KDB: debugger backends: ddb
KDB: current backend: ddb
kernel trap 12 with interrupts disabled

Fatal trap 12: page fault while in kernel mode
cpuid = 0; apic id = 00
fault virtual address = 0xffffffff00000000
fault code = supervisor write data, page not present
instruction pointer = 0x8:0xffffffff0051e885
stack pointer = 0x10:0xffffffff83d743e0
frame pointer = 0x10:0xffffffff83d74408
code segment = base 0x0, limit 0xffffffff, type 0x1b
                DPL 0, pres 1, long 1, def32 0, gran 1
processor eflags = resume, IOPL = 0
current process = 0 ()
[thread pid 0 tid 0 ]
Stopped at pmap_bootstrap+0x322: movq $0, (%r6)
db> bt
Tracing pid 0 tid 0 td 0xffffffff83da3030
pmap_bootstrap() at pmap_bootstrap+0x322
hammer_time() at hammer_time+0xbf7
btext() at btext+0x24
db>
Smash the state
Smash the state, one field at a time
Incremental Interactive Verification of the Correctness of Object-oriented Software

Hannes Mehnert

IT University of Copenhagen

11th October 2013
Incremental Interactive Verification of the Correctness of Object-oriented Software
Incremental Interactive Verification of the Correctness of Object-oriented Software

- Modeling using a pure total language
- Proving that implementation fulfills model
Wait! What?

Incremental Interactive Verification of the Correctness of Object-oriented Software

- Modeling using a pure total language
- Proving that implementation fulfills model
- Aliasing and shared mutable state
Incremental Interactive Verification of the Correctness of Object-oriented Software

- Modeling using a pure total language
- Proving that implementation fulfills model
- Aliasing and shared mutable state
- Handling one class at a time
Incremental Interactive Verification of the Correctness of Object-oriented Software

- Modeling using a pure total language
- Proving that implementation fulfills model
- Aliasing and shared mutable state
- Handling one class at a time
- Guiding discharge of proof obligations
1 Introduction

2 Kopitiam

3 Case studies

4 Aftermath
- Verify the correctness of object-oriented software
- PI: Lars Birkedal and Peter Sestoft
- Grant from Danish Research Council for Technology and Production
Extensions to separation logic (Jonas Buhrkal Jensen)
Coq formalisation of higher-order separation logic (Jonas, Jesper Bengtson, Filip Sieczkowski)
Tools and Methods for Scalable Software Verification

- Charge!
- Foundations
- Case Studies

- Snapshotable trees, composite, planar point location (Jesper, Filip, Hannes Mehnert)
Tools and Methods for Scalable Software Verification

- Charge!
- Kopitiam
- Foundations
- Case Studies

- Extensions to separation logic (Jonas Buhrkal Jensen)
- Coq formalisation of higher-order separation logic (Jonas, Jesper Bengtson, Filip Sieczkowski)
- Snapshotable trees, composite, planar point location (Jesper, Filip, Hannes Mehnert)

- Eclipse plugin (Hannes)
Our approach to verify the correctness

- Hoare logic for imperative programs
Our approach to verify the correctness

- Hoare logic for imperative programs
  - \( \{ x = 2 \} \ x := 5 \ { x = 5 \} \)
Our approach to verify the correctness

- Hoare logic for imperative programs
  - $\{x = 2\} \ x := 5 \ \{x = 5\}$
- Separation logic
Our approach to verify the correctness

- Hoare logic for imperative programs
  - \{x = 2\} x := 5 \{x = 5\}
- Separation logic
  - \(P \ast P'\)
Our approach to verify the correctness

- Hoare logic for imperative programs
  - \( \{ x = 2 \} \ x := 5 \ \{ x = 5 \} \)

- Separation logic
  - \( P \ast P' \)
  - \( o.f \mapsto v \)
Our approach to verify the correctness

- Hoare logic for imperative programs
  - \( \{ x = 2 \} \ x := 5 \ \{ x = 5 \} \)

- Separation logic
  - \( P * P' \)
  - \( o.f \mapsto v \)

\[
\frac{\{ P \} \ s \ \{ Q \} \quad \forall x \in \text{fv} \ R. \ s \text{ does not modify } x}{\{ P * R \} \ s \ \{ Q * R \}} \quad \text{FRAME}
\]
Our approach to verify the correctness

- Hoare logic for imperative programs
  - \( \{ x = 2 \} \ x := 5 \ \{ x = 5 \} \)

- Separation logic
  - \( P \ast P' \)
  - \( o.f \mapsto v \)
  \[
  \frac{\{P\} s \{Q\} \quad \forall x \in fv \ R. \ s \text{ does not modify } x}{\{P \ast R\} s \{Q \ast R\}} \quad \text{FRAME}
  \]

