3.36pt

Compositional verification of a lazy cache coherence protocol with a weak memory model

Chris Banks, Marco Elver, Vijay Nagarajan, Paul Jackson

THE UNIVERSITY *of* EDINBURGH
informatics

icsa | Institute for Computing Systems Architecture

PEPA Club, 4th November 2016

Introduction

- How we verified a lazy cache coherence protocol with a weak memory model:

Introduction

- How we verified a lazy cache coherence protocol with a weak memory model:
  - Intro to TSO-CC and lazy protocols;

Introduction

- How we verified a lazy cache coherence protocol with a weak memory model:
  - Intro to TSO-CC and lazy protocols;
  - why previous verification techniques don't work;

Introduction

- How we verified a lazy cache coherence protocol with a weak memory model:
  - Intro to TSO-CC and lazy protocols;
  - why previous verification techniques don't work;
  - stages of verification:
    - modelling TSO-CC;
    - checking TSO-CC satisfies the memory model (TSO)
    - parameterisation (compositional check for n cores).

Introduction

- How we verified a lazy cache coherence protocol with a weak memory model:
  - Intro to TSO-CC and lazy protocols;
  - why previous verification techniques don't work;
  - stages of verification:
    - modelling TSO-CC;
    - checking TSO-CC satisfies the memory model (TSO)
    - parameterisation (compositional check for n cores).
- Future directions?

TSO-CC and lazy protocols

The basics (all we need to know for this talk):

TSO-CC and lazy protocols

The basics (all we need to know for this talk):

- TSO-CC is a lazy cache coherence protocol,

TSO-CC and lazy protocols

The basics (all we need to know for this talk):

- TSO-CC is a lazy cache coherence protocol,
  - designed to take advantage of weak memory model
  - scale to large numbers of cores.

TSO-CC and lazy protocols

The basics (all we need to know for this talk):

- TSO-CC is a lazy cache coherence protocol,
  - designed to take advantage of weak memory model
  - scale to large numbers of cores.
- Weak memory models (TSO) only require consistency at synchronisation boundaries.

TSO-CC and lazy protocols

The basics (all we need to know for this talk):

- TSO-CC is a lazy cache coherence protocol,
  - designed to take advantage of weak memory model
  - scale to large numbers of cores.
- Weak memory models (TSO) only require consistency at synchronisation boundaries.
- TSO-CC is lazy:
  - does not eagerly invalidate on writes,

TSO-CC and lazy protocols

The basics (all we need to know for this talk):

- TSO-CC is a lazy cache coherence protocol,
  - designed to take advantage of weak memory model
  - scale to large numbers of cores.
- Weak memory models (TSO) only require consistency at synchronisation boundaries.
- TSO-CC is lazy:
  - does not eagerly invalidate on writes,
  - cache lines self-invalidate after so many reads (eventually consistent),

TSO-CC and lazy protocols

The basics (all we need to know for this talk):

- TSO-CC is a lazy cache coherence protocol,
  - designed to take advantage of weak memory model
  - scale to large numbers of cores.
- Weak memory models (TSO) only require consistency at synchronisation boundaries.
- TSO-CC is lazy:
  - does not eagerly invalidate on writes,
  - cache lines self-invalidate after so many reads (eventually consistent),
  - does not need to track sharers, so is more scalable (storage reqs.).

Classical verification fails

- Classical verification techniques rely on local properties:
  - e.g. Single Writer Multiple Reader (SWMR),
  - Data Value Invariant, ...

Classical verification fails

- Classical verification techniques rely on local properties:
  - e.g. Single Writer Multiple Reader (SWMR),
  - Data Value Invariant, ...
- These don't hold in lazy protocols.

