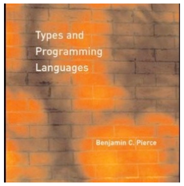
Atomic Sets

Vaziri, Tip, Dolby, Hammer, Vitek

IBM and Purdue

ECOOP 2010



Concurrency control

- Writing correct concurrent code is tricky in language like Java
- Primitives such as `synchronized` are inherently brittle
  - they protect control flow paths
  - it is easy to forget a `synchronized` section
- Data-centric concurrency control turns things around,
  - developers identify *memory locations* which share some *consistency property*,
  - synchronization is inserted by the compiler
  - Approach is *declarative* as it does not prescribe where and what kind of synchronization will be used

Example

- A simple Counter class

```
class Counter {
  int val;
  synchronized int get() { return val; }
  synchronized void dec() { val--; }
  synchronized void inc() { val++; }
}
```

Example

- A simple Counter class

```
class Counter {
  int val;
  synchronized int get() { return val; }
  synchronized void dec() { val--; }
  synchronized void inc() { val++; }
}
```

- What is the consistency property on data?

Example: Multiple Fields

- Atomic set version

```
class Counter {
   atomicset a;
   atomic(a) int val;
  int get() { return val; }
  void dec() { val--; }
  void inc() { val++; }
}
```

Observe: only one annotations needed to specify consistent synchronization policy of the data.

Example: Inheritance

- Atomic sets can be inherited

```
class BackupCounter extends Counter {
   atomic(a) int old;
  void dec() { old=val; val--; }
  void inc() { old=val; val++; }
}
```

Observe: the atomic set declared in the parent is inherited and its state can be extended

Example: Multiple Objects

- Atomic sets spanning multiple objects

```
class PairCounter {
  atomicset b;
  atomic(b) int diff;
  Counter|a=this.b| low = new Counter|a=this.b|();
  Counter|a=this.b| high = new Counter|a=this.b|();
  void incH() {
          high.inc();
          diff = high.get()-low.get();
  }
}
```

Observe: |a=this.b| aliases two atomic sets from different objects.

Example: Multiple Objects

```
class PairCounter {
   atomicset b;
   atomic(b) int diff;
   Counter|a=this.b| low = new Counter|a=this.b|();
   Counter|a=this.b| high = new Counter|a=this.b|();
   void incH() {    high.inc();  diff = high.get()-low.get();    }
}

class Counter {
    atomicset a;
    atomic(a) int val;
   int get() { return val; }
   void dec() { val--; }
   void inc() { val++; }
}
```

Example: Translation to Java

```
class PairCounter {
   Lock b;
   int diff;
   Counter low, high;
   PairCounter() { b = new Lock();  low = new Counter(b); high = new Counter(a); }
   void incH() {  synchronized(b) {  high.inc();  diff = high.get()-low.get();   }  }
}
class Counter {
    Lock a;
    int val;
    Counter(Lock a) { this.a = a; }
    int get() { synchronized (a) { return val; } }
    void dec() { synchronized (a) { val--; } }
    void inc() { synchronized (a) { val++;  } }
}
```

Example: Translating to Java

- Can we use a Counter independently of PairCounter?

```
class Counter {
    atomicset a;
    atomic(a) int val;
   int get() { return val; }
   void dec() { val--; }
   void inc() { val++; }
}
```

- translates to..

```
class Counter {
    Lock a;
    Counter () { a = new Lock(); }
    int val;
    int get() { synchronized (a) { return val; } }
    void dec() { synchronized (a) { val--; } }
    void inc() { synchronized (a) { val++;  } }
}
```

Questions

- Can we hide synchronization from clients?
  - What does that mean?

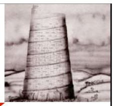
    Hide = Client code not be changed if we change the synchronization policy

```
Counter c = new Counter();   // ok
c.inc();
c.dec();                     // so far so good
c.val++;                     // not so good...
```

Fields marked `atomic` must considered `protected` or `private`

```
class Counter {   ...
  void bad(Counter o) { o.val++; }    // Allowed?
```

Example: Translation to Java



- Can we use a Counter independently and from PairCounter?

```
class Counter {
    atomicset a;
    atomic(a) int val;
    int get() { return val; }
    void dec() { val--; }
    void inc() { val++; }
}
```

- This translates to..

```
class Counter {
    Lock a;
    Counter () { a = new Lock(); }
    Counter (Lock a) { this.a=a; }
    int val;
    int get() { synchronized (a) { return val; } }
    void dec() { synchronized (a) { val--; } }
    void inc() { synchronized (a) { val++;  } }
}
```

Performance

- How many
  - lock acquire/releases performed on a call to incH()?
  - of these are needed?

