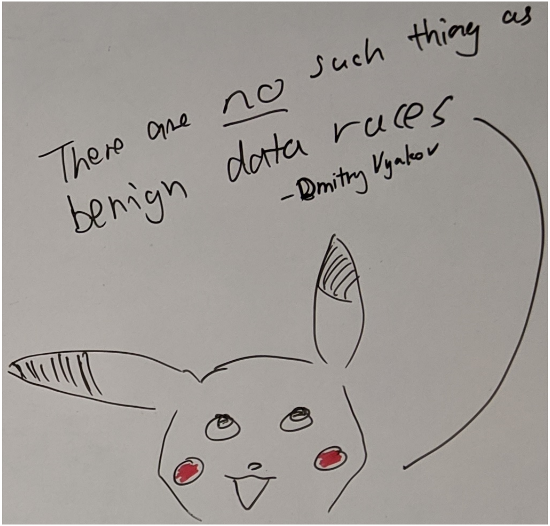
Imperial College London

19 December 2019

Static Detection of Communication Errors and Data Races in Go Programs

Julia Gabet[1] Nobuko Yoshida[1] Nicholas Ng[2]

[1]Imperial College London, [2]Monzo



Drawing by Nicholas Ng

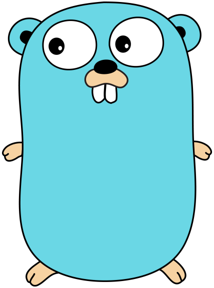
Imperial College London

Motivating example

```
func main() {
  var x int
  ch := make(chan int, 1)
  go f(ch, &x)
  ch <- Lock
  x += 10
  <-ch
  ch <- Lock
  fmt.Println("x is", x)
  <-ch
}

func f(ch chan int, ptr *int) {
  ch <- Lock
  *ptr += 20
  <-ch
}
```

Figure 1: Go programs: safe (size 1)



Imperial College London

Motivating example

```
func main() {
  var x int
  ch := make(chan int, 1)            ⇝ 2
  go f(ch, &x)
  ch <- Lock
  x += 10                            ⇝ race
  <-ch
  ch <- Lock
  fmt.Println("x is", x)
  <-ch
}

func f(ch chan int, ptr *int) {
  ch <- Lock
  *ptr += 20                         ⇝ race
  <-ch
}
```

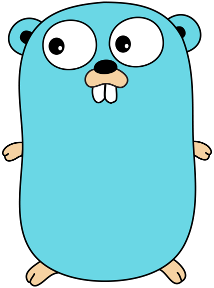


Figure 1: Go programs: safe (size 1) ⇝ race (size 2)

Motivating example: with Mutex

```go
func main() {
  var x int
  m := new(sync.Mutex)
  go f(m, &x)
  m.Lock()
  x += 10
  m.Unlock()
  m.Lock()
  fmt.Println("x is", x)
  m.Unlock()
}

func f(m *sync.Mutex, ptr *int) {
  m.Lock()
  *ptr += 20
  m.Unlock()
}
```

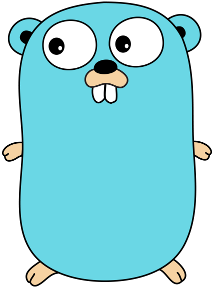


Figure 2: Go program: with a mutual exclusion lock (safe)

Imperial College London

Outline of work

1

Outline of work

1. Abstraction of Go programs with a π-calculus inspired language: MiGo^{+}, a rework of MiGo

---

1

Imperial College London

Outline of work

1. Abstraction of Go programs with a π-calculus inspired language: MiGo$^+$, a rework of MiGo
2. Define desired properties of the MiGo$^+$ processes

1

Imperial College London

Outline of work

1. Abstraction of Go programs with a π-calculus inspired language: MiGo$^+$, a rework of MiGo
2. Define desired properties of the MiGo$^+$ processes
3. Add a type system for the MiGo$^+$ processes

1

Imperial College London

Outline of work

1. Abstraction of Go programs with a π-calculus inspired language: MiGo$^+$, a rework of MiGo
2. Define desired properties of the MiGo$^+$ processes
3. Add a type system for the MiGo$^+$ processes
4. Abstract property verification of the processes to the types

---

1

Imperial College London

Outline of work

1. Abstraction of Go programs with a π-calculus inspired language: MiGo$^+$, a rework of MiGo
2. Define desired properties of the MiGo$^+$ processes
3. Add a type system for the MiGo$^+$ processes
4. Abstract property verification of the processes to the types
5. Model check the types for the desired properties

1

Outline of work

1. Abstraction of Go programs with a π-calculus inspired language: MiGo^{+}, a rework of MiGo
2. Define desired properties of the MiGo^{+} processes
3. Add a type system for the MiGo^{+} processes
4. Abstract property verification of the processes to the types
5. Model check the types for the desired properties
6. Implementation: Extending the Godel Checker[1]

[1] Lange, Ng, Toninho, Yoshida: Fencing off Go: Liveness and Safety for Channel-based Programming (POPL 2017), A Static Verification Framework for Message Passing in Go using Behavioural Types(ICSE 2018)

Imperial College London

MiGo$^+$ Process of our unsafe example

$$\mathbf{P}_{\mathsf{race}} = \left\{ \qquad \right\}$$

in

Imperial College London

MiGo$^+$ Process of our unsafe example

$$\mathbf{P}_{\text{race}} = \left\{ \qquad \right\}$$

$\text{in newvar}(x{:}\text{int});$

Imperial College London

MiGo$^+$ Process of our unsafe example

$$\mathbf{P}_{\mathsf{race}} = \left\{ \qquad \right\}$$

$$\mathsf{in}\;\; \mathsf{newvar}(x{:}\mathsf{int}); \mathsf{newchan}(y{:}\mathsf{int}, 2);$$

