A Generic Process Calculus Approach to Relaxed-Memory Consistency

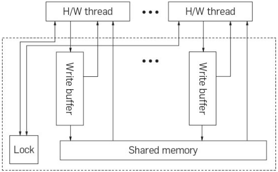
Palle Raabjerg Tjark Weber Nadine Rohde
Lars-Henrik Eriksson Joachim Parrow

UPMARC Workshop on Memory Models (MM'15)
23–24 February 2015, Uppsala, Sweden

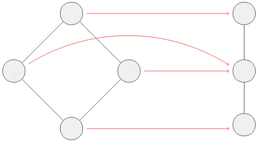
Motivation

Two common ways to specify memory models formally:

1. Operational (events and guards)

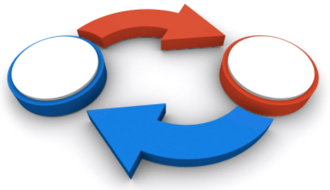


2. Axiomatic (relational logic)



Goal

These different styles make it difficult to apply verification techniques across different models.



We want to specify relaxed consistency models and reason about the behavior of programs in a unified framework that supports formal verification for a range of specification styles.

Approach

Psi calculi	$=$	**process calculi**	$+$	**logics**
		(e.g., pi-calculus)		(e.g., first-order logic)

Psi Calculi

Process Calculi

A process calculus is a language to formally model concurrent systems.

Example (pi-calculus)

$\overline{a}\langle b, c\rangle.\mathbf{0} \mid a(x, y).x(u).\overline{y}u.\mathbf{0}$

- Values: only names of communication channels
- Very expressive, but very basic (cf. lambda calculus)

Process calculi have precise formal semantics.

Process Calculi

A process calculus is a language to formally model concurrent systems.

Example (pi-calculus)

$\overline{a}\langle b, c\rangle.\mathbf{0} \mid a(x, y).x(u).\overline{y}u.\mathbf{0} \quad \xrightarrow{\tau} \quad b(u).\overline{c}u.\mathbf{0}$

- Values: only names of communication channels
- Very expressive, but very basic (cf. lambda calculus)

Process calculi have precise formal semantics.

Process Calculi (cont.)

Pick an existing calculus, adapt it for your application.

Examples:

- pi-calculus + cryptography
- pi-calculus + broadcast communication
- pi-calculus + XML data
- pi-calculus + constraints

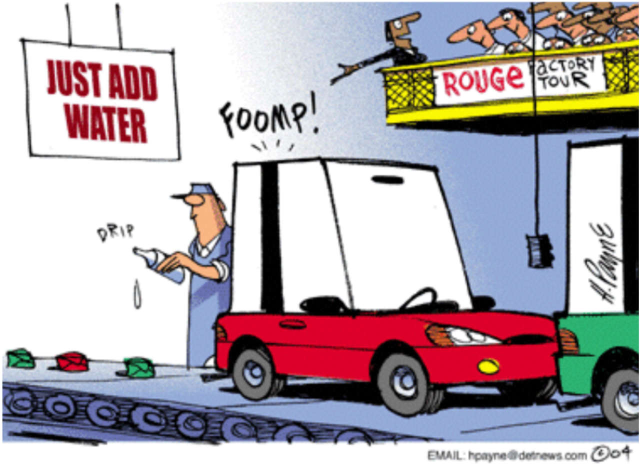
Process Calculi (cntd.)

Repeating the procedure leads to a multitude of slightly different calculi:

Nice for modeling, but how reliable is the analysis?

A Generic Framework for Applied Process Calculi

The Psi calculi framework is a factory for applied calculi: just add data and logics.



UPMARC

A Generic Framework for Applied Process Calculi (cont.)

Instantiating the Psi calculi framework yields

- "pi-calculus extensions" with many nice features (complex channels, arbitrary data structures, broadcast, higher-order, . . .),
- compositional and straight-forward theory (semantics, process equivalence, types), machine-checked in Nominal Isabelle,
- tools for simulation and equivalence checking.

Cooking a Psi Calculus

Three sets: terms Σ, conditions $\mathbf{C}$, assertions $\mathbf{A}$

Substitution on these sets

Four operators:

$$
\begin{array}{ll}
\dot{\leftrightarrow} : \Sigma \times \Sigma \rightarrow \mathbf{C} & \text{(channel equivalence)} \\
\otimes : \mathbf{A} \times \mathbf{A} \rightarrow \mathbf{A} & \text{(composition)} \\
\mathbf{1} : \mathbf{A} & \text{(unit assertion)} \\
\vdash \subseteq \mathbf{A} \times \mathbf{C} & \text{(entailment)}
\end{array}
$$

Psi Calculi Semantics

From these ingredients, we obtain a process calculus: a data type of processes equipped with a structured operational semantics.

