Towards a verified compiler for Distributed PlusCal

Presented at WPTE '25 on July 20, 2025

By Ghilain BERGERON, Horatiu CIRSTEA, Stephan MERZ
(Université de Lorraine, CNRS, Inria, LORIA)

Overview

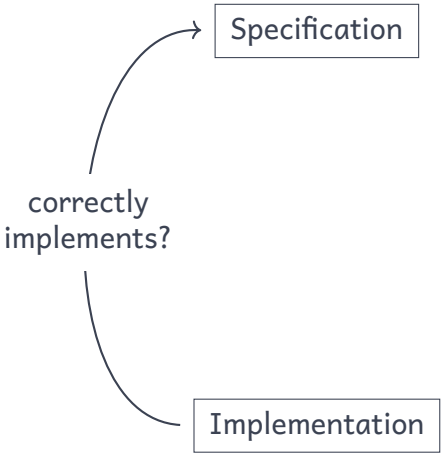
Implementation

Overview

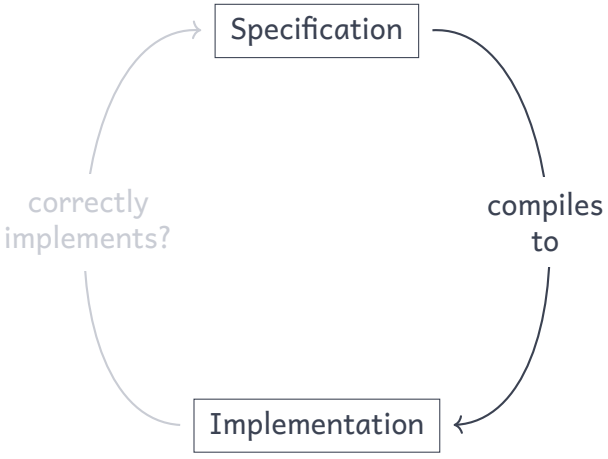
Specification

Implementation

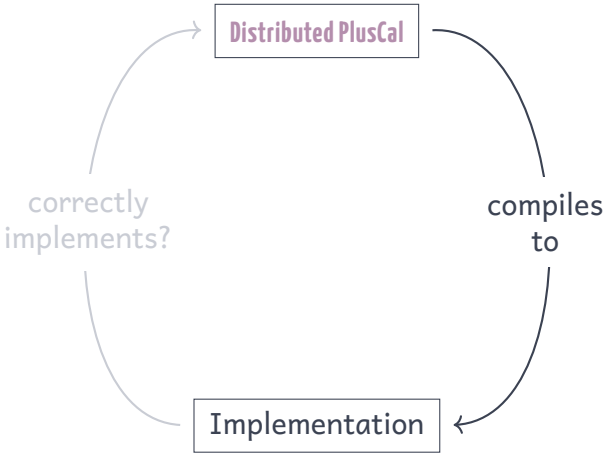
Overview



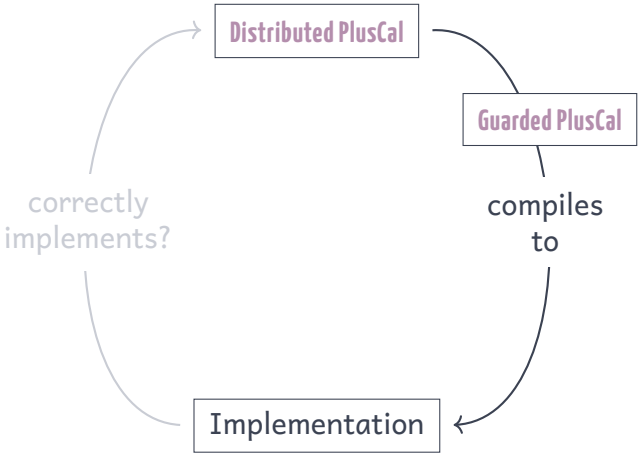
Overview


Specification
compiles to
Implementation
correctly implements?

