Lecture 09 – Synthesis of Synchronization

PROGRAM ANALYSIS & SYNTHESIS

Eran Yahav
Previously

- Predicate abstraction
- Abstraction refinement
  - at a high level
Today

- Synthesis of Synchronization
Verification Challenge

\[ P \models S \]
With abstraction $\alpha$

$P \vdash S$
Now what?

\[ P \not\models_{\alpha} S \]

\[ P \models_{\alpha'} S \]

abstraction refinement
A Standard Approach: Abstraction Refinement

Change the \textit{abstraction} to match the \textit{program}.
Alternatively...

\[ P \not\models_{\alpha} S \]

\[ P' \models_{\alpha} S \]

program modification
Abstraction-Guided Synthesis [VYY-POPL’10]

Change the program to match the abstraction
Alternatively...

\[ P \not\models_{\alpha} S \]

\[ P \models_{\alpha} S' \]

change specification (but to what?)
Focus on Program Modification

- Given a program $P$, specification $S$, and an abstraction $\alpha$ such that $P \not\equiv_{\alpha} S$
- find a modified program $P'$ such that $P' \equiv_{\alpha} S$

- how do you find such a program?
  - very hard in general
  - there is some hope when considering equivalent programs or restricted programs
    - changing a program to add new legal behaviors is hard
  - program restriction natural in concurrent programs
    - reducing permitted schedules
Example

```c
int x, y, sign;
x = ?;  
    x \mapsto \left[ -\infty, \infty \right], \ sign \mapsto \bot
if (x<0) {
    sign = -1;
    x \mapsto \left[ -\infty, -1 \right], \ sign \mapsto \left[ -1, -1 \right]
} else {
    sign = 1;
    x \mapsto \left[ 0, \infty \right], \ sign \mapsto \left[ 1, 1 \right]
}
y = x/sign;
```

Specification S: no division by zero.
Abstract domain $\alpha$: intervals

\[ \theta \in [-1,1] \]

Division by zero seems possible

\[ P \not\models_{\alpha} S \]
Now what?

- Can use a more refined abstract domain
- Disjunctive completion
- abstract value = set (disjunction) of intervals

```c
int x, y, sign;
x = ?;  \quad x \mapsto [-\infty, \infty], \quad \text{sign} \mapsto \bot
if (x<0) {
    \quad \text{sign} = -1;
    \quad x \mapsto [-\infty, -1], \quad \text{sign} \mapsto \{[-1, -1]\}
} else {
    \quad \text{sign} = 1;
    \quad x \mapsto [0, \infty], \quad \text{sign} \mapsto \{[1, 1]\}
}
y = x/\text{sign};
```

\[ \emptyset \not\in [-1, -1] \text{ and } \emptyset \not\in [1,1] \]

Specimen S: no division by zero.
Abstract domain \( \alpha' \): set of intervals

\( P \models_{\alpha'} S \)
Are we done?

- Exponential cost!
- Can we do better?

- Idea: track relationship between $x$ and $\text{sign}$

\[
x < 0 \implies \text{sign} = -1 \\
x \geq 0 \implies \text{sign} = 1
\]

- Problem: again, expensive
- Which relationships should I track?
- Domains such as polyhedra won’t help (convex)
Trace partitioning abstract domain

- refine abstraction by adding some control history
- won’t get into that
- further reading
  - Rival and Mauborgne. The trace partitioning abstract domain. TOPLAS‘07.
  - Holley and Rosen. Qualified data flow problems. POPL'80
We can also change the program

int x, y, sign;
x = ?;
if (x<0) {
    sign = -1;
} else {
    sign = 1;
}
y = x/sign;

\[ P \]

int x, y, sign;
x = ?;
if (x<0) {
    sign = -1;
    y = x/sign;
} else {
    sign = 1;
    y = x/sign;
}

\[ P' \]
Modified Program

\[
\begin{align*}
\text{int } x, y, \text{ sign;} \\
\text{x = ?;} &\quad x \mapsto [-\infty, \infty], \text{ sign } \mapsto 1 \\
\text{if (x<0) } \{ \\
\hspace{1em} \text{sign} = -1; &\quad x \mapsto [-\infty, -1], \text{ sign } \mapsto [-1, -1] \\
\hspace{1em} y = x/\text{sign}; &\quad \theta \notin [-1, -1] \\
\} \text{ else } \{ \\
\hspace{1em} \text{sign} = 1; &\quad x \mapsto [0, \infty], \text{ sign } \mapsto [1, 1] \\
\hspace{1em} y = x/\text{sign}; &\quad \theta \notin [1, 1] \\
\} \\
x \mapsto [-\infty, \infty], \text{ sign } \mapsto [-1, 1]
\end{align*}
\]

