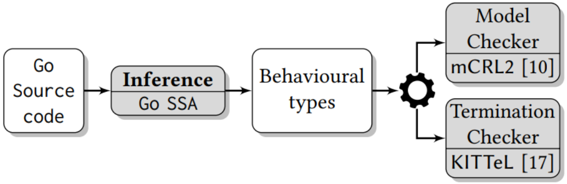
A Static Verification Framework for Message Passing in Go using Behavioural Types

Hugo Moreau

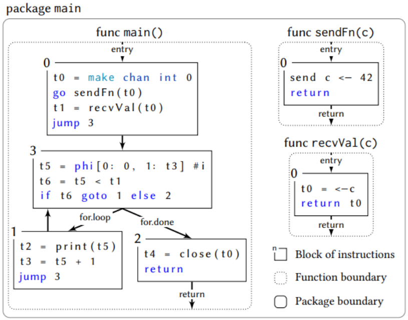
January 2021

Godel Checker, a static analysis toolchain [Julien Lange, 2018]

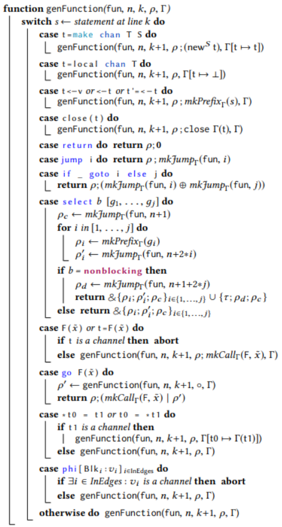


Static Single Assignment [Julien Lange, 2018]

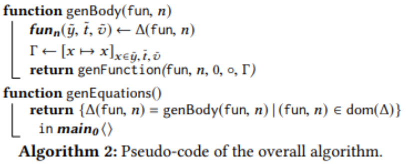
```go
func main() {
  ch := make(chan int) // Create channel
  go sendFn(ch)        // Run as goroutine
  x := recvVal(ch)     // Ordinary func call
  for i := 0; i < x; i++ {
    print(i)
  }
  close(ch) // Close channel ch
}
func sendFn(c chan int) {
  c <- 42 // Send on channel c
}
func recvVal(c chan int) int {
  return <-c // Receive from channel c
}
```



Extracting types [Julien Lange, 2018]



