Structural counter abstraction
Proving fair-termination of depth bounded systems

Kshitij Bansal\textsuperscript{1}

with Eric Koskinen\textsuperscript{1}, Thomas Wies\textsuperscript{1}, Damien Zufferey\textsuperscript{2}

\textsuperscript{1}New York University \textsuperscript{2}IST Austria

March 18, 2013
TACAS, Rome, Italy
Model: Depth-bounded systems
Graph-rewrite based transition systems, which can be used to model concurrent heap-manipulating algorithms, as well as distributed systems.

Problem: Fair-termination problem
Fairness: If a transition is continuously enabled after some point, it is taken infinitely often.

Application: Proving progress properties of concurrent and distributed systems.
Treiber stack

```c
push(s, data):
do {
    pc1: t = s->top; x = new node(t, data);
    pc2:
} while( !CAS(s->top, t, x) );
```

Diagram:

```
stack -- Top --> node \(\rightarrow\) next node \(\rightarrow\) next \(\ldots\)
```
Treiber stack

```
push

stack → node → node → ...  
next  next  ...  

Top

pop
```
push(s, data):
   do {
      pc1: t = s->top; x = new node(t, data);
      pc2: } while( !CAS(s->top, t, x) );
Treiber stack

push(s, data):
    do {
        pc1: t = s->top; x = new node(t, data);
        pc2: }while( !CAS(s->top, t, x) );
Treiber stack

push(s, data):
    do {
        pc1: t = s->top; x = new node(t, data);
        pc2: while( !CAS(s->top, t, x) )
    }
push(s, data):
    do {
        pc1: t = s->top; x = new node(t, data);
        pc2: } while( !CAS(s->top, t, x) );
**Treiber stack**

```plaintext
push(s, data):
   do {
      pc1: t = s->top; x = new node(t, data);
      pc2: }while( !CAS(s->top, t, x) );
```
push(s, data):
  do {
    pc1: t = s->top; x = new node(t, data);
    pc2: } while( !CAS(s->top, t, x) );
push(s, data):
    do {
        pc1: t = s->top; x = new node(t, data);
        pc2: } while( !CAS(s->top, t, x) );
Lock freedom as fair termination

- Treiber stack is lock-free.
  - guarantees global progress: some thread will finish
  - individual threads might starve
- Reduced to termination problem where arbitrarily many but finite number of threads are present.
- A transition which spawns processes at will, along with a fairness constraint can be used to encode this.
Lock freedom as fair termination

- Treiber stack is lock-free.
  - guarantees global progress: some thread will finish
  - individual threads might starve
- Reduced to termination problem where arbitrarily many but finite number of threads are present.
- A transition which spawns processes at will, along with a fairness constraint can be used to encode this.

- **Challenge:** Unbounded number of heap objects and thread objects.
Contribution

- Work with symbolic graphs which can model structures that arise commonly in these systems, and required to be tracked to prove termination.

- **Contribution:** We introduce a counter abstraction derived from these, thus called *structural counter abstraction*. It is sufficiently refined to be able to prove progress properties like lock-freedom of Treiber stack.
Related work

Counter abstraction for concurrent systems

Related work

Counter abstraction for concurrent systems


Graph-based analysis

Outline

Introduction

Model (nested graphs)

Structural counter abstraction

Implementation and conclusion
Graph Transformation Systems

- States: graphs. In our case, symbolic graphs (on next slide).
- Rules: rewrite one subgraph with another.

Other rules:
- Spawn
- CAS succeed
- CAS fail

Prepare rule
Nested graphs

Node

Top

Stack

s

pc1

represent arbitrary number of copies.
Nested graphs
Nested subgraphs represent an arbitrary number of copies of the subgraphs.
Inductive invariant

prepare

...
Inductive invariant
Inductive invariant
Structural counter abstraction

**Input:** Rewrite-rules, Inductive invariant as nested graphs

**Output:** Counter system

- **Graph system** → **Counter system**
  - Nested graphs → Control locations
  - Nodes in nested graphs → Counters
  - Rule applications → Counter updates

**Soundness.** If the graph transition system has a fair non-terminating run, then the counter system will have a fair non-terminating run.
prepare: \((l_1, \{ y'_9 = 1, y'_10 = 1, y'_5 = y_5 - 1, \text{ identity on rest} \}, l_2)\)

cover: \((l_2, \{ y'_1 = y_1 + y_9, y'_9 = 0, y'_2 = y_{10} + y_2, y'_10 = 0, \text{ identity on rest} \}, l_1)\)
Computing Inductive Invariant

- Depth bounded systems: class of well-structured transition systems [Meyer, 2008]. It says if the length of the longest simple path is bounded, then system is well-structured with the ordering given by subgraph homomorphism.