Specification S: no division by zero.
Abstract domain \( \alpha \): intervals
Another Example

while(e) {
    s;
}

... P

if (e) {
    s;
    while(e) {
        s;
    }
}

... P'

...
Challenge: Correct and Efficient Synchronization

- Shared memory concurrent program
- No synchronization: often incorrect (but “efficient”)
- Coarse-grained synchronization: easy to reason about, often inefficient
- Fine-grained synchronization: hard to reason about, programmer often gets this wrong
Challenge: Correct and Efficient Synchronization

- Shared memory concurrent program
- No synchronization: often incorrect (but “efficient”)
- Coarse-grained synchronization: easy to reason about, often inefficient
- Fine-grained synchronization: hard to reason about, programmer often gets this wrong
Challenge: Correct and Efficient Synchronization

- Shared memory concurrent program
- No synchronization: often incorrect (but “efficient”)
- Coarse-grained synchronization: easy to reason about, often inefficient
- Fine-grained synchronization: hard to reason about, programmer often gets this wrong
Challenge

Process 1  Process 2  Process 3

How to synchronize processes to achieve correctness and efficiency?
Synchronization Primitives

- Atomic sections
- Conditional critical region (CCR)
- Memory barriers (fences)
- CAS
- Semaphores
- Monitors
- Locks
- ....
Example: Correct and Efficient Synchronization with Atomic Sections

Safety Specification: S
Example: Correct and Efficient Synchronization with Atomic Sections

Safety Specification: S

Assist the programmer by automatically inferring correct and efficient synchronization
Challenge

- Find minimal synchronization that makes the program satisfy the specification
  - Avoid all bad interleavings while permitting as many good interleavings as possible

- Assumption: we can prove that serial executions satisfy the specification
  - Interested in bad behaviors due to concurrency

- Handle infinite-state programs
Abstraction-Guided Synthesis of Synchronization

- Synthesis of synchronization via abstract interpretation
  - Compute over-approximation of all possible program executions
  - Add minimal synchronization to avoid (over-approximation of) bad interleavings

- Interplay between abstraction and synchronization
  - Finer abstraction may enable finer synchronization
  - Coarse synchronization may enable coarser abstraction
Abstraction-Guided Synthesis

Change the **program** to match the **abstraction**
AGS Algorithm – High Level

**Input:** Program P, Specification S, Abstraction $\alpha$

**Output:** Program P’ satisfying S under $\alpha$

$\varphi = true$

while(true) {

BadTraces = $\{ \pi \mid \pi \in ([P]_\alpha \cap [\varphi]) \text{ and } \pi \not\models S \}$

if (BadTraces is empty) return \textbf{implement}(P, \varphi)

select $\pi \in$ BadTraces

if (?) {

$\psi = avoid(\pi)$

if ($\psi \neq false$) $\varphi = \varphi \land \psi$

else abort

} else {

$\alpha' = refine(\alpha, \pi)$

if ($\alpha' \neq \alpha$) $\alpha = \alpha'$

else abort

}
Avoid and Implement

- Desired output – program satisfying the spec
- Implementability guides the choice of constraint language

Examples
- Atomic sections [POPL’10]
- Conditional critical regions (CCRs) [TACAS’09]
- Memory fences (for weak memory models) [FMCAD’10 + abstractions in progress]
- ...
Avoiding an interleaving with atomic sections

- Adding atomicity constraints
  - Atomicity predicate \([l_1,l_2]\) – no context switch allowed between execution of statements at \(l_1\) and \(l_2\)
  - \(\text{avoid}(\pi)\)
  - A disjunction of all possible atomicity predicates that would prevent \(\pi\)

\[\pi = A_1 B_1 A_2 B_2\]
\[\text{avoid}(\pi) = [A_1, A_2] \lor [B_1, B_2]\]
Avoid and abstraction

