Types for Data Races

Armando Solar-Lezama
Computer Science and Artificial Intelligence Laboratory
MIT
Adapted from Flanagan and Freund
October 20, 2011
Recap

A change in a private input can not affect a public output

Data with a label $L_h$ can not be written to a location with label $L_l$ if $L_l \leq L_h$

```
Wikipedia wp = getWP();
wp.write(rx);
```

```java
Wikipedia{
    void write(String{} txt);
}
```
Data Races

class Account {
    private int bal = 0;
    
    public void deposit(int n) {
        int j = bal;
        bal = j + n;
    }
}

Data Race:

Two threads access the same memory location, one of the accesses is a write, and there is no synchronization in between.
Strategy

How do programmers avoid races?

- Only access shared data while holding the “right” lock
  • all threads must agree on what the right lock for a piece of data is
- The decision of what the right lock is should be easy to describe
  • otherwise it’s easy to get confused

We can make this into a safety policy!
Strategy

In order to avoid races, we will design a type system to enforce the following safety property:

- When a memory location L is accessed by a thread, the set of locks held by the thread must be a superset of the set of locks that protect L.

Challenges:

- Define mechanisms to encode the locks that guard a memory location as part of the type
- Define a type checking algorithm that compares the required locks against a conservative approximation of the set of locks held at a given point in the program
- Define a type inference algorithm that can save you from writing lots of annotations
The language

Start with a simple language with classes and references

\[
\begin{align*}
e & ::= \text{new } c \\
 & \quad | \quad x \\
 & \quad | \quad e.fd \\
 & \quad | \quad e.fd = e \\
 & \quad | \quad e.mn(e^*) \\
 & \quad | \quad \text{let } \text{arg} = e \text{ in } e
\end{align*}
\]

(allocate)
(variable)
(field access)
(field update)
(method call)
(variable binding)

Add threads and synchronization

\[
\begin{align*}
\text{synchronized } e \text{ in } e \\
\text{fork } e
\end{align*}
\]

(synchronization)
(fork)
Java synchronization

Every object has a lock associated with it
A synchronized block acquires and releases the lock of an object

```java
... synchronized(p){
    int bal;
}
...```

We can describe sets of locks by describing sets of objects!
Stating Locking Requirements

class Account {
    private int bal guarded_by this = 0;

    public void deposit(int n) requires this{
        int j = bal;
        bal = j + n;
    }
}

class Account {
    private int bal guarded_by this = 0;

    public void deposit(int n) requires this{
        int j = bal;
        bal = j + n;
    }

    public void transferAll(Account r) requires {
        int j = bal;
        int k = r.bal;
        bal = j+k;
        r.bal = 0;
    }
}
class Account {
    private Guard g
    private int bal guarded_by g = 0;

    public void deposit(int n) requires g{
        int j = bal;
        bal = j + n;
    }

    public void transferAll(Account r) requires g, r.g{
        int j = bal;
        int k = r.bal;
        bal = j+k;
        r.bal = 0;
    }
}
Stating Locking Requirements

class Account {
    private final Guard g;
    private int bal guarded_by g = 0;

    public void deposit(int n) requires g{
        int j = bal;
        bal = j + n;
    }

    public void transferAll(Account r) requires g, r.g{
        int j = bal;
        int k = r.bal;
        bal = j+k;
        r.bal = 0;
    }
}
class Account<Ghost l> {
    private int bal guarded_by l = 0;

    public void deposit(int n) requires l{
        int j = bal;
        bal = j + n;
    }

    public void transferAll(Account<l> r) requires l{
        int j = bal;
        int k = r.bal;
        bal = j+k;
        r.bal = 0;
    }
}
Type Checking

Lock Set must be included as part of the environment

\[ P; E; ls \vdash e : t \]

- Program
- Environment
- Lock set
class Account {
    private int bal guarded_by this = 0;

    public void deposit(int n) requires this{
        int j = bal;
        bal = j + n;
    }

    public void transferAll(Account r) requires this, r{
        int j = bal;
        int k = r.bal;
        bal = j+k;
        r.bal = 0;
        Account a = getAccnt(10220);
        Account b = getAccnt(22123);
        synchronized(a,b){
            a.transferAll(b);
        }
    }
}
class Account {
    private final Guard g;
    private int bal guarded_by g = 0;

    public void deposit(int n) requires g{
        int j = bal;
        bal = j + n;
    }

    public void transferAll(Account r) requires g, r.g{
        int j = bal;
        int k = r.bal;
        bal = j+k;
        r.bal = 0;
    }
}

Account a = getAccnt(10220); Account b = getAccnt(22123);
synchronized(a.g,b.g){
    a.transferAll(b);
}
Typing Rules

\[
\begin{align*}
\text{[EXP FORK]} & \quad P; E; \emptyset \vdash e : t \\
& \quad P; E; ls \vdash \text{fork } e : \text{int}
\end{align*}
\]
Typing Rules

\[
\begin{align*}
\text{[EXP SYNC]} \\
\frac{P; E \vdash_{\text{final}} e_1 : c \quad P; E; ls \cup \{e_1\} \vdash e_2 : t}{P; E; ls \vdash \text{synchronized } e_1 \text{ in } e_2 : t}
\end{align*}
\]
Typing Rules

\[ P; E \vdash t \text{ mn}(\text{arg}_1\ldots_n) \text{ requires } ls \{ e \} \]
Typing Rules

[EXP REF]

\[ P; E; ls \vdash e : c \]
\[ P; E \vdash ([\text{final}]_{\text{opt}} t \text{ fd guarded by } l = e') \in c \]
\[ P; E \vdash [e/\text{this}]l \in ls \]
\[ P; E \vdash [e/\text{this}]t \]
\[ \frac{}{P; E; ls \vdash e.f d : [e/\text{this}]t} \]

[EXP ASSIGN]

\[ P; E; ls \vdash e : c \]
\[ P; E \vdash (t \text{ fd guarded by } l = e'') \in c \]
\[ P; E \vdash [e/\text{this}]l \in ls \]
\[ P; E; ls \vdash e' : [e/\text{this}]t \]
\[ \frac{}{P; E; ls \vdash e.f d = e' : [e/\text{this}]t} \]
class Node<ghost 1>{
    Node<1> next guarded_by 1;
    int v guarded_by 1;
}

class List{
    Node<this> head

    void add(int x) requires this{
        Node<this> t = new Node<this>(x);
        t.next = head;
        head = t;
    }
}

{ List l = getList();
    synchronized(l){ l.add(5); } }
Type Inference

How do we avoid adding all of these annotations?
Reducing Type Inference to SAT

class Ref<ghost g1,g2,...,gn> {
    int i;
    void add(Ref r) {
    i = i + r.i;
    }
}
Reducing Type Inference to SAT

class Ref<ghost g> {
    int i;
    void add(Ref r) {
        i = i + r.i;
    }
}

• Add ghost parameters <ghost g> to each class declaration
Reducing Type Inference to SAT

class Ref<ghost g> {
  int i guarded_by α_i;
  void add(Ref r) {
    i = i + r.i;
  }
}

• Add ghost parameters <ghost g> to each class declaration
• Add guarded_by α_i to each field declaration
  – type inference resolves α_i to some lock
Reducing Type Inference to SAT

