A Brief Overview of Process Calculi

SCCS Research Colloquium

Clément Aubert

Augusta University, School of Computer & Cyber Sciences, GA, USA

Feb. 16, 2024, Augusta, GA, USA

My goals today

In a nutshell

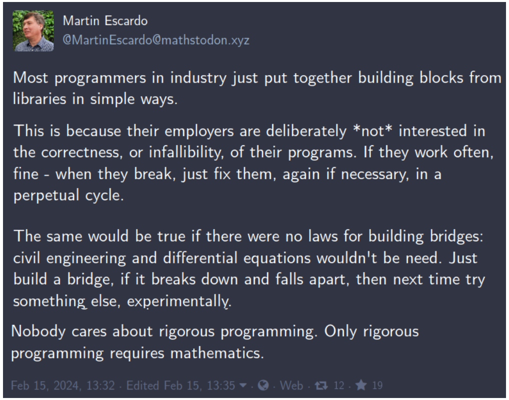
— Explain what my field (formal methods) is,

My goals today

In a nutshell

- Explain what my field (formal methods) is,
- Illustrate (some of) the impacts of process calculi

Why are formal methods ignored?



Martin Escardo
@MartinEscardo@mathstodon.xyz

Most programmers in industry just put together building blocks from libraries in simple ways.

This is because their employers are deliberately *not* interested in the correctness, or infallibility, of their programs. If they work often, fine - when they break, just fix them, again if necessary, in a perpetual cycle.

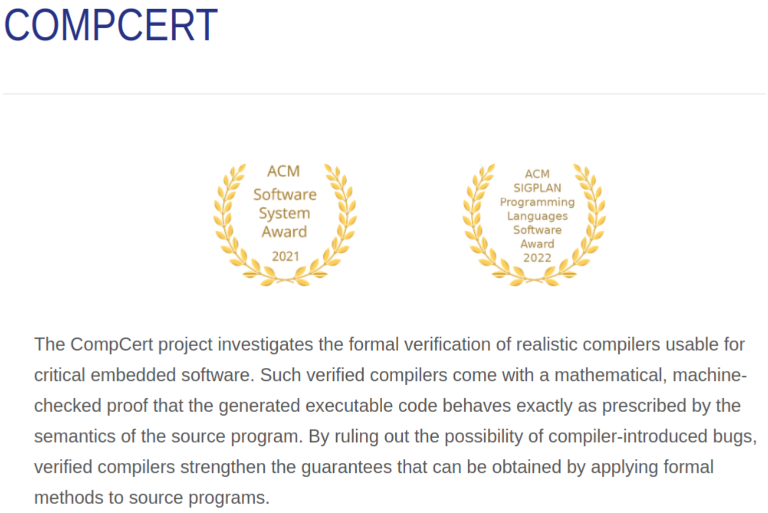
The same would be true if there were no laws for building bridges: civil engineering and differential equations wouldn't be need. Just build a bridge, if it breaks down and falls apart, then next time try something else, experimentally.

Nobody cares about rigorous programming. Only rigorous programming requires mathematics.

Feb 15, 2024, 13:32 · Edited Feb 15, 13:35 · Web · 12 · 19

https://mathstodon.xyz/@MartinEscardo/111936926330740626

What can formal methods bring? – The example of Compcert



COMPCERT

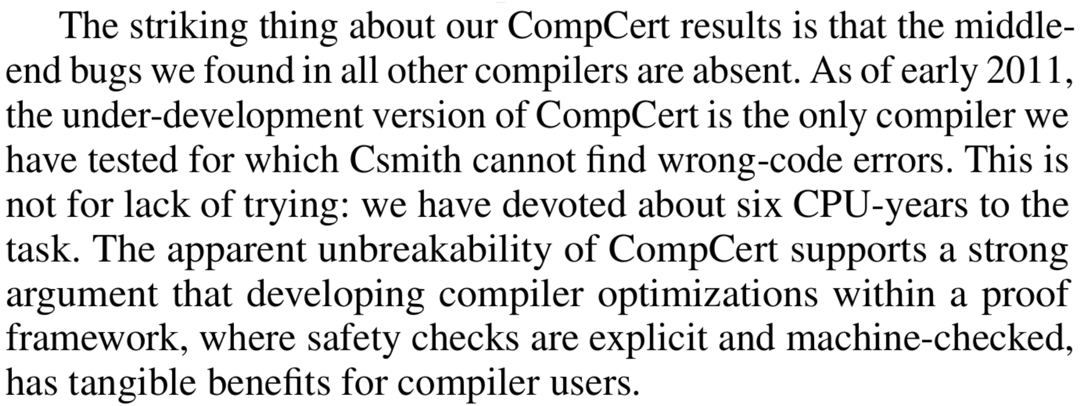
ACM Software System Award 2021

ACM SIGPLAN Programming Languages Software Award 2022

The CompCert project investigates the formal verification of realistic compilers usable for critical embedded software. Such verified compilers come with a mathematical, machine-checked proof that the generated executable code behaves exactly as prescribed by the semantics of the source program. By ruling out the possibility of compiler-introduced bugs, verified compilers strengthen the guarantees that can be obtained by applying formal methods to source programs.