- $\psi = avoid(\pi)$
- Enforcing $\psi$ avoids any abstract trace $\pi'$ such that $\pi' \neq \psi$
- Potentially avoiding “good traces”
- Abstraction may affect our ability to avoid a smaller set of traces
Example

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: x += z</td>
<td>1: z++</td>
<td>1: y1 = f(x)</td>
</tr>
<tr>
<td>2: x += z</td>
<td>2: z++</td>
<td>2: y2 = x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3: assert(y1 != y2)</td>
</tr>
</tbody>
</table>

f(x) {
  if (x == 1) return 3
  else if (x == 2) return 6
  else return 5
}


Example: Concrete Values

```
x += z; x += z; z++; z++; y1 = f(x); y2 = x; assert  \rightarrow  y1 = 5, y2 = 0

z++; x += z; y1 = f(x); z++; x += z; y2 = x; assert  \rightarrow  y1 = 3, y2 = 3
```

Concrete values

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: x += z</td>
<td>1: z++</td>
<td>1: y1 = f(x)</td>
</tr>
<tr>
<td>2: x += z</td>
<td>2: z++</td>
<td>2: y2 = x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3: assert(y1 != y2)</td>
</tr>
</tbody>
</table>

```python
f(x) {
    if (x == 1) return 3
    else if (x == 2) return 6
    else return 5
}
```
Example: Parity Abstraction

Concrete values

Parity abstraction (even/odd)

\[
x += z; x += z; z++; z++; y_1 = f(x); y_2 = x; \text{assert } \Rightarrow y_1 = \text{Odd}, y_2 = \text{Even}
\]

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>f(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: x += z</td>
<td>1: z++</td>
<td>1: y_1 = f(x)</td>
<td>{</td>
</tr>
<tr>
<td>2: x += z</td>
<td>2: z++</td>
<td>2: y_2 = x</td>
<td>if (x == 1) return 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3: assert(y_1 != y_2)</td>
<td>else if (x == 2) return 6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>else return 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>
Example: Avoiding Bad Interleavings

\( \varphi = \text{true} \)

while (true) {
    BadTraces = \{ \pi | \pi \in ([P]_a \cap [\varphi]) \text{ and } \pi \not\in S \}
    if (BadTraces is empty)
        return implement(P, \varphi)
    select \( \pi \in \text{BadTraces} \)
    if (?) {
        \( \varphi = \varphi \land \text{avoid}(\pi) \)
    } else {
        \( \alpha = \text{refine}(\alpha, \pi) \)
    }
}

\( \text{avoid}(\pi_1) = [z++, z++] \)

\( \varphi = \text{true} \)
Example: Avoiding Bad Interleavings

φ = true
while (true) {
    BadTraces = {π | π ∈ ([P]_a ∩ [φ]) and π ∉ S }
    if (BadTraces is empty) {
        return implement(P, φ)
    }
    select π ∈ BadTraces
    if (?) {
        φ = φ ∧ avoid(π)
    } else {
        α = refine(α, π)
    }
}

avoid(π) = [x+=z, x+=z]
φ = [z++, z++] ∧ [x+=z, x+=z]
Example: Avoiding Bad Interleavings

\[ \varphi = \text{true} \]

while (true) {
    BadTraces = \{ \pi | \exists \pi \in ([P]_a \cap [\varphi]) \text{ and } \pi \not\in S \}
    if (BadTraces is empty)
        return implement(P, \varphi)
    select \pi \in BadTraces
    if (?) {
        \varphi = \varphi \land \text{avoid}(\pi)
    } else {
        \alpha = \text{refine}(\alpha, \pi)
    }
}

\[ \varphi = [z++, z++] \land [x+=z, x+=z] \]
Example: Avoiding Bad Interleavings

