Automating Memory Model Metatheory with Intersections

Aristotelis Koutsouridis[1] Michalis Kokologiannakis[2]
Viktor Vafeiadis[1]

[1]MPI-SWS [2]ETH Zurich

September 13, 2024
CONCUR 2024

Reasoning about concurrent programs is hard

Concurrent programs have many outcomes since threads are scheduled non-deterministically

$$
\begin{array}{l||l}
a := x & b := y \\
y := 1 & x := 1
\end{array}
$$

Correctness proofs are hard

How do we prove the correctness of **program transformations**:

$$
\begin{array}{l}
a := x \\
y := 1
\end{array}
\;\middle\|\;
\begin{array}{l}
b := y \\
x := 1
\end{array}
$$

Correctness proofs are hard

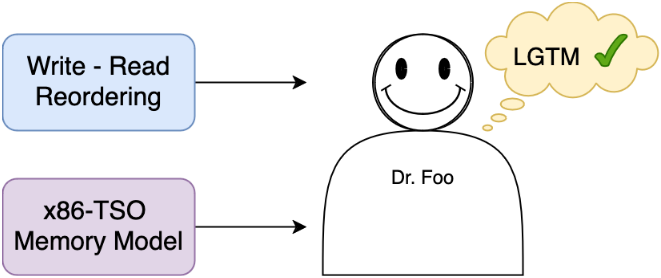
How do we prove the correctness of **program transformations**:

- **compiler optimizations**: Can we reorder $a := x$ with $y := 1$ without altering the overall program behavior?

$$\begin{array}{l} a := x \\ y := 1 \end{array} \;\Bigg\|\; \begin{array}{l} b := y \\ x := 1 \end{array}$$

Correctness proofs

Traditionally, experts would prove the correctness of a specific program transformation for a particular system, e.g:

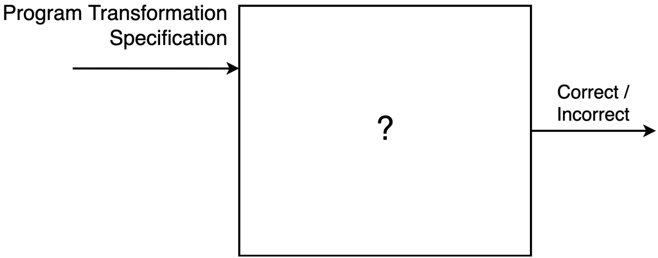


Automating correctness proofs

Instead, design an algorithm parametric on the system and transformation specs:

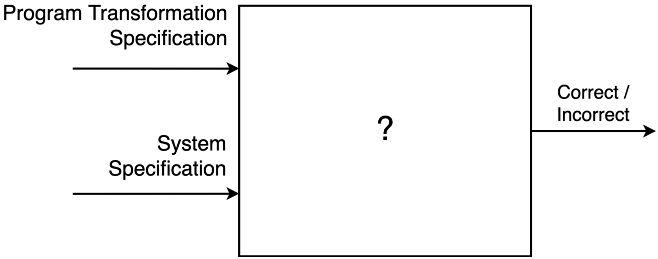
Automating correctness proofs

Instead, design an algorithm parametric on the system and transformation specs:



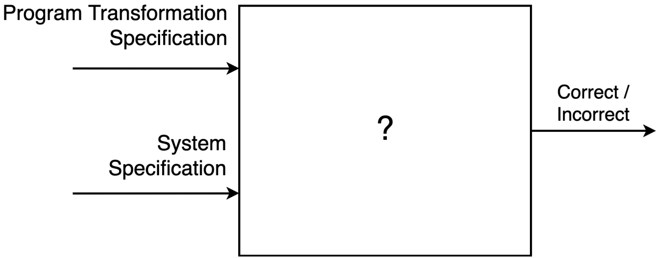
Automating correctness proofs

Instead, design an algorithm parametric on the system and transformation specs:



Automating correctness proofs

Instead, design an algorithm parametric on the system and transformation specs:



System Specification = Memory Model

Memory Consistency Models

Memory models describe a system by **restricting** *the set of program outcomes*

Load buffering (LB)

$$
\begin{array}{l|l}
a := x & b := y \\
y := 1 & x := 1
\end{array}
$$

Outcome: $a = b = 1$

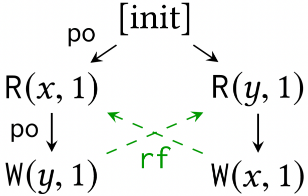
Memory Consistency Models

Memory models describe a system by ***restricting*** *the set of program outcomes*

Load buffering (LB)

$$\begin{array}{c||c} a := x & b := y \\ y := 1 & x := 1 \end{array}$$

Outcome: $a = b = 1$



Memory Consistency Models

Memory models describe a system by ***restricting*** *the set of program outcomes*

Load buffering (LB)

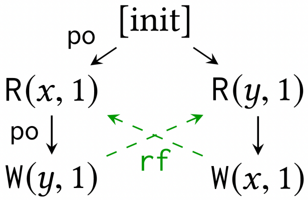
$$\begin{array}{c||c} a := x & b := y \\ y := 1 & x := 1 \end{array}$$

Outcome: $a = b = 1$

Is the graph allowed by the memory model?

✓ *Armv8*

✗ *x86 TSO*



Memory Consistency Models

Memory models describe a system by **restricting** *the set of program outcomes*

Load buffering (LB)

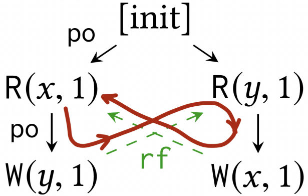
$$\begin{array}{r|l} a := x & b := y \\ y := 1 & x := 1 \end{array}$$

Outcome: $a = b = 1$

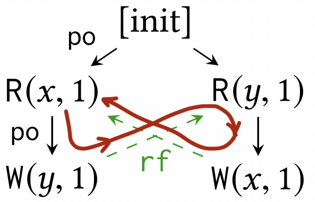
Is the graph allowed by the memory model?

✓ *Armv8*

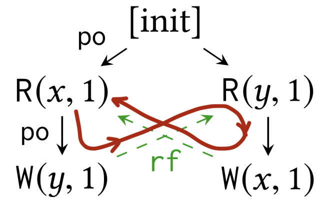
✗ *x86 TSO*



Expressing Constraints using Relational Algebra

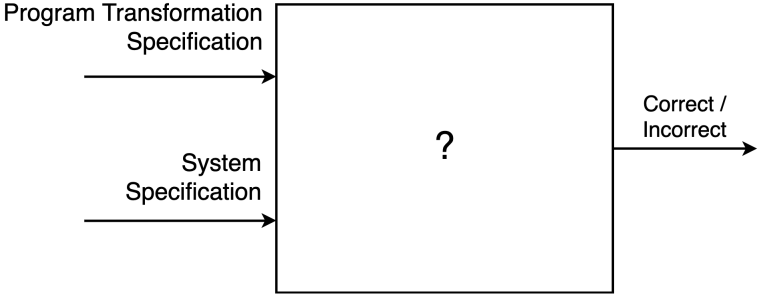


Expressing Constraints using Relational Algebra



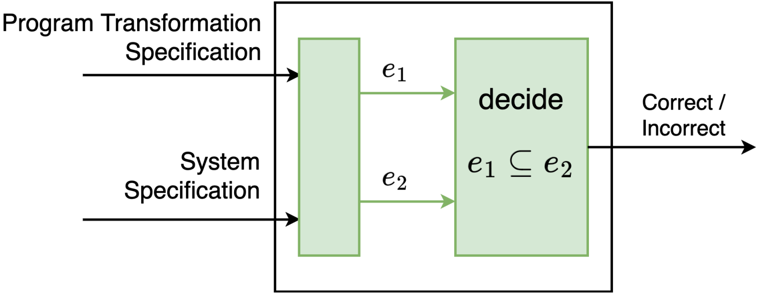
memory model constraint: acyclic($\mathbf{e}$), $\quad \mathbf{e} := \mathsf{po} \cup \mathsf{rf}$

Prior work[1] automates correctness proofs by leveraging relational algebra decision procedures.



