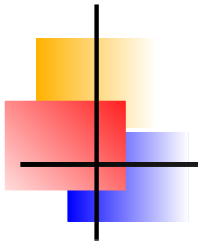
Fast Escape Analysis for Region-Based Memory Management

Guillaume Salagnac[1] , Sergio Yovine[1], Diego Garbervetsky [2]

salagnac@imag.fr, yovine@imag.fr, diegog@dc.uba.ar

(1) Laboratoire Vérimag
Grenoble
France

(2) School of Computer Science
Universidad de Buenos Aires
Argentina.



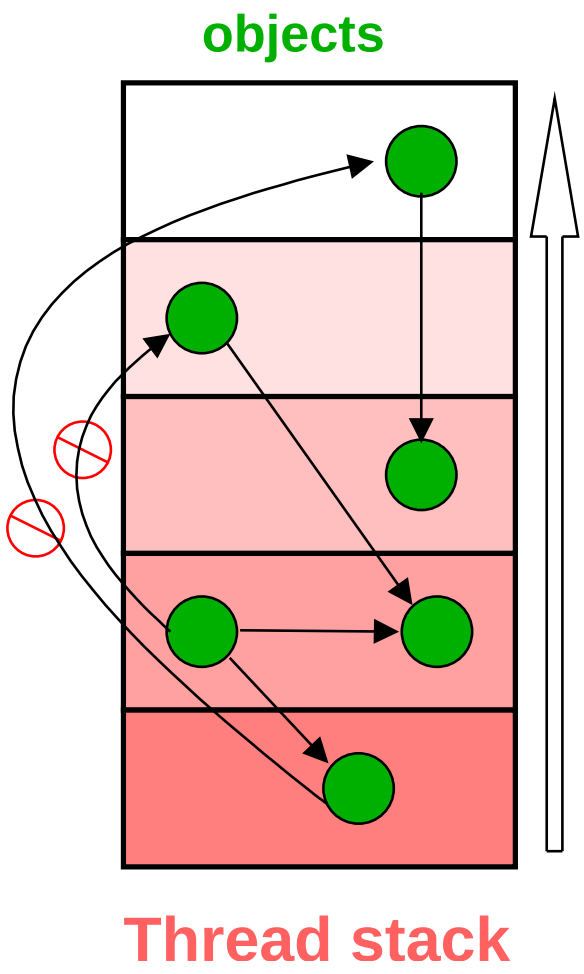
Introduction

- Java
  - Attractive language
  - No manual dynamic memory management
- Use for Real-time : Implementation pitfalls
  - Virtual machine
  - Garbage Collector

Realtime specification for Java

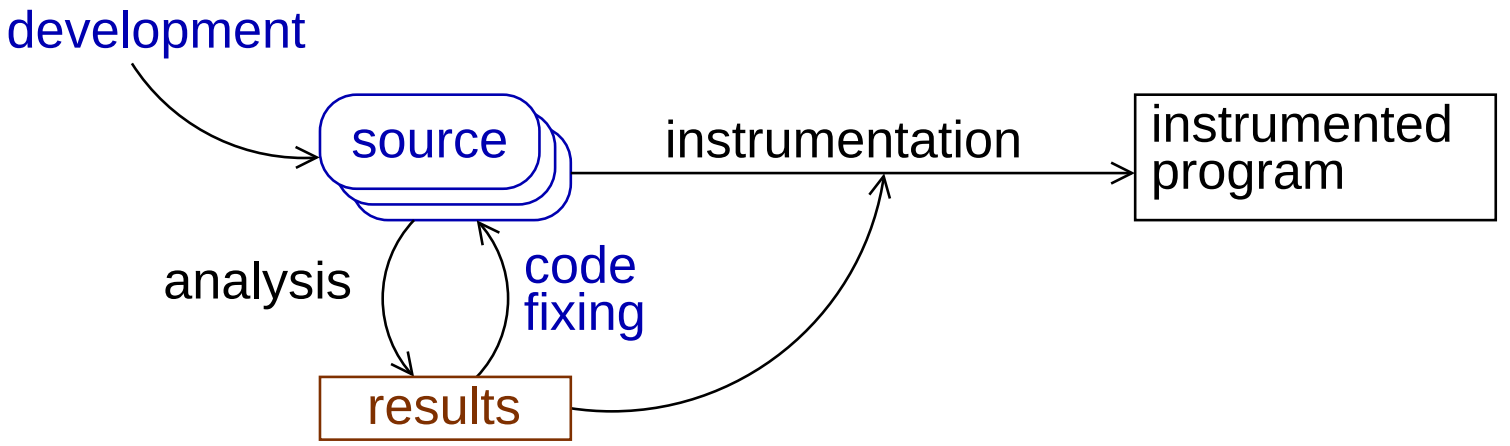
- Allocate objects in *regions*
- One region per `Runnable`
- No more Garbage Collector
- Manual bookkeeping

