Optimistic Synchronization-Based State-Space Reduction

Scott D. Stoller

State University of New York at Stony Brook

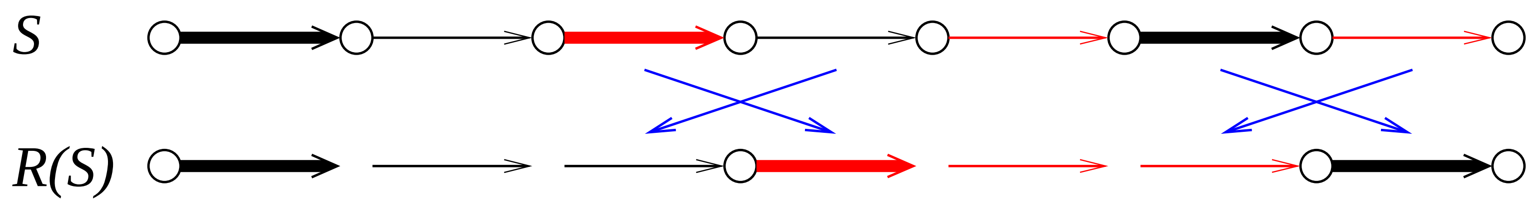
Ernie Cohen

Microsoft

Coarsening the Granularity of Transitions

Each transition of reduced (coarse-grained) system $R(S)$ corresponds to a sequence of transitions of the original system S.

Example: System S has two threads, black and red. Typical executions of S and $R(S)$ look like



Define R so that every execution of S can be re-arranged into an execution of $R(S)$ by swapping transitions that commute.

Verification of coarse-grained systems is easier:
fewer states, simpler invariants

Visible Transitions and States

Derive the transformation R from a classification of transitions (operations) as visible or invisible.

Def.: Each transition of $R(S)$ is a visible transition followed by a maximal sequence of invisible transitions of the same thread.

Relations

u_i: invisible transition relation of thread i
v_i: visible transition relation of thread i

Predicates

d_i: u_i is disabled
d: $(\forall i : d_i)$

visible state: d holds (no invisible transitions are enabled)

Reduced System

Derived Relations

$u = (+i : u_i)$	invisible transition relation of S
$v = (+i : v_i)$	visible transition relation of S
$u + v$	transition relation of S
$(+i : v_i\, u_i^*\, d_i)$	transition relation of $R(S)$

Commutativity Condition: An invisible transition should left-commute with all transitions of other threads: for $i \neq j$,
$s \xrightarrow{u_j + v_j} \xrightarrow{u_i} s'$ implies $s \xrightarrow{u_i} \xrightarrow{u_j + v_j} s'$.
In other words, $(u_j + v_j)u_i \leq u_i(u_j + v_j)$.

Notation: Juxtaposition denotes relational composition.
$\leq$ denotes refinement (subset).

A Simple Traditional Reduction Theorem

Reduction Theorem: If the commutativity condition holds, and all initial states are visible, then each visible state is reachable in S iff it is reachable in $R(S)$. In other words,

$$\frac{i \neq j \Rightarrow (u_j + v_j)u_i \leq u_i(u_j + v_j) \qquad I \leq d}{I\,(u + v)^*\,d = I\,(+i : v_i\,u_i^*\,d_i)^*\,d}$$

Proof Sketch:
$\geq$: immediate from the definitions.
$\leq$: an execution of S can be re-arranged into an execution of $R(S)$ that reaches the same visible state by repeatedly left-commuting invisible transitions past transitions of other threads.

Our optimistic reduction overcomes practical difficulties with traditional reductions.

Which Transitions Are Visible?

Derive classification of transitions from classification of variables.

Simplest Approach:

Classify variables as unshared or shared.

A transition is visible if it possibly accesses a shared variable.

The commutativity condition $(u_j + v_j)u_i \leq u_i(u_j + v_j)$ holds, because $u_j + v_j$ and u_i access disjoint sets of variables.

This reduction is used in model checkers such as Spin.

For increased benefits, try to classify more transitions as invisible.

Synchronization-Based Reduction

Classify variables as:

unshared: accessed by at most one thread

protected: synchronization is used to ensure mutual exclusion for accesses to protected variables

unprotected: all other variables

Exclusive Access Predicate [Flanagan and Qadeer]
e_i^x: thread i has exclusive access to x

(EA1) e_i^x holds in states from which thread i can execute a transition that accesses x.

(EA2) For $i \neq j$, e_i^x and e_j^x are mutually exclusive (disjoint).

(EA3) A thread cannot take away another thread's exclusive access to a variable: for $i \neq j$, $u_j + v_j$ cannot falsify e_i^x.

Examples of Exclusive Access Predicates

Example: x is protected by lock ℓ.

e_i^x : owner(ℓ) $= i$

Example: x is protected by semaphore *s*.

user thread	driver thread
requestData(*buf*)	acceptRequest(*buf*)
down(*s*)	store data in *buf*
use data in buf	up(*s*)

e^{buf}_{user} : program counter of user thread is after "down(*s*)"

e^{buf}_{driver} : program counter of driver thread is before "up(*s*)"

Which Transitions Are Visible?