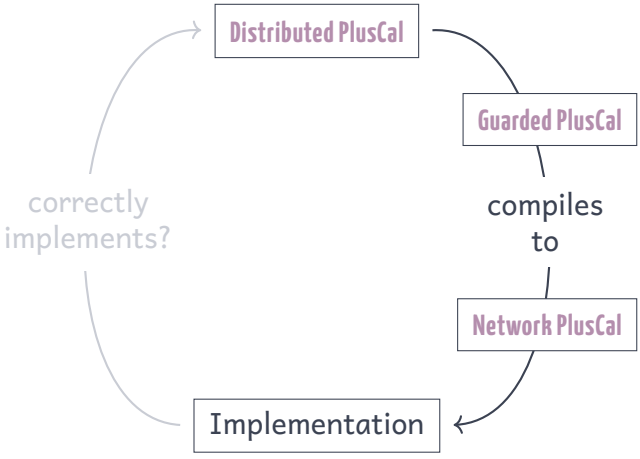
Overview


Distributed PlusCal
compiles to
Implementation
correctly implements?

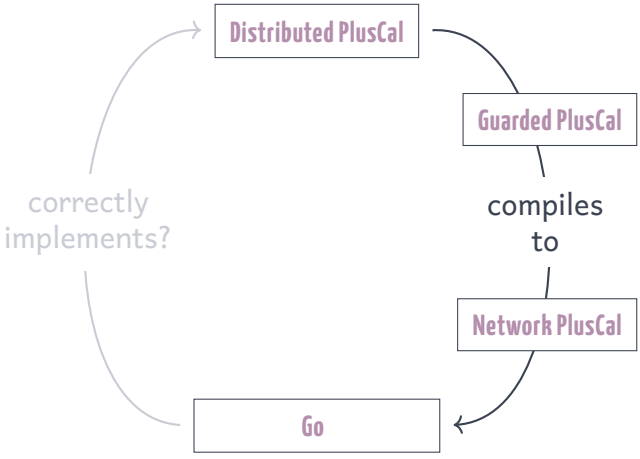
Overview


Distributed PlusCal
Guarded PlusCal
compiles to
Implementation
correctly implements?

Overview


Distributed PlusCal
Guarded PlusCal
compiles
to
Network PlusCal
Implementation
correctly
implements?

Overview


Distributed PlusCal
Guarded PlusCal
compiles
to
Network PlusCal
Go
correctly
implements?

A specification language for distributed/concurrent algorithms...

Distributed PlusCal

A specification language for distributed/concurrent algorithms...

* ...translated to a TLA$^+$ specification.

Distributed PlusCal

A specification language for distributed/concurrent algorithms...

- ...translated to a TLA$^+$ specification.
- ...featuring message-passing and threading primitives.

Distributed PlusCal

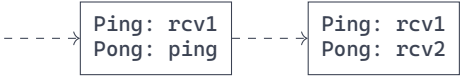
A specification language for distributed/concurrent algorithms...

- ...translated to a TLA$^+$ specification.
- ...featuring message-passing and threading primitives.
- ...that looks like the C language.

```
algorithm PingPong {
    fifos ping, pong;

    process (Ping = 0)
        variable tmp1 = "";
    {
    rcv1:   receive(ping, tmp1);
            await tmp1 = "Ping";
            goto pong;
    pong:   send(pong, "Pong");
            goto rcv1;
    }

    process (Pong = 1)
        variable tmp2 = "";
    {
    ping:   send(ping, "Ping");
            goto rcv2;
    rcv2:   receive(pong, tmp2);
            await tmp2 = "Pong";
            goto ping;
    }
}
```

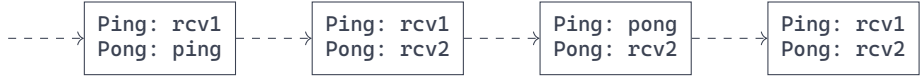
```
algorithm PingPong {
    fifos ping, pong;

    process (Ping = 0)
        variable tmp1 = "";
    {
    rcv1:   receive(ping, tmp1);
            await tmp1 = "Ping";
            goto pong;
    pong:   send(pong, "Pong");
            goto rcv1;
    }

    process (Pong = 1)
        variable tmp2 = "";
    {
    ping:   send(ping, "Ping");
            goto rcv2;
    rcv2:   receive(pong, tmp2);
            await tmp2 = "Pong";
            goto ping;
    }
}
```

```
algorithm PingPong {
    fifos ping, pong;

    process (Ping = 0)
        variable tmp1 = "";
    {
    rcv1:   receive(ping, tmp1);
            await tmp1 = "Ping";
            goto pong;
    pong:   send(pong, "Pong");
            goto rcv1;
    }

    process (Pong = 1)
        variable tmp2 = "";
    {
    ping:   send(ping, "Ping");
            goto rcv2;
    rcv2:   receive(pong, tmp2);
            await tmp2 = "Pong";
            goto ping;
    }
}
```