MiGo^{+} Process of our unsafe example

$$\mathbf{P}_{\text{race}} = \left\{ \phantom{\begin{array}{c} x \\ x \\ x \end{array}} \right\}$$

$$\text{in}\ \ \text{newvar}(x{:}\text{int}); \text{newchan}(y{:}\text{int}, 2); \big(P\langle y, x\rangle \mid Q\langle y, x\rangle\big)$$

MiGo$^+$ Process of our unsafe example

$$\mathbf{P}_{\text{race}} = \left\{ \begin{array}{ll} P(c, z) = c!\langle \text{Lock} \rangle; & c?(u); \\ & \end{array} \right\}$$

$$\text{in newvar}(x{:}\text{int}); \text{newchan}(y{:}\text{int}, 2); \big(P\langle y, x\rangle \mid Q\langle y, x\rangle\big)$$

Imperial College London

MiGo$^+$ Process of our unsafe example

$$\mathbf{P}_{\mathsf{race}} = \left\{ \begin{array}{l} P(c,z) = \; c!\langle \mathsf{Lock}\rangle; t_1 = \mathsf{load}(z); z := t_1 + 10; c?(u); \\ \\ \end{array} \right\}$$

$$\mathsf{in}\;\; \mathsf{newvar}(x{:}\mathsf{int}); \mathsf{newchan}(y{:}\mathsf{int}, 2); \big(P\langle y, x\rangle \mid Q\langle y, x\rangle\big)$$

Imperial College London

MiGo$^+$ Process of our unsafe example

$$\mathbf{P}_{\text{race}} = \left\{ \begin{array}{ll} P(c,z) = & c!\langle \text{Lock} \rangle; t_1 = \text{load}(z); z := t_1 + 10; c?(u); \\ & c!\langle \text{Lock} \rangle; t_2 = \text{load}(z); \tau; c?(u'); \mathbf{0}, \\ & \end{array} \right\}$$

$$\text{in newvar}(x{:}\text{int}); \text{newchan}(y{:}\text{int}, 2); \big(P\langle y, x\rangle \mid Q\langle y, x\rangle\big)$$

MiGo$^+$ Process of our unsafe example

$$\mathbf{P}_{\mathsf{race}} = \left\{ \begin{array}{l} P(c,z) = \; c!\langle \mathsf{Lock}\rangle; t_1 = \mathsf{load}(z); z := t_1 + 10; c?(u); \\ \qquad\qquad\;\; c!\langle \mathsf{Lock}\rangle; t_2 = \mathsf{load}(z); \tau; c?(u'); \mathbf{0}, \\ Q(c,z) = c!\langle \mathsf{Lock}\rangle; t_0 = \mathsf{load}(z); z := t_0 + 20; c?(u''); \mathbf{0} \end{array} \right\}$$

$$\mathsf{in}\;\; \mathsf{newvar}(x{:}\mathsf{int}); \mathsf{newchan}(y{:}\mathsf{int}, 2); \big(P\langle y, x\rangle \mid Q\langle y, x\rangle\big)$$

Imperial College London

Reduction of our MiGo$^+$ example

$$\mathbf{P}_{\text{race}} \stackrel{\text{def}}{=} \begin{array}{l}\mathsf{newvar}(x{:}\mathsf{int});\\ \mathsf{newchan}(y{:}\mathsf{int}, 2);\end{array} \left(\begin{array}{l} \;\; y!\langle \mathsf{Lock}\rangle; t_1 = \mathsf{load}(x); x := t_1 + 10;\\ \qquad\qquad \ldots \\ \mid\; y!\langle \mathsf{Lock}\rangle; t_0 = \mathsf{load}(x); x := t_0 + 20;\\ \qquad\qquad \ldots \end{array}\right)$$

Reduction of our MiGo$^+$ example

$$
\mathbf{P}_{\text{race}} \stackrel{\text{def}}{=} \begin{array}{l}\mathsf{newvar}(x{:}\mathsf{int});\\ \mathsf{newchan}(y{:}\mathsf{int}, 2);\end{array} \left(\begin{array}{l} \quad y!\langle \mathsf{Lock}\rangle; t_1 = \mathsf{load}(x); x := t_1 + 10; \\ \qquad\quad \ldots \\ \mid\ y!\langle \mathsf{Lock}\rangle; t_0 = \mathsf{load}(x); x := t_0 + 20; \\ \qquad\quad \ldots \end{array} \right)
$$

$$
\mathbf{P}_{\text{race}} \to^2 \quad (\nu x c) \left(\begin{array}{l} \quad c!\langle \mathsf{Lock}\rangle; t_1 = \mathsf{load}(x); x := t_1 + 10; \\ \qquad\quad \ldots \\ \mid\ c!\langle \mathsf{Lock}\rangle; t_0 = \mathsf{load}(x); x := t_0 + 20; \\ \qquad\quad \ldots \\ \mid\ [x := 0] \ \mid\ c\langle \mathsf{int}, 2\rangle{::}\emptyset \end{array} \right)
$$

Imperial College London

Reduction of our MiGo$^+$ example

$$\mathbf{P}_{\text{race}} \stackrel{\text{def}}{=} \begin{array}{l}\mathsf{newvar}(x{:}\mathsf{int});\\ \mathsf{newchan}(y{:}\mathsf{int}, 2);\end{array} \left(\begin{array}{rl} & y!\langle \mathsf{Lock}\rangle; t_1 = \mathsf{load}(x); x := t_1 + 10;\\ & \quad \ldots \\ \mid & y!\langle \mathsf{Lock}\rangle; t_0 = \mathsf{load}(x); x := t_0 + 20;\\ & \quad \ldots \end{array}\right)$$

$$\mathbf{P}_{\text{race}} \rightarrow^4 \quad (\nu x c)\left(\begin{array}{rl} & t_1 = \mathsf{load}(x); x := t_1 + 10;\\ & \quad \ldots \\ \mid & t_0 = \mathsf{load}(x); x := t_0 + 20;\\ & \quad \ldots \\ \mid & [x := 0] \;\mid\; c\langle \mathsf{int}, 2\rangle{::}\mathsf{Lock} \cdot \mathsf{Lock} \end{array}\right)$$

