School of Computer Science & Engineering

Trustworthy Systems Group

Mechanising Local Rely-Guarantee Reasoning over the seL4 Trace Monad

For verifying seL4 Microkit based systems

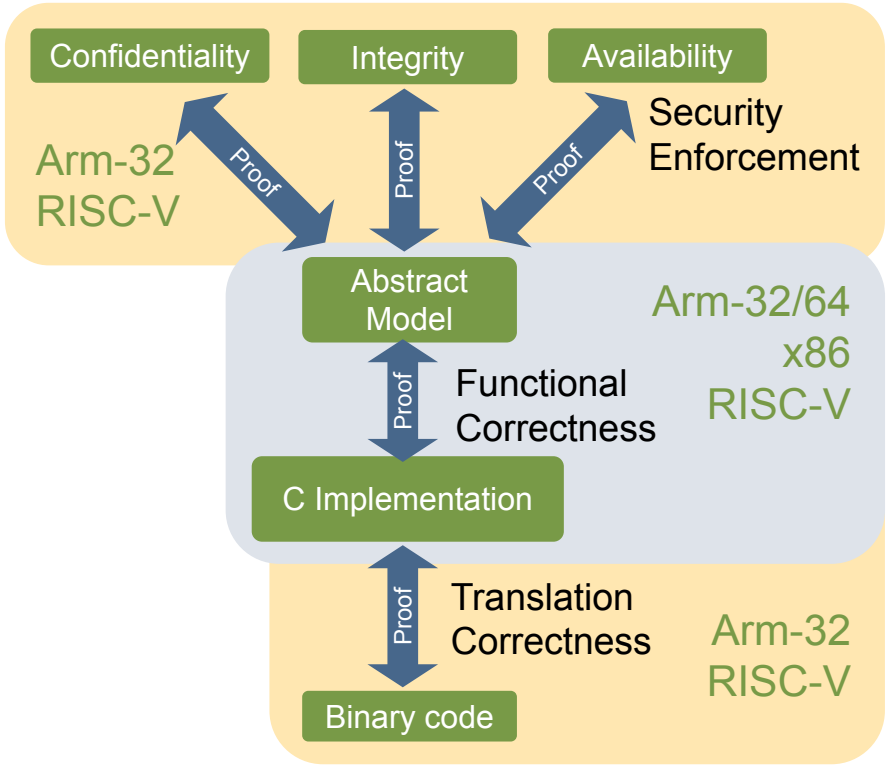
Junming Zhao
junming.zhao@unsw.edu.au

PhD Student
Supervisors: Dr. Rob Sison, Dr. Thomas Sewell, Dr. Miki Tanaka
Trustworthy Systems, UNSW Sydney

seL4 Microkernel

- World's first OS kernel with correctness proof
- Security properties based on functional correctness
- Binary-level Verification
- Fastest microkernel

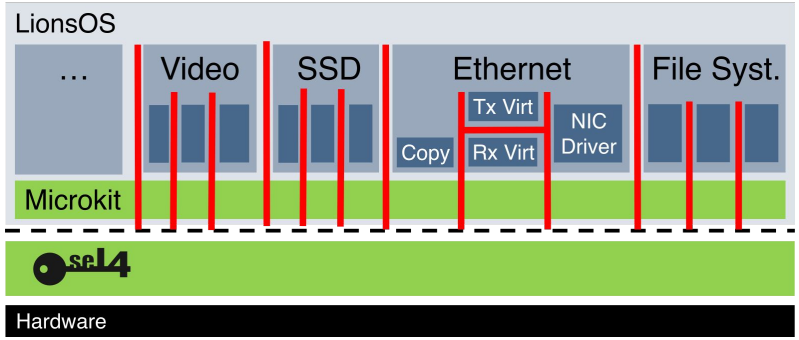
Open Source



Motivation: Verifying seL4 Microkit based systems

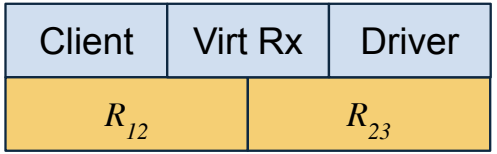
Overview:

- Verified seL4 + Microkit
- User-level PDs: concurrent
- Need a composition rule for PDs interference



Microkit static structure:

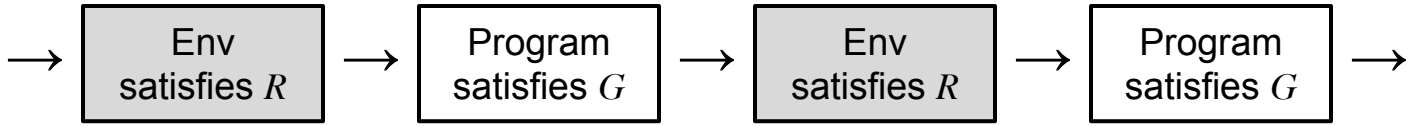
- Fixed PDs
- Fixed memory regions and async channels



Proof goals:

- Prove each component locally
- Frame it into the system state, then finally
- Compose one global property.