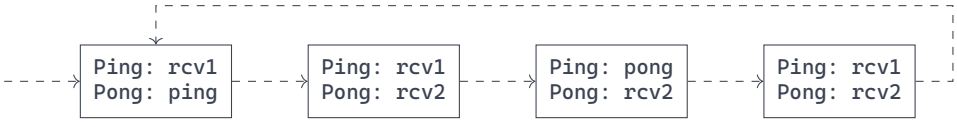


```
algorithm PingPong {
    fifos ping, pong;

    process (Ping = 0)
        variable tmp1 = "";
    {
    rcv1:   receive(ping, tmp1);
            await tmp1 = "Ping";
            goto pong;
    pong:   send(pong, "Pong");
            goto rcv1;
    }

    process (Pong = 1)
        variable tmp2 = "";
    {
    ping:   send(ping, "Ping");
            goto rcv2;
    rcv2:   receive(pong, tmp2);
            await tmp2 = "Pong";
            goto ping;
    }
}
```

```
algorithm PingPong {
    fifos ping, pong;

    process (Ping = 0)
        variable tmp1 = "";
    {
    rcv1:   receive(ping, tmp1);
            await tmp1 = "Ping";
            goto pong;
    pong:   send(pong, "Pong");
            goto rcv1;
    }

    process (Pong = 1)
        variable tmp2 = "";
    {
    ping:   send(ping, "Ping");
            goto rcv2;
    rcv2:   receive(pong, tmp2);
            await tmp2 = "Pong";
            goto ping;
    }
}
```



```
algorithm PingPong {
    fifos ping, pong;

    process (Ping = 0)
        variable tmp1 = "";
    {
    rcv1:   receive(ping, tmp1);
            await tmp1 = "Ping";
            goto pong;
    pong:   send(pong, "Pong");
            goto rcv1;
    }

    process (Pong = 1)
        variable tmp2 = "";
    {
    ping:   send(ping, "Ping");
            goto rcv2;
    rcv2:   receive(pong, tmp2);
            await tmp2 = "Pong";
            goto ping;
    }
}
```



Distributed PlusCal is syntactically close to a programming language.

Distributed PlusCal is syntactically close to a programming language. Yet its semantics is very cumbersome to preserve:

Distributed PlusCal is syntactically close to a programming language. Yet its semantics is very cumbersome to preserve:

```
l: x := f();
   receive(c, y);
   print y;
   await y > 8;
   goto k;
```

Distributed PlusCal is syntactically close to a programming language. Yet its semantics is very cumbersome to preserve:

```
l: x := f();
   receive(c, y);
   print y;
   await y > 8;
   goto k;
```

⚠ Receiving from a channel means consuming a message!

Distributed PlusCal is syntactically close to a programming language. Yet its semantics is very cumbersome to preserve:

```
l: x := f();
   receive(c, y);
   print y;
   await y > 8;
   goto k;
```

- ⚠ Receiving from a channel means consuming a message!
- ⚠ No printing when y <= 8!

Distributed PlusCal is syntactically close to a programming language. Yet its semantics is very cumbersome to preserve:

```
l: x := f();
   receive(c, y);
   print y;
   await y > 8;
   goto k;
```

- ⚠ Receiving from a channel means consuming a message!
- ⚠ No printing when y <= 8!
- ⚠ No rollback mechanisms please!

* Each process may only receive messages from a single (globally unique) channel (called “mailbox”).

* Each process may only receive messages from a single (globally unique) channel (called “mailbox”).
* Each atomic block must start with all its `receive` and `await` statements.

* Each process may only receive messages from a single (globally unique) channel (called “mailbox”).
* Each atomic block must start with all its `receive` and `await` statements.
  - ✔ No more printing if the conditions are not satisfied.

* Each process may only receive messages from a single (globally unique) channel (called “mailbox”).
* Each atomic block must start with all its `receive` and `await` statements.
  - ✔ No more printing if the conditions are not satisfied.
  - ✔ No need to rollback (most) assignments.

From Guarded PlusCal to Network PlusCal

Guarded PlusCal does not avoid all rollbacks.

Guarded PlusCal does not avoid all rollbacks. Consider this example again:

```
l: receive(c, y);
   await y > 8;
   x := f();
   print y;
   goto k;
```

Guarded PlusCal does not avoid all rollbacks. Consider this example again:

```
l: receive(c, y);
   await y > 8;
   x := f();
   print y;
   goto k;
```

What if the first message in the channel c is 3?