https://compcert.org/

What can formal methods bring? – The example of Compcert



The striking thing about our CompCert results is that the middle-end bugs we found in all other compilers are absent. As of early 2011, the under-development version of CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors. This is not for lack of trying: we have devoted about six CPU-years to the task. The apparent unbreakability of CompCert supports a strong argument that developing compiler optimizations within a proof framework, where safety checks are explicit and machine-checked, has tangible benefits for compiler users.

X. Yang, Y. Chen, E. Eide, and J. Regehr. 2011. Finding and understanding bugs in C compilers. PLDI ’11, 10.1145/1993498.1993532

Formal Methods. . .

— . . . is a vast field,

What are formal methods?

Formal Methods. . .

— . . . is a vast field,

— . . . uses different mathematical techniques (type theory, modern algebra, proof techniques, . . .),

What are formal methods?

Formal Methods. . .

— . . . is a vast field,

— . . . uses different mathematical techniques (type theory, modern algebra, proof techniques, . . .),

— . . . takes time (Compcert uses the “calculus of constructions”, created in 1984).

What are formal methods?

Formal Methods. . .

- . . . is a vast field,
- . . . uses different mathematical techniques (type theory, modern algebra, proof techniques, . . .),
- . . . takes time (Compcert uses the “calculus of constructions”, created in 1984).

Our focus today

will be on the calculus of communicating systems (CCS), introduced by Robin Milner around 1980.

CCS

$P, Q := \ 0$ (*Inactive*) $\alpha.P$ (*Prefix*) $P\backslash\alpha$ (*Restriction*)

CCS

$P, Q := \quad 0 \qquad$ (*Inactive*) $\qquad \alpha.P \qquad$ (*Prefix*) $\qquad P\backslash\alpha \qquad$ (*Restriction*)

Action and Restriction

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}\ \text{act.} \qquad\qquad \alpha \notin \{a, \overline{a}\}\ \frac{P \xrightarrow{\alpha} P'}{P\backslash a \xrightarrow{\alpha} P'\backslash a}\ \text{res.}$$

Calculus of Communicating Systems – Presentation

CCS

$P, Q := 0$ (*Inactive*) $\quad \alpha.P$ (*Prefix*) $\quad P\backslash\alpha$ (*Restriction*)

$P \mid Q$ (*Parallel*)

Action and Restriction

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}\ \text{act.} \qquad \alpha \notin \{a, \overline{a}\}\ \frac{P \xrightarrow{\alpha} P'}{P\backslash a \xrightarrow{\alpha} P'\backslash a}\ \text{res.}$$

Parallel Group

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ |_{\mathrm{L}} \qquad \frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\overline{\lambda}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \text{syn.} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}\ |_{\mathrm{R}}$$

Calculus of Communicating Systems – Presentation

CCS

$P, Q := 0$	(*Inactive*)	$\alpha.P$	(*Prefix*)	$P\backslash\alpha$	(*Restriction*)
$P \mid Q$	(*Parallel*)	$P + Q$	(*Sum*)		

Action and Restriction

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}\ \text{act.} \qquad \alpha \notin \{a, \overline{a}\}\ \frac{P \xrightarrow{\alpha} P'}{P\backslash a \xrightarrow{\alpha} P'\backslash a}\ \text{res.}$$

Parallel Group

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ |_{\mathrm{L}} \qquad \frac{P \xrightarrow{\lambda} P' \qquad Q \xrightarrow{\overline{\lambda}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \text{syn.} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}\ |_{\mathrm{R}}$$

Sum Group

$$\frac{P \xrightarrow{\alpha} P'}{Q + P \xrightarrow{\alpha} P'}\ +_{\mathrm{L}} \qquad \frac{Q \xrightarrow{\alpha} Q'}{Q + P \xrightarrow{\alpha} Q'}\ +_{\mathrm{R}}$$

Calculus of Communicating Systems – Presentation

CCS

$$P, Q := 0 \quad (\textit{Inactive}) \qquad \alpha.P \quad (\textit{Prefix}) \qquad P\backslash\alpha \quad (\textit{Restriction})$$
$$P \mid Q \quad (\textit{Parallel}) \qquad P + Q \quad (\textit{Sum}) \qquad !P \quad (\textit{Replication})$$