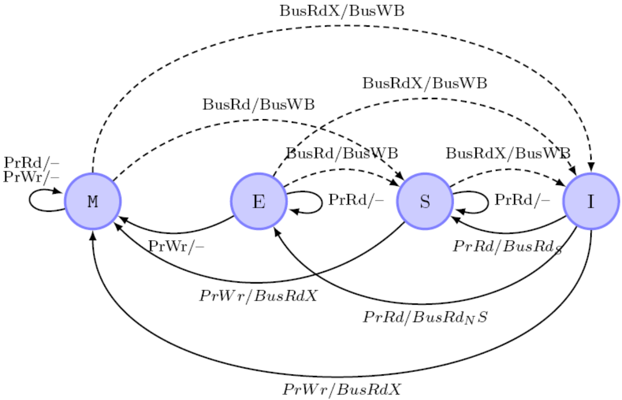
$$\Psi \rhd P \xrightarrow{\alpha} P'$$

"In the environment Ψ, process P can take action α to become P'."

A Psi-Calculi Instance for MESI

The MESI Protocol

MESI is a widely used coherence protocol that supports write-back cache.



MESI Terms

Constants:

$$
\begin{array}{rcll}
c & ::= & \mathbf{M} \mid \mathbf{E} \mid \mathbf{S} \mid \mathbf{I} & \text{(cache line states)} \\
& \mid & \mathbf{read} \mid \mathbf{write} & \text{(CPU requests)} \\
& \mid & \mathbf{bus} & \text{(the shared bus)} \\
& \mid & \mathbf{READ} \mid \mathbf{RWITM} \mid \mathbf{INV} \mid \mathbf{DATA} & \text{(bus communication)} \\
& \mid & \mathbf{memory} & \text{(memory or LLC)} \\
& \mid & 0 \mid 1 & \text{(data values)}
\end{array}
$$

Terms:

$$
\begin{array}{rcll}
S, T & ::= & c & \text{(any constant is a term)} \\
& \mid & x & \text{(any name is a term)} \\
& \mid & (T_1, \ldots, T_n) & \text{(tuples of terms are terms)}
\end{array}
$$

MESI Conditions

φ, ψ	$::=$	$T \mapsto s$	(state of controller $T \in \Sigma$ is s)
	$\mid$	$T \mapsto b$	(data held by controller $T \in \Sigma$ is b)
	$\mid$	M	(some controller is in modified state)
	$\mid$	ES	(some controller is in exclusive or shared state)
	$\mid$	I	(all controllers are in invalid state)
	$\mid$	$S = T$	(equality of terms)
	$\mid$	$\neg\varphi$	(negation)
	$\mid$	$\varphi \wedge \psi$	(conjunction)

MESI Assertions

Let $\mathbb{S} := \{\mathbf{M}, \mathbf{E}, \mathbf{S}, \mathbf{I}\}$ be the set of cache line states, and $\mathbb{B} := \{0, 1\}$ be the set of data values.

Assertions **A**:

$\Psi = (\Psi_{\mathbb{S}}, \Psi_{\mathbb{B}})$ is a pair of functions, where $\Psi_{\mathbb{S}} \colon \Sigma \to \mathbb{S}$ and $\Psi_{\mathbb{B}} \colon \Sigma \to \mathbb{B}$.

The MESI Psi-Calculus

Theorem

MESI *terms, conditions and assertions, together with suitable definitions for entailment and composition, satisfy the requirements on a Psi calculi instance.*

Thus by instantiating the Psi calculi framework with the above parameters, we obtain a process calculus.

Modeling and Verifying the MESI Protocol

We have modeled the MESI protocol as a process in this calculus (extended with broadcast communication and priorities).

1. High-level abstract model, where cache controllers directly transition between states
2. Low-level model, where bus communication is accounted for

We have verified (via a simulation proof) that both models provide sequential consistency.

Conclusions and Future Work

Achieved so far:

- Simple consistency models and coherence protocols implemented in Psi calculi
- Models at different levels of abstraction
- Tools for bisimulation checking in Psi calculi

Expected:

- Realistic memory models (e.g., x86-TSO, Power, C++) implemented in Psi calculi
- Hennessy-Milner style dynamic logics for Psi calculi
- Tools for model checking in Psi calculi

UPMARC