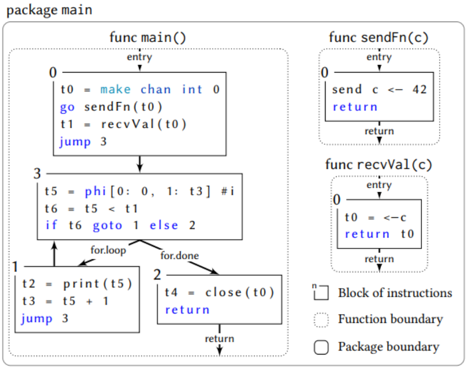
```
function genFunction(fun, $n$, $k$, $\rho$, $\Gamma$)
    switch $s \leftarrow$ statement at line $k$ do
        case t=make chan T S do
            genFunction(fun, $n$, $k$+1, $\rho$ ; (new$^S$ t), $\Gamma$[t $\mapsto$ t])
        case t=local chan T do
            genFunction(fun, $n$, $k$+1, $\rho$, $\Gamma$[t $\mapsto \perp$])
        case t<-v or <-t or t'=<-t do
            genFunction(fun, $n$, $k$+1, $\rho$ ; $mkPrefix_\Gamma(s)$, $\Gamma$)
        case close(t) do
            genFunction(fun, $n$, $k$+1, $\rho$ ; close $\Gamma$(t), $\Gamma$)
        case return do return $\rho$;$\mathbf{0}$
        case jump i do return $\rho$ ; $mkJump_\Gamma$(fun, $i$)
        case if _ goto i else j do
            return $\rho$;($mkJump_\Gamma$(fun, $i$) $\oplus$ $mkJump_\Gamma$(fun, $j$))
        case select $b$ [$g_1, \ldots, g_j$] do
            $\rho_c \leftarrow mkJump_\Gamma$(fun, $n$+1)
            for $i$ in [1, . . . , $j$] do
                $\rho_i \leftarrow mkPrefix_\Gamma(g_i)$
                $\rho'_i \leftarrow mkJump_\Gamma$(fun, $n$+2$*i$)
            if $b$ = nonblocking then
                $\rho_d \leftarrow mkJump_\Gamma$(fun, $n$+1+2$*j$)
                return $\&\{\rho_i; \rho'_i; \rho_c\}_{i \in \{1,\ldots,j\}} \cup \{\tau; \rho_d; \rho_c\}$
            else return $\&\{\rho_i; \rho'_i; \rho_c\}_{i \in \{1,\ldots,j\}}$
        case F($\tilde{x}$) or t=F($\tilde{x}$) do
            if t is a channel then abort
            else genFunction(fun, $n$, $k$+1, $\rho$; $mkCall_\Gamma$(F, $\tilde{x}$), $\Gamma$)
        case go F($\tilde{x}$) do
            $\rho' \leftarrow$ genFunction(fun, $n$, $k$+1, $\circ$, $\Gamma$)
            return $\rho$;($mkCall_\Gamma$(F, $\tilde{x}$) | $\rho'$)
        case *t0 = t1 or t0 = *t1 do
            if t1 is a channel then
                genFunction(fun, $n$, $k$+1, $\rho$, $\Gamma$[t0 $\mapsto$ $\Gamma$(t1)])
            else genFunction(fun, $n$, $k$+1, $\rho$, $\Gamma$)
        case phi[Blk$_i$ : $v_i$]$_{i \in \mathrm{InEdges}}$ do
            if $\exists i \in InEdges : v_i$ is a channel then abort
            else genFunction(fun, $n$, $k$+1, $\rho$, $\Gamma$)
        otherwise do genFunction(fun, $n$, $k$+1, $\rho$, $\Gamma$)
```



```
function genBody(fun, $n$)
    $\mathbf{fun}_n(\tilde{y}, \tilde{t}, \tilde{v}) \leftarrow \Delta$(fun, $n$)
    $\Gamma \leftarrow [x \mapsto x]_{x \in \tilde{y}, \tilde{t}, \tilde{v}}$
    return genFunction(fun, $n$, 0, $\circ$, $\Gamma$)
function genEquations()
    return {$\Delta$(fun, $n$) = genBody(fun, $n$) | (fun, $n$) $\in$ dom($\Delta$)}
     in $\mathbf{main}_0\langle\rangle$
```

Algorithm 2: Pseudo-code of the overall algorithm.

SSA to Behavioural types [Julien Lange, 2018]



$$\boldsymbol{main_0}() = (\text{new}\, t0); (\boldsymbol{sendFn_0}\langle t0\rangle \mid \boldsymbol{recvVal_0}\langle t0\rangle ; \boldsymbol{main_3}\langle t0\rangle)$$
$$\boldsymbol{main_1}(t0) = \boldsymbol{main_3}\langle t0\rangle$$
$$\boldsymbol{main_2}(t0) = \text{close}\, t0; \mathbf{0}$$
$$\boldsymbol{main_3}(t0) = \boldsymbol{main_1}\langle t0\rangle \oplus \boldsymbol{main_2}\langle t0\rangle$$
$$\boldsymbol{sendFn_0}(c) = \bar{c}; \mathbf{0}$$
$$\boldsymbol{recvVal_0}(c) = c; \mathbf{0}$$

Semantic of types [Julien Lange, 2017]

$$\overline{a};T \xrightarrow{\overline{a}} T \qquad a;T \xrightarrow{a} T \qquad \tau;T \xrightarrow{\tau} T$$

$$\mathtt{close}\,a;T \xrightarrow{\mathtt{clo}\,a} T \qquad \lfloor a \rfloor_k^n \xrightarrow{\overline{\mathtt{clo}}\,a} a^\star \qquad a^\star \xrightarrow{a^\star} a^\star$$

$$\frac{i \in \{1,2\}}{T_1 \oplus T_2 \xrightarrow{\tau} T_i} \qquad \frac{\alpha_j;T_j \xrightarrow{\alpha_j} T_j \qquad j \in I}{\&\{\alpha_i;T_i\}_{i\in I} \xrightarrow{\alpha_j} T_j}$$

$$\frac{T \xrightarrow{\alpha} T'}{T \mid S \xrightarrow{\alpha} T' \mid S} \qquad \frac{T \xrightarrow{\alpha} T'}{T;S \xrightarrow{\alpha} T';S} \qquad \mathbf{0};S \xrightarrow{\tau} S$$