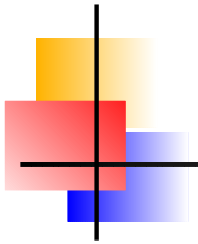
$\Longrightarrow$ changes Java paradigm.



Our approach (work in progress)

- stay close to the Java model
  - not all the advanced features
  - no manual dynamic memory management
- semi-automatized development cycle





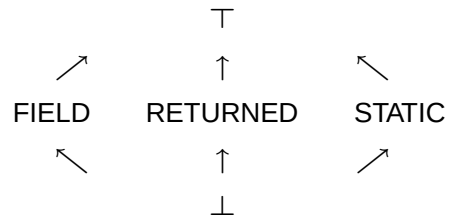
The Analysis

- tradeoff between precision and performance
- variable-based VS points-to based
- start with the simpliest/fastest algorithm and add information
  - alias analysis
  - points-to analysis
  - parameter binding at method call

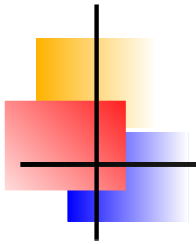
[G&S] D. Gay and B. Steensgaard. Fast escape analysis and stack allocation for object-based programs. In *CC'00*. Springer-Verlag, 2000.

The lattices

- $\forall \mathsf{v}, \mathsf{escape}(\mathsf{v}) \in Escape$



- $\forall \mathsf{v}, \mathsf{def}(\mathsf{v}) \in \{\text{NEW}, \text{RETVAL}, \text{PARAM}, \text{STATIC}, \text{COPY}, \text{PHI}, \text{FIELD}, \text{CONSTANT}\}$
- $\forall \mathsf{v}, \mathsf{fielduse}(\mathsf{v}) \subseteq \{\text{variables of } \mathtt{m}\}$
- $\forall \mathsf{v}, \mathsf{sites}(\mathsf{v}) \subseteq AllocationSites \cup \{\text{UNKNOWN}\}$
- $\mathsf{mrefs} \subseteq AllocationSites \times Fields \times AllocationSites$
- $\forall \mathsf{v}, \mathsf{side}(\mathsf{v}) \in \{\text{INSIDE}, \text{OUTSIDE}\}$



The algorithm

```
class RefObject {
    Object f;
}
class Example1 {
    void m0() {
        Object e=m1();
    }
    Object m1() {
        RefObject c=m2();
        Object d=c.f;
        return d;
    }
}
```

```
RefObject m2() {
    RefObject a=new RefObject();
    Object b=new Object();
    a.f=b;
    return a;
}
```