But we can also refine the abstraction...
parity
(a) T1 \( x+=z \)
   \( x+=z \)
T2 \( z++; \)
   \( z++; \)
T3 \( y1=f(x) \)
   \( y2=x \)
assert 
\( y1!= y2 \)

parity
(b) T1 \( x+=z \)
   \( x+=z \)
T2 \( z++; \)
   \( z++; \)
T3 \( y1=f(x) \)
   \( y2=x \)
assert 
\( y1!= y2 \)

parity
(c) T1 \( x+=z \)
   \( x+=z \)
T2 \( z++; \)
   \( z++; \)
T3 \( y1=f(x) \)
   \( y2=x \)
assert 
\( y1!= y2 \)

interval
(d) T1 \( x+=z \)
   \( x+=z \)
T2 \( z++; \)
   \( z++; \)
T3 \( y1=f(x) \)
   \( y2=x \)
assert 
\( y1!= y2 \)

interval
(e) T1 \( x+=z \)
   \( x+=z \)
T2 \( z++; \)
   \( z++; \)
T3 \( y1=f(x) \)
   \( y2=x \)
assert 
\( y1!= y2 \)

interval
(f) T1 \( x+=z \)
   \( x+=z \)
T2 \( z++; \)
   \( z++; \)
T3 \( y1=f(x) \)
   \( y2=x \)
assert 
\( y1!= y2 \)

octagon
(g) T1 \( x+=z \)
   \( x+=z \)
T2 \( z++; \)
   \( z++; \)
T3 \( y1=f(x) \)
   \( y2=x \)
assert 
\( y1!= y2 \)

octagon
(h) T1 \( x+=z \)
   \( x+=z \)
T2 \( z++; \)
   \( z++; \)
T3 \( y1=f(x) \)
   \( y2=x \)
assert 
\( y1!= y2 \)
Multiple Solutions

- Performance: smallest atomic sections

- Interval abstraction for our example produces the atomicity constraint:

\[
([x+=z, x+=z] \lor [z++, z++]) \land ([y_1=f(x), y_2=x] \lor [x+=z, x+=z] \lor [z++, z++])
\]

- Minimal satisfying assignments
  - \( \Gamma_1 = [z++, z++] \)
  - \( \Gamma_2 = [x+=z, x+=z] \)
AGS Algorithm – More Details

**Input:** Program P, Specification S

**Output:** Program P' satisfying S

\[ \varphi = \text{true} \]

\[
\text{while (true) } \{
\]
\[
\text{BadTraces} = \{ \pi \mid \pi \in ([P], \cap [\varphi]) \text{ and } \pi \notin S \}
\]

if (BadTraces is empty) return implement (P, \varphi)

select \pi \in BadTraces

if (?) {

\[ \psi = \text{avoid} (\pi) \]

if (\psi \neq \text{false}) \[ \varphi = \varphi \wedge \psi \]
else abort

} else {

\[ a' = \text{refine} (a, \pi) \]

if (a' \neq a) \[ a = a' \]
else abort

}

Order of selection matters

Forward Abstract Interpretation, taking \( \varphi \) into account for pruning infeasible interleavings

Backward exploration of invalid interleavings using \( \varphi \) to prune infeasible interleavings

Up to this point did not commit to a synchronization mechanism

AGS Algorithm – More Details

Order of selection matters

Forward Abstract Interpretation, taking \( \varphi \) into account for pruning infeasible interleavings

Backward exploration of invalid interleavings using \( \varphi \) to prune infeasible interleavings

Up to this point did not commit to a synchronization mechanism
Choosing a trace to avoid

T1
0: if (y==0) goto L
1: x++
2: L:

T2
0: y=2
1: x+=1
2: assert x != y

Legend:
- pc1, pc2
- x, y
Implementability

- Separation between schedule constraints and how they are realized
  - Can realize in program: atomic sections, locks,...
  - Can realize in scheduler: benevolent scheduler

- No program transformations (e.g., loop unrolling)
- Memoryless strategy
Atomic sections results

- If we can show disjoint access we can avoid synchronization
- Requires abstractions rich enough to capture access pattern to shared data

<table>
<thead>
<tr>
<th>Program</th>
<th>Refine Steps</th>
<th>Avoid Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double buffering</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Defragmentation</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>3D array update</td>
<td>2</td>
<td>23</td>
</tr>
<tr>
<td>Array Removal</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>Array Init</td>
<td>1</td>
<td>56</td>
</tr>
</tbody>
</table>
AGS with guarded commands

