid \hat{v};\ \hat{\mathcal{X}},\hat{\mathcal{M}}\oplus\{\!|\hat{D}^{\mathbf{a}}(a)|\!\})}$$

$$\hat{D}^{\mathbf{a}}(a) := \begin{cases}\mathrm{hot}^{\mathbf{a}}(a) & (\hat{D} = \mathrm{hot})\\ \mathrm{cold}_\beta(a) & (\hat{D} = \mathrm{cold}_\beta)\end{cases}$$

$$\frac{\mathrm{readout}^{\mathbf{a}}_{\mathbf{H},\hat{D}}(*a :: T[\mu X.T/X] \mid \hat{v};\ \hat{\mathcal{X}},\hat{\mathcal{M}})}{\mathrm{readout}^{\mathbf{a}}_{\mathbf{H},\hat{D}}(*a :: \mu X.T \mid \hat{v};\ \hat{\mathcal{X}},\hat{\mathcal{M}})}$$

$$\frac{\mathbf{H}(a) = n}{\mathrm{readout}^{\mathbf{a}}_{\mathbf{H},\hat{D}}(*a :: \mathsf{int} \mid n;\ \varnothing,\{\!|\hat{D}^{\mathbf{a}}(a)|\!\})} \qquad \mathrm{readout}^{\mathbf{a}}_{\mathbf{H},\hat{D}}(*a :: \mathsf{unit} \mid ();\ \varnothing,\varnothing)$$

$$\frac{\begin{array}{c}\mathbf{H}(a) = i \in [2] \quad \mathrm{readout}^{\mathbf{a}}_{\mathbf{H},\hat{D}}(*(a+1) :: T_i \mid \hat{v};\ \hat{\mathcal{X}},\hat{\mathcal{M}}) \quad n_0 = (\#T_{1-i}-\#T_i)_{\geq 0}\\ \text{for any } k \in [n_0],\ \mathbf{H}(a+1+\#T_i+k) = 0 \quad \hat{\mathcal{M}}_0 = \{\!|\hat{D}^{\mathbf{a}}(a+1+\#T_i+k) \mid k \in [n_0]|\!\}\end{array}}{\mathrm{readout}^{\mathbf{a}}_{\mathbf{H},\hat{D}}(*a :: T_0+T_1 \mid \mathsf{inj}_i\,\hat{v};\ \hat{\mathcal{X}},\hat{\mathcal{M}}\oplus\{\!|\hat{D}^{\mathbf{a}}(a)|\!\}\oplus\hat{\mathcal{M}}_0)}$$

$$\frac{\mathrm{readout}^{\mathbf{a}}_{\mathbf{H},\hat{D}}(*a :: T_0 \mid \hat{v}_0;\ \hat{\mathcal{X}}_0,\hat{\mathcal{M}}_0) \quad \mathrm{readout}^{\mathbf{a}}_{\mathbf{H},\hat{D}}(*(a+\#T_0) :: T_1 \mid \hat{v}_1;\ \hat{\mathcal{X}}_1,\hat{\mathcal{M}}_1)}{\mathrm{readout}^{\mathbf{a}}_{\mathbf{H},\hat{D}}(*a :: T_0\times T_1 \mid (\hat{v}_0,\hat{v}_1);\ \hat{\mathcal{X}}_0\oplus\hat{\mathcal{X}}_1,\ \hat{\mathcal{M}}_0\oplus\hat{\mathcal{M}}_1)}$$

$$\mathrm{summary}^{\dagger\alpha}_{D}(\boldsymbol{x} :: T \mid \{\mathrm{take}^{\dagger\alpha}(\boldsymbol{x} :: T)\}) \qquad \frac{\mathrm{summary}^{\mathbf{a}}_{D\cdot\check{P}}(\hat{v} :: T \mid \mathcal{X})}{\mathrm{summary}^{\mathbf{a}}_{D}(\langle\hat{v}\rangle :: \check{P}\,T \mid \mathcal{X})}$$

$$D\cdot\mathsf{own} := D \quad D\cdot\mathsf{immut}_\beta := \mathrm{cold}$$

$$\frac{\mathrm{summary}^{\mathbf{a}}_{\mathrm{hot}}(\hat{v} :: T \mid \mathcal{X})}{\mathrm{summary}^{\mathbf{a}}_{\mathrm{hot}}(\langle\hat{v},\boldsymbol{x}\rangle :: \mathsf{mut}_\beta\,T \mid \mathcal{X}\oplus\{\mathrm{give}_\beta(\boldsymbol{x} :: T)\})} \qquad \frac{\mathrm{summary}^{\mathbf{a}}_{\mathrm{cold}}(\hat{v} :: T \mid \mathcal{X})}{\mathrm{summary}^{\mathbf{a}}_{\mathrm{cold}}(\langle\hat{v},\boldsymbol{x}\rangle :: \mathsf{mut}_\beta\,T \mid \mathcal{X})}$$