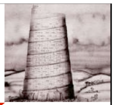
```
class PairCounter {   …
   void incH() {  high.inc();  diff = high.get()-low.get();   }
}
class Counter { ...
   int get() {  return val; }
   void inc() { val++;  }
}
```

Performance

- How many
  - lock acquire/releases performed on a call to incH()?
  - of these are needed?

```
class PairCounter {  …
   void incH() {  synchronized(b) {  high.inc();  diff = high.get()-low.get();   }  }
}
class Counter { ...
   int get() { synchronized (a) { return val; } }
   void inc() { synchronized (a) { val++;  } }
}
```

Performance



- Translation optimizes internal locking away

```
class PairCounter { …
   void incH(){
        synchronized(b) {  high.inc$intern();
                           diff=high.get$intern() - low.get$intern();  }  }
}

class Counter { ...
   int get() { synchronized (a) { return val; } }
   void inc() { synchronized (a) { val++;  } }
   int get$intern() { return val; }
   void inc$intern() {  val++;  }
}
```

Observe: intern methods are only called when lock is held, they can call intern methods. Intern methods must not be observable by client code.

Performance

- As an additional optimization we support internal objects

```
class PairCounter {
   atomicset b;
   atomic(b) int diff;
   Counter|a=this.b| low = new Counter|a=this.b|();
   …
}

internal class Counter {
    atomicset a;
    atomic(a) int val;
   …
}
```

Observe: an internal class is never directly manipulated by client code. Thus it does not have to have a lock and all of its methods can be called without synchronization as they are always accessed with the owner’s lock held

Performance

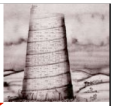
- The optimization is correct as long as the following is not allowed

```
class PairCounter { …
    void bad(PairCounter p) {   low = p.low;  }
```

- How do we prevent this?
  - Types of course!
  - We need a formalization of Java with threads, state, and atomics sets

AJ

- The AJ calculus is a simple object calculus modeled on FJ + state + threads
- Some simplifications make AJ easier to work with
  - all local variables are declared at method entry
  - no nested expression, only simple ones + assignment
  - no implicit up-casts in assignment or method invocation
  - no thread creation/destruction
  - single atomicset per class
  - …



Formalizing AJ

- Syntax

$$
\begin{array}{llll}
p & ::= & \overline{cd} & \textit{program} \\
cd & ::= & \iota\ \text{class C extends D}\ \{ as\ \overline{fd}\ \overline{md} \} & \textit{class} \\
as & ::= & \text{atomicset a} \mid \epsilon & \\
fd & ::= & \alpha\ \tau\ \text{f} & \textit{field} \\
md & ::= & \tau\ \text{m}\ (\overline{\tau\ \text{x}})\ \{ \overline{\tau\ \text{z};}\ \text{s;return y} \} & \textit{method} \\
\text{s} & ::= & \text{s;s} \mid \text{skip} \mid \text{x =this.f} \mid \text{x =}(\tau)\text{y} \mid & \textit{statement} \\
 & & \text{this.f =z} \mid \text{x =new}\ \tau\ () \mid \text{x =y.m}\ (\overline{\text{z}}) &
\end{array}
$$

Formalizing AJ

- Syntax

$$
\begin{array}{lll}
\tau & ::= \mathsf{C}|\mathsf{a} = \mathsf{this.b}| \;|\; \mathsf{C} & \textit{type} \\
\alpha & ::= \mathsf{atomic\ (a)} \;|\; \epsilon & \\
\iota & ::= \mathsf{internal} \;|\; \epsilon & \\
 & & \\
E & ::= [] \;|\; E[x : \tau] & \textit{type env}
\end{array}
$$

Auxiliaries

Subtyping:

$$\frac{}{\mathsf{C} <: \mathsf{C}} \qquad \frac{\mathsf{C}\ \mathsf{extends}\ \mathsf{D}}{\mathsf{C} <: \mathsf{D}} \qquad \frac{\mathsf{C} <: \mathsf{C}' \quad \mathsf{C}' <: \mathsf{D}}{\mathsf{C} <: \mathsf{D}}$$

$$\frac{\mathsf{C} <: \mathsf{D}}{\mathsf{C}|\mathsf{a}{=}\mathsf{this.b}| <: \mathsf{D}|\mathsf{a}{=}\mathsf{this.b}|}$$