The algorithm

```
class RefObject {
    Object f;
}
class Example1 {
    void m0() {
        Object e=m1();
    }
    Object m1() {
        RefObject c=m2();
        Object d=c.f;
        return d;
    }
}
```

```
RefObject m2() {
    RefObject a=new RefObject();
    Object b=new Object();
    a.f=b;
    return a;
}
```

	escape
m0	⊥
e	⊥
m1	RETURNED
c	⊥
d	RETURNED
m2	RETURNED
a	RETURNED
b	FIELD

The algorithm

```
class RefObject {
    Object f;
}
class Example1 {
    void m0() {
        Object e=m1();
    }
    Object m1() {
        RefObject c=m2();
        Object d=c.f;
        return d;
    }
}
```

```
RefObject m2() {
    RefObject a=new RefObject();
    Object b=new Object();
    a.f=b;
    return a;
}
```

	escape	def
m0	⊥	
e	⊥	RETVAL
m1	RETURNED	
c	⊥	RETVAL
d	RETURNED	FIELD
m2	RETURNED	
a	RETURNED	NEW
b	FIELD	NEW

The algorithm

```
class RefObject {
    Object f;
}
class Example1 {
    void m0() {
        Object e=m1();
    }
    Object m1() {
        RefObject c=m2();
        Object d=c.f;
        return d;
    }
}
```

```
RefObject m2() {
    RefObject a=new RefObject();
    Object b=new Object();
    a.f=b;
    return a;
}
```

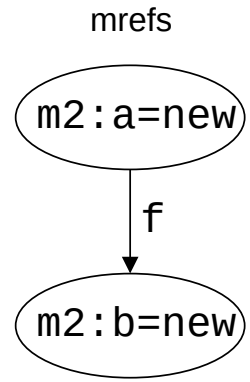
	escape	def	fielduse
m0	⊥		
e	⊥	RETVAL	∅
m1	RETURNED		
c	⊥	RETVAL	∅
d	RETURNED	FIELD	∅
m2	RETURNED		
a	RETURNED	NEW	∅
b	FIELD	NEW	[a]

The algorithm

```
class RefObject {
    Object f;
}
class Example1 {
    void m0() {
        Object e=m1();
    }
    Object m1() {
        RefObject c=m2();
        Object d=c.f;
        return d;
    }
}
```

```
RefObject m2() {
    RefObject a=new RefObject();
    Object b=new Object();
    a.f=b;
    return a;
}
```

	escape	def	fielduse	sites
m0	⊥			∅
e	⊥	RETVAL	∅	[m2:b=new]
m1	RETURNED			[m2:b=new]
c	⊥	RETVAL	∅	[m2:a=new]
d	RETURNED	FIELD	∅	[m2:b=new]
m2	RETURNED			[m2:a=new]
a	RETURNED	NEW	∅	[m2:a=new]
b	FIELD	NEW	[a]	[m2:b=new]

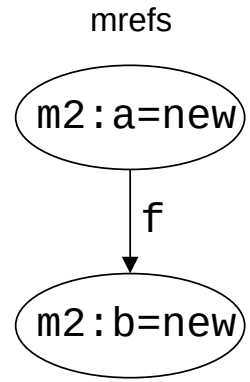


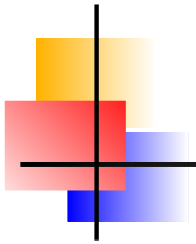
The algorithm

```
class RefObject {
    Object f;
}
class Example1 {
    void m0() {
        Object e=m1();
    }
    Object m1() {
        RefObject c=m2();
        Object d=c.f;
        return d;
    }
}
```

```
RefObject m2() {
    RefObject a=new RefObject();
    Object b=new Object();
    a.f=b;
    return a;
}
```

	escape	def	fielduse	sites	side
m0	⊥			∅	
e	⊥	RETVAL	∅	[m2:b=new]	INSIDE
m1	RETURNED			[m2:b=new]	
c	⊥	RETVAL	∅	[m2:a=new]	INSIDE
d	RETURNED	FIELD	∅	[m2:b=new]	OUTSIDE
m2	RETURNED			[m2:a=new]	
a	RETURNED	NEW	∅	[m2:a=new]	OUTSIDE
b	FIELD	NEW	[a]	[m2:b=new]	OUTSIDE