$$\frac{\mathrm{summary}^{\mathbf{a}}_{D}(\hat{v} :: T[\mu X.T/X] \mid \mathcal{X})}{\mathrm{summary}^{\mathbf{a}}_{D}(\hat{v} :: \mu X.T/X \mid \mathcal{X})} \qquad \mathrm{summary}^{\mathbf{a}}_{D}(\mathit{const} :: T \mid \varnothing)$$

$$\frac{\mathrm{summary}^{\mathbf{a}}_{D}(\hat{v} :: T_i \mid \mathcal{X})}{\mathrm{summary}^{\mathbf{a}}_{D}(\mathsf{inj}_i\,\hat{v} :: T_0+T_1 \mid \mathcal{X})} \qquad \frac{\mathrm{summary}^{\mathbf{a}}_{D}(\hat{v}_0 :: T_0 \mid \mathcal{X}_0) \quad \mathrm{summary}^{\mathbf{a}}_{D}(\hat{v}_1 :: T_1 \mid \mathcal{X}_1)}{\mathrm{summary}^{\mathbf{a}}_{D}((\hat{v}_0,\hat{v}_1) :: T_0\times T_1 \mid \mathcal{X}_0\oplus\mathcal{X}_1)}$$

$$\frac{\mathcal{X}(x) = \{\!|\mathrm{give}_\alpha(\boldsymbol{x} :: T),\ \mathrm{take}^{\dagger\beta}(\boldsymbol{x} :: T')|\!\} \quad T \sim_{\mathbf{A}} T' \quad \alpha \leq_{\mathbf{A}} \beta}{\mathrm{safe}_{\mathbf{A}}(\boldsymbol{x},\mathcal{X})} \qquad \frac{\mathcal{X}(\boldsymbol{x}) = \varnothing}{\mathrm{safe}_{\mathbf{A}}(\boldsymbol{x},\mathcal{X})}$$

$\mathcal{X}(\boldsymbol{x})$: the multiset of the items of form ‘$\mathrm{give}_\gamma(\boldsymbol{x} :: U)$’/‘$\mathrm{take}^\gamma(\boldsymbol{x} :: U)$’ in $\mathcal{X}$

# Table of Contents

- Ownership and Borrow in Rust
- Our Method
- Formalization and Correctness Proof
- **Experiments and Evaluation**

# Implementation and Experiments

- Implemented a **prototype verifier** **RustHorn**
  - Uses **MIR** (mid-level intermediate representation) of the Rust compiler
  - Generates CHCs with a **simple algorithm** based on **our method**

- Conducted **experiments** on benchmarks
  - Tested **RustHorn** and **SeaHorn** [Gurfinkel+, 2015]
    - SeaHorn is a standard CHC-based verifier for C programs
  - Made **benchmark** verification problems in **Rust** & **C**
    - Took ones from SeaHorn and also wrote ones featuring pointers
  - Used **Z3/Spacer** [Komuravelli+, 2014] and **HoIce** [Champion+, 2018] as a backend CHC solver