Guarded PlusCal does not avoid all rollbacks. Consider this example again:

```
l: receive(c, y);
   await y > 8;
   x := f();
   print y;
   goto k;
```

What if the first message in the channel `c` is $\not\exists$ `10`?

Guarded PlusCal but `receive` statements are replaced with local message buffering.

Guarded PlusCal but `receive` statements are replaced with local message buffering.

- ✔ No more mandatory consuming!

Guarded PlusCal but `receive` statements are replaced with local message buffering.

- ✔ No more mandatory consuming!
- ✔ No need for rollback mechanisms anymore!

1. Compile `receive` into buffer indexing.

1. Compile `receive` into buffer indexing.

```
process (Ping = 0)
    variable tmp1 = "";
{
rcv1:   receive(ping, tmp1);
        await tmp1 = "Ping";
        goto pong;
pong:   send(pong, "Pong");
        goto rcv1;
}
```

1. Compile `receive` into buffer indexing.

```
process (Ping = 0)
    variable tmp1 = "";
{
rcv1:   receive(ping, tmp1);
        await tmp1 = "Ping";
        goto pong;
pong:   send(pong, "Pong");
        goto rcv1;
}
```

1. Compile `receive` into buffer indexing.

```
process (Ping = 0)
    variable tmp1 = "";
{
rcv1:   receive(ping, tmp1);
        await tmp1 = "Ping";
        goto pong;
pong:   send(pong, "Pong");
        goto rcv1;
}
```

$\Rightarrow \mathcal{E}$

```
process (Ping = 0)
    variables tmp1 = "", inbox = <<>>;
{
rcv1:   await Len(inbox) > 0;
        tmp1 := Head(inbox);
        inbox := Tail(inbox);
        await tmp1 = "Ping";
        goto pong;
pong:   send(pong, "Pong");
        goto rcv1;
}
{
rx†:    receive(ping, rcv†);
        inbox := Append(inbox, rcv†);
        goto rx†;
}
```

1. Compile `receive` into buffer indexing.

```
process (Ping = 0)
    variable tmp1 = "";
{
rcv1:   receive(ping, tmp1);
        await tmp1 = "Ping";
        goto pong;
pong:   send(pong, "Pong");
        goto rcv1;
}
```

$\twoheadrightarrow \mathcal{E}$

$\twoheadrightarrow \mathcal{M}$

```
process (Ping = 0)
    variables tmp1 = "", inbox = <<>>;
{
rcv1:   await Len(inbox) > 0;
        tmp1 := Head(inbox);
        inbox := Tail(inbox);
        await tmp1 = "Ping";
        goto pong;
pong:   send(pong, "Pong");
        goto rcv1;
}
{
rx†:    receive(ping, rcv†);
        inbox := Append(inbox, rcv†);
        goto rx†;
}
```

2. Reorder statements to conform to Network PlusCal syntactical restrictions.

2. Reorder statements to conform to Network PlusCal syntactical restrictions.

```
process (Ping = 0)
    variables tmp1 = "", inbox = <<>>;
{
rcv1:   await Len(inbox) > 0;
        tmp1 := Head(inbox);
        inbox := Tail(inbox);
        await tmp1 = "Ping";
        goto pong;
pong:   send(pong, "Pong");
        goto rcv1;
}
{
rx†:    receive(ping, rcv†);
        inbox := Append(inbox, rcv†);
        goto rx†;
}
```

2. Reorder statements to conform to Network PlusCal syntactical restrictions.

```
process (Ping = 0)
    variables tmp1 = "", inbox = <<>>;
{
rcv1:   await Len(inbox) > 0;
        tmp1 := Head(inbox);
        inbox := Tail(inbox);
        await tmp1 = "Ping";
        goto pong;
pong:   send(pong, "Pong");
        goto rcv1;
}
{
rx†:    receive(ping, rcv†);
        inbox := Append(inbox, rcv†);
        goto rx†;
}
```

2. Reorder statements to conform to Network PlusCal syntactical restrictions.

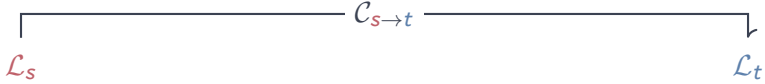
```
process (Ping = 0)
    variables tmp1 = "", inbox = <<>>;
{
rcv1:   await Len(inbox) > 0;
        tmp1 := Head(inbox);
        inbox := Tail(inbox);
        await tmp1 = "Ping";
        goto pong;
pong:   send(pong, "Pong");
        goto rcv1;
}
{
rx†:    receive(ping, rcv†);
        inbox := Append(inbox, rcv†);
        goto rx†;
}
```