A transition is visible if it possibly:
- accesses an unprotected variable, or
- changes the value of an exclusive access predicate.

The commutativity condition $(u_j + v_j)u_i \leq u_i(u_j + v_j)$ holds, because $u_j + v_j$ and u_i access disjoint sets of variables.

Proof (sketch) by contradiction: If they accessed the same protected variable x, then (EA1) would imply that e_i^x and e_j^x both hold in the starting state; this would contradict (EA2).

The detailed proof takes into account that a transition may access different variables when executed from different states.
Example: **if (x==0) then y=y+1 else z=z+1**

Using the Reduction: Static Analysis

1. Classify the variables using static analysis.
2. Check correctness properties on $R(S)$.

Reduction theorem implies the results also hold for S.

Problem

Classifying variables as unshared, protected, or unprotected is difficult due to dynamic allocation, references, method calls, etc.

Static analysis is conservative, making some transitions visible unnecessarily, decreasing the benefit of the reduction.

Using the Reduction: Exact Approach

1. Classify the variables, manually or with automated heuristics. (The classification includes exclusive access predicates for protected variables.)

2. Check that the classification C is valid for S, that is, variables are accessed according to C in all executions of S.
Use model-checking, theorem-proving, ...
If C is invalid, revise it and re-check.

3. Check correctness properties on $R_C(S)$.

Problem

Step 2 may be expensive or difficult. (If we could check properties of S directly, we wouldn't need a reduction at all.)

Using the Reduction: Optimistic Approach

1. Classify the variables, manually or with automated heuristics.

2. Check that the classification C is valid for $R_C(S)$, that is, variables are accessed according to C in all executions of $R_C(S)$.
Use model-checking, theorem-proving, ...
If C is invalid, revise it and re-check.

Theorem: For a large class of systems, a classification C of variables is valid for S iff it is valid for $R_C(S)$.

3. Check correctness properties on $R_C(S)$.
Reduction theorem implies the results also hold for S.

Optimistic Coarsening Theorem

Optimistic Coarsening Theorem: Given a system and a synchronization discipline C, if appropriate commutativity conditions hold, then a violation of the synchronization discipline is reachable in S iff a violation is reachable in $R_C(S)$.

$$\frac{i \neq j \;\Rightarrow\; u_j u_i \leq u_i\, u_j + u_i\, q\, \top + u_i\, u_j\, q\, \top) \qquad \cdots}{d\,(u+v)^*\, q \leq d\,(+i : v_i\, u_i^*\, d_i\, \bar{q})^*\, R\, q\, \top}$$

q: the synchronization discipline has been violated.
$\top$: the full relation. $R = 1 + (+i : v_i\, u_i^*)$.

The theorem and formal proof are in omega algebra [Cohen].

Theorem: The commutativity conditions hold for unshared-protected-unprotected synchronization disciplines.

Implementation

Implementation in Java PathFinder (JPF) [Visser et al.], an explicit-state model checker for Java.

Focus on mutex provided by Java's built-in locks.

User-supplied classif.: sets of unshared vars, unprotected vars. Other variables are implicitly classified as protected by locks.

Modify scheduler to use coarse-grained transitions $v_i u_i^* d_i$, by adding an inner loop.

Instrument object accesses, etc., with code that checks classification of vars. Use lockset alg [Savage et al.] to determine which locks protect each protected variable. Incorrect classification can be refined automatically.

Experiments

HaltException, Clean: synchronization skeletons (shared counter + wait/notify; shared buffer) from JPF developers.

Xtango-DP, Xtango-QS : animation of a dining philosophers algorithm and quicksort, with calls to java.awt eliminated.

Application	Mem. Red.	Mem. ByCo	$\frac{\text{MemByCo}}{\text{MemRed}}$	Mem. Line	$\frac{\text{MemLine}}{\text{MemRed}}$
HaltException	2.1	45.9	22	4.3	2.0
Clean	2.3	8.2	3.6	3.6	1.6
Xtango-DP	236	>1800	>7.6	609	2.6
Xtango-QS	91	>1800	>20	346	3.8

Memory is in MB.

Related Work

Reduction Theorems [Lipton, Lamport, Schneider, Cohen, ...]
These theorems require checking commutativity conditions on S. Optimistic coarsening allows checking commutativity conditions (classification of vars) on $R(S)$.

Partial-Order Methods [Valmari, Godefroid, Peled, ...]
These methods avoid exploring some interleavings of transitions, based on conservative static analysis (for example, algorithms that compute stubborn sets). Optimistic coarsening is based on an efficient exact check of conditions for coarsening. It is more effective for "complicated" systems.

Types for Atomicity [Flanagan and Qadeer]
The reduction R is defined and justified by type annotations, supplied by the user. Our method is more automatic.