Concurrent Programming via Access Permissions

Sven Stork

Carnegie Mellon University
svens@cs.cmu.edu
Supervisor: Jonathan Aldrich

University of Coimbra
stork@dei.uc.pt
Supervisor: Paulo Marques

October 7, 2009
Motivation
Era of Concurrency is upon us!

- concurrency is (basically) everywhere
- major paradigm shift in technology
- “The free lunch is over” (Sutter, 2005)
- programs need to become concurrent to achieve higher performance
The Problem

What is the problem with concurrent programming?

explicit concurrent programming

- **default** in mainstream object oriented languages
- “manual” management of concurrent (e.g., threads, locks, …)
- concurrent issues **additionally** to sequential problems
- **complex** and **error prone** reasoning about ordering constraints

implicit concurrent programming

- tell the system **what to do** and **not how to do it**
- functional programming languages (**no state**)  
- works well for certain domains (e.g., data parallel)
  - does not work for general purpose tasks (e.g., web server)
The Problem

Research Questions

- Can you have an implicit concurrency model for a general purpose object oriented programming language?
  - Can you have a language that automatically infers concurrency information?
  - Can you have a language that protect against common pitfalls (e.g., race conditions)?

- What are the requirements for a runtime systems, to execute such programs efficiently?
  - How to use the mixture fine - coarse grain concurrency information to execute a program efficiently?

- How much concurrency can be extracted via such a high level abstraction?
  - How much performance improvements can real world applications achieve?
Approach
Approach

Research Question

- Can you have an implicit concurrency model for a general purpose object oriented programming language?
  - Can you have a language that automatically infers concurrency information?
  - Can you have a language that protect against common pitfalls (e.g., race conditions)?

Our Approach

- **Problem**: lack of exact dependency information caused by aliasing
- **Remedy**: use access permissions
  1. provide aliasing and data access information
  2. calculate more precise dependencies
  3. infer concurrency information obeying dependencies (dataflow graph)
  4. check against correct usage
Approach

Research Question

- Can you have an implicit concurrency model for a general purpose object oriented programming language?
  - Can you have a language that automatically infers concurrency information?
  - Can you have a language that protects against common pitfalls (e.g., race conditions)?

Our Approach

- Problem: lack of exact dependency information caused by aliasing
- Remedy: use access permissions
  - provide aliasing and data access information
  - calculate more precise dependencies
  - infer concurrency information obeying dependencies (dataflow graph)
  - check against correct usage

programming paradigm: concurrency by default
programming language: ÆMINIUM
# Access Permissions

## What kind of Access Permissions do we use?

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| **unique** | - there is only one reference to the object  
               - exclusive access  
               - no synchronization required |
| **immutable** | - the object **cannot** be modified through an immutable reference  
               - there might be several alias reference to the object, but all of them are immutable  
               - no synchronization required |
| **shared** | - the object **can** be modified through an shared reference  
               - there might be several alias reference to the object, but all of them are shared  
               - access to shared objects requires synchronization |
## Access Permissions

**What kind of Access Permissions do we use?**

<table>
<thead>
<tr>
<th>Access Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>unique</strong></td>
<td>there is only one reference to the object</td>
</tr>
<tr>
<td></td>
<td>exclusive access</td>
</tr>
<tr>
<td></td>
<td>no synchronization required</td>
</tr>
<tr>
<td><strong>immutable</strong></td>
<td>the object <em>cannot</em> be modified through an immutable reference</td>
</tr>
<tr>
<td></td>
<td>there might be several alias reference to the object, but all of them are immutable</td>
</tr>
<tr>
<td></td>
<td>no synchronization required</td>
</tr>
<tr>
<td><strong>shared</strong></td>
<td>the object <em>can</em> be modified through a shared reference</td>
</tr>
<tr>
<td></td>
<td>there might be several alias reference to the object, but all of them are shared</td>
</tr>
<tr>
<td></td>
<td>access to shared objects requires synchronization</td>
</tr>
</tbody>
</table>

*Concurrent Programming via Access Permissions*
Access Permissions

How to extract concurrency out of access permissions?