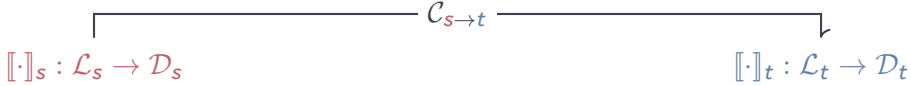
$\Rightarrow_{\mathcal{S}}$

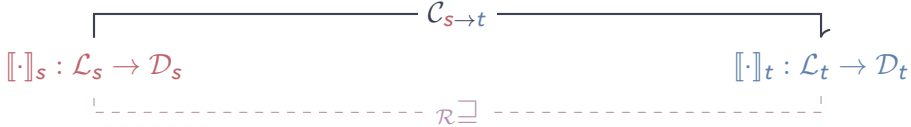
```
process (Ping = 0)
    variables tmp1 = "", inbox = <<>>;
{
rcv1:   await Len(inbox) > 0;
        await Head(inbox) = "Ping";
        tmp1 := Head(inbox);
        inbox := Tail(inbox);
        goto pong;
pong:   send(pong, "Pong");
        goto rcv1;
}
{
rx†:    receive(ping, rcv†);
        inbox := Append(inbox, rcv†);
        goto rx†;
}
```

$\mathcal{L}_s$

$\mathcal{L}_t$


$\mathcal{C}_{s \to t}$
$\mathcal{L}_s$
$\mathcal{L}_t$


$\mathcal{C}_{s \to t}$
$[\![\cdot]\!]_s : \mathcal{L}_s \to \mathcal{D}_s$
$[\![\cdot]\!]_t : \mathcal{L}_t \to \mathcal{D}_t$



Definition (Behavior justification)

For $\mathcal{R} \subseteq \mathcal{D}_s \times \mathcal{D}_t$, a behavior $\sigma_t \in [\![P_t]\!]_t$ is $\mathcal{R}$-*justified by* a behavior $\sigma_s \in [\![P_s]\!]_s$ if $(\sigma_s, \sigma_t) \in \mathcal{R}$.

Definition (Partial correctness)

$\mathcal{C}_{s \to t}$ *preserves the semantics of* $\mathcal{L}_s$ if for all programs $P \in \mathcal{L}_s$, every (terminating) behavior of $[\![\mathcal{C}_{s \to t}(P)]\!]_t$ is $\mathcal{R}$-justified by some behavior of $[\![P]\!]_s$.

Define $[\![\cdot]\!] : \text{PlusCal} \to \mathit{State} \times \mathit{Trace} \times \mathit{State}$

Define $[\![\cdot]\!] : \text{PlusCal} \to \mathit{State} \times \mathit{Trace} \times \mathit{State}$ where

* *State* is a tuple $\mathit{Memory} \times \mathit{Channels}$

Define $[\![\cdot]\!] : \text{PlusCal} \to \mathit{State} \times \mathit{Trace} \times \mathit{State}$ where

* *State* is a tuple $\mathit{Memory} \times \mathit{Channels}$
* *Trace* is a sequence of events $\mathit{out}(v)$ or $c!v$

Define $[\![\cdot]\!] : \text{PlusCal} \to \mathit{State} \times \mathit{Trace} \times \mathit{State}$ where

* *State* is a tuple *Memory* × *Channels*
* *Trace* is a sequence of events $\mathit{out}(v)$ or $c!v$

$$\frac{M \vdash e \Downarrow v \qquad \tau = \mathit{out}(v)}{\langle\langle M, C\rangle, \tau, \langle M, C\rangle\rangle \in [\![\text{print } e]\!]} \qquad \frac{M \vdash e \Downarrow \text{TRUE} \qquad \tau = \varepsilon}{\langle\langle M, C\rangle, \tau, \langle M, C\rangle\rangle \in [\![\text{await } e]\!]}$$

$$\frac{\langle\langle M, C\rangle, \tau', \langle M'', C''\rangle\rangle \in [\![S]\!] \qquad \langle\langle M'', C''\rangle, \tau'', \langle M', C'\rangle\rangle \in [\![B]\!] \qquad \tau = \tau' * \tau''}{\langle\langle M, C\rangle, \tau, \langle M', C'\rangle\rangle \in [\![S; B]\!]}$$

$$\frac{C(c) = v \bullet vs \qquad C' = C(c := vs) \qquad M' = M(x := v) \qquad \tau = \varepsilon}{\langle\langle M, C\rangle, \tau, \langle M', C'\rangle\rangle \in [\![\text{receive}(c, x)]\!]}$$