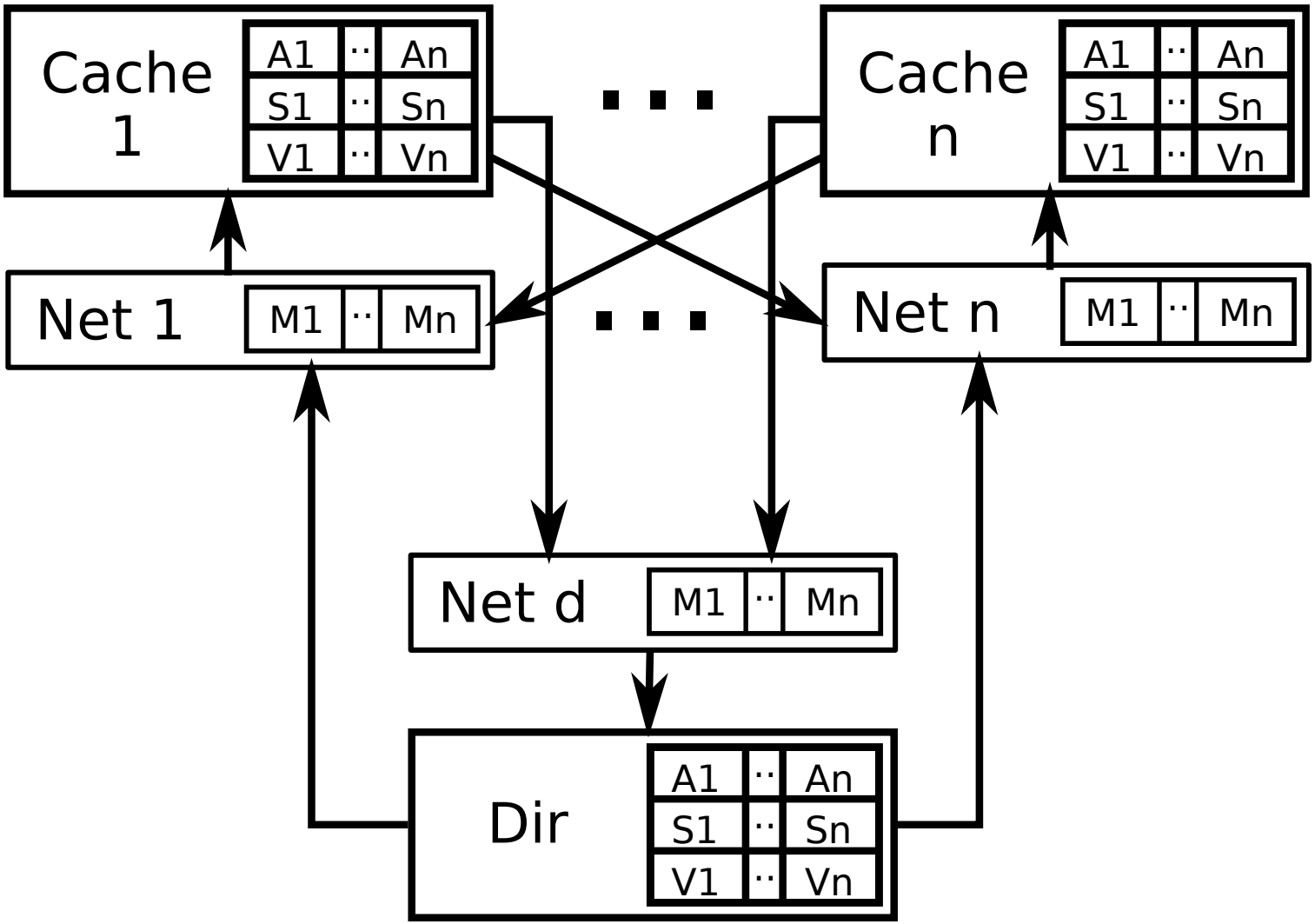
Classical verification fails

- Classical verification techniques rely on local properties:
  - e.g. Single Writer Multiple Reader (SWMR),
  - Data Value Invariant, ...
- These don't hold in lazy protocols.
- What do we verify against then?

Classical verification fails

- Classical verification techniques rely on local properties:
  - e.g. Single Writer Multiple Reader (SWMR),
  - Data Value Invariant, ...
- These don't hold in lazy protocols.
- What do we verify against then?
- Answer: the memory model.

Modelling TSO-CC



Modelling TSO-CC

Murϕ example rules:

$$c[a].\texttt{state} = \texttt{I} \implies \texttt{SendGetS}(c, \mathit{Dir}, a);$$
$$c[a].\texttt{state} := \mathit{WS} \qquad \text{(Read I)}$$

$$c[a].\texttt{state} = \texttt{E} \implies c[a].\texttt{val} := v$$
$$c[a].\texttt{state} := \mathit{M}; \qquad \text{(Write E)}$$

Rules to handle processor actions.