- Access Permissions encode data access and aliasing information
  - automatically splitting/joining of permissions
    e.g., unique ⇔ immutable ⊗ immutable
    e.g., unique ⇔ shared ⊗ shared
  - use linear logic for management access permissions
- “reverse” this approach and infer which operations can be executed concurrently

We define that ...

- pieces of code can be executed concurrently, iff they dependent on
  immutable permissions ⇝ only read operations
  shared permissions ⇝ access must synchronized
- we propose data groups for specifying additional ordering
Motivation

Approach

Related Work

Road Ahead

Access Permissions

**Program - main**

```c
void main() {
    Collection c = readData()
    printCollection(c)
    Statistics s = compStats(c)
    Dependencies d = compDeps(c)
    removeDuplicates(c)
    printCollection(c)
    ...
}
```

Question

Is there concurrency in this program?

Concurrent Programming via Access Permissions
Access Permissions

Program - main

```java
void main() {
    Collection c = readData()
    printCollection(c)
    Statistics s = compStats(c)
    Dependencies d = compDeps(c)
    removeDuplicates(c)
    printCollection(c)
...
}
```

Question

- Is there concurrency in this program?
Access Permissions

Program - Types

Collection readData()
: () → Collection

void removeDuplicates(Collection c)
: Collection → ()

void printCollection(Collection c)
: Collection → ()

Dependencies compDeps(Collection c)
: Collection → Dependencies

Statistics compStats(Collection c)
: Collection → Statistics
Access Permissions

Program - Types

Collection readData()
: () → Collection

void removeDuplicates(Collection c)
: Collection → ()

void printCollection(Collection c)
: Collection → ()

Dependencies compDeps(Collection c)
: Collection → Dependencies

Statistics compStats(Collection c)
: Collection → Statistics

Question
- Typing specifies input/output of behavior
- Q: Can we run compDeps and compStats?
- A: Depends if there are dependencies between them.
Program - Access Permissions

Collection readData()
  : unit ⇒ unique(result)

void removeDuplicates(Collection c)
  : unique(c) ⇒ unique(c)

void printCollection(Collection c)
  : immutable(c) ⇒ immutable(c)

Dependencies compDeps(Collection c)
  : immutable(c) ⇒ immutable(c), unique(result)

Statistics compStats(Collection c)
  : immutable(c) ⇒ immutable(c), unique(result)
Access Permissions

Program - Access Permissions

Collection readData()
  : unit ⇒ unique(result)

void removeDuplicates(Collection c)
 : unique(c)⇒ unique(c)

void printCollection(Collection c)
 : immutable(c)⇒ immutable(c)

Dependencies compDeps(Collection c)
 : immutable(c)⇒ immutable(c), unique(result)

Statistics compStats(Collection c)
 : immutable(c)⇒ immutable(c), unique(result)

Solution
- permissions specify data access
- compStats and compDeps do not and cannot modify the collection
- permissions specification describes contract
Access Permissions

- Motivation
- Approach
- Related Work
- Road Ahead

Concurrent Programming via Access Permissions
Access Permissions

Motivation

Approach

Related Work

Road Ahead

Concurrent Programming via Access Permissions
### Research Question

- What are the requirements for a runtime systems, to execute such programs efficiently?

  ⇝ How to use the **mixture fine - coarse grain concurrency information to execute a program efficiently**?

### Possible Approaches

- using a **cost semantics** to estimate best granularity (static)
- system dependent **cost model** at start of program (semi-dynamic)
- **profiling** and **adaption** during **runtime** (dynamic)
- a bit of all above …
Approach

Research Question

- How much concurrency can be extracted via such a high level abstraction?
- How much performance improvements can real world applications achieve?