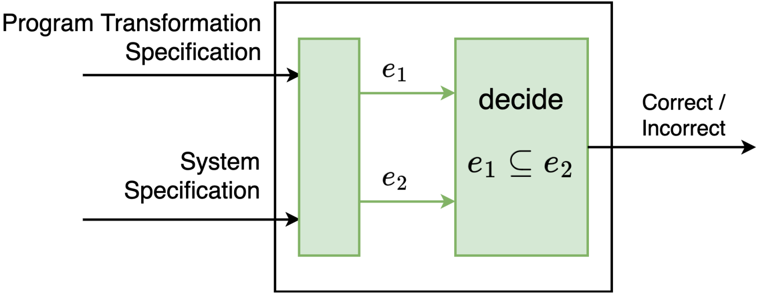
[1] Kokologiannakis, Lahav, and Vafeiadis, "Kater: Automating Weak Memory Model Metatheory and Consistency Checking".

Prior work[1] automates correctness proofs by leveraging relational algebra decision procedures.



[1] Kokologiannakis, Lahav, and Vafeiadis, "Kater: Automating Weak Memory Model Metatheory and Consistency Checking".

Prior work[1] automates correctness proofs by leveraging relational algebra decision procedures.



1. **KAT**(Kleene Algebra with Tests) expressions for the inputs
2. Leverage decidable theory of **KAT** to check $e_1 \subseteq e_2$

[1] Kokologiannakis, Lahav, and Vafeiadis, "Kater: Automating Weak Memory Model Metatheory and Consistency Checking".

KAT does not capture all memory models

KAT has limited expressiveness:

$$\begin{aligned} e_1 \cup e_2 &: \textit{union} \\ e_1 ; e_2 &: \textit{relational composition} \\ e^* &: \textit{reflexive-transitive closure} \end{aligned}$$

KAT does not capture all memory models

KAT has limited expressiveness:

$$e_1 \cup e_2 : \textit{union}$$
$$e_1 ; e_2 : \textit{relational composition}$$
$$e^* : \textit{reflexive-transitive closure}$$

Therefore, cannot capture interesting memory models like:

- ✗ LKMM: Linux Kernel Memory Model
- ✗ RC11: Repaired C11 Model

Memory model definitions use relational intersection

What do the definitions of models like LKMM and RC11 contain that cannot be expressed in KAT?

$e_1 \cap e_2$: *relational* **intersection**

Memory model definitions use relational intersection

What do the definitions of models like LKMM and RC11 contain that cannot be expressed in KAT?

$e_1 \cap e_2$: *relational* **intersection**

Challenge: relational algebra with intersections is hard

Memory model definitions use relational intersection

What do the definitions of models like LKMM and RC11 contain that cannot be expressed in KAT?

$e_1 \cap e_2$: *relational* **intersection**

Challenge: relational algebra with intersections is hard

Observation: memory models only use intersection with primitive relations e.g: $- \cap \textit{sameloc}$, $- \cap \textit{samethread}$ but not $- \cap (\mathsf{po} \cup \mathsf{rf})$

Memory model definitions use relational intersection

What do the definitions of models like LKMM and RC11 contain that cannot be expressed in KAT?