Reduction of our MiGo$^+$ example

$$\mathbf{P}_{\mathsf{race}} \stackrel{\text{def}}{=} \begin{array}{l}\mathsf{newvar}(x{:}\mathsf{int});\\ \mathsf{newchan}(y{:}\mathsf{int}, 2);\end{array} \left(\begin{array}{l} \ \ y!\langle \mathsf{Lock}\rangle; t_1 = \mathsf{load}(x); x := t_1 + 10;\\ \qquad \ldots \\ \mid y!\langle \mathsf{Lock}\rangle; t_0 = \mathsf{load}(x); x := t_0 + 20;\\ \qquad \ldots \end{array}\right)$$

$$\mathbf{P}_{\mathsf{race}} \to^6 \quad (\nu xc)\left(\begin{array}{lr} & \boxed{x := 0 + 10};\\ \qquad \ldots & \\ \mid & \boxed{x := 0 + 20};\\ \qquad \ldots & \\ \mid [x := 0] \ \mid \ c\langle \mathsf{int}, 2\rangle{::}\mathsf{Lock} \cdot \mathsf{Lock} & \end{array}\right)$$

The happens-before relation: $P \triangleright o_1 \mapsto o_2$

A first example:

$$P \quad = \quad c!\langle \mathsf{Lock} \rangle; t_1 = \mathsf{load}(x); P'$$

Imperial College London

The happens-before relation: $P \triangleright o_1 \mapsto o_2$

A first example:

$$\begin{aligned} P &= c!\langle \mathsf{Lock} \rangle; t_1 = \mathsf{load}(x); P' \\ &\xrightarrow{\overline{c}, \mathsf{Lock}} t_1 = \mathsf{load}(x); P' \end{aligned}$$

The happens-before relation: $P \triangleright o_1 \mapsto o_2$

A first example:

$$\begin{array}{rcl} P & = & c!\langle \mathsf{Lock} \rangle; t_1 = \mathsf{load}(x); P' \downarrow_{\overline{c}} \\ & \xrightarrow{\overline{c}, \mathsf{Lock}} & t_1 = \mathsf{load}(x); P' \end{array}$$

The happens-before relation: $P \triangleright o_1 \mapsto o_2$

A first example:

$$
\begin{aligned}
P \quad &= \quad c!\langle \mathsf{Lock}\rangle; t_1 = \mathsf{load}(x); P' \downarrow_{\overline{c}} \\
&\xrightarrow{\overline{c},\mathsf{Lock}} t_1 = \mathsf{load}(x); P' \\
&\xrightarrow{\mathsf{r}\langle x\rangle,0} P' \left\{{}^{0}/_{t_1}\right\}
\end{aligned}
$$

Imperial College London

The happens-before relation: $P \triangleright o_1 \mapsto o_2$

A first example:

$$
\begin{aligned}
P \quad &= \quad c!\langle \mathsf{Lock}\rangle; t_1 = \mathsf{load}(x); P' \downarrow_{\overline{c}} \\
&\xrightarrow{\overline{c},\mathsf{Lock}} t_1 = \mathsf{load}(x); P' \downarrow_{\mathsf{r}\langle x\rangle} \\
&\xrightarrow{\mathsf{r}\langle x\rangle,0} P' \left\{{}^{0}/_{t_1}\right\}
\end{aligned}
$$

Imperial College London

The happens-before relation: $P \triangleright o_1 \mapsto o_2$

A first example:

$$\begin{aligned} P \quad &= \quad c!\langle \mathsf{Lock} \rangle; t_1 = \mathsf{load}(x); P' \downarrow_{\overline{c}} \\ &\xrightarrow{\overline{c},\mathsf{Lock}} t_1 = \mathsf{load}(x); P' \downarrow_{\mathsf{r}\langle x \rangle} \\ &\xrightarrow{\mathsf{r}\langle x \rangle, 0} P' \{^0/_{t_1}\} \end{aligned}$$

$$\boxed{P \triangleright \overline{c} \mapsto \mathsf{r}\langle x \rangle}$$

Imperial College London

The happens-before relation: $P \triangleright o_1 \mapsto o_2$

A first example:

$$
\begin{aligned}
P \quad &= \quad c!\langle \mathsf{Lock}\rangle;\, t_1 = \mathsf{load}(x);\, P' \downarrow_{\overline{c}} \\
&\xrightarrow{\overline{c},\mathsf{Lock}} t_1 = \mathsf{load}(x);\, P' \downarrow_{\mathsf{r}\langle x\rangle} \\
&\xrightarrow{\mathsf{r}\langle x\rangle,0} P' \left\{{}^{0}/_{t_1}\right\}
\end{aligned}
$$

$$\boxed{P \triangleright \overline{c} \mapsto \mathsf{r}\langle x\rangle}$$

An other example:

$Q = x := 10;\, Q_1 \mid x := 20;\, Q_2$
$Q \downarrow_{(\mathsf{w}\langle x\rangle,1.*)}$
$Q \downarrow_{(\mathsf{w}\langle x\rangle,2.*)}$

The happens-before relation: $P \triangleright o_1 \mapsto o_2$

A first example:

$$\begin{aligned} P &= c!\langle \mathsf{Lock}\rangle; t_1 = \mathsf{load}(x); P' \downarrow_{\overline{c}} \\ &\xrightarrow{\overline{c},\mathsf{Lock}} t_1 = \mathsf{load}(x); P' \downarrow_{\mathsf{r}\langle x\rangle} \\ &\xrightarrow{\mathsf{r}\langle x\rangle,0} P' \left\{{}^{0}/_{t_1}\right\} \end{aligned}$$

$$\boxed{P \triangleright \overline{c} \mapsto \mathsf{r}\langle x\rangle}$$

An other example:

$Q = x := 10; Q_1 \mid x := 20; Q_2$
$Q \downarrow_{(\mathsf{w}\langle x\rangle,1.*)}$
$Q \downarrow_{(\mathsf{w}\langle x\rangle,2.*)}$

$$\boxed{\begin{array}{l} \neg(P \triangleright (\mathsf{w}\langle x\rangle, 1.*) \mapsto (\mathsf{w}\langle x\rangle, 2.*)) \\ \neg(P \triangleright (\mathsf{w}\langle x\rangle, 2.*) \mapsto (\mathsf{w}\langle x\rangle, 1.*)) \end{array}}$$

Defining Data Races

$$
\begin{tikzcd}
 & & & Q_1 \downarrow_{(\mathsf{w}\langle x\rangle,\iota)} \\
\text{(1) if } \mathbf{P} \arrow[r, "*"] & (\nu\tilde{u})P \text{ and } & P \arrow[ur, "*"] \arrow[dr, "*"'] & & \text{such that} \\
 & & & Q_2 \downarrow_{(\mathsf{w}\langle x\rangle,\iota')}
\end{tikzcd}
$$

Defining Data Races

$$
\text{(1) if } \mathbf{P} \longrightarrow^* (\nu \tilde{u})P \quad \text{and} \quad
\begin{tikzcd}[column sep=large]
 & Q_1 \downarrow_{\color{red}(\mathsf{w}\langle x\rangle,\iota)} \arrow[d, mapsto, "\times" description, color=magenta] \\
P \arrow[ur, "*"] \arrow[dr, "*"'] & \\
 & Q_2 \downarrow_{\color{blue}(\mathsf{w}\langle x\rangle,\iota')}
\end{tikzcd}
\quad \text{such that}
$$

$$\neg(P \triangleright (\mathsf{w}\langle x\rangle, \iota) \mapsto (\mathsf{w}\langle x\rangle, \iota')))$$

Defining Data Races

(1) if $\mathbf{P} \longrightarrow^* (\nu\tilde{u})P$ and

$$\begin{tikzcd}
 & Q_1 \downarrow_{(\mathsf{w}\langle x\rangle,\iota)} \arrow[d, shift right, "\nmapsto"'] \\
P \arrow[ur, "*"] \arrow[dr, "*"'] & \\
 & Q_2 \downarrow_{(\mathsf{w}\langle x\rangle,\iota')} \arrow[u, shift right, "\nmapsto"']
\end{tikzcd}$$

such that

$\neg(P \triangleright (\mathsf{w}\langle x\rangle,\iota) \mapsto (\mathsf{w}\langle x\rangle,\iota'))$ and $\neg(P \triangleright (\mathsf{w}\langle x\rangle,\iota') \mapsto (\mathsf{w}\langle x\rangle,\iota))$

Defining Data Races

(1) if $\mathbf{P} \longrightarrow^* (\nu\tilde{u})P$ and

$$\begin{tikzcd} & Q_1 \downarrow_{(\mathsf{w}\langle x\rangle,\iota)} \arrow[dd, mapsto, shift right, "\times" marking] \\ P \arrow[ur, "*"] \arrow[dr, "*"'] & \\ & Q_2 \downarrow_{(\mathsf{w}\langle x\rangle,\iota')} \arrow[uu, mapsto, shift right, "\times" marking] \end{tikzcd}$$

such that

$\neg(P \triangleright (\mathsf{w}\langle x\rangle, \iota) \mapsto (\mathsf{w}\langle x\rangle, \iota')))$ and $\neg(P \triangleright (\mathsf{w}\langle x\rangle, \iota') \mapsto (\mathsf{w}\langle x\rangle, \iota)))$
then $\mathbf{P}$ has a data race

Defining Data Races

(1) if $\mathbf{P} \longrightarrow^* (\nu \tilde{u})P$ and

$$\begin{tikzcd}[row sep=small] & Q_1 \downarrow_{(\mathsf{w}\langle x\rangle,\iota)} \\ P \arrow[ur, "*"] \arrow[dr, "*"'] & \\ & Q_2 \downarrow_{(\mathsf{w}\langle x\rangle,\iota')} \end{tikzcd}$$

such that

$\neg(P \triangleright (\mathsf{w}\langle x\rangle, \iota) \mapsto (\mathsf{w}\langle x\rangle, \iota')) $ and $\neg(P \triangleright (\mathsf{w}\langle x\rangle, \iota') \mapsto (\mathsf{w}\langle x\rangle, \iota))$
then $\mathbf{P}$ has a data race

(2) if $\mathbf{P} \longrightarrow^* (\nu \tilde{u})P'$ and $\begin{cases} P' \downarrow_{(\mathsf{w}\langle x\rangle,\iota)} \\ \text{and} \\ P' \downarrow_{(\mathsf{w}\langle x\rangle,\iota')} \end{cases}$ then $\mathbf{P}$ has a data race

Theorem

Charactisations (1) and (2) of data races are equivalent.

Liveness and Safety: the Example of Mutex

Definition (Mutex Safety)

If $\mathbf{P} \rightarrow^* (\nu\tilde{u})P$ and $P \downarrow_{\mathsf{ul}\langle m \rangle}$, then $P \equiv P' \mid \lceil m \rceil^{\star}$

Definition (Mutex Liveness)

If $\mathbf{P} \rightarrow^* (\nu\tilde{u})P$ and $P \downarrow_{\mathsf{l}\langle m \rangle}$, then $\exists P \rightarrow^* P' \xrightarrow{\tau_m}$

Liveness and Safety: the Example of Mutex

Definition (Mutex Safety)

If $\mathbf{P} \rightarrow^* (\nu\tilde{u})P$ and $P \downarrow_{\mathsf{ul}\langle m\rangle}$, then $P \equiv P' \mid \lceil m \rceil^\star$

a mutual exclusion lock can only be unlocked if it is already locked.