Extends:

$$\frac{CT(\mathsf{C}) = \iota\ \mathsf{class}\ \mathsf{C}\ \mathsf{extends}\ \mathsf{D}\ \{\ as\ \overline{fd}\ \overline{md}\ \}}{\mathsf{C}\ \mathsf{extends}\ \mathsf{D}}$$

Type lookup:

$$\frac{\tau\ \mathsf{m}(\overline{\tau_{\mathsf{x}}\ \mathsf{x}})\ \{\ \overline{\tau_{\mathsf{z}}\ \mathsf{z}};\ \mathsf{s};\ \mathsf{return}\ \mathsf{y}\ \} \in methods(\mathsf{C})}{typeof(\mathsf{C.m}) = \overline{\tau_{\mathsf{x}}} \rightarrow \tau}$$

$$\frac{\tau\ \mathsf{f} \in fields(\mathsf{C})}{typeof(\mathsf{C.f}) = \tau}$$

Method lookup:

$$\frac{\tau\ \mathsf{m}(\overline{\tau_{\mathsf{x}}\ \mathsf{x}})\ \{\ \overline{\tau_{\mathsf{z}}\ \mathsf{z}};\ \mathsf{s};\ \mathsf{return}\ \mathsf{y}\ \} \in methods(\mathsf{C})}{mbody(\mathsf{C.m}) = (\overline{\tau_{\mathsf{x}}\ \mathsf{x}};\ \overline{\tau_{\mathsf{z}}\ \mathsf{z}};\ \mathsf{s};\ \mathsf{return}\ \mathsf{y})}$$

Local vars:

$$\frac{\begin{array}{c} H(F(\mathsf{this})) = \mathsf{C}|\omega|(\overline{r'}) \\ mbody(\mathsf{C.m}) = (\overline{\tau_{\mathsf{x}}\ \mathsf{x}};\ \overline{\tau_{\mathsf{z}}\ \mathsf{z}};\ \mathsf{s};\ \mathsf{return}\ \mathsf{y}) \\ E \equiv \overline{\mathsf{x} : \tau_{\mathsf{x}}}, \overline{\mathsf{z} : \tau_{\mathsf{z}}}, \mathsf{this} : \mathsf{C} \end{array}}{locals(\mathsf{m}, F) = E}$$

Internal lookup:

$$\frac{CT(\mathsf{C}) = \mathsf{internal}\ \mathsf{class}\ \mathsf{C}\ \mathsf{extends}\ \mathsf{D}\ \{\ \ldots\ \}}{\mathsf{C}\ is\ \mathsf{internal}}$$

Fields lookup:

$$\frac{}{fields(\mathsf{Object}) = \epsilon}$$

$$\frac{\begin{array}{c} CT(\mathsf{C}) = \iota\ \mathsf{class}\ \mathsf{C}\ \mathsf{extends}\ \mathsf{D}\ \{\ as\ \overline{fd}\ \overline{md}\ \} \\ fields(\mathsf{D}) = \overline{fd'} \end{array}}{fields(\mathsf{C}) = \overline{fd'}\ \overline{fd}}$$

Methods lookup:

$$\frac{}{methods(\mathsf{Object}) = \epsilon}$$

$$\frac{\begin{array}{c} CT(\mathsf{C}) = \iota\ \mathsf{class}\ \mathsf{C}\ \mathsf{extends}\ \mathsf{D}\ \{\ as\ \overline{fd}\ \overline{md}\ \} \\ methods(\mathsf{D}) = \overline{md'} \quad \overline{md''} = \overline{md'} - \overline{md} \end{array}}{methods(\mathsf{C}) = \overline{md}\ \overline{md''}}$$

Valid Method overriding:

$$\frac{\begin{array}{c} typeof(\mathsf{C.m}) = \overline{\tau'} \rightarrow \tau'\ \ implies \\ \overline{\tau} = \overline{\tau'}\ and\ \tau = \tau' \end{array}}{override(\mathsf{m}, \mathsf{C}, \overline{\tau} \rightarrow \tau)}$$