- Proof assistant Coq
Research Areas of this Dissertation

- Formal verification
- Separation logic
Part II

- Formal verification
- Separation logic
- Software engineering
Part II

- Formal verification
- Separation logic
- Software engineering
- Plug-in development, user interfaces for theorem proving
Part II

- Formal verification
- Separation logic
- Software engineering
- Plug-in development, user interfaces for theorem proving
- Feature model, feature analysis
Part II

- Formal verification
- Separation logic
- Software engineering
- Plug-in development, user interfaces for theorem proving
- Feature model, feature analysis
- Software evolution, software lifecycle
Part II

- Formal verification
- Separation logic
- Software engineering
- Plug-in development, user interfaces for theorem proving
- Feature model, feature analysis
- Software evolution, sofware lifecycle
- Empirical evaluation (Technology Acceptance Model)
Part III

- Formal verification
- Separation logic
- Software engineering
- Plug-in development, user interfaces for theorem proving
- Feature model, feature analysis
- Software evolution, software lifecycle
- Empirical evaluation (Technology Acceptance Model)
- Copy-on-write data structure
- Non-observable sharing
Part III

- Formal verification
- Separation logic
- Software engineering
- Plug-in development, user interfaces for theorem proving
- Feature model, feature analysis
- Software evolution, software lifecycle
- Empirical evaluation (Technology Acceptance Model)
- Copy-on-write data structure
- Non-observable sharing
- Geometry
Part III

- Formal verification
- Separation logic
- Software engineering
- Plug-in development, user interfaces for theorem proving
- Feature model, feature analysis
- Software evolution, software lifecycle
- Empirical evaluation (Technology Acceptance Model)
- Copy-on-write data structure
- Non-observable sharing
- Geometry
- Typestate
- Access Permissions
Modular interactive verification of correctness of object-oriented software is achievable and can be tightly integrated into the software development workflow.

Contributions:

- *Kopitiam* – feature model, design, implementation, evaluation
- Case studies
  - Formalised proof of snapshotable tree data structure
  - Informal proof of planar point location
Motivation for this Dissertation

If I can’t click it, it doesn’t exist

Tool development is crucial for testing ideas
1 Introduction

2 Kopitiam

3 Case studies

4 Aftermath
- *kopi* is Malay/Hokkien for coffee
- *tiam* is Hokkien/Hakka for shop
- Integration of development of correctness proofs and programs
- Standard IDE features and tools still in place
Kopitiam development timeline

July 4th 2010 - October 6th 2013
Commits to master, excluding merge commits

Contribution Type: Commits
Feature Model of Verification Tools

Verification Tool
Feature Model of Verification Tools

- Trusted Code Base
- User Interface
- Verification Tool
- Verification Back-end
- Target Language
- Specification Logic
Feature Model of Verification Tools

- Algorithms Proven
- Custom Prover
- LCF
- Trusted Code Base
- Verification Tool
  - User Interface
  - Verification Back-end
  - Specification Logic
  - Target Language
Feature Model of Verification Tools

- Algorithms Proven
- Custom Prover
- LCF
- Trusted Code Base
- Verification Tool
- Automated
- Interactive
- User Interface
- Verification Back-end
- Target Language
- Specification Logic
Feature Model of Verification Tools

- Algorithms Proven
- Custom Prover
- LCF
- Trusted Code Base
- Verification Tool
- Target Language
- Generated Proof Obligations
- Automated
- User Interface
- Verification Back-end
- Specification Logic
- Embedded in Proover
- Interactive
- Complete IDE
Feature Model of Verification Tools

- Algorithms Proven
- Custom Prover
- Trusted Code Base
- LCF
- Verification Tool
- Verification Back-end
- Generated Proof Obligations
- Automated
- User Interface
- LCF
- SMT
- Custom rules
- Embedded in Proover
- Interactive
- Complete IDE
- Target Language
- Specification Logic
- Incremental Interactive Verification of the Correctness of Object-oriented Software
Feature Model of Verification Tools

Verification Tool

- Algorithms Proven
- Custom Prover
- Trusted Code Base
- Generated Proof Obligations
- Automated
- User Interface
- LCF
- Complete IDE
- Embedded in Proozer
- Interactive
- Specification Logic
- Custom rules
- SMT
- Separation Logic
- (Multi-sorted) FOL