Definition ($\sim \subseteq \mathit{State} \times \mathit{State}$)

$$\langle M_G, C_G \rangle \sim \langle M_N, C_N \rangle \overset{\text{def}}{\Leftrightarrow} \bigwedge \begin{cases} \forall\, v \neq \mathit{inbox}.\ M_G(v) = M_N(v) \\ \forall\, c \neq \mathit{mailbox}.\ C_G(c) = C_N(c) \\ C_G(\mathit{mailbox}) = M_N(\mathit{inbox}) +\!\!+ C_N(\mathit{mailbox}) \end{cases}$$

Definition ($\sim \subseteq \mathit{State} \times \mathit{State}$)

$$\langle M_G, C_G \rangle \sim \langle M_N, C_N \rangle \overset{\text{def}}{\Leftrightarrow} \bigwedge \begin{cases} \forall\, v \neq \mathit{inbox}.\ M_G(v) = M_N(v) \\ \forall\, c \neq \mathit{mailbox}.\ C_G(c) = C_N(c) \\ C_G(\mathit{mailbox}) = M_N(\mathit{inbox}) +\!\!+ C_N(\mathit{mailbox}) \end{cases}$$

Theorem (Partial correctness of $\mathcal{C}_{G \to N}$)

$\forall\, P \in$ Guarded PlusCal. $[\![\mathcal{C}_{G \to N}(P)]\!] \sqsubseteq_{\sim} [\![P]\!]$

Definition ($\sim \subseteq \mathit{State} \times \mathit{State}$)

$$\langle M_G, C_G \rangle \sim \langle M_N, C_N \rangle \overset{\text{def}}{\Leftrightarrow} \bigwedge \begin{cases} \forall\, v \neq \mathit{inbox}.\ M_G(v) = M_N(v) \\ \forall\, c \neq \mathit{mailbox}.\ C_G(c) = C_N(c) \\ C_G(\mathit{mailbox}) = M_N(\mathit{inbox}) +\!\!+ C_N(\mathit{mailbox}) \end{cases}$$

Theorem (Partial correctness of $\mathcal{C}_{G \to N}$)

$\forall\, P \in$ Guarded PlusCal. $[\![\mathcal{C}_{G \to N}(P)]\!] \sqsubseteq_{\sim} [\![P]\!]$

Proof idea:

Prove partial correctness of $\mathcal{E}$, $\mathcal{S}$ and $\mathcal{M}$ separately, then compose.

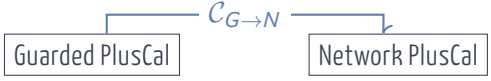
Conclusion

Conclusion

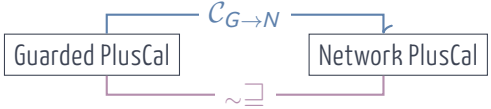
Guarded PlusCal

Network PlusCal

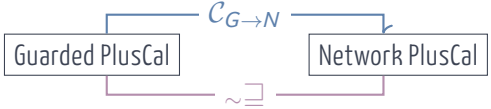
Conclusion



Conclusion

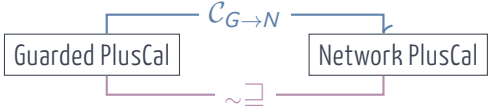

$\mathcal{C}_{G \to N}$
Guarded PlusCal
Network PlusCal
$\sim\!\sqsupseteq$

Conclusion



Fully mechanized/developed in the LEAN proof assistant/programming language!

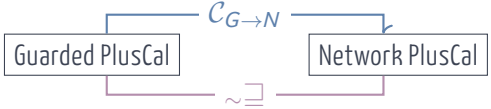
Conclusion



Fully mechanized/developed in the LEAN proof assistant/programming language!

* Totalling ~**11k** LoC (~**8k** LoC of proofs)

Conclusion

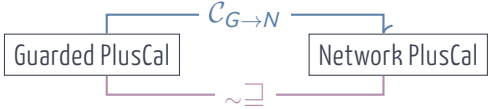


Fully mechanized/developed in the Coq proof assistant/programming language!