Action and Restriction

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}\ \text{act.} \qquad \alpha \notin \{a, \overline{a}\}\ \frac{P \xrightarrow{\alpha} P'}{P\backslash a \xrightarrow{\alpha} P'\backslash a}\ \text{res.}$$

Parallel Group

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ |_{\mathrm{L}} \qquad \frac{P \xrightarrow{\lambda} P' \qquad Q \xrightarrow{\overline{\lambda}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \text{syn.} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}\ |_{\mathrm{R}}$$

Sum Group

$$\frac{P \xrightarrow{\alpha} P'}{Q + P \xrightarrow{\alpha} P'}\ +_{\mathrm{L}} \qquad \frac{Q \xrightarrow{\alpha} Q'}{Q + P \xrightarrow{\alpha} Q'}\ +_{\mathrm{R}}$$

CCS

CCS $\longrightarrow$ π-Calculi

— π-Calculi adds *name-passing abilities* (messages can contain channel name),

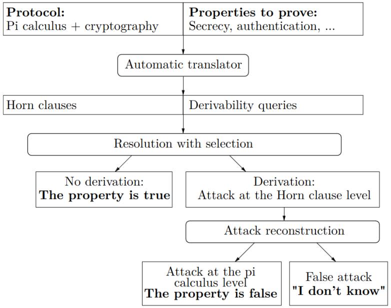
Calculus of Communicating Systems – Applications

CCS $\longrightarrow$ π-Calculi $\longrightarrow$ Applied π-Calculi

- π-Calculi adds *name-passing abilities* (messages can contain channel name),
- *Applied* π-Calculi adds *aliases* (messages can contain encrypted information),

CCS $\longrightarrow$ π-Calculi $\longrightarrow$ Applied π-Calculi $\longrightarrow$ ProVerif

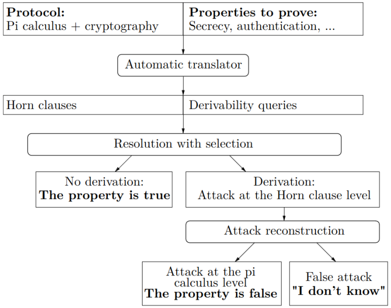
- π-Calculi adds *name-passing abilities* (messages can contain channel name),
- *Applied* π-Calculi adds *aliases* (messages can contain encrypted information),
- ProVerif is an implementation:



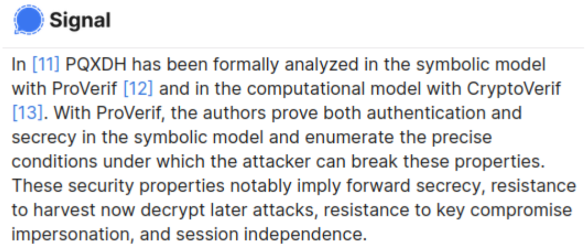
Calculus of Communicating Systems – Applications

CCS $\longrightarrow$ π-Calculi $\longrightarrow$ Applied π-Calculi $\longrightarrow$ ProVerif $\longrightarrow$, ,

- π-Calculi adds *name-passing abilities* (messages can contain channel name),
- *Applied* π-Calculi adds *aliases* (messages can contain encrypted information),
- ProVerif is an implementation:



Protocol: Pi calculus + cryptography
Properties to prove: Secrecy, authentication, ...
Automatic translator
Horn clauses
Derivability queries
Resolution with selection
No derivation: **The property is true**
Derivation: Attack at the Horn clause level
Attack reconstruction
Attack at the pi calculus level **The property is false**
False attack **"I don't know"**



Signal

In [11] PQXDH has been formally analyzed in the symbolic model with ProVerif [12] and in the computational model with CryptoVerif [13]. With ProVerif, the authors prove both authentication and secrecy in the symbolic model and enumerate the precise conditions under which the attacker can break these properties. These security properties notably imply forward secrecy, resistance to harvest now decrypt later attacks, resistance to key compromise impersonation, and session independence.

CCS is also connected to . . .

Petri nets, biological systems, session types, choregraphies, category theory, . . .

Conclusion: my work & thanks!

CCS is also connected to . . .

Petri nets, biological systems, session types, choregraphies, category theory, . . .

My contributions revolve around

— “Better” applied π-calculus,

Conclusion: my work & thanks!

CCS is also connected to . . .

Petri nets, biological systems, session types, choregraphies, category theory, . . .

My contributions revolve around

— “Better” applied π-calculus,

— *Reversible* CCS,

Conclusion: my work & thanks!

CCS is also connected to . . .

Petri nets, biological systems, session types, choregraphies, category theory, . . .

My contributions revolve around

— "Better" applied π-calculus,

— *Reversible* CCS,

— (and others in the field of formal method, but not related to process calculi – ask Neea Rusch about those!)

Thanks!

Feel free to reach out to

`caubert@augusta.edu`