Definition (Mutex Liveness)

If $\mathbf{P} \rightarrow^* (\nu\tilde{u})P$ and $P \downarrow_{\mathsf{l}\langle m\rangle}$, then $\exists P \rightarrow^* P' \xrightarrow{\tau_m}$

a mutual exclusion lock will always eventually answer a lock request.

Behavioural Typing System for MiGo$^+$

$$\mathbf{P}_{\mathsf{race}} \stackrel{\text{def}}{=} \begin{array}{l}\mathsf{newvar}(x{:}\mathsf{int});\\ \mathsf{newchan}(y{:}\mathsf{int}, 2);\end{array} \left(\begin{array}{l} \;\; y!\langle \mathsf{Lock}\rangle; t_1 = \mathsf{load}(x); x := t_1 + 10;\\ \qquad \ldots \\ \mid y!\langle \mathsf{Lock}\rangle; t_0 = \mathsf{load}(x); x := t_0 + 20;\\ \qquad \ldots \end{array}\right)$$

$$\mathbf{T}_{\mathsf{race}} \stackrel{\text{def}}{=} (\nu \mathsf{v}\, x)(\nu\, y^2)\left(\begin{array}{l} \;\; \overline{y}; \mathsf{r}(x); \mathsf{w}(x); \ldots \\ \mid \overline{y}; \mathsf{r}(x); \mathsf{w}(x); \ldots \end{array}\right)$$

Behavioural Typing System for MiGo$^+$

$$\mathbf{P}_{\mathsf{race}} \stackrel{\mathsf{def}}{=} \begin{array}{l}\mathsf{newvar}(x{:}\mathsf{int});\\ \mathsf{newchan}(y{:}\mathsf{int}, 2);\end{array} \left(\begin{array}{rl} & y!\langle \mathsf{Lock}\rangle;\ t_1 = \mathsf{load}(x);\ x := t_1 + 10;\\ & \quad \dots \\ \mid & y!\langle \mathsf{Lock}\rangle;\ t_0 = \mathsf{load}(x);\ x := t_0 + 20;\\ & \quad \dots \end{array}\right)$$

$$\mathbf{T}_{\mathsf{race}} \stackrel{\mathsf{def}}{=} (\nu \mathsf{v}\, x)(\nu\, y^2)\left(\begin{array}{rl} & \overline{y};\ \mathsf{r}(x);\ \mathsf{w}(x);\ \dots \\ \mid & \overline{y};\ \mathsf{r}(x);\ \mathsf{w}(x);\ \dots \end{array}\right)$$

$$\mathbf{T}_{\mathsf{race}} \to^2 (\nu x c)\left(\begin{array}{rl} & \overline{c};\ \mathsf{r}(x);\ \mathsf{w}(x);\ \dots \\ \mid & \overline{c};\ \mathsf{r}(x);\ \mathsf{w}(x);\ \dots \\ \mid & x^{\blacksquare} \quad \mid \quad \lfloor c \rfloor_0^2 \end{array}\right)$$

Behavioural Typing System for MiGo$^+$

$$\mathbf{P}_{\text{race}} \stackrel{\text{def}}{=} \begin{array}{l}\text{newvar}(x{:}\text{int});\\ \text{newchan}(y{:}\text{int}, 2);\end{array} \left(\begin{array}{l} \;\; y!\langle\text{Lock}\rangle; t_1 = \text{load}(x); x := t_1 + 10;\\ \qquad\quad \ldots \\ \mid\; y!\langle\text{Lock}\rangle; t_0 = \text{load}(x); x := t_0 + 20;\\ \qquad\quad \ldots \end{array}\right)$$

$$\mathbf{T}_{\text{race}} \stackrel{\text{def}}{=} (\nu \text{v}\, x)(\nu\, y^2)\left(\begin{array}{l} \;\; \overline{y}; \text{r}(x); \text{w}(x); \ldots \\ \mid\; \overline{y}; \text{r}(x); \text{w}(x); \ldots \end{array}\right)$$

$$\mathbf{T}_{\text{race}} \rightarrow^4 (\nu x c)\left(\begin{array}{l} \qquad \text{r}(x); \text{w}(x); \ldots \\ \mid \quad \text{r}(x); \text{w}(x); \ldots \\ \mid\; x^{\blacksquare} \;\mid\; \lfloor c \rfloor_2^2 \end{array}\right)$$

Behavioural Typing System for MiGo^{+}

$$\mathbf{P}_{\mathrm{race}} \stackrel{\mathrm{def}}{=} \begin{array}{l}\mathrm{newvar}(x{:}\mathrm{int});\\ \mathrm{newchan}(y{:}\mathrm{int}, 2);\end{array} \left(\begin{array}{l} \;\; y!\langle \mathrm{Lock}\rangle; t_1 = \mathrm{load}(x); x := t_1 + 10;\\ \qquad \ldots \\ \mid y!\langle \mathrm{Lock}\rangle; t_0 = \mathrm{load}(x); x := t_0 + 20;\\ \qquad \ldots \end{array}\right)$$

$$\mathbf{T}_{\mathrm{race}} \stackrel{\mathrm{def}}{=} (\nu \mathrm{v}\, x)(\nu\, y^2)\left(\begin{array}{l} \;\; \overline{y}; \mathrm{r}(x); \mathrm{w}(x); \ldots \\ \mid \overline{y}; \mathrm{r}(x); \mathrm{w}(x); \ldots \end{array}\right)$$

$$\mathbf{T}_{\mathrm{race}} \rightarrow^6 (\nu x c)\left(\begin{array}{lll} & & \mathrm{w}(x); \ldots \\ \mid & & \mathrm{w}(x); \ldots \\ \mid x^{\blacksquare} & \mid & \lfloor c \rfloor_2^2 \end{array}\right)$$

Properties of our Type System

Our type system has reduction transitions that follow almost exactly the reduction of the MiGo^{+} processes, expect for $\text{IF}/\text{THEN}/\text{ELSE}$ constructs.
It also does not care about the content of the data.
Because of that, it admits the following properties:

Properties of our Type System

Our type system has reduction transitions that follow almost exactly the reduction of the MiGo^{+} processes, expect for $\text{IF}/\text{THEN}/\text{ELSE}$ constructs. It also does not care about the content of the data.
Because of that, it admits the following properties:

Theorem (Subject reduction)

If $\Gamma \vdash P \blacktriangleright T$ and P reduces to P', then T has a reduction T' such that $\Gamma \vdash P' \blacktriangleright T'$.