Atomic set lookup:

$$\frac{\begin{array}{c} CT(\mathsf{C}) = \iota\ \mathsf{class}\ \mathsf{C}\ \mathsf{extends}\ \mathsf{D}\ \{\ as\ \overline{fd}\ \overline{md}\ \} \\ as = \epsilon \qquad \mathsf{D}\ has\ \mathsf{a} \end{array}}{\mathsf{C}\ has\ \mathsf{a}}$$

$$\frac{\begin{array}{c} CT(\mathsf{C}) = \iota\ \mathsf{class}\ \mathsf{C}\ \mathsf{extends}\ \mathsf{D}\ \{\ as\ \overline{fd}\ \overline{md}\ \} \\ as = \mathsf{atomicset}\ \mathsf{a} \end{array}}{\mathsf{C}\ has\ \mathsf{a}}$$

Atomic lookup:

$$\frac{\mathsf{atomic}(\mathsf{a})\ \tau\ \mathsf{f} \in fields(\mathsf{C})}{\mathsf{C.f}\ is\ \mathsf{atomic}}$$

Subtyping

- Subtyping is the familiar reflexive, transitive closure of the extends relation, with the added rule

$$\frac{\mathsf{C} <: \mathsf{D}}{\mathsf{C}|\mathsf{a}=\mathsf{this.b}| <: \mathsf{D}|\mathsf{a}=\mathsf{this.b}|}$$

Observe: aliased types are only in subtype relation if they are aliasing the same atomic sets

Adaption

- The view point adaption is used to describe how types are viewed outside of their declaring context:

$$adapt(\mathsf{C}, \tau) \;=\; \mathsf{C}$$

$$adapt(\mathsf{C}|\mathsf{a}{=}\mathsf{this.b}|, \mathsf{D}|\mathsf{b}{=}\mathsf{this.c}|) \;=\; \mathsf{C}|\mathsf{a}{=}\mathsf{this.c}|$$

Observe: this predicate takes two types, the type being observed and the context in which it is being looked at and returns the type in the observing context.
When the predicate is undefined this means that the type is not accessible

Type Checking Method Calls

$$\frac{E(\mathsf{y}) = \tau_{\mathsf{y}} \quad \mathit{typeof}(\tau_{\mathsf{y}}.\mathsf{m}) = \overline{\tau} \to \tau \quad E(\overline{\mathsf{z}}) = \overline{\tau_{\mathsf{z}}} \\ \overline{\tau_{\mathsf{z}}} = \mathit{adapt}(\overline{\tau}, \tau_{\mathsf{y}}) \quad \tau' = \mathit{adapt}(\tau, \tau_{\mathsf{y}}) \quad E(\mathsf{x}) = \tau'}{E \vdash \mathsf{x} = \mathsf{y}.\mathsf{m}(\overline{\mathsf{z}})}$$

Adaptation

- A Trio of classes

```
class T { atomicset b;
    void main() {
        W|a=this.b| w = new W|a=this.b|;
        M|c=this.b| m = new M|c=this.b|;
        w.see(m);
}
class W { atomicset a;
     void see(M|c=this.a| t) {…
class M { atomicset c;
```

Adaptation

- A Trio of classes

```
E(m) |- M|c=this.b|        typeof(W|a=this.b|.see) = M|c=this.a|->void
E(w) |- W|a=this.b|
M|c=this.a| = adapt(M|c=this.b|, W|a=this.b|)
-----------------------------------------------------------------------------------------
    E |-    m.see(w);
```

Observe: adaption is used to unify the annotations

Types

$$\frac{E(\mathsf{this}) = \tau \quad E(\mathsf{x}) = \tau_{\mathsf{f}} \quad \mathit{typeof}(\tau.\mathsf{f}) = \tau_{\mathsf{f}}}{E \vdash \mathsf{x} = \mathsf{this.f}} \qquad \frac{E(\mathsf{this}) = \tau \quad E(\mathsf{y}) = \tau_{\mathsf{f}} \quad \mathit{typeof}(\tau.\mathsf{f}) = \tau_{\mathsf{f}}}{E \vdash \mathsf{this.f} = \mathsf{y}}$$

Observe: fields can only be selected from this (i.e. strongly private)