$e_1 \cap e_2$: *relational* **intersection**

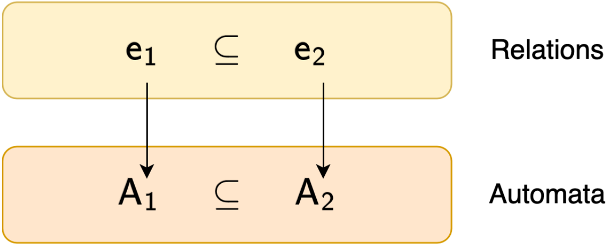
Challenge: relational algebra with intersections is hard

Observation: memory models only use intersection with primitive relations e.g: $- \cap \mathit{sameloc}$, $- \cap \mathit{samethread}$ but not $- \cap (\mathsf{po} \cup \mathsf{rf})$

Solution: decision procedure for $e_1 \subseteq e_2$ with restricted intersections

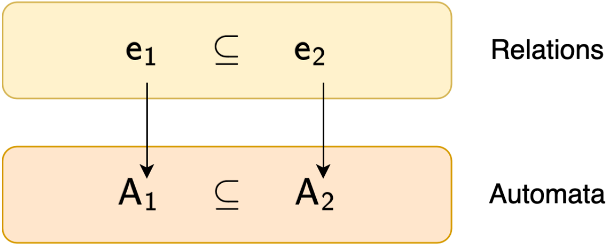
Relational inclusion is checked using automata

1. Construct automata out of KAT**I** expressions
2. Check inclusion between automata



Relational inclusion is checked using automata

1. Construct automata out of KAT**I** expressions
2. Check inclusion between automata



We **prove the equivalence** of the two inclusion problems.

Takeaways

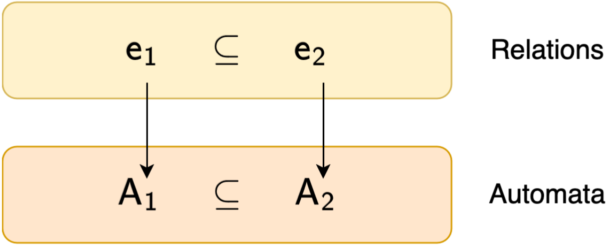
From a practical standpoint:

- We enabled automated reasoning for complex memory models like LKMM and RC11

From a theoretical viewpoint:

- Devised decision procedure for a fragment of relational algebra with restricted intersections.

Roadmap



1. Relational Interpretation
2. Novel Language Interpretation
3. Automata

Semantics of Inclusion

To validate correctness of program transformations, it may suffice to check e.g.:

$$(\mathsf{po} \cup \mathsf{rf})^* \subseteq \mathsf{po}^*; (\mathsf{rf}; \mathsf{po}^*)^*$$

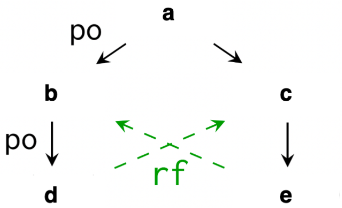
Semantics of Inclusion

To validate correctness of program transformations, it may suffice to check e.g.:

$$(\mathsf{po} \cup \mathsf{rf})^* \subseteq \mathsf{po}^*; (\mathsf{rf}; \mathsf{po}^*)^*$$

i.e. **for all programs, for all executions**.

Interpreting KAT expressions over a graph G

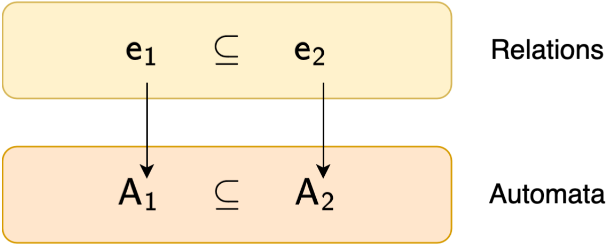


Execution Graph G

$$(\mathsf{po} \cup \mathsf{rf})^* \subseteq \mathsf{po}^*; (\mathsf{rf}; \mathsf{po}^*)^*$$

A relation that relates events connected by paths of 0 or more edges, each of which is either po or rf.

Roadmap



1. Relational Interpretation
2. **Novel Language Interpretation**
3. Automata

Interpretation over Languages

We map an expression e to a regular language $L(e)$ over the alphabet of primitive relations $\Sigma = \{\mathsf{po}, \mathsf{rf}, \ldots\}$.

