Coordination and safe program execution: problems, solutions and open issues.

Minas Charalambides

October 29, 2014
Motivation

Parallel programming

Too many schedules lead to
- Atomicity violations
- Data races
- Deadlocks
Parallel programming

Too many schedules lead to
- Atomicity violations
- Data races
- Deadlocks
Idea: Restrict the set of schedules.

Before
- Locks
- Synchronizers
- Synchronization languages

Recently
- Data-centric synchronization
- Session types
PickUpConstraint(c1, c2, phil)
{
    atomic((c1.pick(sender) where sender = phil),
           (c2.pick(sender) where sender = phil)),
    (c1.pick where sender = phil) stops
}
class Buffer {
    int size = 0;
    Object[] array = new Object[10];

    void put(Object o) [size < 0] {
        array[size++] = obj;
    }

    Object get() [size > 0] {
        return array[--size];
    }
}
class ArrayList {
    int size;
    Object[] entries;

    Object get(int i) {
        synchronized(lock) {
            if (0 <= i && i < this.size) {
                return this.entries[i];
            } else {
                return null;
            }
        }
    }

    void addAll(ArrayList al) {
        synchronized(lock) {
            this.size += al.size;
            // copy elements
        }
    }
}
class ArrayList {
    atomicset(L);
    atomic(L) int size;
    atomic(L) Object[] entries;

    Object get(int i) {
        if (0 <= i && i < this.size) {
            return this.entries[i];
        } else {
            return null;
        }
    }

    void addAll(unitfor(L) ArrayList al) {
        this.size += al.size;
        // copy elements
    }
}
Converting a Program to Atomic Sets Requires Understanding its Concurrency Structure

- Data-centric synchronization looks beneficial
- But: must *understand* old synchronization to convert it

**Conversion Experience of Dolby et al.**
- Takes several hours for rather simple programs
- 2 out of 6 programs lack synchronization of some classes
- 2 out of 6 programs accidentally introduced global locks

**Our Algorithm**

Avoid conversion errors by automatically determining annotations from program traces.
First attempt for automated conversion

Algorithm Strategy and Properties
- Based on set membership for classification
- Analysis done during replay of observation trace
- Observation data scales in size of execution

Evaluation
- Qualitative evaluation shows promising results and room for improvement
- Inferred annotations mostly agree with manual annotations
- Bugs found in manual annotations

Desirable Properties of Atomic Set Inference Algorithms Using Traces

- Take observation *distance* and *scope* into account
- Produce stable annotations that do not vary wildly between different runs
- Observation data bounded by size of codebase
- Produce atomic set estimates on-the-fly
- Improve analysis with longer execution traces
Second attempt: probabilistic inference of atomic sets

Assumptions about Input Programs

- Methods perform semantically meaningful operations
- Data fields that a method accesses are likely connected by invariant

Algorithm Idea

- Observe which pairs of fields a method accesses atomically
  - This is (Bayesian) evidence that fields are connected through a semantic invariant
- Store current beliefs for all field pairs in affinity matrices

P. Dinges, K. Palmskog, G. Agha – work in progress
### Indirect Access and Distance

- High-level semantic operations use low-level operations
- Example: `get()` might call `getSize()` instead of accessing field `size`
- Propagate observed access to caller’s scope
- Quantify directness of access as *distance*
The Analysis Algorithm Must Support Indirect Field Access and Access Paths

**Indirect Access and Distance**
- High-level semantic operations use low-level operations
- Example: `get()` might call `getSize()` instead of accessing field `size`
- Propagate observed access to caller’s scope
- Quantify directness of access as *distance*

**Access Paths**
- Methods traverse the object graph
- Track *access paths* instead of field names
- Example: `this.urls.size`
Bayesian Inference Variables

\(H\): “\(f\) and \(g\) are connected through an invariant” [Hypothesis]
\(e_k\): “\(f\), \(g\) accessed (non-)atomically with distance \(d_k\)” [evidence]

Consider a sequence of observations \(e_1, \ldots, e_n\) w.r.t. \(f\) and \(g\).
Want to know probability that \(H\) holds given \(e_1, \ldots, e_n\).
Consider a sequence of observations $e_1, \ldots, e_n$ w.r.t. $f$ and $g$. Want to know *probability that $H$ holds given $e_1, \ldots, e_n$*

$$\frac{P(H|e_1, \ldots, e_n)}{P(-H|e_1, \ldots, e_n)} = \frac{P(e_1, \ldots, e_n|H)}{P(e_1, \ldots, e_n|-H)} \times \frac{P(H)}{P(-H)}$$

 updated info = info from observations $\times$ original info

 posterior odds = likelihood ratio $\times$ prior odds

$$O(H|e_1, \ldots, e_n) = L(e_1, \ldots, e_n|H) \times O(H)$$
Conditional Independence