Rely-Guarantee



Rely-Guarantee — $\{P\}\ \{R\}\ f\ \{G\}\ \{Q\}$

- **Pre-condition** (P): Initial state
- **Post-condition** (Q): Final state
- **Rely** (R): Environmental steps
- **Guarantee** (G): Component's program steps

Parallel composition checks compatibility:

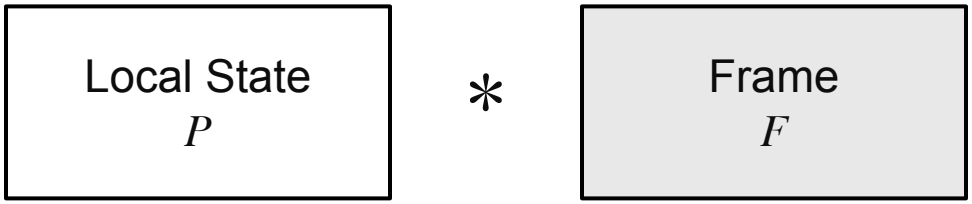
$$\{P_1\}\ \{R_1\}\ f_1\ \{G_1\}\ \{Q_1\}$$

$$\{P_2\}\ \{R_2\}\ f_2\ \{G_2\}\ \{Q_2\}$$

$$G_2 \Rightarrow R_1 \qquad G_1 \Rightarrow R_2$$

$$\{P_1 \wedge P_2\}\ \{R_1 \wedge R_2\}\ f_1 \parallel f_2\ \{G_1 \vee G_2\}\ \{Q_1 \wedge Q_2\}$$

Separation Logic



$(P * F)(s)$ iff s splits into disjoint sP and sF,
with $P(sP)$ and $F(sF)$

$$\frac{\{P\}\ f\ \{Q\} \quad f \text{ does not touch } F}{\{P * F\}\ f\ \{Q * F\}}$$

Separation Logic

$$\frac{\{P\}\ \{R\}\ f\ \{G\}\ \{Q\} \quad f \text{ does not “touch” } F}{\{P*F\}\ \{R*F\}\ f\ \{G*F\}\ \{Q*F\}}$$

Local rely-guarantee reasoning

Author:  Xinyu Feng | Authors Info & Claims

POPL '09: Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages
Pages 315 - 327 • https://doi.org/10.1145/1480881.1480922

Trace-Monad Semantics

$$f : \Sigma \to \mathscr{P}(\text{Trace} \times \text{Out}(\alpha))$$

Given $\sigma \in \Sigma$, $f\,\sigma \subseteq \text{Trace} \times \text{Out}(\alpha)$

$$\text{Trace} = (\text{Id} \times \Sigma)^*$$

$$\text{Id} = \{\texttt{Me}, \texttt{Env}\}$$

$$\text{Out}(\alpha) = \{\ \texttt{Failed}, \texttt{Incomplete}, \text{Result}\ (\alpha \times \Sigma)\}$$

R filters Env-labelled steps; G filters Me-labelled steps.

Trace-Monad Semantics

Example Primitives

```
return v = λs.{([], Result(v,s))}
  -- returns v, emits no trace step

modify f = λs.{([], Result((),f s))}
  -- applies modification f to current
state, emits no trace step

commit_step = λs.{([], Incomplete),
([(Me, s)], Result((),s))}
  -- emits one Me step

modify_commit f = modify f; commit_step
  -- one committed Me-labelled update
```

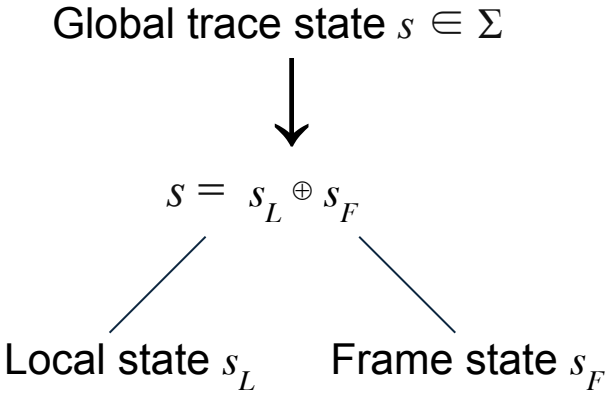
Example Program

```
step : c_state => c_state
step σ =
  σ(count := count σ + 1)

prog =
  whileLoop T
    (λ_. modify_commit step) ()
```