```java
class Ref<ghost g> {
  int i guarded_by α₁;
  void add(Ref<α₂> r)
  {
    i = i + r.i;
  }
}
```

- Add ghost parameters `<ghost g>` to each class declaration
- Add `guarded_by αᵢ` to each field declaration
  - type inference resolves `αᵢ` to some lock
- Add `<α₂>` to each class reference
Reducing Type Inference to SAT

class Ref<ghost g> {
    int i guarded_by α₁;
    void add(Ref<α₂> r)
        requires β
    {
        i = i + r.i;
    }
}

- Add ghost parameters <ghost g> to each class declaration
- Add guarded_by αᵢ to each field declaration
  - type inference resolves αᵢ to some lock
- Add <α₂> to each class reference
- Add requires βᵢ to each method
  - type inference resolves βᵢ to some set of locks
Reducing Type Inference to SAT

class Ref<ghost g> {
    int i guarded_by α₁;
    void add(Ref<α₂> r) requires β {
        i = i + r.i;
    }
}

Constraints:
α₁ ∈ { this, g }
α₂ ∈ { this, g }
β ⊆ { this, g, r }
α₁ ∈ β
α₁[this := r, g := α₂] ∈ β
Reducing Type Inference to SAT

```java
class Ref<ghost g> {  
    int i guarded_by α₁;
    void add(Ref<α₂> r)  
        requires β  
    {  
        i = i + r.i;
    }
}
```

**Constraints:**
- $α₁ ∈ \{ \text{this, g} \}$
- $α₂ ∈ \{ \text{this, g} \}$
- $β ⊆ \{ \text{this, g, r} \}$
- $α₁ ∈ β$
- $α₁[\text{this := r, g := α₂}] ∈ β$

**Encoding:**
- $α₁ = (b₁ ? \text{this : g })$
- $α₂ = (b₂ ? \text{this : g })$
- $β = \{ b₃ ? \text{this, b₄ ? g, b₅ ? r } \}$

Use boolean variables $b₁, ..., b₅$ to encode choices for $α₁, α₂, β$
Reducing Type Inference to SAT

class Ref<ghost g> {
    int i guarded_by α₁;
    void add(Ref<α₂> r)
        requires β
    {
        i = i + r.i;
    }
}

Constraints:
- $\alpha_1 \in \{ \text{this, g} \}$
- $\alpha_2 \in \{ \text{this, g} \}$
- $\beta \subseteq \{ \text{this, g, r} \}$

Encoding:
- $\alpha_1 = (b_1 \ ? \ \text{this} : g)$
- $\alpha_2 = (b_2 \ ? \ \text{this} : g)$
- $\beta = \{ b_3 \ ? \ \text{this}, b_4 \ ? \ g, b_5 \ ? \ r \}$

Use boolean variables $b_1, \ldots, b_5$ to encode choices for $\alpha_1$, $\alpha_2$, $\beta$
Reducing Type Inference to SAT