Modelling TSO-CC

Rules to handle receipt of messages, functions for actions upon receipt:

$$
\begin{aligned}
\texttt{DirectoryReceive}(\mathit{msg}, a) = \ & \textbf{if } \mathit{Dir}[a].\texttt{state} = \texttt{I} \wedge \mathit{msg}.\texttt{type} = \texttt{GetS} \\
& \textbf{then } \texttt{SendDataS}(\mathit{msg}.\texttt{src}, a, \ldots); \\
& \quad \texttt{ReplaceOwner}(\mathit{msg}.\texttt{src}, a); \\
& \quad \mathit{Dir}[a].\texttt{state} := \texttt{WE1} \\
& \textbf{else } \ldots
\end{aligned}
$$

Modelling TSO-CC

We do some clever things like abstracting counters:

$$c[a].\texttt{state} = \texttt{S} \implies \texttt{SendGetS}(c, \mathit{Dir}, a);$$
$$c[a].\texttt{state} := \mathit{WS} \qquad \text{(Read S (ACount=MAX))}$$

$$c[a].\texttt{state} = \texttt{S} \implies \text{//do nothing} \qquad \text{(Read S (ACount<MAX))}$$

(This becomes important later.)

TSO-CC $\models$ TSO

- How do we show TSO-CC $\models$ TSO?

TSO-CC $\models$ TSO

- How do we show TSO-CC $\models$ TSO?
- Answer: show TSO-CC is *weakly simulated by* TSO.

TSO-CC $\models$ TSO

- How do we show TSO-CC $\models$ TSO?
- Answer: show TSO-CC is *weakly simulated by* TSO.
- To show a weak simulation relation we need an operational model of TSO.

TSO-CC $\models$ TSO

- How do we show TSO-CC $\models$ TSO?
- Answer: show TSO-CC is *weakly simulated by* TSO.
- To show a weak simulation relation we need an operational model of TSO.
- Existing operational models of TSO have potentially infinite store buffers.

TSO-CC $\models$ TSO

- How do we show TSO-CC $\models$ TSO?
- Answer: show TSO-CC is *weakly simulated by* TSO.
- To show a weak simulation relation we need an operational model of TSO.
- Existing operational models of TSO have potentially infinite store buffers.
- TSO-LB has finite load buffers.

TSO-CC $\models$ TSO

- How do we show TSO-CC $\models$ TSO?
- Answer: show TSO-CC is *weakly simulated by* TSO.
- To show a weak simulation relation we need an operational model of TSO.
- Existing operational models of TSO have potentially infinite store buffers.
- TSO-LB has finite load buffers.
- TSO-LB $\implies$ TSO (proof).

TSO-CC $\models$ TSO-LB

- $\texttt{local} : (S \times P \times A) \to V$ is a function where $\texttt{local}(s, p, a)$ is the value at address a in the local buffer of p in state s.
- $\texttt{global} : (S \times A) \to V$ is a function where $\texttt{global}(s, a)$ is the value at address a in the global buffer in state s.

$$\frac{\texttt{local}(s, p, a) = v}{s \xrightarrow{\mathsf{Read}(p,a,v)} s} \textsc{Read}$$

$$\frac{\top}{s \xrightarrow{\mathsf{Write}(p,a,v)} s' \wedge \texttt{local}(s', p, a) \mapsto v \wedge \texttt{global}(s', a) \mapsto v} \textsc{Write}$$

$$\frac{\top}{s \xrightarrow{\mathsf{Prop}(p)} s' \wedge \forall a.\texttt{local}(s', p, a) \mapsto \texttt{global}(s, a)} \textsc{Propagate}$$