- Implementation mechanism: conditional critical region (CCR)

\[
\text{guard} \rightarrow \text{stmt}
\]

- Constraint language: Boolean combinations of equalities over variables \((x == c)\)

- Abstraction: what variables a guard can observe
Avoiding a transition using a guard

- Add guard to $z = y+1$ to prevent execution from state $s_1$
- Guard for $s_1$: $(x \neq 1 \lor y \neq 1 \lor z \neq 0)$
- Can affect other transitions where $z = y+1$ is executed
Example: full observability

T1
1: x = z + 1

T2
1: y = x + 1

T3
1: z = y + 1

Specification:
• \( \neg (y = 2 \land z = 1) \)
• No Stuck States

Abstraction:
\{ x, y, z \}
Build Transition System

---

Diagram showing a transition system with states and transitions labeled with equations:
- $x = z + 1$
- $y = x + 1$
- $z = y + 1$

Legend:
- PC1
- PC2
- PC3

Nodes represent states with labels:
- $1,1,1$
- $0,0,0$
- $e,1,1$
- $1,0,0$
- $1,e,1$
- $0,1,0$
- $1,1,e$
- $0,0,1$
- $1,1,e$
- $0,1,1$
- $e,e,e$
- $1,2,0$
- $e,e,e$
- $1,0,1$
- $e,1,e$
- $1,1,0$
- $e,e,1$
- $1,1,2$
- $e,e,e$
- $1,1,2$
- $e,e,e$
- $1,2,3$
- $e,e,e$
- $1,2,1$
- $e,e,e$
- $3,1,2$
- $e,e,e$
- $2,3,1$
- $e,e,e$
- $2,1,1$

Connections indicate transitions between states with the labeled equations.
Avoid transition
Result is valid

Correct and Maximally Permissive

\( x = z + 1 \)
\( y = x + 1 \)
\( z = y + 1 \)

\( x \neq 1 \lor y \neq 0 \lor z \neq 0 \Rightarrow z = y + 1 \)

\( x = z + 1 \)
\( y = x + 1 \)
\( z = y + 1 \)

\( x = z + 1 \)
\( y = x + 1 \)
\( x = z + 1 \)

\( x = z + 1 \)
\( y = x + 1 \)
\( x = z + 1 \)

\( x = z + 1 \)
\( y = x + 1 \)
\( x = z + 1 \)
Resulting program

T1
1: x = z + 1

T2
1: y = x + 1

T3
1: z = y + 1

Specification:
• \( \forall (y = 2 \land z = 1) \)
• No Stuck States

Abstraction:
\{ x, y, z \}

T1
1: x = z + 1

T2
1: y = x + 1

T3
1: (x \neq 1 \lor y \neq 0 \lor z \neq 0) \Rightarrow z = y + 1
Example: limited observability

Specification:
- \( \forall (y = 2 \land z = 1) \)
- No Stuck States

Abstraction:
\[ \{x, z\} \]
Build transition system
Avoid bad state
Select all equivalent transitions

Implementability
Result has stuck states

x = z + 1

y = x + 1

x ≠ 1 ∨ z ≠ 0 → z = y + 1

side-effects
Select transition to remove

\[ x = z + 1 \]

\[ y = x + 1 \]
\[ z = y + 1 \]
Result is valid

Correct and Maximally Permissive

\[ x = z + 1 \]
\[ y = x + 1 \]
\[ z = y + 1 \]
Resulting program

Specification:
• !(y = 2 && z = 1)
• No Stuck States

Abstraction:
{ x, z }

```
T1  T2  T3
1: x = z + 1 1: y = x + 1 1: z = y + 1

T1  T2  T3
1: (x ≠ o ∨ z ≠ o) →
  x = z + 1 1: y = x + 1 1: (x ≠ 1 ∨ z ≠ o) →
  z = y + 1
```
\[
\begin{align*}
\{x\} & \quad \text{T1} \quad x = z + 1 \quad \text{T2} \\
& \quad \text{T3} \quad y = x + 1 \quad \text{T3} \\
& \quad z = y + 1 \quad (x \neq 1) \Rightarrow z = y + 1
\end{align*}
\]

