Compositional Reasoning for WMMs

COV889 Course Presentation

Ramneet Singh
Mrunmayi Bhalerao

IIT Delhi

May 2023

Section 1

Introduction

Rely/Guarantee Reasoning

- Verification of concurrent programs with shared resources is challenging due to **combinatorial explosion**
- **Abstraction to the rescue!**
- Everything the environment can do: $\mathcal{R}$
- Everything you can do: $\mathcal{G}$

$$\mathcal{R}, \mathcal{G} \vdash P\{\ c\ \}Q$$

- *Compositional!*

Extension to Weak Memory Models

- Judgements using earlier techniques are valid under **sequentially consistent semantics**
  - Can be directly used for data-race free code executing on weak memory models
  - But, lots of code has data races! `seqlock`, `java.util.concurrent.ConcurrentLinkedQueue` ...
- How do we extend them to weak memory models?
- What If: We could find a condition under which sequentially consistent rely-guarantee reasoning can be *soundly* preserved

$$(\vdash P\{\ c\ \}Q) \wedge ?? \implies \vdash P\{\ c_{WM}\ \}Q$$

- Benefits:
  - Reuse existing verification techniques
  - Deal with the complexity of weak memory separately as a *side-condition*

What are WMMs anyway?

- Relaxing the memory consistency guarantees provided by hardware enables optimisations
  - Store forwarding (will see later)
  - Write buffers
- (Part 1) **Multicopy Atomic**: One thread's stores become observable to all other threads at the same time.
  - x86-TSO, ARMv8, RISC-V
- (Part 2) **Non-Multicopy Atomic**: Each component has its own *view* of the global memory.
  - Older ARM versions, POWER, C11
- Challenge: Two types of interference now – **Inter-Thread** + **Intra-Thread** (due to reordering)
- How will we deal with this? . . .

Teaser

- We want a compositional approach through thread-local reasoning.
- Exploit the reordering semantics of Colvin and Smith: **multicopy atomic memory models can be captured in terms of instruction reordering.**
    - Combinatorial explosion? (n reorderable instructions in a thread $\implies$ $n!$ behaviours)
    - Introduce *reordering interference freedom* between $(\frac{n(n-1)}{2})$ pairs of instructions (Stay tuned. . .)
- In non-multicopy atomic WMMs, there is **no global shared state**(!!)
    - Judgement for each thread is applicable to *its view* (depends on *propagation* of writes by hardware)
    - How do we know it holds in other threads' views?
    - Represent the semantics using reordering **between** different threads
    - No longer compositional? Hardest part of the talk – *global reordering interference freedom*: use the rely abstraction to represent reorderings between threads

Section 2

Abstract Language

Syntax

- Individual (atomic) instructions α
- Commands (or programs)

$$c \ := \ \epsilon \mid \alpha \mid c_1 ; c_2 \mid c_1 \ \sqcap \ c_2 \mid c^* \mid c_1 \mid\mid c_2$$

- Iteration, choice are non-deterministic
- Empty program ϵ represents termination

Semantics: Commands

- Each atomic instruction α has a relation $\mathrm{beh}(\alpha)$ (over pre- and post-states) specifying its behaviour
- Program execution is defined by a small-step semantics over commands
- Iteration, non-deterministic choice are dealt with at a higher level (see next slide)

$$\frac{}{\alpha \mapsto_\alpha \epsilon} \qquad \frac{c_1 \mapsto_\alpha c_1'}{c_1 ; c_2 \mapsto_\alpha c_1' ; c_2}$$

$$\frac{c_1 \mapsto_\alpha c_1'}{c_1 \mid\mid c_2 \mapsto_\alpha c_1' \mid\mid c_2} \qquad \frac{c_2 \mapsto_\alpha c_2'}{c_1 \mid\mid c_2 \mapsto_\alpha c_1 \mid\mid c_2'}$$