Types

$$\frac{E(\mathsf{x}) = \mathsf{C} \qquad \mathsf{C}\ \mathit{not}\ \mathsf{internal}}{E \vdash \mathsf{x} = \mathsf{new}\ \mathsf{C}()}$$

$$\frac{E(\mathsf{x}) = \mathsf{C}|\mathsf{a}{=}\mathsf{this.b}| \qquad \mathsf{C}\ \mathit{has}\ \mathsf{a} \quad E(\mathsf{this})\ \mathit{has}\ \mathsf{b}}{E \vdash \mathsf{x} = \mathsf{new}\ \mathsf{C}|\mathsf{a}{=}\mathsf{this.b}|()}$$

Types

$$\frac{E(\mathsf{x}) = \mathsf{D} \quad E(\mathsf{y}) = \mathsf{C} \quad \mathsf{D} <: \mathsf{C}}{E \vdash \mathsf{y} = (\mathsf{C})\mathsf{x}}$$

$$\frac{\begin{array}{c} E(\mathsf{x}) = \mathsf{C}|\mathsf{a}{=}\mathsf{this.b}| \quad \mathsf{C}\ \mathit{not}\ \mathsf{internal} \\ E(\mathsf{y}) = \mathsf{C} \end{array}}{E \vdash \mathsf{y} = (\mathsf{C})\mathsf{x}}$$

$$\frac{\begin{array}{c} E(\mathsf{x}) = \mathsf{D}|\mathsf{a}{=}\mathsf{this.b}| \quad E(\mathsf{y}) = \mathsf{C}|\mathsf{a}{=}\mathsf{this.b}| \\ \mathsf{C}\ \mathit{has}\ \mathsf{a} \quad E(\mathsf{this})\ \mathit{has}\ \mathsf{b} \quad \mathsf{D} <: \mathsf{C} \end{array}}{E \vdash \mathsf{y} = (\mathsf{C}|\mathsf{a}{=}\mathsf{this.b}|)\mathsf{x}}$$

Types

$$\textsc{(T-Class)}\quad \dfrac{\begin{array}{c}\overline{fd}\ \text{OK in}\ \mathsf{C} \quad methods(\mathsf{C}) = \overline{md'} \quad \overline{md'}\ \text{OK in}\ \mathsf{C} \quad (\mathsf{D}\ has\ \mathsf{a}\ \ implies\ \ as = \epsilon) \\ (\iota = \mathsf{internal}\ \ implies\ \ \mathsf{C}\ has\ \mathsf{a}) \quad (\mathsf{D}\ is\ \mathsf{internal}\ implies\ \iota = \mathsf{internal})\end{array}}{\iota\ \mathsf{class\ C\ extends\ D}\ \{\, as\ \overline{fd}\ \overline{md}\,\}\ \text{OK}}$$

$$\textsc{(T-Field)}\quad \dfrac{(\tau \equiv \mathsf{D}|\mathsf{a}{=}\mathsf{this.b}|\ implies\ \mathsf{D}\ has\ \mathsf{a}\ and\ \mathsf{C}\ has\ \mathsf{b}) \quad (\alpha = \mathsf{atomic}\,(\mathsf{a})\ implies\ \ \mathsf{C}\ has\ \mathsf{a})}{\alpha\,\tau\,\mathsf{f}\quad \text{OK in}\ \mathsf{C}}$$

$$\textsc{(T-Method)}\quad \dfrac{\begin{array}{c}E \equiv \overline{\mathsf{x} : \tau_{\mathsf{x}}}, \overline{\mathsf{z} : \tau_{\mathsf{z}}}, \mathsf{this} : \tau_{\mathsf{this}} \quad E\ \vdash\ \mathsf{s; return\, y} \quad E(\mathsf{y}) = \tau \quad \mathsf{C\ extends\ D} \\ (if\ \mathsf{C}\ has\ \mathsf{a}\ \ then\ \ \tau_{\mathsf{this}} \equiv \mathsf{C}|\mathsf{a}{=}\mathsf{this.a}|\ \ else\ \ \tau_{\mathsf{this}} \equiv \mathsf{C}) \quad override(\mathsf{m}, \mathsf{D}, \overline{\tau_{\mathsf{X}}} \to \tau)\end{array}}{\tau\ \mathsf{m}(\overline{\tau_{\mathsf{x}}\,\mathsf{x}})\,\{\,\overline{\tau_{\mathsf{z}}\,\mathsf{z}};\ \mathsf{s; return\, y}\,\}\quad \text{OK in}\ \mathsf{C}}$$