```
class Ref<ghost g> {  
  int i guarded_by α₁;  
  void add(Ref<α₂> r)  
      requires β  
      {  
        i = i + r.i;  
      }  
}  
```

**Constraints:**
- $\alpha_1 \in \{\text{this, g}\}$
- $\alpha_2 \in \{\text{this, g}\}$
- $\beta \subseteq \{\text{this, g, r}\}$
- $\alpha_1[\text{this := r, g:=}\alpha_2] \in \beta$

**Encoding:**
- $\alpha_1 = (b_1 ? \text{this : g})$
- $\alpha_2 = (b_2 ? \text{this : g})$
- $\beta = \{b_3 ? \text{this, b_4 ? g, b_5 ? r}\}$

**Use boolean variables $b_1,\ldots,b_5$ to encode choices for $\alpha_1, \alpha_2, \beta$.**
Reducing Type Inference to SAT

class Ref<ghost g> {
    int i guarded_by α₁;
    void add(Ref<α₂> r) requires β {
        { i = i + r.i;
        }
    }
}

Constraints:
α₁ ∈ { this, g }
α₂ ∈ { this, g }
β ⊆ { this, g, r }

Encoding:
α₁ = (b₁ ? this : g )
α₂ = (b₂ ? this : g )
β = { b₃ ? this, b₄ ? g, b₅ ? r }

Use boolean variables b₁,...,b₅ to encode choices for α₁, α₂, β

α₁[this := r, g:=α₂] ∈ β
(b₁ ? this : g )[this := r, g:=α₂] ∈ β
(b₁ ? r : α₂) ∈ β
Reducing Type Inference to SAT

class Ref<ghost g> {  
    int i guarded_by α₁;
    void add(Ref<α₂> r)  
    {  
        i = i  
        + r.i;
    }  
}  

Constraints:

\[ \alpha₁ \in \{ \text{this, g} \} \]
\[ \alpha₂ \in \{ \text{this, g} \} \]
\[ \beta \subseteq \{ \text{this, g, r} \} \]

Encoding:

\[ \alpha₁ = (b₁ ? \text{this : g}) \]
\[ \alpha₂ = (b₂ ? \text{this : g}) \]
\[ \beta = \{ b₃ \ ? \text{this}, b₄ \ ? \text{g}, b₅ \ ? \text{r} \} \]

Use boolean variables \( b₁, \ldots, b₅ \) to encode choices for \( \alpha₁, \alpha₂, \beta \):

\[ \alpha₁[\text{this := r, g := } \alpha₂] \in \beta \]
\[ (b₁ ? \text{this : g})[\text{this := r, g := } \alpha₂] \in \beta \]
\[ (b₁ ? r \ : \alpha₂) \in \beta \]
\[ (b₁ ? r : (b₂ ? \text{this : g} )) \in \{ b₃ \ ? \text{this}, b₄ \ ? \text{g}, b₅ \ ? \text{r} \} \]
Reducing Type Inference to SAT

$$\alpha_1 \in \{ \text{this, g} \}$$
$$\alpha_2 \in \{ \text{this, g} \}$$
$$\beta \subseteq \{ \text{this, g, r} \}$$

Use boolean variables $b_1, \ldots, b_5$ to encode choices for $\alpha_1, \alpha_2, \beta$

Clauses:

$$b_1 \Rightarrow b_5$$
$$(\neg b_1 \land b_2 \Rightarrow b_3)$$
$$(\neg b_1 \land \neg b_2 \Rightarrow b_4)$$

Encoding:

$$\alpha_1 = (b_1 ? \text{this} : g)$$
$$\alpha_2 = (b_2 ? \text{this} : g)$$
$$\beta = \{ b_3 ? \text{this}, b_4 ? g, b_5 ? r \}$$
Overview of Type Inference

Add Unknowns:

class Ref<ghost g> {
  int i guarded_by α1;
  ...
}

Unannotated Program:

class Ref {
  int i;
  ...
}

Annotated Program:

class Ref<ghost g> {
  int i guarded_by g;
  ...
}

Constraints:

α1 ∈ { this, g }
...

Error: potential race on field i

SAT problem:

(b1 ⇒ b5)
...

b1, ... encodes choice for α1, ...

Error: potential race on field i

SAT solver

unsatisfiable

satisfiable

SAT soln:

b1 = false
...

Constraint Solution:

α1 = g
...

C. Hawegan

Types for Race Freedom