SL Over Trace Monad

Trace monad $f : \Sigma \to \mathcal{P}(\text{Trace} \times \text{Out}(\alpha))$



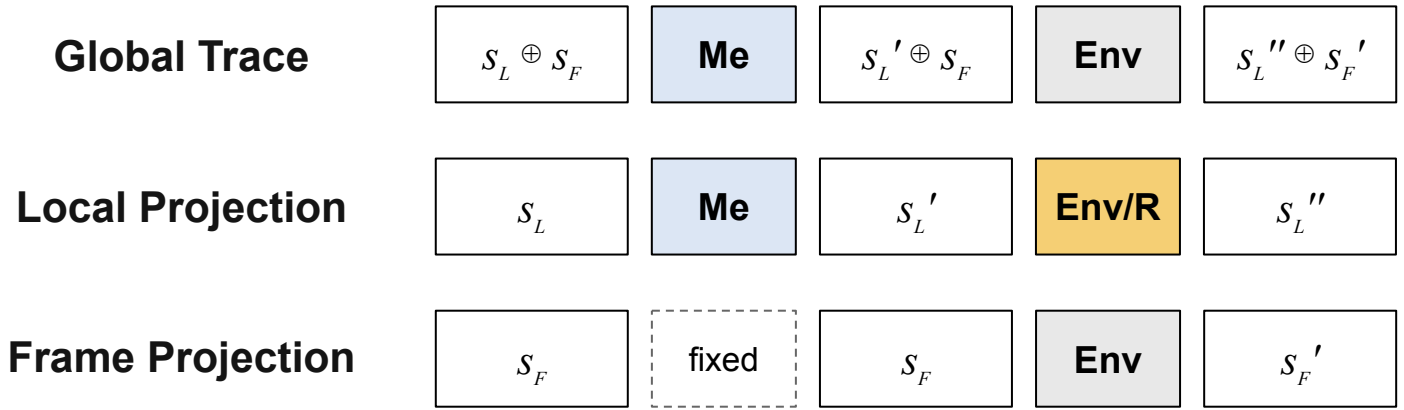
Σ has:

ε	empty
$s_1 \# \# s_2$	disjoint
$s_1 \oplus s_2$	merge

Same trace semantics; each state can now be viewed as local $\oplus$ frame

Trace Splitting

A global run is split into a local projection and a frame projection.



Separating and Fencing Actions

Action separating conjunction $*_A$

$$\frac{s = s_L \oplus s_F \qquad s' = s'_L \oplus s'_F \qquad R_L\ \ s_L\ s'_L \qquad R_F\ \ s_F\ s'_F}{(R_L *_A R_F)\ (s, s')}$$

$*_A$ splits a transition into local + frame steps

Fenced Action $\triangleright$

$$\frac{I \text{ precise} \qquad R\ s\ s' \Rightarrow I\ s \wedge I\ s' \qquad I\ s \Rightarrow R\ s\ s}{I \triangleright R}$$

$\triangleright$ makes “acts only on this resource” precise

RG Frame Rule

$$\frac{\{P\}\ \{R\}\ f\ \{G\}\ \{Q\}\quad \text{locality}(I, R, R', F, f) \qquad I \triangleright R \quad I' \triangleright R' \quad \text{sta}(F, R') \quad P \le I \quad F \le I'}{\{P * F\} \qquad \{R *_{\text{A}} R'\}\ f\ \{G *_{\text{A}} G'\}\ \{Q * F\}}$$

Locality: f doesn't "touch" other resources

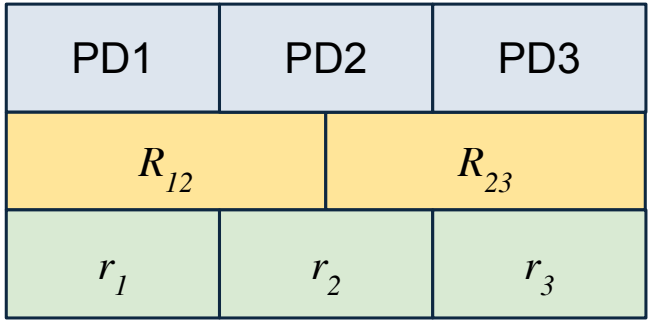
Every allowed global run of f from $s_L \oplus s_F$ (allowed by the global rely $R *_{\text{A}} R'$) splits into:

- a local run of f from s_L (allowed by R)
- a frame trace preserving F (allowed by R')

Case Study: 3-PD Microkit Pipeline

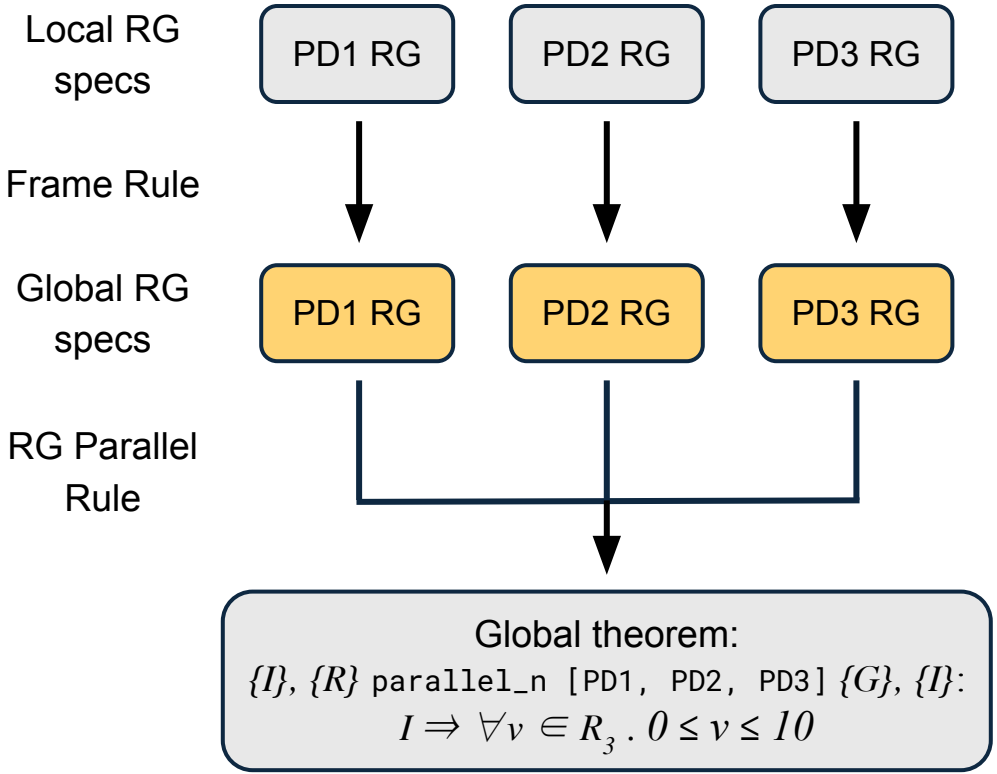
PD1: producer - produces integers smaller than 10
PD2: filter - forwards only $v \geq 0$
PD3: consumer - appends to private log

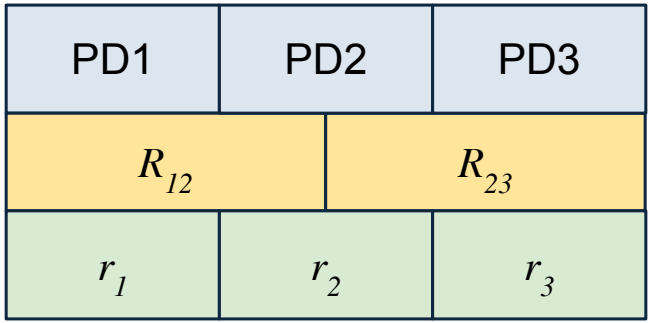
R_{12} mailbox: int ≤ 10
R_{23} mailbox: int ≥ 0



Goal: $\forall v \in r_3 \,.\, 0 \leq v \leq 10$

Case Study: 3-PD Microkit Pipeline





Conclusion & Future Work

Conclusion

- Mechanised LRG for seL4 non-deterministic trace monad semantics
- Minimalistic logic framework for Microkit purposes
- Establish verification story to seL4 microkit based system

Future Work

- Connect local RG proof to ATP like Viper
- Clearer interface template & proof obligations for user
- Nested parallel combinator
- More realistic Microkit Examples

Thank You!
Questions?