EVERYTHING YOU NEED TO KNOW ABOUT POINTER ANALYSIS

10 RULES

George Kastrinis ~ University of Athens ~ PLAST lab

BASED ON WORK FROM

- “Efficient and Effective Handling of Exceptions in Java Points-To Analysis”
  Kastrinis G., Smaragdakis Y. (CC'13)
- “Pick Your Contexts Well: Understanding Object-Sensitivity”
  Smaragdakis Y., Bravenboer M., Lhoták O. (POPL'11)
- “Strictly Declarative Specification of Sophisticated Points-to Analyses”
  Bravenboer M., Smaragdakis Y. (OOPSLA'09)

University of Athens ~ PL lab (PLAST)

Yannis Smaragdakis George Kastrinis George Balatsouras

Aggelos Biboudis Kostas Ferles George Kollias Prodromos Gerakios

IN SHORT: WHAT OBJECTS CAN A VARIABLE POINT TO?

IN SHORT: WHAT OBJECTS CAN A VARIABLE POINT TO?

```
void foo() {
    a = new A1();
    b = id(a);
}

void bar() {
    a = new A2();
    b = id(a);
}

A id(A a) {
    return a;
}
```

IN SHORT: WHAT OBJECTS CAN A VARIABLE POINT TO?

```
void foo() {
    a = new A1();
    b = id(a);
}

void bar() {
    a = new A2();
    b = id(a);
}

A id(A a) {
    return a;
}
```

```
foo::a → new A1()
bar::a → new A2()
```

IN SHORT: WHAT OBJECTS CAN A VARIABLE POINT TO?

```
void foo() {
    a = new A1();
    b = id(a);
}

void bar() {
    a = new A2();
    b = id(a);
}

A id(A a) {
    return a;
}
```

```
foo::a → new A1()
bar::a → new A2()
```

Represent objects as allocation sites

IN SHORT: WHAT OBJECTS CAN A VARIABLE POINT TO?

```
void foo() {
    a = new A1();
    b = id(a);
}

void bar() {
    a = new A2();
    b = id(a);
}

A id(A a) {
    return a;
}
```

```
foo::a → new A1()
bar::a → new A2()
 id::a → new A1(), new A2()
```

IN SHORT: WHAT OBJECTS CAN A VARIABLE POINT TO?

```
void foo() {
    a = new A1();
    b = id(a);
}

void bar() {
    a = new A2();
    b = id(a);
}

A id(A a) {
    return a;
}
```

```
foo::a → new A1()
bar::a → new A2()
 id::a → new A1(), new A2()
foo::b → new A1(), new A2()
bar::b → new A1(), new A2()
```

IN SHORT: WHAT OBJECTS CAN A VARIABLE POINT TO?

```
void foo() {
    a = new A1();
    b = id(a);
}

void bar() {
    a = new A2();
    b = id(a);
}

A id(A a) {
    return a;
}
```

```
foo::a → new A1()
bar::a → new A2()
 id::a → new A1(), new A2()
foo::b → new A1(), new A2()
bar::b → new A1(), new A2()
```

Not the most precise, right?

IN SHORT: WHAT OBJECTS CAN A VARIABLE POINT TO?

```
void foo() {
    a = new A1();
    b = id(a);
}

void bar() {
    a = new A2();
    b = id(a);
}

A id(A a) {
    return a;
}
```

```
foo::a → new A1()
bar::a → new A2()
 id::a → new A1(), new A2()
```

IN SHORT: WHAT OBJECTS CAN A VARIABLE POINT TO?

```
void foo() {
    a = new A1();
    b = id(a);
}

void bar() {
    a = new A2();
    b = id(a);
}

A id(A a) {
    return a;
}
```

```
foo::a → new A1()
bar::a → new A2()
 id::a → new A1(), new A2()
```

Add “context” to variables

IN SHORT: WHAT OBJECTS CAN A VARIABLE POINT TO?

```
void foo() {
    a = new A1();
    b = id(a);
}

void bar() {
    a = new A2();
    b = id(a);
}

A id(A a) {
    return a;
}
```

```
foo::a → new A1()
bar::a → new A2()
 id::a (foo) → new A1()
 id::a (bar) → new A2()
```

Add “context” to variables

IN SHORT: WHAT OBJECTS CAN A VARIABLE POINT TO?

```
void foo() {
    a = new A1();
    b = id(a);
}

void bar() {
    a = new A2();
    b = id(a);
}

A id(A a) {
    return a;
}
```

```
foo::a → new A1()
bar::a → new A2()
 id::a (foo) → new A1()
 id::a (bar) → new A2()
foo::b → new A1()
bar::b → new A2()
```

Add “context” to variables

10 RULES?
NO ALGORITHMS?

10 RULES?
NO ALGORITHMS?

Using Datalog

10 RULES?
NO ALGORITHMS?

~~Using Datalog~~ Purely declarative

de·clar·a·tive

/di'kle(ə)rətiv/

Adjective

Computing denoting high-level programming languages which can be used to solve problems without requiring the programmer to specify an exact procedure to be followed.