Dynamics

H ::= $[] \mid H[r \mapsto v]$	*heap*	F ::= $[] \mid F[\mathsf{y} \mapsto r]$	*stack frame*
T ::= $\rho\, S \mid \rho\, \mathsf{NPE}$	*thread*	v ::= $\mathsf{C}\vert\omega\vert(\overline{r})$	*object*
S ::= $\epsilon \mid S\,\langle \mathsf{m}\, F\ \ \mathsf{s}\rangle$	*stack*	ω ::= $r \mid \epsilon$	*owner atomic set*

$$\frac{H; \overline{T}\, \overline{T'}\, T \xrightarrow{\ell}_{\rho} H'; \overline{T}\, \overline{T'}\, T'}{H; \overline{T}\, T\, \overline{T'} \xrightarrow{\ell}_{\rho} H'; \overline{T}\, \overline{T'}\, T'}$$

$$\frac{F(\mathsf{y}) = r \quad F(\mathsf{this}) = r'}{H; \overline{T}\, \rho\, S\, \langle \mathsf{m}'\ F'\ \ \mathsf{x} = \mathsf{y}'.\mathsf{m}(\overline{\mathsf{z}}); \mathsf{s}'\rangle \langle \mathsf{m}\, F\ \ \mathsf{return}\, \mathsf{y}\rangle \xrightarrow{\leftarrow r'.\mathsf{m}}_{\rho} H; \overline{T}\, \rho\, S\, \langle \mathsf{m}'\ F'[\mathsf{x} \mapsto r]\ \ \mathsf{s}'\rangle}$$

Dynamics

$$\frac{F(\mathsf{this}) = r \quad H(r.\mathsf{f}_i) = r_i}{H;\overline{T}\,\rho\,S\,\langle \mathsf{m}\,F\;\;\mathsf{x}{=}\mathsf{this.f}_i;\mathsf{s}\rangle \xrightarrow{\uparrow r.\mathsf{f}_i}_{\rho} \quad H;\overline{T}\,\rho\,S\,\langle \mathsf{m}\,F[\mathsf{x}\mapsto r_i]\;\;\mathsf{s}\rangle}$$

(D-UPDATE)

$$\frac{F(\mathsf{this}) = r \quad F(\mathsf{x}) = r_{\mathsf{x}} \quad H(r) = \mathsf{C}|\omega|(\overline{r}, r_i, \overline{r'}) \quad H' \equiv H[r \mapsto \mathsf{C}|\omega|(\overline{r}, r_{\mathsf{x}}, \overline{r'})]}{H;\overline{T}\,\rho\,S\,\langle \mathsf{m}\,F\;\;\mathsf{this.f}_i{=}\mathsf{x};\mathsf{s}\rangle \xrightarrow{\downarrow r.\mathsf{f}_i}_{\rho} H';\overline{T}\,\rho\,S\,\langle \mathsf{m}\,F\;\;\mathsf{s}\rangle}$$

Dynamics

(D-NEW-PLAIN)

$$\frac{v \equiv \mathsf{C}|\epsilon|(\mathsf{null}_1 ... \mathsf{null}_n) \quad r \text{ is fresh} \quad \mathit{not}\ \mathsf{C}\ \mathit{has}\ \mathsf{a} \qquad H' \equiv H[r \mapsto v] \quad |\mathit{fields}(\mathsf{C})| = n \quad F' \equiv F[\mathsf{x} \mapsto r]}{H; \overline{T}\, \rho\, S\, \langle \mathsf{m}\, F\ \ \mathsf{x} = \mathsf{new}\, \mathsf{C}(); \mathsf{s}\rangle \xrightarrow{\epsilon}_{\rho} H'; \overline{T}\, \rho\, S\, \langle \mathsf{m}\, F'\ \ \mathsf{s}\rangle}$$

(D-NEW-SELF)

$$\frac{v \equiv \mathsf{C}|r|(\mathsf{null}_1 ... \mathsf{null}_n) \quad r \text{ is fresh} \quad \mathsf{C}\ \mathit{has}\ \mathsf{a} \quad H' \equiv H[r \mapsto v] \qquad |\mathit{fields}(\mathsf{C})| = n \quad F' \equiv F[\mathsf{x} \mapsto r]}{H; \overline{T}\, \rho\, S\, \langle \mathsf{m}\, F\ \ \mathsf{x} = \mathsf{new}\, \mathsf{C}(); \mathsf{s}\rangle \xrightarrow{\epsilon}_{\rho} H'; \overline{T}\, \rho\, S\, \langle \mathsf{m}\, F'\ \ \mathsf{s}\rangle}$$