- Totalling ~**11k** LoC (~**8k** LoC of proofs)
- ~**400** lemmas & theorems

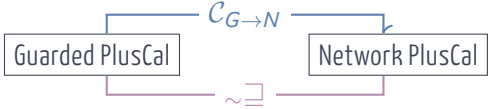
What next?

This is only the beginning!



What next?

This is only the beginning!



Some challenges remain to be (fully) solved:

What next?

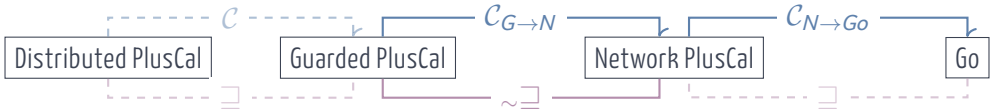
This is only the beginning!

Some challenges remain to be (fully) solved:

* Correctness of $\mathcal{C}_{N\to Go}$?

What next?

This is only the beginning!



Some challenges remain to be (fully) solved:

* Correctness of $\mathcal{C}_{N\to Go}$?
* How to relax the restrictions of Guarded PlusCal to improve on ergonomics?

Appendix

Behavior refinement $\sqsubseteq_{\mathcal{R}}$

- $[\![\cdot]\!] \subseteq \mathit{State} \times \mathit{Trace} \times \mathit{State}$: terminating (error-free) behaviors

Behavior refinement $\sqsubseteq_{\mathcal{R}}$

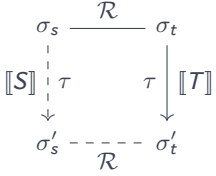
- $[\![\cdot]\!] \subseteq \mathit{State} \times \mathit{Trace} \times \mathit{State}$: terminating (error-free) behaviors
- $[\![\cdot]\!]^{\perp} \subseteq \mathit{State} \times \mathit{Trace}$: erroneous behaviors

Behavior refinement $\sqsubseteq_{\mathcal{R}}$

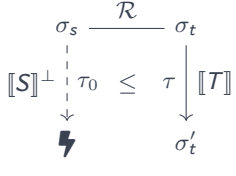
- $[\![\cdot]\!] \subseteq \mathit{State} \times \mathit{Trace} \times \mathit{State}$: terminating (error-free) behaviors
- $[\![\cdot]\!]^{\perp} \subseteq \mathit{State} \times \mathit{Trace}$: erroneous behaviors
- $|\cdot| \in \mathit{State} \to \mathbb{N}$: some measure on states

Behavior refinement $\sqsubseteq_{\mathcal{R}}$

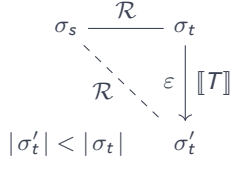
* $[\![\cdot]\!] \subseteq \mathit{State} \times \mathit{Trace} \times \mathit{State}$: terminating (error-free) behaviors
* $[\![\cdot]\!]^{\perp} \subseteq \mathit{State} \times \mathit{Trace}$: erroneous behaviors
* $|\cdot| \in \mathit{State} \to \mathbb{N}$: some measure on states



or

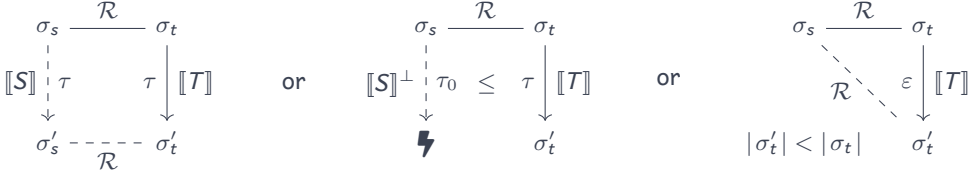


or



Behavior refinement $\sqsubseteq_{\mathcal{R}}$

* $[\![\cdot]\!] \subseteq \mathit{State} \times \mathit{Trace} \times \mathit{State}$: terminating (error-free) behaviors
* $[\![\cdot]\!]^{\perp} \subseteq \mathit{State} \times \mathit{Trace}$: erroneous behaviors
* $|\cdot| \in \mathit{State} \to \mathbb{N}$: some measure on states



Appendix

Relaxing syntactical restrictions of Guarded PlusCal

* Writing specifications in Guarded PlusCal is cumbersome.

Relaxing syntactical restrictions of Guarded PlusCal

* Writing specifications in Guarded PlusCal is cumbersome.
* We would like the full expressive power of loops and non-deterministic branching as in Distributed PlusCal.