~ Oxford dictionaries

de·clar·a·tive

/di'kle(ə)rətiv/

From algorithms to specifications

Adjective

Computing denoting high-level programming languages which can be used to solve problems without requiring the programmer to specify an exact procedure to be followed.

~ Oxford dictionaries

Datalog Rules

```
INTERPROCASSIGN(to, calleeCtx, from, callerCtx) ←
    CALLGRAPH(invo, callerCtx, meth, calleeCtx),
    ACTUALARG(invo, i, from), FORMALARG(meth, i, to).
```

Datalog Rules

head

```
INTERPROCASSIGN(to, calleeCtx, from, callerCtx) ←
    CALLGRAPH(invo, callerCtx, meth, calleeCtx),
    ACTUALARG(invo, i, from), FORMALARG(meth, i, to).
```

Datalog Rules

$\textsc{InterProcAssign}(\mathit{to},\ \mathit{calleeCtx},\ \mathit{from},\ \mathit{callerCtx}) \leftarrow$
$\quad \underbrace{\textsc{CallGraph}(\mathit{invo},\ \mathit{callerCtx},\ \mathit{meth},\ \mathit{calleeCtx}),\ \textsc{ActualArg}(\mathit{invo},\ i,\ \mathit{from}),\ \textsc{FormalArg}(\mathit{meth},\ i,\ \mathit{to}).}_{\text{body}}$

Datalog Rules

Output relations in red

```
INTERPROCASSIGN(to, calleeCtx, from, callerCtx) ←
    CALLGRAPH(invo, callerCtx, meth, calleeCtx),
    ACTUALARG(invo, i, from), FORMALARG(meth, i, to).
```

Datalog Rules

```
INTERPROCASSIGN(to, calleeCtx, from, callerCtx) ←
    CALLGRAPH(invo, callerCtx, meth, calleeCtx),
    ACTUALARG(invo, i, from), FORMALARG(meth, i, to).
```

Input relations in blue

#1

Datalog Rules

meth(..,from,..)

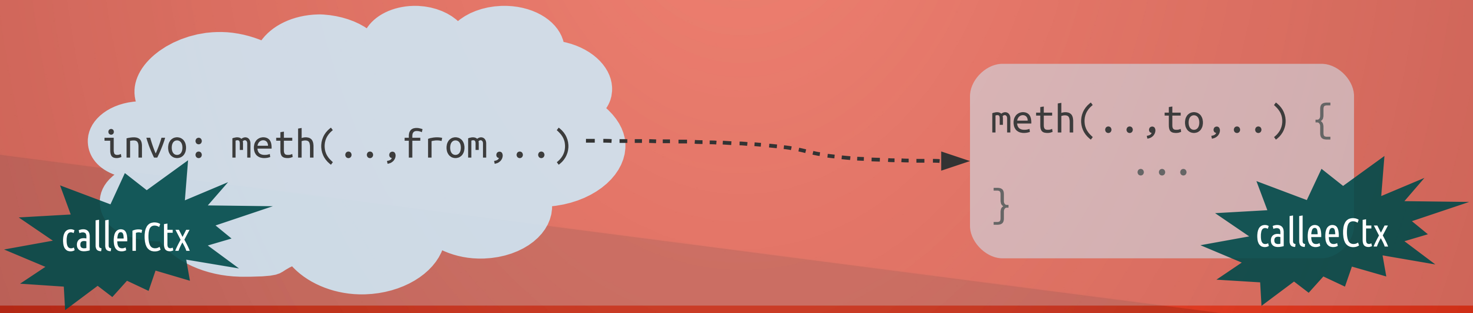
```
INTERPROCASSIGN(to, calleeCtx, from, callerCtx) ←
    CALLGRAPH(invo, callerCtx, meth, calleeCtx),
    ACTUALARG(invo, i, from), FORMALARG(meth, i, to).
```