Semantics: Configurations

- *Configuration* (c, σ) of a program
  - Command c to be executed
  - State σ (map from variables to values)
- *Action Step*: Performed by component, changes state

$$(c, \sigma) \xrightarrow{as} (c', \sigma') \iff \exists \alpha . c \mapsto_{\alpha} c' \wedge (\sigma, \sigma') \in \mathrm{beh}(\alpha)$$

- *Silent Step*: Performed by component, doesn't change state

$$(c_1 \sqcap c_2, \sigma) \rightsquigarrow (c_1, \sigma) \quad (c_1 \sqcap c_2, \sigma) \rightsquigarrow (c_2, \sigma)$$

$$(c^*, \sigma) \rightsquigarrow (\epsilon, \sigma) \quad (c^*, \sigma) \rightsquigarrow (c; c^*, \sigma)$$

- *Program Step*: Action Step or Silent Step
- *Environment Step*: Performed by environment, changes state. $(c, \sigma) \xrightarrow{es} (c, \sigma')$.

Section 3

Basic Proof System

Definitions

- Associate a verification condition $\mathrm{vc}(\alpha)$ with each instruction α: Provides finer-grained control (just set to $\top$ if not needed)
- Hoare triple

$$P\{\ \alpha\ \}Q \overset{\text{def}}{=} P \subseteq \mathrm{vc}(\alpha) \cap \{\sigma \mid \forall\, \sigma',\ (\sigma, \sigma') \in \mathrm{beh}(\alpha) \Longrightarrow \sigma' \in Q\}$$

- A rely-guarantee pair $(\mathcal{R}, \mathcal{G})$ is well-formed if
  - $\mathcal{R}$ is reflexive and transitive
  - $\mathcal{G}$ is reflexive
- Stability of predicate P under rely condition $\mathcal{R}$

$$\mathrm{stable}_{\mathcal{R}}(P) \overset{\text{def}}{=} P \subseteq \{\sigma \in P \mid \forall\, \sigma',\ (\sigma, \sigma') \in \mathcal{R} \Longrightarrow \sigma' \in P$$

- Instruction α satisfies guarantee condition $\mathcal{G}$

$$\mathrm{sat}(\alpha, \mathcal{G}) \overset{\text{def}}{=} \{\sigma \mid \forall\, \sigma',\ (\sigma, \sigma') \in \mathrm{beh}(\alpha) \Longrightarrow (\sigma, \sigma') \in \mathcal{G}\}$$

- Now introduce rely/guarantee judgements at three levels

Instruction Level ($\vdash_a$)

$$\mathcal{R}, \mathcal{G} \vdash_a P\{\ c\ \}Q \overset{\text{def}}{=} \mathrm{stable}_{\mathcal{R}}(P) \wedge \mathrm{stable}_{\mathcal{R}}(Q) \wedge \mathrm{vc}(\alpha) \subseteq \mathrm{sat}(\alpha, \mathcal{G}) \wedge P\{\ c\ \}Q$$

- Interplay between environmental interference and pre-,post-conditions handled through stability

Component Level ($\vdash_c$)

$$\text{Atom}\ \frac{\mathcal{R},\mathcal{G} \vdash_a P\{\ \alpha\ \}Q}{\mathcal{R},\mathcal{G} \vdash_c P\{\ \alpha\ \}Q}$$

$$\text{Seq}\ \frac{\mathcal{R},\mathcal{G} \vdash_c P\{\ c_1\ \}M \quad \mathcal{R},\mathcal{G} \vdash_c M\{\ c_2\ \}Q}{\mathcal{R},\mathcal{G} \vdash_c P\{\ c_1; c_2\ \}Q}$$

$$\text{Choice}\ \frac{\mathcal{R},\mathcal{G} \vdash_c P\{\ c_1\ \}Q \quad \mathcal{R},\mathcal{G} \vdash_c P\{\ c_2\ \}Q}{\mathcal{R},\mathcal{G} \vdash_c P\{\ c_1 \sqcap c_2\ \}Q}$$