Relaxing syntactical restrictions of Guarded PlusCal

* Writing specifications in Guarded PlusCal is cumbersome.
* We would like the full expressive power of loops and non-deterministic branching as in Distributed PlusCal.
* Sometimes it is more convenient to write assignments before `await`s.

Relaxing syntactical restrictions of Guarded PlusCal

```
* if (e) { B } else { B' } ⟹
  either { await e; B } or { await \lnot e; B' }
```

Relaxing syntactical restrictions of Guarded PlusCal

* `if (e) { B } else { B' }` $\Longrightarrow$
  `either { await e; B } or { await \lnot e; B' }`
* `l: while (e) { B }; B'` $\Longrightarrow$ `l: if (e) { B; goto l } else { B' }`

Relaxing syntactical restrictions of Guarded PlusCal

* `if (e) { B } else { B' }` $\Longrightarrow$
  `either { await e; B } or { await \lnot e; B' }`
* `l: while (e) { B }; B'` $\Longrightarrow$ `l: if (e) { B; goto l } else { B' }`
* `x := e; await e'` $\Longrightarrow$ `await e'[x\e]; x := e`

Relaxing syntactical restrictions of Guarded PlusCal

* `if (e) { B } else { B' }` $\Longrightarrow$
  `either { await e; B } or { await \lnot e; B' }`
* `l: while (e) { B }; B'` $\Longrightarrow$ `l: if (e) { B; goto l } else { B' }`
* `x := e; await e'` $\Longrightarrow$ `await e'[x\e]; x := e`
* `x := e; receive(c, x)` $\Longrightarrow$ `receive(c, x)`

Relaxing syntactical restrictions of Guarded PlusCal

* `if (e) { B } else { B' }` $\Longrightarrow$
  `either { await e; B } or { await \lnot e; B' }`
* `l: while (e) { B }; B'` $\Longrightarrow$ `l: if (e) { B; goto l } else { B' }`
* `x := e; await e'` $\Longrightarrow$ `await e'[x\e]; x := e`
* `x := e; receive(c, x)` $\Longrightarrow$ `receive(c, x)`
* `x := e; receive(c, y)` $\Longrightarrow$ `receive(c, y); x := e`

Relaxing syntactical restrictions of Guarded PlusCal

* `if (e) { B } else { B' }` $\Longrightarrow$
  `either { await e; B } or { await \lnot e; B' }`
* `l: while (e) { B }; B'` $\Longrightarrow$ `l: if (e) { B; goto l } else { B' }`
* `x := e; await e'` $\Longrightarrow$ `await e'[x\e]; x := e`
* `x := e; receive(c, x)` $\Longrightarrow$ `receive(c, x)`
* `x := e; receive(c, y)` $\Longrightarrow$ `receive(c, y); x := e`
* `print e; await e'` $\Longrightarrow$ `await e'; print e`

Relaxing syntactical restrictions of Guarded PlusCal

* `if (e) { B } else { B' }` $\Longrightarrow$
  `either { await e; B } or { await \lnot e; B' }`
* `l: while (e) { B }; B'` $\Longrightarrow$ `l: if (e) { B; goto l } else { B' }`
* `x := e; await e'` $\Longrightarrow$ `await e'[x\e]; x := e`
* `x := e; receive(c, x)` $\Longrightarrow$ `receive(c, x)`
* `x := e; receive(c, y)` $\Longrightarrow$ `receive(c, y); x := e`
* `print e; await e'` $\Longrightarrow$ `await e'; print e`
* `either ... or { B; either B1 or ... or Bn; B' } or ...` $\Longrightarrow$
  `either ... or { B; B1; B' } or ... or { B; Bn; B' } or ...`

Relaxing syntactical restrictions of Guarded PlusCal

* `if (e) { B } else { B' }` $\Longrightarrow$
  `either { await e; B } or { await \lnot e; B' }`
* `l: while (e) { B }; B'` $\Longrightarrow$ `l: if (e) { B; goto l } else { B' }`
* `x := e; await e'` $\Longrightarrow$ `await e'[x\e]; x := e`
* `x := e; receive(c, x)` $\Longrightarrow$ `receive(c, x)`
* `x := e; receive(c, y)` $\Longrightarrow$ `receive(c, y); x := e`
* `print e; await e'` $\Longrightarrow$ `await e'; print e`
* `either ... or { B; either B1 or ... or Bn; B' } or ...` $\Longrightarrow$
  `either ... or { B; B1; B' } or ... or { B; Bn; B' } or ...`

Appendix