Properties of our Type System

Our type system has reduction transitions that follow almost exactly the reduction of the MiGo$^+$ processes, expect for $\text{IF}/\text{THEN}/\text{ELSE}$ constructs.
It also does not care about the content of the data.
Because of that, it admits the following properties:

Theorem (Subject reduction)

If $\Gamma \vdash P \blacktriangleright T$ and P reduces to P', then T has a reduction T' such that $\Gamma \vdash P' \blacktriangleright T'$.

Theorem (Progress)

If $\Gamma \vdash P \blacktriangleright T$ and T reduces to T_0, then P has a reduction P' and there exists a reduction T' of T such that $\Gamma \vdash P' \blacktriangleright T'$.

Verifying Processes through their Types

Theorem (Process-Type relation)

1. **P** *is safe (resp. data race free) iff* **T** *is safe (resp. data race free)*

Processes

Imperial College London

Verifying Processes through their Types

Theorem (Process-Type relation)

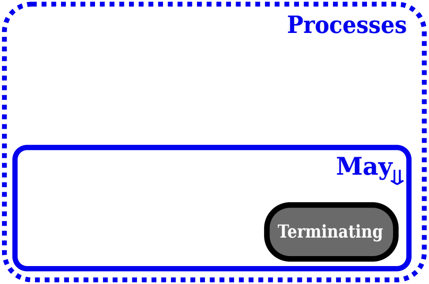
1. **P** *is safe (resp. data race free) iff* **T** *is safe (resp. data race free)*
2. **P** *is live iff* **T** *is live and*

Processes

Verifying Processes through their Types

Theorem (Process-Type relation)

1. **P** *is safe (resp. data race free) iff* **T** *is safe (resp. data race free)*
2. **P** *is live iff* **T** *is live and*
   - **P** $\in$ May$_{\Downarrow}$

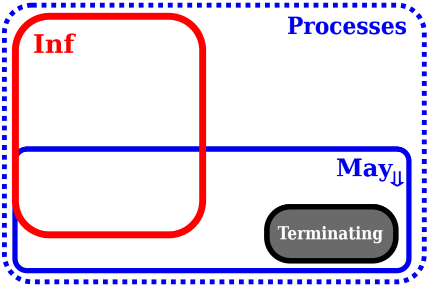


Imperial College London

Verifying Processes through their Types

Theorem (Process-Type relation)

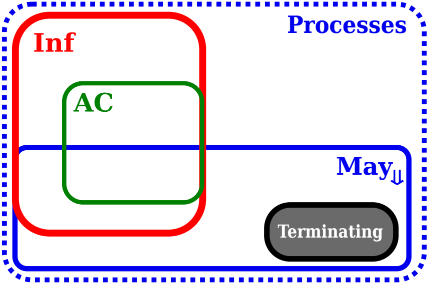
1. **P** *is safe (resp. data race free) iff* **T** *is safe (resp. data race free)*
2. **P** *is live iff* **T** *is live and*
   - **P** $\in$ May$\Downarrow$ *or*
   - **P** $\notin$ Inf



Verifying Processes through their Types

Theorem (Process-Type relation)

1. **P** *is safe (resp. data race free) iff* **T** *is safe (resp. data race free)*
2. **P** *is live iff* **T** *is live and*
   - **P** $\in$ May$\Downarrow$ *or*
   - **P** $\notin$ Inf *or*
   - **P** $\in$ AC



Type Verification: the Modal μ-calculus

Definition: $T \downarrow_o$ iff T can execute action o immediately.

$$\mathbf{T}_{\mathsf{race}} \rightarrow^2 (\nu xc) \left(\begin{array}{l} \quad \overline{c}; \mathsf{r}(x); \mathsf{w}(x); \ldots \\ \mid \ \overline{c}; \mathsf{r}(x); \mathsf{w}(x); \ldots \\ \mid \ x^{\blacksquare} \ \mid \ \lfloor c \rfloor_0^2 \end{array} \right) = (\nu xc) T$$

Imperial College London

Type Verification: the Modal μ-calculus

Definition: $T \downarrow_o$ iff T can execute action o immediately.

$$\mathbf{T}_{\text{race}} \to^2 (\nu xc) \begin{pmatrix} \overline{c}; \mathsf{r}(x); \mathsf{w}(x); \ldots \\ \mid \overline{c}; \mathsf{r}(x); \mathsf{w}(x); \ldots \\ \mid x^{\blacksquare} \;\mid\; \lfloor c \rfloor_0^2 \end{pmatrix} = (\nu xc) T$$

$$\begin{array}{c} T \\ \Big| \\ \boxed{\downarrow_{\overline{c}} \ \downarrow_{x^{\blacksquare}} \ \downarrow_{\bullet c} \ \downarrow_{\tau_c}} \end{array}$$

Type Verification: the Modal μ-calculus

Definition: $T \downarrow_o$ iff T can execute action o immediately.