\[
\begin{align*}
\{x, z\} & \quad \text{T1} \quad x = z + 1 \quad \text{T2} \\
& \quad \text{T3} \quad y = x + 1 \quad \text{T3} \\
& \quad z = y + 1 \quad (x \neq 1 \lor z \neq 0) \Rightarrow z = y + 1
\end{align*}
\]

\[
\begin{align*}
\{x, y, z\} & \quad \text{T1} \quad x = z + 1 \quad \text{T2} \\
& \quad \text{T3} \quad y = x + 1 \quad \text{T3} \\
& \quad z = y + 1 \quad (x \neq 1 \lor y \neq 0 \lor z \neq 0) \Rightarrow z = y + 1
\end{align*}
\]
AGS with memory fences

- Avoidable transition more tricky to define
  - Operational semantics of weak memory models

- Special abstraction required to deal with potentially unbounded store-buffers
  - Even for finite-state programs

- Informally “finer abstraction = fewer fences”
Synthesis Results

- Mostly mutual exclusion primitives
- Different variations of the abstraction

<table>
<thead>
<tr>
<th>Program</th>
<th>FD $k=0$</th>
<th>FD $k=1$</th>
<th>PD $k=0$</th>
<th>PD $k=1$</th>
<th>PD $k=2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sense0</td>
<td>✓</td>
<td>☹</td>
<td>✓</td>
<td>☹</td>
<td>☹</td>
</tr>
<tr>
<td>Pet0</td>
<td>☹</td>
<td>☹</td>
<td>✓</td>
<td>✓</td>
<td>☹</td>
</tr>
<tr>
<td>Dek0</td>
<td>☹</td>
<td>☹</td>
<td>✓</td>
<td>✓</td>
<td>☹</td>
</tr>
<tr>
<td>Lam0</td>
<td>T/O</td>
<td>T/O</td>
<td>✓</td>
<td>✓</td>
<td>T/O</td>
</tr>
<tr>
<td>Fast0</td>
<td>T/O</td>
<td>T/O</td>
<td>☹</td>
<td>T/O</td>
<td>T/O</td>
</tr>
<tr>
<td>Fast1a</td>
<td>✓</td>
<td>✓</td>
<td>☹</td>
<td>☹</td>
<td>☹</td>
</tr>
<tr>
<td>Fast1b</td>
<td>✓</td>
<td>✓</td>
<td>☹</td>
<td>✓</td>
<td>☹</td>
</tr>
<tr>
<td>Fast1c</td>
<td>T/O</td>
<td>T/O</td>
<td>☹</td>
<td>✓</td>
<td>☹</td>
</tr>
</tbody>
</table>
Chase-Lev Work-Stealing Queue

```c
1 int take() {
2     long b = bottom - 1;
3     item_t * q = wsq;
4     bottom = b
5     fence("store-load");
6     long t = top
7     if (b < t) {
8         bottom = t;
9         task = q->ap[b % q->size] = task;
10        fence("store-store");
11     } else if (b > t) {
12         return task
13     }
14     if (!CAS(&top, t, t+1))
15         return ABORT;
16     return task;
}

1 int push(int task) {
2     long b = bottom;
3     long t = top;
4     item_t * q = wsq;
5     fence("load-load");
6     item_t * q = wsq;
7     if (t >= b) {
8         return EMPTY;
9     } else if (!CAS(&top, t, t+1))
10         return ABORT;
11     return task;
}

1 int steal() {
2     long t = top;
3     fence("load-load");
4     long b = bottom;
5     fence("load-load");
6     item_t * q = wsq;
7     if (t >= b) {
8         return EMPTY;
9     } else if (!CAS(&top, t, t+1))
10         return ABORT;
11     return task;
}
```