Example2

```
class RefObject {
    Object f;
}
class Example1 {
    void m0() {
        Object e=m1();
    }
    Object m1() {
        RefObject c=m2();
        Object d=c.f;
        return d;
    }
}
static Object s;

RefObject m2() {
    RefObject a=new RefObject();
    Object b=new Object();
    a.f=b;
    s=b;
    return a;
}
```

Example2

```
class RefObject {
    Object f;
}
class Example1 {
    void m0() {
        Object e=m1();
    }
    Object m1() {
        RefObject c=m2();
        Object d=c.f;
        return d;
    }
    static Object s;

    RefObject m2() {
        RefObject a=new RefObject();
        Object b=new Object();
        a.f=b;
        s=b;
        return a;
    }
}
```

	escape
m0	⊥
e	⊥
m1	RETURNED
c	⊥
d	RETURNED
m2	RETURNED
a	RETURNED
b	⊤

Example2

```
class RefObject {
    Object f;
}
class Example1 {
    void m0() {
        Object e=m1();
    }
    Object m1() {
        RefObject c=m2();
        Object d=c.f;
        return d;
    }
    static Object s;

    RefObject m2() {
        RefObject a=new RefObject();
        Object b=new Object();
        a.f=b;
        s=b;
        return a;
    }
}
```

	escape	def
m0	⊥	
e	⊥	RETVAL
m1	RETURNED	
c	⊥	RETVAL
d	RETURNED	FIELD
m2	RETURNED	
a	RETURNED	NEW
b	⊤	NEW

Example2

```
class RefObject {
    Object f;
}
class Example1 {
    void m0() {
        Object e=m1();
    }
    Object m1() {
        RefObject c=m2();
        Object d=c.f;
        return d;
    }
}
```

```
static Object s;

RefObject m2() {
    RefObject a=new RefObject();
    Object b=new Object();
    a.f=b;
    s=b;
    return a;
}
```

	escape	def	fielduse
m0	⊥		
e	⊥	RETVAL	∅
m1	RETURNED		
c	⊥	RETVAL	∅
d	RETURNED	FIELD	∅
m2	RETURNED		
a	RETURNED	NEW	∅
b	⊤	NEW	[a]

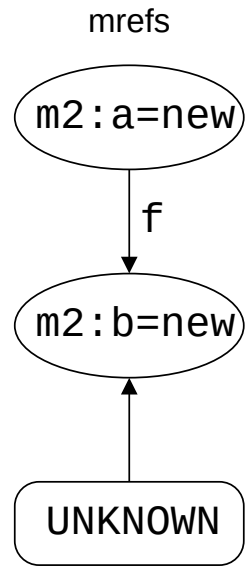
Example2

```
class RefObject {
    Object f;
}
class Example1 {
    void m0() {
        Object e=m1();
    }
    Object m1() {
        RefObject c=m2();
        Object d=c.f;
        return d;
    }
}
```

```
static Object s;

RefObject m2() {
    RefObject a=new RefObject();
    Object b=new Object();
    a.f=b;
    s=b;
    return a;
}
```

	escape	def	fielduse	sites
m0	⊥			∅
e	⊥	RETVAL	∅	[m2:b=new]
m1	RETURNED			[m2:b=new]
c	⊥	RETVAL	∅	[m2:a=new]
d	RETURNED	FIELD	∅	[m2:b=new]
m2	RETURNED			[m2:a=new]
a	RETURNED	NEW	∅	[m2:a=new]
b	⊤	NEW	[a]	[m2:b=new]