- Analysis to compute over-approximation of set of reachable states of the WSTS [Ideal abstraction, Zufferey, Wies, Henzinger, 2012]. This overapproximation is a downward closed set, also inductive, given as finite union of states represented by the nested graphs.

- Many concurrent and distributed process can be modeled as depth-bounded processes for proving termination (Treiber stack without next, etc.)
Implementation

**Input**: Graph rewrite system.

1. Picasso\(^1\) computes the inductive invariant as nested graphs[ZWH'12].
2. Picasso extended to compute the counter abstraction from the invariant[this work].
3. Counter program is fed to termination prover for counter systems, ARMC [Andrey Rybalchenko, Andreas Podelski].

We also use Z3 [Leonardo de Moura, Nikolaj Bjorner] and Princess[Philipp Rümmer] for variable elimination to optimize counter abstraction.

\(^1\)http://pub.ist.ac.at/~zufferey/picasso/
Experimental Results

<table>
<thead>
<tr>
<th>Example</th>
<th>#loc</th>
<th>#v</th>
<th>#t</th>
<th>$\hat{I}$</th>
<th>$N$</th>
<th>ARMc</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Split/merge</td>
<td>4</td>
<td>3</td>
<td>9</td>
<td>1.5</td>
<td>6.8</td>
<td>0.1</td>
<td>8.4</td>
</tr>
<tr>
<td>Work stealing, 3 processors</td>
<td>4</td>
<td>4</td>
<td>20</td>
<td>1.7</td>
<td>13.1</td>
<td>0.2</td>
<td>15.0</td>
</tr>
<tr>
<td>Work stealing, parameterized</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1.5</td>
<td>5.6</td>
<td>0.1</td>
<td>6.2</td>
</tr>
<tr>
<td>Compute server job queue</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>1.6</td>
<td>6.1</td>
<td>0.1</td>
<td>7.8</td>
</tr>
<tr>
<td>Chat room</td>
<td>5</td>
<td>34</td>
<td>80</td>
<td>9.8</td>
<td>61.3</td>
<td>5 min</td>
<td>6 min</td>
</tr>
<tr>
<td>Map reduce</td>
<td>6</td>
<td>10</td>
<td>15</td>
<td>2.0</td>
<td>8.8</td>
<td>0.2</td>
<td>11.0</td>
</tr>
<tr>
<td>Map reduce with failure</td>
<td>6</td>
<td>15</td>
<td>21</td>
<td>2.3</td>
<td>11.1</td>
<td>0.9</td>
<td>14.3</td>
</tr>
<tr>
<td>Treiber’s stack (coarse-grained)</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>1.9</td>
<td>7.2</td>
<td>0.2</td>
<td>9.3</td>
</tr>
<tr>
<td>Treiber’s stack (fine-grained)</td>
<td>3</td>
<td>14</td>
<td>13</td>
<td>2.7</td>
<td>14.2</td>
<td>1.2</td>
<td>17.1</td>
</tr>
<tr>
<td>Herlihy/Wing queue</td>
<td>3</td>
<td>16</td>
<td>25</td>
<td>3.8</td>
<td>24.9</td>
<td>6.5</td>
<td>34.2</td>
</tr>
<tr>
<td>Michael/Scott queue (dequeue only)</td>
<td>4</td>
<td>7</td>
<td>23</td>
<td>2.8</td>
<td>13.0</td>
<td>0.6</td>
<td>16.4</td>
</tr>
<tr>
<td>Michael/Scott queue (enqueue only)</td>
<td>7</td>
<td>15</td>
<td>53</td>
<td>3.8</td>
<td>43.7</td>
<td>7.6</td>
<td>55.1</td>
</tr>
<tr>
<td>Michael/Scott queue</td>
<td>9</td>
<td>31</td>
<td>224</td>
<td>25.0</td>
<td>265.0</td>
<td>3 wks</td>
<td>3 wks</td>
</tr>
</tbody>
</table>

Table: The columns show the number of locations, variables, and transitions in the counter abstraction, and the running times, in seconds, for computing the inductive invariant, constructing the abstraction, and for proving termination.
Related work

Conclusion

- Novel technique for proving fair termination of DBS that can be used to prove progress properties of concurrent data structures and distributed systems.
- An analysis that is both practical and sufficiently precise built on top of existing termination provers for counter systems.