(D-NEW-ALIAS)

$$\frac{H(F(\mathsf{this})) = \mathsf{D}|r'|(\overline{r}) \quad r \text{ is fresh} \quad v \equiv \mathsf{C}|r'|(\mathsf{null}_1 ... \mathsf{null}_n) \quad H' \equiv H[r \mapsto v] \qquad |\mathit{fields}(\mathsf{C})| = n \quad T \equiv \rho\, S\, \langle \mathsf{m}\, F[\mathsf{x} \mapsto r]\ \ \mathsf{s}\rangle}{H; \overline{T}\, \rho\, S\, \langle \mathsf{m}\, F\ \ \mathsf{x} = \mathsf{new}\, \mathsf{C}|\mathsf{a} = \mathsf{this.b}|(); \mathsf{s}\rangle \xrightarrow{\epsilon}_{\rho} H'; \overline{T}\, T}$$

Dynamics

(D-CALL)

$$\frac{F(\mathsf{y}) = r \quad F(\overline{\mathsf{z}}) = \overline{r} \quad H(r) = \mathsf{C}|\omega|(\overline{r'}) \quad mbody(\mathsf{C.m}) = (\overline{\tau_{\mathsf{x}}\,\mathsf{x}'};\ \overline{\tau_{\mathsf{y}}\,\mathsf{y}};\mathsf{s}';\mathsf{return}\ \mathsf{y}') \qquad F' \equiv [\overline{\mathsf{y} \mapsto \mathsf{null}}][\overline{\mathsf{x}' \mapsto r}][\mathsf{this} \mapsto r] \quad S' \equiv S\,\langle \mathsf{m}'\ F\ \ \mathsf{x}\!=\!\mathsf{y.m}(\overline{\mathsf{z}});\mathsf{s}\rangle\langle \mathsf{m}\,F'\ \ \mathsf{s}';\mathsf{return}\ \mathsf{y}'\rangle}{H;\overline{T}\,\rho\,S\,\langle \mathsf{m}'\ F\ \ \mathsf{x}\!=\!\mathsf{y.m}(\overline{\mathsf{z}});\mathsf{s}\rangle \xrightarrow{\rightarrow r.\mathsf{m}}_{\rho} H;\overline{T}\,\rho\,S'}$$

(D-CALL-NPE)

$$\frac{}{H;\overline{T}\,\rho\,S\,\langle \mathsf{m}'\ F[\mathsf{y} \mapsto \mathsf{null}]\ \ \mathsf{x}\!=\!\mathsf{y.m}(\overline{\mathsf{z}});\mathsf{s}\rangle \xrightarrow{\epsilon}_{\rho} H;\overline{T}\,\rho\,\mathsf{NPE}}$$

Properties

Theorem 1. *Preservation. If* $H; \overline{T}\, T\, \overline{T'}$ *is WF and* $H; \overline{T}\, T\, \overline{T'} \xrightarrow{\ell}_{\rho} H'; \overline{T}\, \overline{T'}\, T'$, *then* $H; \overline{T}\, \overline{T'}\, T'$ *is WF.*

We define the notion of an *active* thread as a thread that it has not stumbled on a NPE or returned from its bottommost stack frame.

Definition 1. *A thread* $T \equiv \rho\, S$ *is* active, *denoted* $active(T)$, *if* $S \not\equiv$ *NPE and* $S \not\equiv \langle$*run* F *return y*$\rangle$.

Progress requires that if there exists an active thread in a well-formed configuration, this thread should be allowed to make a step.

Theorem 2. *Progress. If* $H; \overline{T}\, T\, \overline{T'}$ *is WF and* $active(T)$, *then* $H; \overline{T}\, T\, \overline{T'} \xrightarrow{\ell}_{\rho} H'; \overline{T}\, \overline{T'}\, T'$.