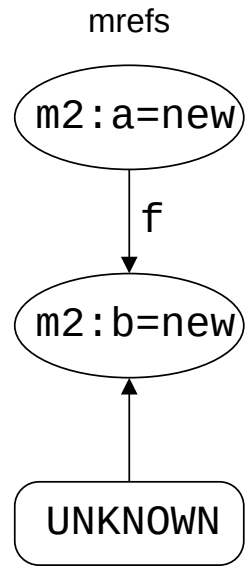
Example2

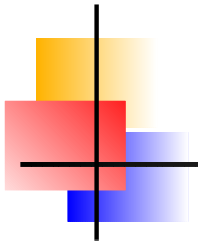
```
class RefObject {
    Object f;
}
class Example1 {
    void m0() {
        Object e=m1();
    }
    Object m1() {
        RefObject c=m2();
        Object d=c.f;
        return d;
    }
}
```

```
static Object s;

RefObject m2() {
    RefObject a=new RefObject();
    Object b=new Object();
    a.f=b;
    s=b;
    return a;
}
```

	escape	def	fielduse	sites	side
m0	⊥			∅	
e	⊥	RETVAL	∅	[m2:b=new]	OUTSIDE
m1	RETURNED			[m2:b=new]	
c	⊥	RETVAL	∅	[m2:a=new]	INSIDE
d	RETURNED	FIELD	∅	[m2:b=new]	OUTSIDE
m2	RETURNED			[m2:a=new]	
a	RETURNED	NEW	∅	[m2:a=new]	OUTSIDE
b	⊤	NEW	[a]	[m2:b=new]	OUTSIDE





Example3

```
class RefObject {
    Object f;
}
class Example3 {
    void m0() {
        RefObject t1=new RefObject();
        Object t2=new Object();
        t1.f=t2;
        m1(t1);
    }
    void m1(RefObject x)
    {
        Object a=x.f;
        Object b=a;
    }
}
```

Example3

```
class RefObject {
    Object f;
}
class Example3 {
    void m0() {
        RefObject t1=new RefObject();
        Object t2=new Object();
        t1.f=t2;
        m1(t1);
    }
    void m1(RefObject x)
    {
        Object a=x.f;
        Object b=a;
    }
}
```

	escape
m0	⊥
t1	⊥
t2	FIELD
m1	⊥
x	⊥
a	⊥
b	⊥

Example3

```
class RefObject {
    Object f;
}
class Example3 {
    void m0() {
        RefObject t1=new RefObject();
        Object t2=new Object();
        t1.f=t2;
        m1(t1);
    }
    void m1(RefObject x)
    {
        Object a=x.f;
        Object b=a;
    }
}
```

	escape	def
m0	⊥	
t1	⊥	NEW
t2	FIELD	NEW
m1	⊥	
x	⊥	PARAM
a	⊥	FIELD
b	⊥	COPY

Example3

```
class RefObject {
    Object f;
}
class Example3 {
    void m0() {
        RefObject t1=new RefObject();
        Object t2=new Object();
        t1.f=t2;
        m1(t1);
    }
    void m1(RefObject x)
    {
        Object a=x.f;
        Object b=a;
    }
}
```

	escape	def	IsD	uP
m0	⊥			
t1	⊥	NEW	true	true
t2	FIELD	NEW	false	false
m1	⊥			
x	⊥	PARAM	true	false
a	⊥	FIELD	false	false
b	⊥	COPY	false	false

Example3

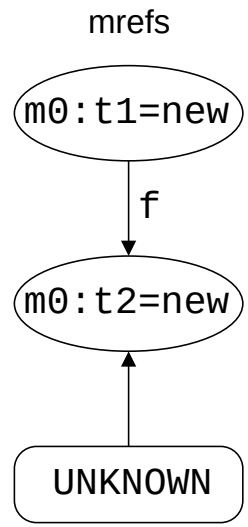
```
class RefObject {
    Object f;
}
class Example3 {
    void m0() {
        RefObject t1=new RefObject();
        Object t2=new Object();
        t1.f=t2;
        m1(t1);
    }
    void m1(RefObject x)
    {
        Object a=x.f;
        Object b=a;
    }
}
```

	escape	def	IsD	uP	fielduse
m0	⊥				
t1	⊥	NEW	true	true	∅
t2	FIELD	NEW	false	false	[t1]
m1	⊥				
x	⊥	PARAM	true	false	∅
a	⊥	FIELD	false	false	∅
b	⊥	COPY	false	false	∅