Weak simulation checking

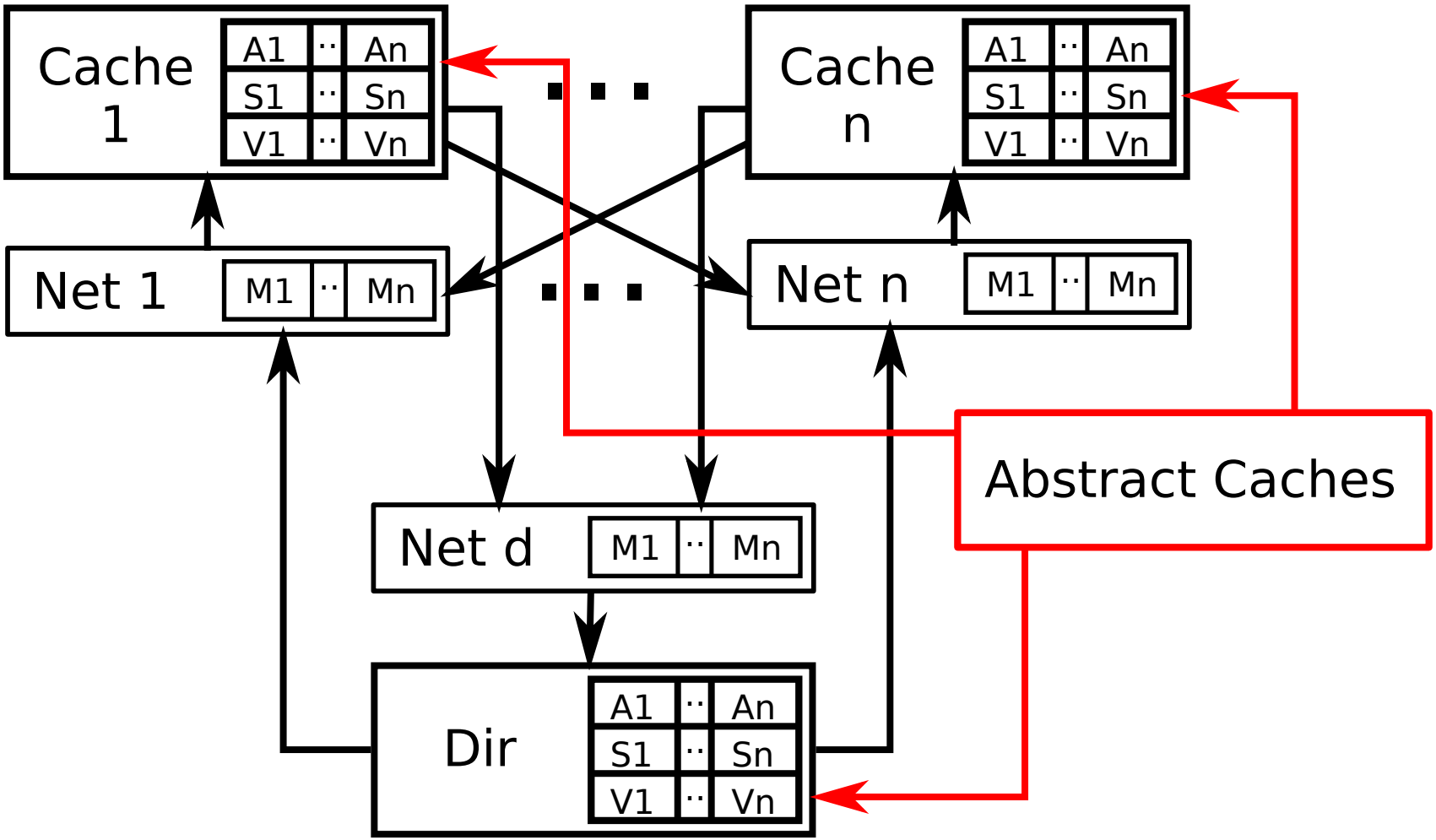
Add invocation of TSO-LB from TSO-CC (in the model checker).

$$c[a].\texttt{state} = \texttt{E} \implies c[a].\texttt{val} := v$$
$$c[a].\texttt{state} := M;$$
$$\boxed{\texttt{TSOStore}(c, a, v)} \qquad \text{(Write E)}$$