$$\mathbf{T}_{\text{race}} \rightarrow^3 (\nu xc) \left(\begin{array}{l} \quad\ \ \mathsf{r}(x); \mathsf{w}(x); \ldots \\ \mid\ \overline{c}; \mathsf{r}(x); \mathsf{w}(x); \ldots \\ \mid\ x^{\blacksquare} \ \mid\ \lfloor c \rfloor_1^2 \end{array} \right) = (\nu xc)\, T'$$

$$\begin{tikzcd}
T \arrow[rr, "\tau_c"] \arrow[d, dash] & & T' \arrow[d, dash] \\
\boxed{\downarrow_{\overline{c}}\ \downarrow_{x^{\blacksquare}}\ \downarrow_{\bullet c}\ \downarrow_{\tau_c}} & & \boxed{\begin{array}{c} \downarrow_{\mathsf{r}\langle x \rangle}\ \downarrow_{\overline{c}}\ \downarrow_{x^{\blacksquare}} \\ \downarrow_{\bullet c}\ \downarrow_{c \bullet}\ \downarrow_{\tau_x}\ \downarrow_{\tau_c} \end{array}}
\end{tikzcd}$$

Modal properties are logical propositions guarded by modalities parametrised by the flags and the synchronisations. The two modalities are "there exists an action" (diamond) and "for all actions" (box).

Type Verification: the Modal μ-calculus

Definition: $T \downarrow_o$ iff T can execute action o immediately.

$$\mathbf{T}_{\mathsf{race}} \rightarrow^4 (\nu xc)\begin{pmatrix} \mathsf{w}(x);\ldots \\ \mid \overline{c};\mathsf{r}(x);\mathsf{w}(x);\ldots \\ \mid x^{\blacksquare} \quad \mid \quad \lfloor c \rfloor_1^2 \end{pmatrix} = (\nu xc)T''$$

$$\begin{tikzcd}
T \arrow[r, "\tau_c"] \arrow[d, dash] & T' \arrow[r, "\tau_x"] \arrow[d, dash] & T'' \arrow[r, "\tau_x"] \arrow[d, dash] & \cdots \\
\boxed{\downarrow_{\overline{c}} \ \downarrow_{x^{\blacksquare}} \ \downarrow_{\bullet c} \ \downarrow_{\tau_c}} & \boxed{\begin{array}{c}\downarrow_{\mathsf{r}\langle x\rangle} \ \downarrow_{\overline{c}} \ \downarrow_{x^{\blacksquare}} \\ \downarrow_{\bullet c} \ \downarrow_{c\bullet} \ \downarrow_{\tau_x} \ \downarrow_{\tau_c}\end{array}} & \boxed{\begin{array}{c}\downarrow_{(\mathsf{w}\langle x\rangle,1.*)} \ \downarrow_{\overline{c}} \ \downarrow_{x^{\blacksquare}} \\ \downarrow_{\bullet c} \ \downarrow_{c\bullet} \ \downarrow_{\tau_x}, \downarrow_{\tau_c}\end{array}} &
\end{tikzcd}$$

Modal properties are logical propositions guarded by modalities parametrised by the flags and the synchronisations. The two modalities are "there exists an action" (diamond) and "for all actions" (box).

Model Checking the Types and Processes

Formula $\Psi(\phi)$ means "ϕ is true in every reachable state"

Theorem (Model Checking of MiGo$^+$ processes)

Suppose $\Gamma \vdash \mathbf{P} \blacktriangleright \mathbf{T}$.

Processes

Model Checking the Types and Processes

Formula $\Psi(\phi)$ means "ϕ is true in every reachable state"

Theorem (Model Checking of MiGo$^+$ processes)

Suppose $\Gamma \vdash \mathbf{P} \blacktriangleright \mathbf{T}$.

1. *If* $\mathbf{T} \models \Psi(\phi)$ *where* ϕ *is a safety property, then* $\mathbf{P} \models \Psi(\phi)$.

Processes

Imperial College London

Model Checking the Types and Processes

Formula $\Psi(\phi)$ means "ϕ is true in every reachable state"

Theorem (Model Checking of MiGo$^+$ processes)

Suppose $\Gamma \vdash \mathbf{P} \blacktriangleright \mathbf{T}$.

1. *If* $\mathbf{T} \models \Psi(\phi)$ *where* ϕ *is a safety property, then* $\mathbf{P} \models \Psi(\phi)$.
2. *If* $\mathbf{T} \models \Psi(\phi)$ *where* ϕ *is a liveness property, and either*

   then $\mathbf{P} \models \Psi(\phi)$.

Processes

Model Checking the Types and Processes

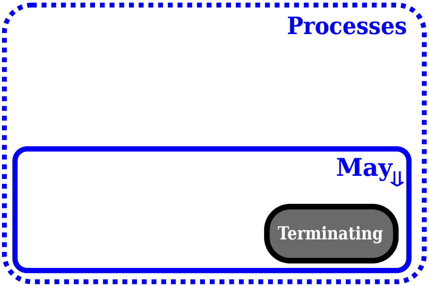
Formula $\Psi(\phi)$ means "ϕ is true in every reachable state"

Theorem (Model Checking of MiGo$^+$ processes)

Suppose $\Gamma \vdash \mathbf{P} \blacktriangleright \mathbf{T}$.

1. *If* $\mathbf{T} \models \Psi(\phi)$ *where* ϕ *is a safety property, then* $\mathbf{P} \models \Psi(\phi)$.
2. *If* $\mathbf{T} \models \Psi(\phi)$ *where* ϕ *is a liveness property, and either*
   (a) $\mathbf{P} \in \mathrm{May}_{\Downarrow}$

   then $\mathbf{P} \models \Psi(\phi)$.



Model Checking the Types and Processes

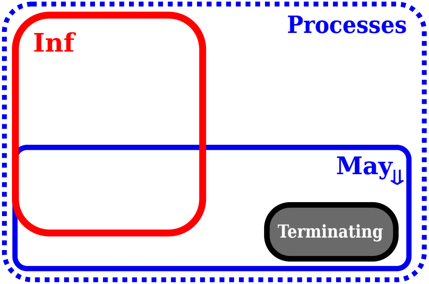
Formula $\Psi(\phi)$ means "ϕ is true in every reachable state"

Theorem (Model Checking of MiGo$^+$ processes)

Suppose $\Gamma \vdash \mathbf{P} \blacktriangleright \mathbf{T}$.