$$\frac{\alpha \in \{\overline{a}, a^\star, a^\bullet\} \qquad T \xrightarrow{\alpha} T' \qquad S \xrightarrow{\beta} S' \qquad \beta \in \{{}^\bullet a, a\}}{T \mid S \xrightarrow{\tau_a} T' \mid S'}$$

$$\frac{T \equiv_\alpha T' \qquad T \xrightarrow{\alpha} T''}{T' \xrightarrow{\alpha} T''} \qquad \frac{T\{\tilde{a}/\tilde{x}\} \xrightarrow{\alpha} T' \qquad \mathbf{t}(\tilde{x}) = T}{\mathbf{t}\langle\tilde{a}\rangle \xrightarrow{\alpha} T'}$$

$$\frac{T \xrightarrow{\mathtt{clo}\,a} T' \qquad S \xrightarrow{\overline{\mathtt{clo}}\,a} S'}{T \mid S \xrightarrow{\tau} T' \mid S'} \qquad \frac{k < n}{\lfloor a \rfloor_k^n \xrightarrow{{}^\bullet a} \lfloor a \rfloor_{k+1}^n} \qquad \frac{k \geq 1}{\lfloor a \rfloor_k^n \xrightarrow{a^\bullet} \lfloor a \rfloor_{k-1}^n}$$

$$\begin{array}{rcl} \alpha & \coloneqq & \overline{u} \mid u \mid \tau \qquad\qquad \boldsymbol{T} \coloneqq \{\mathbf{t}_i(\tilde{y}_i) = T_i\}_{i\in I} \ \mathtt{in}\ S \\ T,S & \coloneqq & \alpha;T \mid T;S \mid T \oplus S \mid \&\{\alpha_i;T_i\}_{i\in I} \mid (T \mid S) \mid \mathbf{0} \\ & \mid & (\mathtt{new}^n\ a);T \mid \mathtt{close}\,u;T \mid \mathbf{t}\langle\tilde{u}\rangle \mid \lfloor a \rfloor_k^n \mid a^\star \end{array}$$

$\overline{a}$ / a	send / receive on channel a
τ_a	synchronisation over a
τ	silent action
$\mathtt{clo}\,a$ / $\overline{\mathtt{clo}}\,a$	request to close a / closing a
$a^\star$	channel a is closed
${}^\bullet a$ / $a^\bullet$	push / pop on buffer a
$\tilde{o}$	waiting to synchronise over the actions in $\tilde{o}$

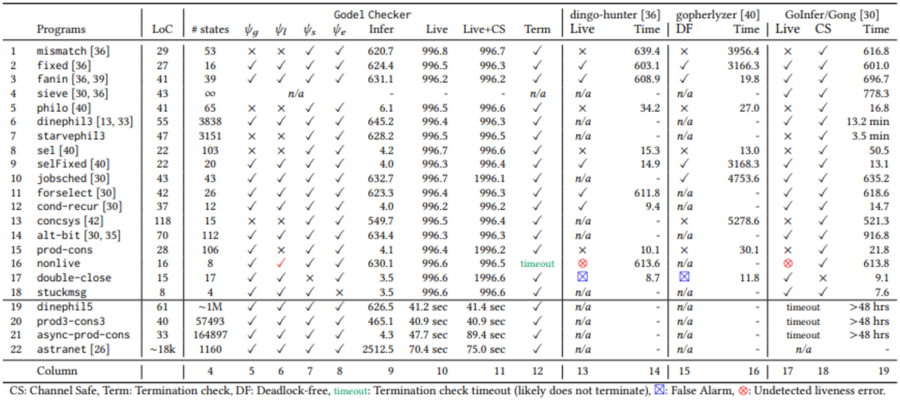
Verification process

1. Type verifier (LTS)
2. Model checking (mCRL2)
3. Termination checking (KITTeL)

Remark

Loops in Go programs generates **types with conditional branching**.