Possible Approaches

- depends on granularity … (see previous question)
- build prototype and re-(write) real world applications and benchmark
Standing on the Shoulders of Giants
## Standing on the Shoulders of Giants

### Access Permissions

<table>
<thead>
<tr>
<th>Name</th>
<th>Contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boyland</td>
<td>- Verification of concurrent programs (simple system with just unique/immutable)</td>
</tr>
<tr>
<td></td>
<td>- Verification of correct lock usage (nested permissions)</td>
</tr>
<tr>
<td>Bierhoff</td>
<td>- Verification of API protocol conformance</td>
</tr>
<tr>
<td>Beckman</td>
<td>- Verification of correct usage of atomic blocks</td>
</tr>
<tr>
<td></td>
<td>- Optimizations of Software Transactional Memory system</td>
</tr>
<tr>
<td>Terauchi</td>
<td>- Verification of correct usage of locks</td>
</tr>
</tbody>
</table>
## Standing on the Shoulders of Giants

### Concurrent Programming Languages

<table>
<thead>
<tr>
<th>Language</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fortress</strong></td>
<td>- concurrent by default semantics for loops and tuples</td>
</tr>
<tr>
<td></td>
<td>- no check for correct synchronization</td>
</tr>
<tr>
<td><strong>Axum</strong></td>
<td>- programs as dataflow graph</td>
</tr>
<tr>
<td></td>
<td>- manual generation of graph</td>
</tr>
<tr>
<td><strong>ML, etc.</strong></td>
<td>- exact dependency information</td>
</tr>
<tr>
<td></td>
<td>- no state</td>
</tr>
<tr>
<td><strong>NESL, etc.</strong></td>
<td>- implicit concurrency model</td>
</tr>
<tr>
<td></td>
<td>- largely limited to domain specific areas</td>
</tr>
</tbody>
</table>
Road Ahead
Road Ahead

Current Status

- worked on AtomicPower!
  - use access permissions to optimize STM
- developed core concepts of ÆMINIUM
  - concurrency by default
  - access permissions
  - data groups
- developed core language grammar
- published a position paper in Onward! conference
### Road Ahead

<table>
<thead>
<tr>
<th>Next Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Core Language</strong> (2\textsuperscript{nd} – 3\textsuperscript{rd} year)</td>
</tr>
<tr>
<td>- develop core language calculus</td>
</tr>
<tr>
<td>- proof core language calculus</td>
</tr>
<tr>
<td><strong>Runtime System</strong> (3\textsuperscript{nd} – 4\textsuperscript{th} year)</td>
</tr>
<tr>
<td>- first step develop interpreter to gather experience</td>
</tr>
<tr>
<td>- implement efficient runtime system</td>
</tr>
<tr>
<td>- alternatively find existing runtime system</td>
</tr>
<tr>
<td>- Axum, Dryad, …</td>
</tr>
<tr>
<td><strong>Evaluation</strong> (4\textsuperscript{th} – 5\textsuperscript{th} year)</td>
</tr>
<tr>
<td>- evaluate performance via application benchmarks</td>
</tr>
<tr>
<td>- user studies to evaluate usability and 'real world code'</td>
</tr>
</tbody>
</table>
Impact ... What if we achieve Nirvana ...

- Develop an programming languages (approach) that helps to write less error prone concurrent code.

- Educate or change mind set of programmers:
  - Permission require some rigor when designing interfaces
    \[\Rightarrow\] better design
  - Reasoning about dependencies instead of execution orders

- An innovative runtime system, that executes programs as efficient as dependencies and hardware permits.
Summary

Concurrent Programming via Access Permissions
Summary

Proposed a new programming paradigm: Concurrency by Default
- use access permissions and data groups to specify dependencies and extract concurrency information

ÆMINIUM a new programming language based on concurrency by default
- identified core features
- developed grammar

Future Work:
- develop and proof formal system of ÆMINIUM
- develop an efficient runtime system for ÆMINIUM
- evaluation of the system and the overall approach
Thanks for the attention!
Questions?