WF

$$\frac{H \text{ is WF in } H \quad \overline{T} \text{ is WF in } H \quad \vdash CT}{H; \overline{T} \text{ is WF}}\ \text{(WF-CONFIGURATION)}$$

$$\frac{}{[]\ \text{ is WF in } H}\ \text{(WF-EMPTY-HEAP)}$$

$$\frac{}{\rho\, \mathsf{NPE} \text{ is WF in } H}\ \text{(WF-NPE-THREAD)}$$

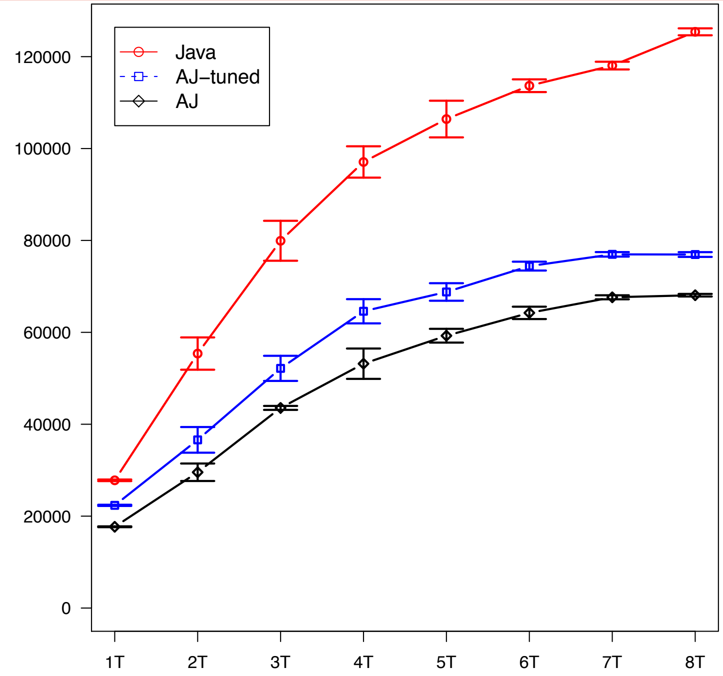
$$\frac{\begin{array}{c}\langle \mathsf{run}\, F\ \ \mathsf{s}\rangle \text{ is WF in } H \\ \mathit{not\ internal}_H(F(\mathsf{this}))\end{array}}{\rho\, \langle \mathsf{run}\, F\ \ \mathsf{s}\rangle \text{ is WF in } H}\ \text{(WF-THREAD-BOT)}$$

$$\frac{\begin{array}{c}\langle \mathsf{m}\, F\ \ \mathsf{s}\rangle \text{ is WF in } H \quad S \equiv S'\langle \mathsf{m}'\, F'\ \ \mathsf{x} = \mathsf{y}.\mathsf{m}(\overline{\mathsf{z}'}); \mathsf{s}''\rangle \\ \rho\, S \text{ is WF in } H \\ (\exists \langle \mathsf{m}''\, F''\ \ s''\rangle \in S\langle \mathsf{m}\, F\ \ \mathsf{s}\rangle,\ \mathit{not\ internal}_H(F''(\mathsf{this})) \\ \mathit{and\ owner}_H(F''(\mathsf{this})) = \mathit{owner}_H(F(\mathsf{this})))\end{array}}{\rho\, S\langle \mathsf{m}\, F\ \ \mathsf{s}\rangle \text{ is WF in } H}\ \text{(WF-THREAD)}$$

$$\frac{\begin{array}{c}(\mathsf{C}\ \mathit{has}\ \mathsf{a}\ \mathit{implies}\ \omega \neq \epsilon) \quad H' \text{ is WF in } H \\ \mathit{fields}(\mathsf{C}) = \overline{\alpha\, \tau\, \mathsf{f}} \quad \overline{r\mathsf{z} <:_{r,H} \tau}\end{array}}{H'[r \mapsto \mathsf{C}|\omega|(\overline{r\mathsf{z}})]\ \text{ is WF in } H}\ \text{(WF-HEAP)}$$

$$\frac{\begin{array}{c}\mathit{locals}(\mathsf{m}, F) = E \quad E \vdash \mathsf{s} \\ \forall\, \mathsf{x} \in \mathit{dom}(F), F(\mathsf{x}) <:_{F(\mathsf{this}),H} E(\mathsf{x})\end{array}}{\langle \mathsf{m}\, F\ \ \mathsf{s}\rangle \text{ is WF in } H}\ \text{(WF-FRAME)}$$

SPECJBB



Java
AJ–tuned
AJ
120000
100000
80000
60000
40000
20000
0
1T
2T
3T
4T
5T
6T
7T
8T