$(\mathsf{po} \cup \mathsf{rf})^*$:

Interpretation over Languages

We map an expression e to a regular language $L(e)$ over the alphabet of primitive relations $\Sigma = \{\mathsf{po}, \mathsf{rf}, \ldots\}$.

$(\mathsf{po} \cup \mathsf{rf})^*$: All strings consisting of a sequence of 0 or more po, rf.

e.g. ϵ, $'\mathsf{po}'$, $'\mathsf{po}\,\mathsf{rf}\,\mathsf{rf}'$ ...

Attempting to interpret intersection over languages

Unfortunately, cannot use $\cap$ on languages:

- $\mathsf{po} \cap \mathsf{rf} \neq \emptyset$ for relations
- However, $L(\mathsf{po}) \cap L(\mathsf{rf}) = \emptyset$ holds

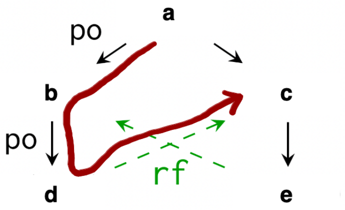
Attempting to interpret intersection over languages

Unfortunately, cannot use $\cap$ on languages:

- $\mathrm{po} \cap \mathrm{rf} \neq \emptyset$ for relations
- However, $L(\mathrm{po}) \cap L(\mathrm{rf}) = \emptyset$ holds

We can make some observations about relational $\cap$ that will help us!

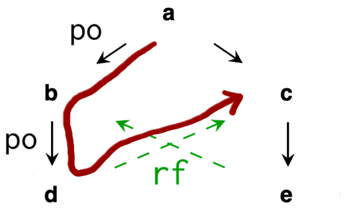
Relational Interpretation of Intersection



Execution Graph G

$$(\mathrm{po}; \mathrm{po}; \mathrm{rf}) \cap \mathrm{po}$$

Relational Interpretation of Intersection



Execution Graph G

$(\mathsf{po}; \mathsf{po}; \mathsf{rf}) \cap \mathsf{po}$

Key Observation: The **endpoints** of the paths are also related by a po edge.

Key Idea of Interpretation

For each primitive relation r, introduce a new pair of symbols: $<_r, >_r$

$$L(e \cap r) = <_r \cdot L(e) \cdot >_r$$

Key Idea of Interpretation

For each primitive relation r, introduce a new pair of symbols: $<_r, >_r$

$$L(e \cap r) = <_r \cdot L(e) \cdot >_r$$

e.g. : $L(\mathrm{rf} \cap \mathrm{po}) = \{" <_{\mathrm{po}}\ \mathrm{rf}\ >_{\mathrm{po}} "\}$

Key Idea of Interpretation

For each primitive relation r, introduce a new pair of symbols: $<_r, >_r$

$$L(e \cap r) = <_r \cdot L(e) \cdot >_r$$

e.g. : $L(\text{rf} \cap \text{po}) = \{" <_{\text{po}} \text{ rf } >_{\text{po}} "\}$

But still, we cannot validate properties of relational algebra:

- $\text{rf} \cap \text{po} \subseteq \text{rf}$ holds for relations
- However, $L(\text{rf} \cap \text{po}) = \{" <_{\text{po}} \text{ rf } >_{\text{po}} "\} \not\subseteq \{"\text{rf}"\}$

Saturation of expressions

Key idea: saturate right-hand-side with brackets.

Saturation of expressions

Key idea: saturate right-hand-side with brackets.

When faced with the decision problem:

$$\ldots <_r \cdot e \cdot >_r \ldots \;\subseteq\; \ldots e' \ldots$$

Saturation of expressions

Key idea: saturate right-hand-side with brackets.