If $e_1, \ldots, e_n$ are conditionally independent given $H$, we can write

$$P(e_1, \ldots, e_n|H) = \prod_{k=1}^{n} P(e_k|H)$$

and similarly for $\neg H$, whereby

$$O(H|e_1, \ldots, e_n) = O(H) \prod_{k=1}^{n} L(e_k|H)$$

Adding one more piece of evidence $e_{n+1}$, we get

$$O(H|e_1, \ldots, e_n, e_{n+1}) = L(e_{n+1}|H) O(H|e_1, \ldots, e_n)$$

Hence, if we have independence, know $O(H)$, and can compute $L(e_k|H)$, we can update odds on-the-fly when observing!

Coarse-grained hypothesis space: $H \cup \neg H$

With conditional independence, $e_1, \ldots, e_n$ should depend only on hypothesis, not on systematic external influence.

However, we have at least the following external factors:
- Workload
- Scheduler
Working assumption: good workload and long executions minimize external influence

Currently comparing results of instrumented workload with plain workloads

Given good enough driver programs (test suites) that run long enough, scheduler and driver influence becomes minimal

It is always safe to place $f$ and $g$ in an atomic set even if no invariant holds
For all $f$ and $g$, we let

$$O(H) = \frac{P(H)}{P(\neg H)} = 1$$

Then, $f$ and $g$ equally likely \textit{in} as \textit{not in} atomic set.
The effect of access distance

Coordination: solutions and open issues.
Given access observation $e_k$ for fields $f$ and $g$ with operation distance $d$, need to compute $L(e_k|H)$.

$L(e_k|H)$ should increase as $d$ decreases up to some maximum, after which it is flat.

$L(e_k|H)$ should decrease as $d$ increases down to some minimum, after which it is flat.
Given access observation $e_k$ for fields $f$ and $g$ with operation distance $d$, need to compute $L(e_k|H)$.

$L(e_k|H)$ should increase as $d$ decreases up to some maximum, after which it is flat.

$L(e_k|H)$ should decrease as $d$ increases down to some minimum, after which it is flat.

We use a linear approximation of a logistic curve.
To achieve good results in practice:

- How much should prior odds be revised when observing interleaving access?
- How steep should the logistic curve be?
- How much penalty to distance should, e.g., call and return statements give?
Advantages of On-the-Fly Bayesian Inference

- Likelihoods incorporate scope and distance of observations
- Beliefs can be revised by new evidence, and thus improve with longer executions
- Analysis becomes robust and insensitive to outlier observations
- Size of observation data is in the size of the codebase, not size of execution
- Infers aliases similarly to atomic sets, which is hard to do statically
Implementation Toolchain

start → Program

Instr. Bytecode → Affinity matrices → Atomic sets

Workload

Aliases

Coordination: solutions and open issues.
**Evaluation Strategy:** Qualitative Comparison Against Manual Annotations of 6 Programs

### Qualitative Evaluation
- Currently comparing inferred annotations against manual annotations for 6 programs
- Manual annotations by AJ developers

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
<th>kLoC</th>
<th>Cls.</th>
</tr>
</thead>
<tbody>
<tr>
<td>collections</td>
<td>OpenJDK collections</td>
<td>11.1</td>
<td>171</td>
</tr>
<tr>
<td>elevator</td>
<td>Elevator simulation</td>
<td>0.3</td>
<td>6</td>
</tr>
<tr>
<td>jcurzez1</td>
<td>Console window library</td>
<td>2.7</td>
<td>78</td>
</tr>
<tr>
<td>jcurzez2</td>
<td>Console window library</td>
<td>2.8</td>
<td>79</td>
</tr>
<tr>
<td>tsp2</td>
<td>Traveling salesman</td>
<td>0.5</td>
<td>6</td>
</tr>
<tr>
<td>weblech</td>
<td>Web site crawler</td>
<td>1.3</td>
<td>12</td>
</tr>
</tbody>
</table>
Result: Inferred Annotations are Similar to Manual Annotations

- Inferred annotations mostly agree with manual annotations
- Mistakes in manually annotated programs were revealed in some cases
- Only a few mistakes, due to workload (fuzzer) incompleteness
Session types

- Apply to all parallel programming models
- We focus on Actors
- Determine adherence to communication protocols at compile time
Example - Limited resource sharing

client 1 \ldots \text{client } k

(server)

\begin{itemize}
\item \text{(lock-ack-unlock)}^*
\end{itemize}

n \text{ resources}
Example - Continued

// globals
param n
param k
struct lock[n] = { }
struct unlock[n] = { }
struct ack[n] = { }

