Types for Data Races

Armando Solar-Lezama
Computer Science and Artificial Intelligence Laboratory
MIT
With slides by Flanagan and Freund.
Used with permission.
October 19, 2015
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 $L_l \leq L_h$.

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

\[
e ::= \text{new } c \quad (\text{allocate})
\]

\[
| x \quad (\text{variable})
\]

\[
| e.fd \quad (\text{field access})
\]

\[
| e.fd = e \quad (\text{field update})
\]

\[
| e.mn(e^*) \quad (\text{method call})
\]

\[
| \text{let } \text{arg} = e \text{ in } e \quad (\text{variable binding})
\]

Add threads and synchronization

\[
| \text{synchronized } e \text{ in } e \quad (\text{synchronization})
\]

\[
| \text{fork } e \quad (\text{fork})
\]
Java synchronization

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

```java
Account

int bal;

synchronized(p){
    ...
}

... We can describe sets of locks by describing sets of objects!
```
class Account {
    private int bal guarded_by this = 0;

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

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

    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;
    }
}

These expressions need to be final.
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
Type Checking

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);
}

14
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

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

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

\[ \text{METHOD} \]

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

[EXP REF]

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

[EXP ASSIGN]

\[
P; E; ls \vdash e : c \\
P; E \vdash (t \; fd \; \text{guarded_by} \; l = e'') \in c \\
P; E \vdash [e/\text{this}]l \in ls \\
P; E; ls \vdash e' : [e/\text{this}]t \\
\hline \\
P; E; ls \vdash e.fd = e' : [e/\text{this}]t
\]
Example

class Node<ghost l>{
    Node<l> next guarded_by l;
    int v guarded_by l;
}

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

Add ghost parameters `<ghost g>` to each class declaration

```java
class Ref<ghost g> {
    int i;
    void add(Ref r) {
        i = i + r.i;
    }
}
```
Reducing Type Inference to SAT

class Ref<ghost g> {
  int i guarded_by $\alpha_1$;
  void add(Ref r) {
    i = i + r.i;
  }
}

- Add ghost parameters <ghost g> to each class declaration
- Add guarded_by $\alpha_i$ to each field declaration
  - type inference resolves $\alpha_i$ to some lock
Reduction Type Inference to SAT

```
class Ref<ghost g> {
    int i guarded_by \alpha_1;
    void add(Ref<\alpha_2> r) {
        i = i + r.i;
    }
}
```

- Add ghost parameters <ghost g> to each class declaration
- Add guarded_by \alpha_i to each field declaration
  - type inference resolves \alpha_i to some lock
- Add <\alpha_2> to each class reference
Reducing Type Inference to SAT

class Ref<ghost g> {
    int i guarded_by \alpha_1;
    void add(Ref<\alpha_2> r)
        requires \beta
    {
        i = i + r.i;
    }
}

- Add ghost parameters <ghost g> to each class declaration
- Add guarded_by \alpha_i to each field declaration
  - type inference resolves \alpha_i to some lock
- Add <\alpha_2> to each class reference
- Add requires \beta_i to each method
  - type inference resolves \beta_i to some set of locks
Reducing Type Inference to SAT

class Ref<ghost g> {
    int i guarded_by \alpha_1;
    void add(Ref<\alpha_2> r)
        requires \beta
    {
        i = i + r.i;
    }
}

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

α₁[this := r, g := α₂] ∈ β

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
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:
- $α_1 \in \{ \text{this, g} \}$
- $α_2 \in \{ \text{this, g} \}$
- $β \subseteq \{ \text{this, g, r} \}$
- $α_1[\text{this := r, g := } α_2] \in β$

Encoding:
- $α_1 = (b1 ? \text{this : g})$
- $α_2 = (b2 ? \text{this : g})$
- $β = \{ b3 ? \text{this, b4 ? g, b5 ? r} \}$

Use boolean variables $b_1, ..., b_5$ to encode choices for $α_1, α_2, β$
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:

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

α₁ ∈ β
α₁[this := r, g := α₂] ∈ β

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

C. Flanagan

Types for Race Freedom
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

```java
class Ref<ghost g> {
    int i guarded_by \( \alpha_1 \);
    void add(Ref<\alpha_2> r)
        requires \( \beta \)
    {
        i = i + r.i;
    }
}
```

Constraints:

- \( \alpha_1 \in \{ \text{this, g} \} \)
- \( \alpha_2 \in \{ \text{this, g} \} \)
- \( \beta \subseteq \{ \text{this, g, r} \} \)
- \( \alpha_1 \in \beta \)
- \( \alpha_1[\text{this := r, g := } \alpha_2] \in \beta \)
- \( (b1 \ ? \text{this : g})[\text{this := r, g := } \alpha_2] \in \beta \)
- \( (b1 \ ? \alpha_2) \in \beta \)
- \( (b1 \ ? \ r : (b2 \ ? \text{this : g})) \in \{ \text{b3 ? this, b4 ? g, b5 ? r} \} \)

Encoding:

- \( \alpha_1 = (b1 \ ? \text{this : g}) \)
- \( \alpha_2 = (b2 \ ? \text{this : g}) \)
- \( \beta = \{ \text{b3 ? this, b4 ? g, b5 ? r} \} \)

Use boolean variables \( b1, \ldots, b5 \) to encode choices for \( \alpha_1, \alpha_2, \beta \)
Reducing Type Inference to SAT

```
class Ref<ghost g> {
    int i guarded_by \( \alpha_1 \);
    void add(Ref<\alpha_2> r) {
        requires \( \beta \)
        {
            i = i + r.i;
        }
    }
}
```

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

**Encoding:**
- \( \alpha_1 = (b1 ? \text{this : g} ) \)
- \( \alpha_2 = (b2 ? \text{this : g} ) \)
- \( \beta = \{ b3 ? \text{this, b4 ? g, b5 ? r } \} \)

**Use boolean variables**
- \( b1, \ldots, b5 \) to encode choices for \( \alpha_1, \alpha_2, \beta \)

**Clauses:**
- \( (b1 \Rightarrow b5) \)
- \( (\neg b1 \land b2 \Rightarrow b3) \)
- \( (\neg b1 \land \neg b2 \Rightarrow b4) \)

C. Flanagan
Overview of Type Inference

Add Unkowns:
class Ref<ghost g> {
  int i guarded_by \alpha_1 ;
  ...
}

Unannotated Program:
class Ref {
  int i;
  ...
}

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

Constraints:
\alpha_1 \in \{ \text{this, g} \}
...

SAT problem:
(b_1 \Rightarrow b_5)
...

b_1, ... encodes choice for
\alpha_1, ...

Error: potential race on field i

Constraint Solution:
\alpha_1 = g
...

SAT soln: b_1=false
...

Types for Race Freedom