check out `github.com/hopv/rust-horn`

# Experimental Results

| *Group* | *Instance* | *Property* | RustHorn *w/Spacer* | RustHorn *w/HoIce* | SeaHorn *w/Spacer* *as is* | SeaHorn *w/Spacer* *modified* |
|---|---|---|---|---|---|---|
| `simple` | `01` | safe | <0.1 | <0.1 | <0.1 | |
| | `04-recursive` | safe | 0.5 | timeout | 0.8 | |
| | `05-recursive` | unsafe | <0.1 | <0.1 | <0.1 | |
| | `06-loop` | safe | timeout | 0.1 | timeout | |
| | `hhk2008` | safe | timeout | 40.5 | <0.1 | |
| | `unique-scalar` | unsafe | <0.1 | <0.1 | <0.1 | |
| `bmc` | `1` | safe | 0.2 | <0.1 | <0.1 | |
| | | unsafe | 0.2 | <0.1 | <0.1 | |
| | `2` | safe | timeout | 0.1 | <0.1 | |
| | | unsafe | <0.1 | <0.1 | <0.1 | |
| | `3` | safe | <0.1 | <0.1 | <0.1 | |
| | | unsafe | <0.1 | <0.1 | <0.1 | |
| | `diamond-1` | safe | 0.1 | <0.1 | <0.1 | |
| | | unsafe | <0.1 | <0.1 | <0.1 | |
| | `diamond-2` | safe | 0.2 | <0.1 | <0.1 | |
| | | unsafe | <0.1 | <0.1 | <0.1 | |
| `inc-max` | `base` | safe | <0.1 | <0.1 | false alarm | <0.1 |
| | | unsafe | <0.1 | <0.1 | <0.1 | <0.1 |
| | `base/3` | safe | <0.1 | <0.1 | false alarm | |
| | | unsafe | 0.1 | <0.1 | <0.1 | |
| | `repeat` | safe | 0.1 | timeout | false alarm | 0.1 |
| | | unsafe | <0.1 | 0.4 | <0.1 | <0.1 |
| | `repeat/3` | safe | 0.2 | timeout | <0.1 | |
| | | unsafe | <0.1 | 1.3 | <0.1 | |
| `swap-dec` | `base` | safe | <0.1 | <0.1 | false alarm | <0.1 |
| | | unsafe | 0.1 | timeout | <0.1 | <0.1 |
| | `base/3` | safe | 0.2 | timeout | false alarm | <0.1 |
| | | unsafe | 0.4 | 0.9 | <0.1 | 0.1 |
| | `exact` | safe | 0.1 | 0.5 | false alarm | timeout |
| | | unsafe | <0.1 | 26.0 | <0.1 | <0.1 |
| | `exact/3` | safe | timeout | timeout | false alarm | false alarm |
| | | unsafe | <0.1 | 0.4 | <0.1 | <0.1 |
| `just-rec` | `base` | safe | <0.1 | <0.1 | <0.1 | |
| | | unsafe | <0.1 | 0.1 | <0.1 | |
| `linger-dec` | `base` | safe | <0.1 | <0.1 | false alarm | |
| | | unsafe | <0.1 | 0.1 | <0.1 | |
| | `base/3` | safe | <0.1 | <0.1 | false alarm | |
| | | unsafe | <0.1 | 7.0 | <0.1 | |
| | `exact` | safe | <0.1 | <0.1 | false alarm | |
| | | unsafe | <0.1 | 0.2 | <0.1 | |
| | `exact/3` | safe | <0.1 | <0.1 | false alarm | |
| | | unsafe | <0.1 | 0.6 | <0.1 | |
| `lists` | `append` | safe | tool error | <0.1 | false alarm | |
| | | unsafe | tool error | 0.2 | 0.1 | |
| | `inc-all` | safe | tool error | <0.1 | false alarm | |
| | | unsafe | tool error | 0.3 | <0.1 | |
| | `inc-some` | safe | tool error | <0.1 | false alarm | |
| | | unsafe | tool error | 0.3 | 0.1 | |
| | `inc-some/2` | safe | tool error | timeout | false alarm | |
| | | unsafe | tool error | 0.3 | 0.4 | |
| `trees` | `append-t` | safe | tool error | <0.1 | timeout | |
| | | unsafe | tool error | 0.3 | 0.1 | |
| | `inc-all-t` | safe | tool error | timeout | timeout | |
| | | unsafe | tool error | 0.1 | <0.1 | |
| | `inc-some-t` | safe | tool error | timeout | timeout | |
| | | unsafe | tool error | 0.3 | 0.1 | |
| | `inc-some/2-t` | safe | tool error | timeout | false alarm | |
| | | unsafe | tool error | 0.4 | 0.1 | |

**Table 1.** Benchmarks and experimental results on RustHorn and SeaHorn, with Spacer/Z3 and HoIce. "timeout" denotes timeout of 180 seconds; "false alarm" means reporting 'unsafe' for a safe program; "tool error" is a tool error of Spacer, which currently does not deal with recursive types well.

- RustHorn+HoIce **handles recursive data types quite well**
- The output of RustHorn is **very reliable**
- RustHorn is well **comparable** to SeaHorn in performance

# Conclusion

- **Novel translation** from **Rust** programs to CHCs
  - Models a **mutable reference** as a pair of the current value and the **future value**, reminiscent of a **prophecy variable**
  - **Applicable to a wide class** of Rust programs
  - We formalized and proved its correctness and confirmed the effectiveness by experiments

We believe that this work establishes the foundation of **verification leveraging borrow-based ownership**.

full paper: `arxiv.org/abs/2002.09002`