// server
actor s = {
  spawn (i = 1..n
    message l : lock[i]
    message a : ack[i]
    message u : unlock[i]
  while NotDone {
    select (j = 1..k
      recv(c[j], l);
      send(c[j], a);
      recv(c[j], u);
    })
  })
}

// client
actor c[k] = {
  spawn (i = 1..n
    message l : lock[i]
    message a : ack[i]
    message u : unlock[i]
  while NotDone {
    send(s, l);
    recv(s, a);
    // compute ...
    send(s, u);
  })
}
Parameterization

Example (System-A type)

\[
\prod_{i=1}^{n} \left( \bigoplus_{j=1}^{k} \left( c_j \xrightarrow{\text{lock}_i} s ; s \xrightarrow{\text{ack}_i} c_j ; c_j \xrightarrow{\text{unlock}_i} s \right) \right)^{*}
\]

client 1 \ldots client k

(lock-ack-unlock)^* 

server

n resources
System-A Global type syntax

\[ G :::= \quad a \xrightarrow{t} b \quad (G\text{-Interaction}) \quad | \quad G ; G \quad (G\text{-Seq}) \]
\[ \quad | \quad G \oplus G \quad (G\text{-Choice}) \quad | \quad \bigoplus_{i=1}^{n} G_i \quad (G\text{-Choice}) \]
\[ \quad | \quad G \parallel G \quad (G\text{-Parallel}) \quad | \quad \bigparallel_{i=1}^{n} G_i \quad (G\text{-Parallel-N}) \]
\[ \quad | \quad G \otimes G \quad (G\text{-Shuffle}) \quad | \quad \bigotimes_{i=1}^{n} G_i \quad (G\text{-Shuffle-N}) \]
\[ \quad | \quad G^* \quad (G\text{-KleeneStar}) \quad | \quad \bigparallel G \quad (G\text{-Parallel-Star}) \]
\[ \quad | \quad (G) \quad (G\text{-Paren}) \]

Charalambides et al. *Parameterized Concurrent Multi-Party Session Types.* FOCLASA 2012
Previously

How to infer types?
Previously

Like this!

\[
\begin{align*}
\text{(Inf-Select-N)} & \quad S \vdash c_1 : \text{cons} \quad S \vdash c_2 : \text{cons} \\
& \quad \Gamma \vdash R[i/k] : L_i \quad k \in \text{fv}(R) \\
& \quad \text{first}(L_i) = x?y \quad x = a_i \lor y = m_i \\
& \quad \Gamma \vdash \text{select}(k = c_1..c_2 R) : (\bigoplus_{i = c_1}^{c_2} L_i)
\end{align*}
\]

\[
\begin{align*}
\text{(Inf-Select)} & \quad \Gamma \vdash R_1 : L_1 \quad \Gamma \vdash R_2 : L_2 \\
& \quad \text{first}(L_1) = a?m \quad \text{first}(L_2) = a'?m' \\
& \quad a \neq a' \lor m \neq m' \\
& \quad \Gamma \vdash \text{select}(R_1 R_2) : (L_1 \oplus L_2)
\end{align*}
\]

\[
\begin{align*}
\text{(Inf-Repeat)} & \quad \Gamma \vdash R : L \quad S \vdash n : \text{cons} \\
& \quad \Gamma \vdash \text{repeat } n R : (L^*)
\end{align*}
\]

\[
\begin{align*}
\text{(Inf-While)} & \quad \Gamma \vdash R : L \quad S \vdash \text{exp} : \text{Boolean} \\
& \quad \Gamma \vdash \text{while } \text{exp } R : (L^*)
\end{align*}
\]

\[
\begin{align*}
\text{(Inf-If)} & \quad \Gamma \vdash R_1 : L_1 \quad \Gamma \vdash R_2 : L_2 \\
& \quad S \vdash \text{exp} : \text{Boolean} \\
& \quad \Gamma \vdash \text{if } \text{exp } R_1 \text{ else } R_2 : (L_1 \oplus L_2)
\end{align*}
\]

\[
\begin{align*}
\text{(Inf-Environment)} & \quad \Pi = \{a \mid \text{“actor } a = \{R_a\} \in P\} \\
& \quad \forall a \in \Pi, \quad \Gamma \vdash R_a : L_a \\
& \quad \Gamma \vdash P : \Delta \text{ where } \Delta = \{a : L_a \mid a \in \Pi\},
\end{align*}
\]
Previously

- All required information lies in program syntax
- Makes for an awkward programming model
Now

Current work

- More natural programming model
- Actual actor language
- Static analysis infers global types
- Dynamic process creation
Parallel programming is hard. Need to restrict schedules.

From control-centric to data-centric synchronization

Session types

Session type inference

Open problem: Actor creation
Thank you.

Questions?