When faced with the decision problem:

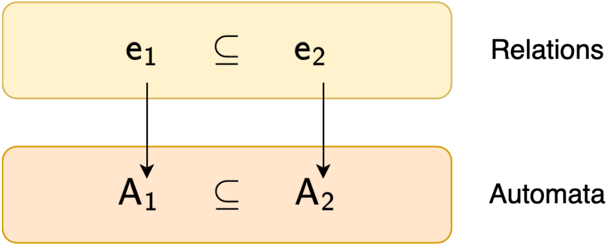
$$\ldots <_r \cdot e \cdot >_r \ldots \subseteq \ldots e' \ldots$$

Instead decide whether:

$$\ldots <_r \cdot e \cdot >_r \ldots \subseteq \ldots <_{\mathbf{r}} \cdot \mathbf{e'} \cdot >_{\mathbf{r}} \ldots$$

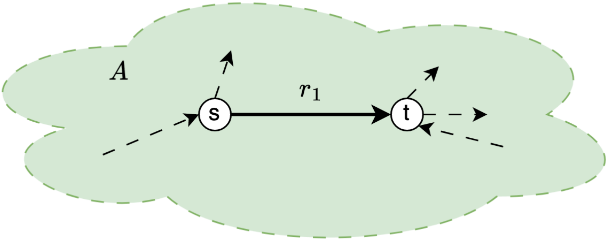
We calculate the saturation on an automaton instead of an expression.

Roadmap

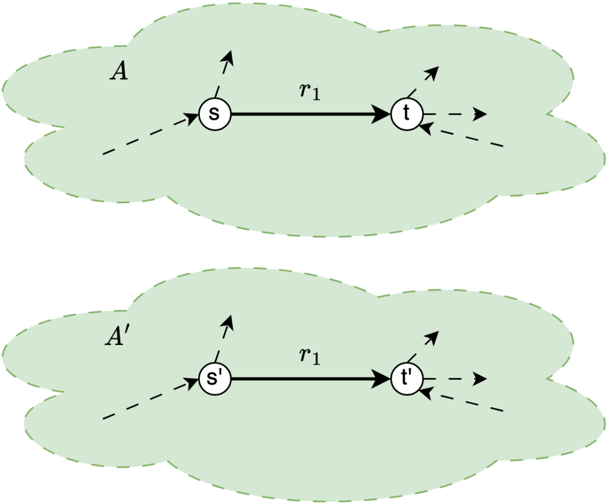


1. Relational Interpretation
2. Novel Language Interpretation
3. **Automata**

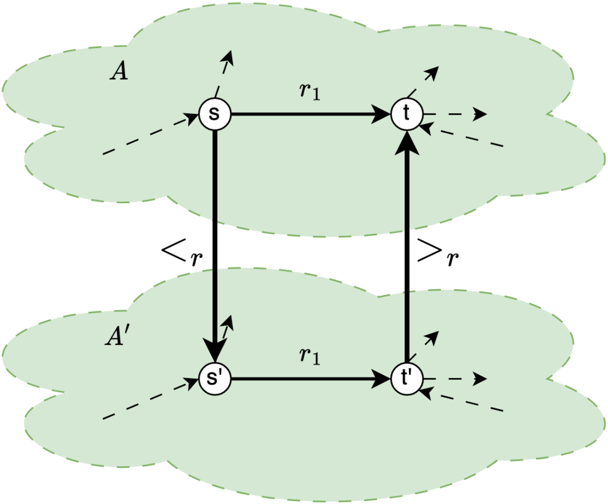
Saturation on automata



Saturation on automata



Saturation on automata



Decision procedure with saturation

When checking

$$e_1 \subseteq e_2$$

Construct automata A_1, A_2 and the saturated automaton $BR(A_2)$.
Use language inclusion algorithms:

$$L(A_1) \subseteq L(BR(A_2))$$

Conclusion and Future Work

We developed a new decision procedure for an extended KAT with $\cap$ with primitive relations.

Future work includes:

- implementation of decision procedure
- intersections with equivalence relations