1. *If* $\mathbf{T} \models \Psi(\phi)$ *where* ϕ *is a safety property, then* $\mathbf{P} \models \Psi(\phi)$.
2. *If* $\mathbf{T} \models \Psi(\phi)$ *where* ϕ *is a liveness property, and either*
   (a) $\mathbf{P} \in \mathsf{May}_{\Downarrow}$ *or*
   (b) $\mathbf{P} \notin \mathsf{Inf}$

   then $\mathbf{P} \models \Psi(\phi)$.



Imperial College London

Model Checking the Types and Processes

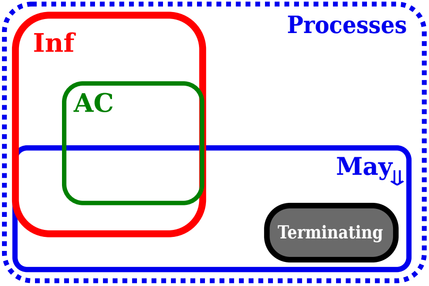
Formula $\Psi(\phi)$ means "ϕ is true in every reachable state"

Theorem (Model Checking of MiGo$^+$ processes)

Suppose $\Gamma \vdash \mathbf{P} \blacktriangleright \mathbf{T}$.

1. *If* $\mathbf{T} \models \Psi(\phi)$ *where* ϕ *is a safety property, then* $\mathbf{P} \models \Psi(\phi)$.
2. *If* $\mathbf{T} \models \Psi(\phi)$ *where* ϕ *is a liveness property, and either*
   (a) $\mathbf{P} \in \mathsf{May}_{\Downarrow}$ *or*
   (b) $\mathbf{P} \notin \mathsf{Inf}$ *or*
   (c) $\mathbf{P} \in \mathsf{AC}$

   then $\mathbf{P} \models \Psi(\phi)$.



Implementation: Godel 2

Go source code

Figure 3: Workflow of the verification toolchain.

Implementation: Godel 2

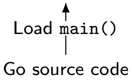
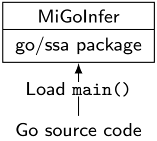


Figure 3: Workflow of the verification toolchain.

Implementation: Godel 2

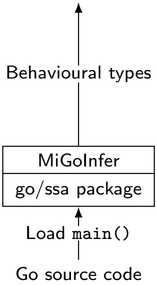


MiGoInfer *written in* **Go**

This tool loads source code, type-checks and builds SSA IR using the go/ssa package, then extracts communication, mutexes and shared variables from the SSA IR as behavioural types.

Figure 3: Workflow of the verification toolchain.

Implementation: Godel 2

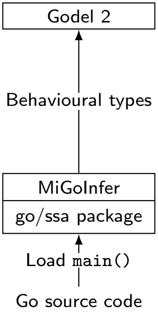


MiGoInfer *written in* **Go**

This tool loads source code, type-checks and builds SSA IR using the `go/ssa` package, then extracts communication, mutexes and shared variables from the SSA IR as behavioural types.

Figure 3: Workflow of the verification toolchain.

Implementation: Godel 2



Godel 2 *written in* **Haskell**

This tool uses either KiTTEL to check for termination of the input behavioural types, or mCRL2 to check for properties like liveness, safety and data-race freedom of the types.

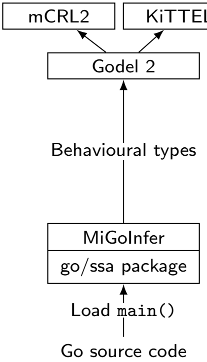
MiGoInfer *written in* **Go**

This tool loads source code, type-checks and builds SSA IR using the `go/ssa` package, then extracts communication, mutexes and shared variables from the SSA IR as behavioural types.

Figure 3: Workflow of the verification toolchain.

Imperial College London

Implementation: Godel 2



Godel 2 *written in* **Haskell**

This tool uses either KiTTEL to check for termination of the input behavioural types, or mCRL2 to check for properties like liveness, safety and data-race freedom of the types.

MiGoInfer *written in* **Go**

This tool loads source code, type-checks and builds SSA IR using the `go/ssa` package, then extracts communication, mutexes and shared variables from the SSA IR as behavioural types.

Figure 3: Workflow of the verification toolchain.

Table 1: Go Programs Verified by the Toolchain.

Programs	LoC	Sum	Safe	Live	DRF	time (ms)
`no-race`	15	9	✓	✓	✓	691.45
`no-race-mutex`	24	33	✓	✓	✓	785.57
`no-race-mut-bad`	23	20	✓	×	✓	721.77
`simple-race`	13	8	✓	✓	×	701.93
`simple-race-fix`	19	17	✓	✓	✓	731.73
`deposit-race`[1]	18	14	✓	✓	×	697.90
`deposit-fix`[1]	24	27	✓	✓	✓	727.43
`ch-as-lock-race`[2]	19	20	✓	✓	×	753.99
`ch-as-lock-fix`[2]	19	20	✓	✓	✓	745.64
`ch-as-lock-bad`	19	20	✓	×	✓	749.97
`prod-cons-race`	38	156	✓	✓	×	1,903.52
`prod-cons-fix`	40	188	✓	✓	✓	1,971.26
`dinephil5-race`	59	2672	✓	✓	×	$\sim$ 185mn
`dinephil5-fix`	59	2688	✓	✓	✓	$\sim$ 645mn

[1]Donovan, A.A., Kernighan, B.W.: The Go Programming Language (2015), [2]Running example, LoC: Lines of Code, DRF: Data Race Free, Sum: Summands,
✓: Formula is true, ×: Formula is false

Questions?