Target Language

- C
- Java

Interface

- Dynamic Frames
- Generic Array
- Callback

Language

- First Order
- Higher Order
Feature Model of Verification Tools

jStar

- Algorithms Proven
- Custom Prover
- LCF
- Trusted Code Base
- Generated Proof Obligations
- Embedded in Proover
- Complete IDE
- Automated
- Interactive
- User Interface
- LCF
- SMT
- Verification Tool
- Verification Back-end
- Custom rules
- Specification Logic
- (Multi-sorted) FOL
- First Order
- Higher Order
- Separation Logic
- Dynamic Frames
- Interface
- Callback
- Generic Array
- C
- Java
- Target Language
Incremental Interactive Verification of the Correctness of Object-oriented Software
class List {
    Node next;
    static class Node {
        int value;
        Node next;
    }
}
class List {
    Node next;
    static class Node {
        int value;
        Node next;
    }
}

Fixpoint Node_list (p : val) (lst : list val) : asn :=
class List {
    Node next;
    static class Node {
        int value;
        Node next;
    }
}

Fixpoint Node_list (p : val) (lst : list val) : asn :=
    match lst with
    | nil ⇒ p = null
In-place list reversal

class List {
    Node next;
    static class Node {
        int value;
        Node next;
    }
}

Fixpoint Node_list (p : val) (lst : list val) : asn :=
    match lst with
    | nil       ⇒ p = null
    | x :: xs    ⇒ typeof p Node ∧

class List {
    Node next;
    static class Node {
        int value;
        Node next;
    }
}

Fixpoint Node_list (p : val) (lst : list val) : asn :=
    match lst with
    | nil    ⇒ p = null
    | x :: xs ⇒ typeof p Node ∧ p.value→x *
class List {
    Node next;
    static class Node {
        int value;
        Node next;
    }
}

Fixpoint Node_list (p : val) (lst : list val) : asn :=
    match lst with
    | nil       ⇒ p = null
    | x :: xs    ⇒ typeof p Node ∧ p.value → x *
        ∃ v : val. p.next → v *

class List {
    Node next;
    static class Node {
        int value;
        Node next;
    }
}

Fixpoint Node_list (p : val) (lst : list val) : asn :=
    match lst with
    | nil => p = null
    | x :: xs => typeof p Node \ p.value → x *
                \exists v : val. p.next → v * Node_list v xs
    end.
In-place list reversal

class List {
    Node next;
    static class Node {
        int value;
        Node next;
    }
}

Fixpoint Node_list (p : val) (lst : list val) : asn :=
match lst with
    | nil       ⇒ p = null
    | x :: xs    ⇒ typeof p Node ∧ p.value → x *
        ∃ v : val. p.next → v * Node_list v xs
end.

Definition List_rep (p : val) (lst : list val) : asn :=
class List {
    Node next;
    static class Node {
        int value;
        Node next;
    }
}

Fixpoint Node_list (p : val) (lst : list val) : asn :=
match lst with
| nil         ⇒ p = null
| x :: xs     ⇒ typeof p Node ∧ p.value⇒x *
                  ∃v : val. p.next⇒v * Node_list v xs
end.

Definition List_rep (p : val) (lst : list val) : asn :=
typeof p List ∧ ∃h : val. p.head⇒h *
class List {
    Node next;
    static class Node {
        int value;
        Node next;
    }
}

Fixpoint Node_list (p : val) (lst : list val) : asn :=
    match lst with
    | nil ⇒ p = null
    | x :: xs ⇒ typeof p Node ∧ p.value→x ∗
        ∃v : val. p.next→v ∗ Node_list v xs
    end.

Definition List_rep (p : val) (lst : list val) : asn :=
    typeof p List ∧ ∃h : val. p.head→h ∗ Node_list h lst
coz I like living on the edge
Synchronised of software and its correctness proofs
Interactively proving using Coq and separation logic
Eclipse plugin, several generations
Empirically evaluated perceived usefulness with positive results
Open source, available on github
Snapshotable Tree

- Developed by Driscoll, Sarnak, Sleator, Tarjan in 1989
- Binary search tree
- Persistent read-only snapshots
Snapshotable Tree

- Developed by Driscoll, Sarnak, Sleator, Tarjan in 1989
- Binary search tree
- Persistent read-only snapshots

```java
Tree t = new Tree();
t.add("a"); t.add("b"); t.add("c"); t.add("d");
Tree S1 = t.snapshot();
t.remove("d"); t.add("e");
Tree S2 = t.snapshot();
```

![Diagram of trees S1 and S2 showing the differences after operations]
Snapshotable Tree - Client and Implementor View