$$\text{Iteration}\ \frac{\mathcal{R},\mathcal{G} \vdash_c P\{\ c\ \}P \quad \mathrm{stable}_{\mathcal{R}}(P)}{\mathcal{R},\mathcal{G} \vdash_c P\{\ c^*\ \}Q}$$

$$\text{Conseq}\ \frac{\mathcal{R},\mathcal{G} \vdash_c P\{\ c\ \}Q \quad P' \subseteq P \quad \mathcal{R}' \subseteq \mathcal{R} \quad Q \subseteq Q' \quad \mathcal{G} \subseteq \mathcal{G}'}{\mathcal{R}',\mathcal{G}' \vdash_c P'\{\ c\ \}Q'}$$

Global Level ($\vdash$)

- Global satisfiability needs component satisfiability + **interference check**

$$\text{Comp}\,\frac{\mathcal{R},\mathcal{G} \vdash_c P\{\ c\ \}Q \quad \mathrm{rif}(\mathcal{R},\mathcal{G},c)}{\mathcal{R},\mathcal{G} \vdash P\{\ c\ \}Q}$$

- Usual parallel rule

$$\text{Par}\,\frac{\mathcal{R},\mathcal{G} \vdash_c P\{\ c\ \}Q \quad \mathrm{rif}(\mathcal{R},\mathcal{G},c)}{\mathcal{R},\mathcal{G} \vdash P\{\ c\ \}Q}$$

Section 4

Multicopy Atomic Memory Models

Reordering Semantics: Basics

- Multicopy atomic memory models can be characterised using a *reordering* relation $\hookleftarrow$ over pairs of instructions in a component
- $\hookleftarrow$ is syntactically derivable based on the specific memory model. E.g., in ARMv8
  - Two instructions which don't access (read or write) a common variable can be reordered
  - Various types of memory barriers prevent reordering
- *Forwarding* is another complication
  - $\beta = \texttt{x := 3}; \alpha = \texttt{y := x}$. Can forward the value 3 to y, losing dependence between α, β.
  - `x := 3 ; y := x` $\Longrightarrow$ `y := 3 ; x := 3`
  - Denote α with the value written in an earlier instruction forwarded to it as $\alpha_{<\beta>}$.
- Forwarding may continue arbitrarily and can span multiple instructions

Reordering Semantics: Formal

- $\alpha_{<c>}$: cumulative forwarding effects of the instructions in command c on α
- Ternary relation $\gamma < c < \alpha$: Reordering of instruction α prior to command c, with cumulative forwarding effects producing γ.
- Definition by induction

$$\alpha_{<\beta>} < \beta < \alpha \overset{\text{def}}{=} \beta \hookleftarrow \alpha_{<\beta>}$$

$$\alpha_{<c_1;c_2>} < c_1; c_2 < \alpha \overset{\text{def}}{=} \alpha_{<c_1;c_2>} < c_1 < \alpha_{<c_2>} \wedge \alpha_{<c_2>} < c_2 < \alpha$$

- Example: $\alpha = (\texttt{y := x}), \beta = (\texttt{x := 3}), \gamma = (\texttt{z := 5})$.
  $\alpha_{<\beta>} = (\texttt{y := 3})$,$\alpha_{<\gamma;\beta>} = (\texttt{y := 3})$.

  $\texttt{y := 3} < \texttt{x := 3} < \texttt{y := x} \quad \texttt{y := 3} < \texttt{z := 5 ; x := 3} < \texttt{y := x}$

- Can execute an instruction which occurs later in the program if reordering and forwarding can bring it (in its new form γ) to the beginning

$$\text{Reorder}\,\frac{c_2 \mapsto_\alpha \quad \gamma < c < \alpha}{c_1; c_2 \mapsto_\gamma c_1; c_2'}$$