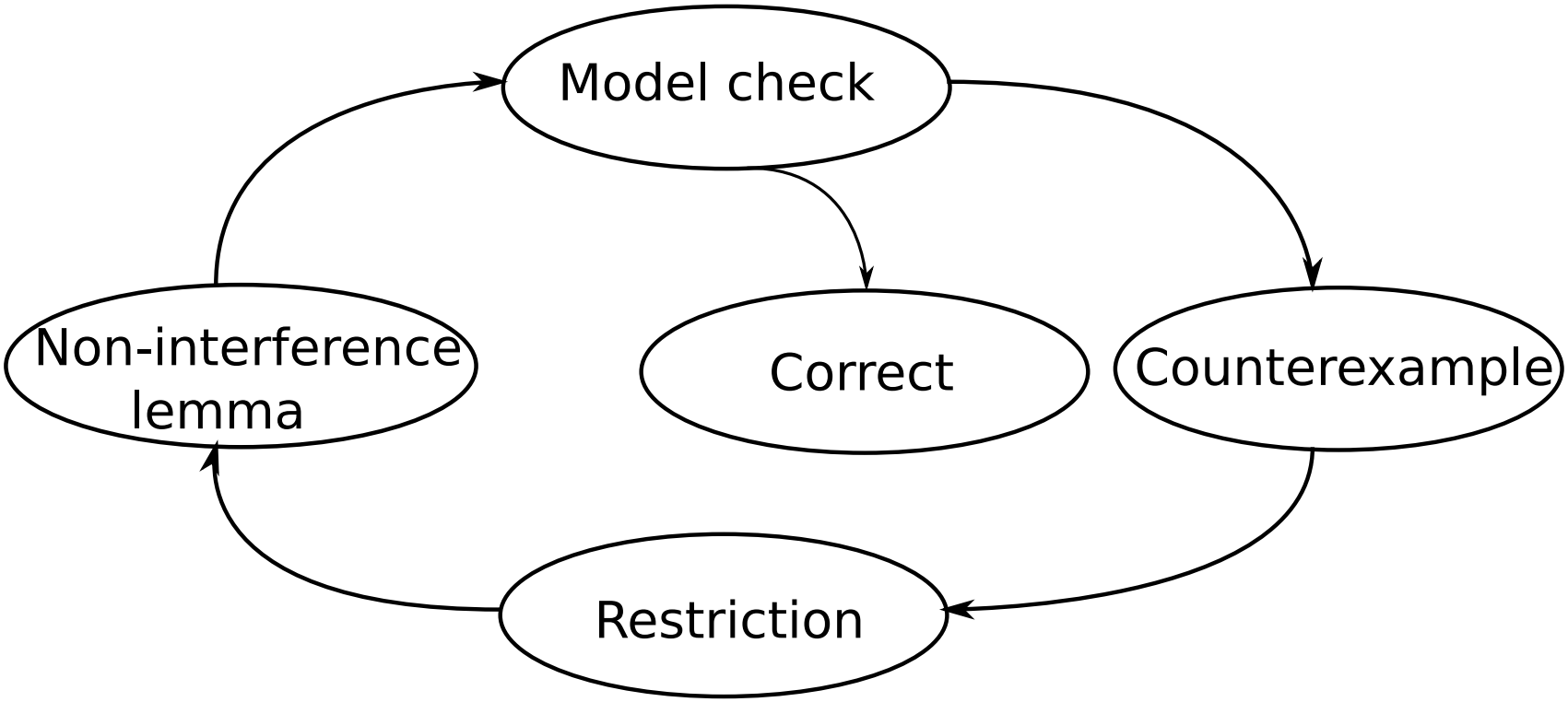
$$c[a].\texttt{state} = \texttt{S} \implies \text{//do nothing;}$$
$$\boxed{\texttt{Assert}(\texttt{TSOVerify}(c, a, c[a].\texttt{val}))} \qquad \text{(Read S)}$$

Thus we show weak simulation by model checking.

Parameterisation



Parameterisation



Parameterisation

Add rules for messages received from the abstract cache, e.g.:

$$
\begin{aligned}
c[a].\texttt{state} = \texttt{WX} \\
\quad \Longrightarrow\ & \texttt{SendAck}(c, \mathit{Dir}, a); \\
& c[a].\texttt{state} := \texttt{M}; \\
& \texttt{InvalidateAllOtherLines}(c, a)
\end{aligned}
$$

(Cache Recv DataX Abs)

+ all combinations of node state and message type.

Abstract cache initially produces all possible messages (*over-approximation*).

Parameterisation

Model check; get counterexample; add restriction:

$$c[a].\texttt{state} = \texttt{WX} \;\boxed{\wedge \quad \texttt{IsOwner}(c, a)}$$
$$\implies \texttt{SendAck}(c, \mathit{Dir}, a);$$
$$c[a].\texttt{state} := \texttt{M};$$
$$\texttt{InvalidateAllOtherLines}(c, a)$$

(Cache Recv DataX Abs)

Add non-interference lemma:

$$\forall n \forall a \forall i.\ \mathit{net}[n][a][i].\texttt{msgType} = \texttt{DataX} \implies \texttt{IsOwner}(n, a)$$

Repeat. (31 restriction/lemma pairs!)

Summary

Problems solved:

- How to verify against TSO – *weak simulation*.
- How to show weak simulation in the model checker.
- Finite state operational model of TSO (TSO-LB).
- How to extend this to compositional model checking.

Validation:

- bug found: only in the full n-process model!
- Trivial bug in model, but shows it catches more bugs.