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 $\Longrightarrow$ $n!$ 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)

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

- 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

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

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

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