Example3

```
class RefObject {
    Object f;
}
class Example3 {
    void m0() {
        RefObject t1=new RefObject();
        Object t2=new Object();
        t1.f=t2;
        m1(t1);
    }
    void m1(RefObject x)
    {
        Object a=x.f;
        Object b=a;
    }
}
```

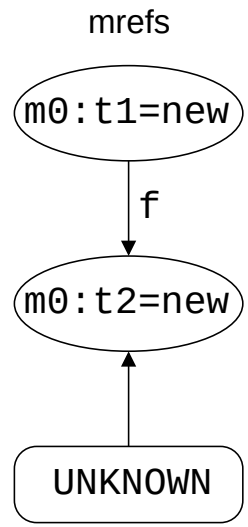
	escape	def	IsD	uP	fielduse	sites
m0	⊥					∅
t1	⊥	NEW	true	true	∅	[m0:t1=new]
t2	FIELD	NEW	false	false	[t1]	[m0:t2=new]
m1	⊥					∅
x	⊥	PARAM	true	false	∅	[UNKNOWN]
a	⊥	FIELD	false	false	∅	[UNKNOWN]
b	⊥	COPY	false	false	∅	[UNKNOWN]

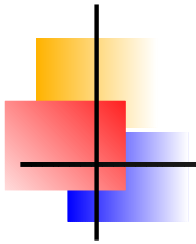


Example3

```
class RefObject {
    Object f;
}
class Example3 {
    void m0() {
        RefObject t1=new RefObject();
        Object t2=new Object();
        t1.f=t2;
        m1(t1);
    }
    void m1(RefObject x)
    {
        Object a=x.f;
        Object b=a;
    }
}
```

	escape	def	IsD	uP	fielduse	sites	side
m0	⊥					∅	
t1	⊥	NEW	true	true	∅	[m0:t1=new]	INSIDE
t2	FIELD	NEW	false	false	[t1]	[m0:t2=new]	OUTSIDE
m1	⊥					∅	
x	⊥	PARAM	true	false	∅	[UNKNOWN]	OUTSIDE
a	⊥	FIELD	false	false	∅	[UNKNOWN]	OUTSIDE
b	⊥	COPY	false	false	∅	[UNKNOWN]	OUTSIDE

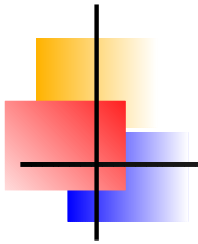




Benchmarks

Program	Lines	Allocation sites	INSIDE variables	INSIDE sites	G&S's *stackable* variables	INSIDE sites vs. total sites	INSIDE variables vs. *stackable* vars.
bh	1128	41	34	21	23	51%	147%
bisort	340	10	7	7	7	70%	100%
em3d	462	26	13	11	11	42%	118%
health	562	28	18	13	10	46%	180%
mst	473	16	8	8	7	50%	114%
perimeter	745	13	7	7	7	53%	100%
power	765	21	9	9	5	42%	180%
treeadd	195	11	6	6	6	54%	100%
tsp	545	12	7	7	7	58%	100%
voronoi	1000	35	34	20	31	57%	109%

[JOlden benchmarks] B. Cahoon and K. S. McKinley. Data flow analysis for software prefetching linked data structures in java controller. In *PACT'01*, p. 280–291, Sep. 2001.



Perspectives

- Work in progress
  - more precise than G&S
  - nearly as efficient
  - no calling context $\rightarrow$ lack of precision
- To be done
  - use results to experiment at runtime
  - take more information into account (parameter binding at method calls...)
  - thorough comparison with other tools (like Flex)