Specification: no lost items, no phantom items, memory safety
## Results

|                  | Initial | Client | $|E|$ Bound | RMO States | RMO Edges | PSO States | PSO Edges | TSO States | TSO Edges | SC States | SC Edges |
|------------------|---------|--------|----------|------------|-----------|------------|-----------|------------|-----------|-----------|-----------|
| **MSN**          | empty   | e|d      | $\infty$ | 1219      | 2671       | 455       | 743        | 1         | 228       | 316       |
|                  | empty   | e|e      | $\infty$ | 4934      | 12670      | 2678      | 6354       | 1         | 586       | 994       |
|                  | empty   | ee|dd     | $\infty$ | 24194     | 61514      | 7025      | 13689      | 2         | 1724      | 2512      |
|                  | empty   | ed|ed     | $\infty$ | 86574     | 242822     | 15450     | 35362      | 2         | 2476      | 3972      |
|                  | empty   | ed|de     | $\infty$ | 59119     | 167067     | 11023     | 24362      | 2         | 2570      | 4010      |
|                  | empty   | e|e|d     | $\infty$ | 233414    | 653094     | 51990     | 119050     | 2         | 9638      | 16822     |
| **Chase-Lev**    | empty   | pppt|(tpt|sss) | $\infty$ | 386283    | 1030857    | 74533     | 256613     | -         | 12348     | 20004     |
| **WSQ**          | empty   | tt(ttt|sss) | $\infty$ | 1048498   | 2819355    | 124455    | 255390     | -         | 6418      | 9380      |
|                  | empty   | pppt|(tpt|sss) | $\infty$ | 281314    | 878880     | 66960     | 241814     | -         | 10564     | 16317     |
|                  | empty   | ttt|(tt|sss) | $\infty$ | 1325858   | 4150650    | 361855    | 1080835    | -         | 9878      | 13956     |
|                  | empty   | ttt|(tt|sss) | $\infty$ | 280396    | 698398     | 29573     | 54696      | -         | 9197      | 14499     |
| **"LIFO"**       | 2/2     | tp|ss    | $\infty$ | 2151      | 3190       | 882       | 1171       | 1         | 676       | 852       |
| **WSQ**          | 2/2     | tp|ss    | $\infty$ | 9721      | 16668      | 3908      | 5811       | 1         | 2256      | 3116      |
|                  | 2/2     | pt|ss    | $\infty$ | 89884     | 195246     | 31289     | 64133      | 3         | 4045      | 5688      |
|                  | 2/2     | pt|ss    | $\infty$ | 85104     | 198353     | 29920     | 62020      | 3         | 4130      | 5987      |
|                  | 1/1     | pt|ss    | $\infty$ | 23913     | 48997      | 9976      | 18002      | 3         | 2353      | 3269      |
| **Dekker**       | -       | -      | 1        | 1388      | 2702       | 1388      | 2702       | 2         | 489       | 674       |
|                  | -       | -      | 10       | 7504      | 14477      | 7504      | 14477      | 2         | 2560      | 3750      |
|                  | -       | -      | 20       | 13879     | 26422      | 13879     | 26422      | 2         | 4845      | 7115      |
|                  | -       | -      | 50       | 33004     | 62257      | 33004     | 62257      | 2         | 11770     | 17210     |
| **Treiber**      | empty   | p|t      | $\infty$ | 71        | 93         | 71        | 93         | 2         | 43        | 48        |
|                  | empty   | pt|tp     | $\infty$ | 3054      | 6190       | 3041      | 6167       | 2         | 407       | 609       |
|                  | empty   | pp|tt     | $\infty$ | 1276      | 2250       | 1276      | 2250       | 2         | 325       | 407       |
| **VYSet**        | empty   | ar|ra     | 10        | 4079      | 6247       | 4079      | 6247       | 2         | 1088      | 1308      |
|                  | empty   | aa|rr     | 10        | 20034     | 31623      | 20034     | 31623      | 2         | 1168      | 1411      |
|                  | empty   | ar|ar     | 10        | 6093      | 9905       | 6093      | 9905       | 2         | 1671      | 1968      |
|                  | empty   | aaa|rrr    | 10        | 41520     | 66533      | 41520     | 66533      | 2         | 3311      | 4072      |
## Performance