Godel Checker's benchmark



			Godel Checker									dingo-hunter [36]		gopherlyzer [40]		GoInfer/Gong [30]		
	Programs	LoC	# states	ψ_g	ψ_l	ψ_s	ψ_e	Infer	Live	Live+CS	Term	Live	Time	DF	Time	Live	CS	Time
1	`mismatch` [36]	29	53	×	×	✓	✓	620.7	996.8	996.7	✓	×	639.4	×	3956.4	×	✓	616.8
2	`fixed` [36]	27	16	✓	✓	✓	✓	624.4	996.5	996.3	✓	✓	603.1	✓	3166.3	✓	✓	601.0
3	`fanin` [36, 39]	41	39	✓	✓	✓	✓	631.1	996.2	996.2	✓	✓	608.9	✓	19.8	✓	✓	696.7
4	`sieve` [30, 36]	43	∞		*n/a*			-	-	-	*n/a*	*n/a*	-	*n/a*	-	✓	✓	778.3
5	`philo` [40]	41	65	×	×	✓	✓	6.1	996.5	996.6	✓	×	34.2	×	27.0	×	✓	16.8
6	`dinephil3` [13, 33]	55	3838	✓	✓	✓	✓	645.2	996.4	996.3	✓	*n/a*	-	*n/a*	-	✓	✓	13.2 min
7	`starvephil3`	47	3151	×	×	✓	✓	628.2	996.5	996.5	✓	*n/a*	-	*n/a*	-	×	✓	3.5 min
8	`sel` [40]	22	103	×	×	✓	✓	4.2	996.7	996.6	✓	×	15.3	×	13.0	×	✓	50.5
9	`selFixed` [40]	22	20	✓	✓	✓	✓	4.0	996.3	996.4	✓	✓	14.9	✓	3168.3	✓	✓	13.1
10	`jobsched` [30]	43	43	✓	✓	✓	✓	632.7	996.7	1996.1	✓	*n/a*	-	✓	4753.6	✓	✓	635.2
11	`forselect` [30]	42	26	✓	✓	✓	✓	623.3	996.4	996.3	✓	✓	611.8	*n/a*	-	✓	✓	618.6
12	`cond-recur` [30]	37	12	✓	✓	✓	✓	4.0	996.2	996.2	✓	✓	9.4	*n/a*	-	✓	✓	14.7
13	`concsys` [42]	118	15	×	×	✓	✓	549.7	996.5	996.4	✓	*n/a*	-	×	5278.6	×	✓	521.3
14	`alt-bit` [30, 35]	70	112	✓	✓	✓	✓	634.4	996.3	996.3	✓	*n/a*	-	*n/a*	-	✓	✓	916.8
15	`prod-cons`	28	106	✓	×	✓	✓	4.1	996.4	1996.2	✓	×	10.1	×	30.1	×	✓	21.8
16	`nonlive`	16	8	✓	✓	✓	✓	630.1	996.6	996.5	timeout	⊗	613.6	*n/a*	-	⊗	✓	613.8
17	`double-close`	15	17	✓	✓	×	✓	3.5	996.6	1996.6	✓	⊠	8.7	⊠	11.8	✓	×	9.1
18	`stuckmsg`	8	4	✓	✓	✓	×	3.5	996.6	996.6	✓	*n/a*	-	*n/a*	-	✓	✓	7.6
19	`dinephil5`	61	~1M	✓	✓	✓	✓	626.5	41.2 sec	41.4 sec	✓	*n/a*	-	*n/a*	-	timeout		>48 hrs
20	`prod3-cons3`	40	57493	✓	✓	✓	✓	465.1	40.9 sec	40.9 sec	✓	*n/a*	-	*n/a*	-	timeout		>48 hrs
21	`async-prod-cons`	33	164897	✓	✓	✓	✓	4.3	47.7 sec	89.4 sec	✓	*n/a*	-	*n/a*	-	timeout		>48 hrs
22	`astranet` [26]	~18k	1160	✓	✓	✓	✓	2512.5	70.4 sec	75.0 sec	✓	*n/a*	-	*n/a*	-	*n/a*		-
	Column		4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

CS: Channel Safe, Term: Termination check, DF: Deadlock-free, timeout: Termination check timeout (likely does not terminate), ⊠: False Alarm, ⊗: Undetected liveness error.

Recap

- Support dynamic spawning of goroutines.
- Handling uninitialised channels.
- Using behavioural types to check for safety and liveness properties (not just deadlock-freedom)
- Better performance for larger programs.

What's next?

What has been done:

- Read the bibliography.
- Read some research articles.

What's next?

What has been done:

- Read the bibliography.
- Read some research articles.

What should be done next :

- Find what we can take from it
- And probably continue to read articles...

References

Julien Lange, Nicholas Ng, B. T. N. Y. (2017).
Fencing off go: liveness and safety for channel-based programming.
pages 748–761.

Julien Lange, Nicholas Ng, B. T. N. Y. (2018).
A static verification framework for message passing in go using behavioural types.