Client:

```
S1.root
b
a c
  d
S2.root
b
a c
  e
```

Implementation:

```
S1.root
b
a c
  d
S2.root
b
a c
  e
d
```

Incremental Interactive Verification of the Correctness of Object-oriented Software
interface ITree {
  H({Tree(this, τ)} ⊔ φ) contains(x) ret = x ∈ τ ∧
  H({Tree(this, τ)} ⊔ φ)

  H({Snap(this, τ)} ⊔ φ) contains(x) ret = x ∈ τ ∧
  H({Snap(this, τ)} ⊔ φ)

  H({Tree(this, τ)} ⊔ φ) add(x) ret = x ∉ τ ∧
  H({Tree(this, {x} ∪ τ)} ⊔ φ)

  H({Tree(this, {x} ∪ τ)} ⊔ φ) remove(x) ret = true ∧
  H({Tree(this, τ)} ⊔ φ)

  H({Tree(this, τ)} ⊔ φ) ∧
  remove(x) ret = false ∧
  H({Tree(this, τ)} ⊔ φ)

  H({Tree(this, τ)} ⊔ φ) snapshot() H({Snap(ret, τ), Tree(this, τ)} ⊔ φ)
}
(a) \( H(\{ Tree(t, \tau) \} \uplus \phi) \vdash t : C \)
(b) \( H(\{ Snap(s, \tau) \} \uplus \phi) \vdash s : C \)
(c) \( \tau = \tau' \land H(\{ Tree(t, \tau) \} \uplus \phi) \vdash H(\{ Tree(t, \tau') \} \uplus \phi) \)
(d) \( H(\{ Snap(s, \tau) \} \uplus \phi) \vdash H(\phi) \)
Snapshotable Tree - Conclusion

- Implementation strategies
  - Path copy persistence
  - Node copy persistence
- Orthogonal rebalancing
- Interface specification is valid for all implementation
- Formally verified path copy persistence
Planar Point Location

Given a plane (statically) subdivided into non-overlapping polygons, find the smallest polygon containing a (dynamic) query point.
Given a plane (statically) subdivided into non-overlapping polygons, find the smallest polygon containing a (dynamic) query point.
Developed by Sarnak and Tarjan, 1986
Used in real applications
  - Locating country of a coordinate on a geographical map
  - Find HTML element where mouse was clicked
Motivation for Planar Point Location

- Developed by Sarnak and Tarjan, 1986
- Used in real applications
  - Locating country of a coordinate on a geographical map
  - Find HTML element where mouse was clicked
- Given \( n \) line segments
- Static space consumption \( O(n) \)
- Dynamic query time consumption \( O(\log n) \)
Planar Point Location Algorithm
Planar Point Location Algorithm
Planar Point Location Algorithm
Planar Point Location Algorithm
Planar Point Location Algorithm
Planar Point Location - Conclusion

- Refactored code from C5 library
- Using integral numbers instead of floating point numbers
- Stateless comparators
- Work in progress: verification of correctness
  - Specification
  - Informal sketch of verification
  - Formalisation needs to be done
1 Introduction

2 Kopitiam

3 Case studies

4 Aftermath
Future Work

- **Kopitiam**
  - Coq perspective already improved tremendously (builder, interaction, ...)
  - Java features: arrays, generics, exceptions, concurrency
  - Navigation, completion, and refactoring (common IDE features)
  - Develop more case studies
  - More user evaluation

- Verify correctness of other snapshotable tree implementations
- Formalise verification of planar point location
- Verify time and space complexity
Modular interactive verification of correctness of object-oriented software is achievable and can be tightly integrated into the software development workflow.

- Kopitiam integrates development with correctness proofs
  - Proof is done interactively using separation logic
  - Proof is checked by Coq
- Case studies
Think *before* you start to prove
Think *before* you start to prove

Programming with verification in mind gives rise to *stateless* coding style
Think *before* you start to prove

Programming with verification in mind gives rise to *stateless* coding style

*Smash the state, one field at a time*
Due to Kulturnatten, the reception will be on the 1st floor of the canteen.
Trusted Code Base

- Coq implementation: Ocaml code
- Ocaml compiler
- Operating system
- Kopitiam: Translation from Java into Coq presentation
- Java compiler and bytecode interpreter (because we verify source code)
Snapshotable Tree - Iterator specification

\[
H(\{ \text{Snap}(\text{this}, \tau) \} \cup \phi) \text{ iterator()}
\]

\[
H(\{ \text{Iter}(\text{ret}, [\tau]), \text{Snap}(\text{this}, \tau) \} \cup \phi) \\
\land \text{ret} \ll : \text{Iterator}
\]

\[
H(\{ \text{Iter}(\text{it}, \alpha) \} \cup \phi) \vdash H(\phi)
\]