| Initial State | Client | $|E|$ Bnd | Time (sec.) |
|---------------|--------|--------|-------------|
|               |        |        | total | build | form. | solve |
| MSN           | empty  | e|d     | ∞     | 0.83  | 0.45  | 0.14  | 0.23  |
|               | empty  | e|e     | ∞     | 1.78  | 1.15  | 0.54  | 0.07  |
|               | empty  | ee|dd    | ∞     | 5.21  | 2.87  | 2.12  | 0.21  |
|               | empty  | ed|ed    | ∞     | 13.05 | 9.14  | 3.73  | 0.18  |
|               | empty  | ed|de    | ∞     | 9.26  | 6.54  | 2.52  | 0.19  |
|               | empty  | e|e|d    | ∞     | 31.43 | 23.03 | 8.2   | 0.18  |
| Chase-Lev     | empty  | pppt(tp|sss) | ∞     | 97.22 | 56.1  | 41.11 | -     |
| WSQ           | empty  | tttt(pt|sss) | ∞     | 255.5 | 160.06 95.44 | -     |
|               | empty  | pppt(tp|sss) | ∞     | 90.28 | 45.17 | 45.11 | -     |
|               | empty  | tttt(tp|sss) | ∞     | 355.95 | 212.29 | 143.65 | -     |
|               | empty  | tttt(tp|tpp|sss) | ∞     | 37.98 | 31.34 | 6.63  | -     |
| "LIFO"        | 2/2    | tp|ss    | ∞     | 0.69  | 0.5   | 0.14  | 0.05  |
| WSQ           | 2/2    | tpt|ss    | ∞     | 1.94  | 1.27  | 0.61  | 0.06  |
|               | 2/2    | ptp|ss    | ∞     | 11.41 | 8.27  | 3.05  | 0.09  |
|               | 2/2    | ptt|ss    | ∞     | 11.31 | 8.26  | 2.99  | 0.06  |
|               | 1/1    | ptt|ss    | ∞     | 4.07  | 2.57  | 1.44  | 0.06  |
| Dekker        | -      | -      | 1     | 0.64  | 0.39  | 0.19  | 0.05  |
|               | -      | -      | 10    | 2.13  | 1.15  | 0.92  | 0.05  |
|               | -      | -      | 20    | 2.71  | 1.68  | 0.98  | 0.05  |
|               | -      | -      | 50    | 5.99  | 4.46  | 1.46  | 0.05  |
| Treiber       | empty  | pt|t    | ∞     | 1     | 0.07  | 0.02  | 0     |
|               | empty  | pt|tp    | ∞     | 1.02  | 0.76  | 0.26  | 0     |
|               | empty  | pp|tt    | ∞     | 0.6   | 0.46  | 0.14  | 0     |
| VYSet         | empty  | ar|ra    | 10    | 1.98  | 1.49  | 0.41  | 0.08  |
|               | empty  | aa|rr    | 10    | 4.56  | 3.41  | 1.1   | 0.05  |
|               | empty  | ar|ar    | 10    | 2.19  | 1.7   | 0.43  | 0.05  |
|               | empty  | aaa|rrr  | 10    | 7.98  | 5.61  | 2.24  | 0.13  |
Some AGS instances

<table>
<thead>
<tr>
<th>Avoid</th>
<th>Implementation Mechanism</th>
<th>Abstraction Space (examples)</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context switch</td>
<td>Atomic sections</td>
<td>Numerical abstractions</td>
<td>[POPL’10]</td>
</tr>
<tr>
<td>Transition</td>
<td>Conditional critical regions (CCRs)</td>
<td>Observability</td>
<td>[TACAS’09]</td>
</tr>
<tr>
<td>Reordering</td>
<td>Memory fences</td>
<td>Partial-Coherence abstractions for Store buffers</td>
<td>[FMCAD’10], [PLDI’11]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
General Setting Revisited

Change the specification to match the abstraction.
Summary

- Modifying the program to fit an abstraction
  - examples inspired by the trace partitioning domain
  - (trace partitioning domain can capture these effects, and more, without changing the program)
- An algorithm for Abstraction-Guided Synthesis
  - Synthesize efficient and correct synchronization
  - Handles infinite-state systems based on abstract interpretation
  - Refine the abstraction and/or restrict program behavior
  - Interplay between abstraction and synchronization
  - Separate characterization of solution from choosing optimal solutions (e.g., smallest atomic sections)