#1

Datalog Rules

meth(..,from,..)

```
INTERPROCASSIGN(to, calleeCtx, from, callerCtx) ←
    CALLGRAPH(invo, callerCtx, meth, calleeCtx),
    ACTUALARG(invo, i, from), FORMALARG(meth, i, to).
```

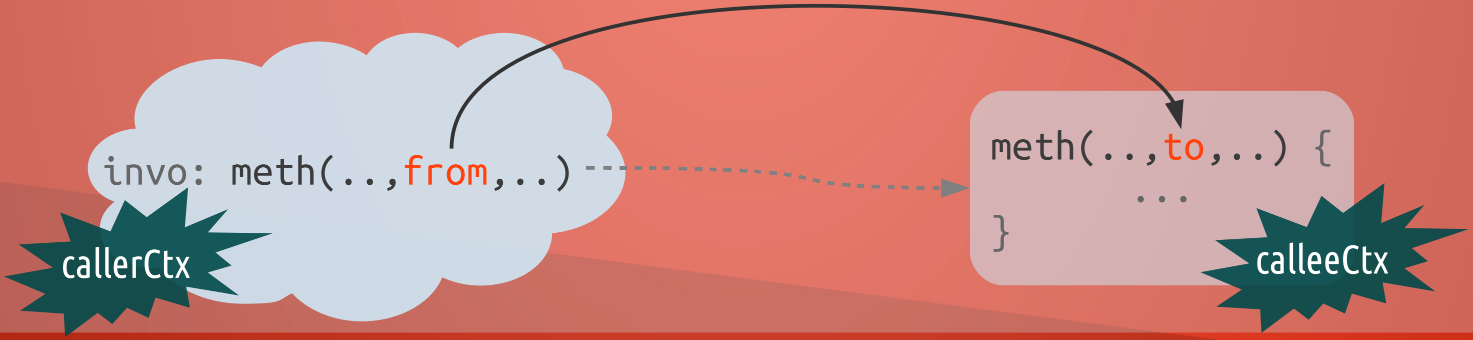


#1

Datalog Rules

meth(..,from,..)

```
INTERPROCASSIGN(to, calleeCtx, from, callerCtx) ←
    CALLGRAPH(invo, callerCtx, meth, calleeCtx),
    ACTUALARG(invo, i, from), FORMALARG(meth, i, to).
```



#2

Datalog Rules

to = meth(..)

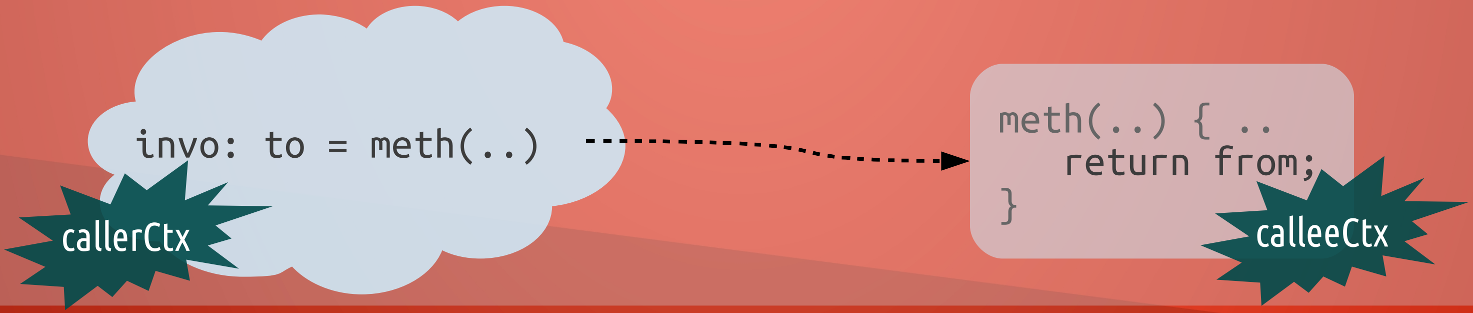
```
INTERPROCASSIGN(to, callerCtx, from, calleeCtx) ←
    CALLGRAPH(invo, callerCtx, meth, calleeCtx),
    ACTUALRETURN(invo, to), FORMALRETURN(meth, from).
```

#2

Datalog Rules

to = meth(..)

```
INTERPROCASSIGN(to, callerCtx, from, calleeCtx) ←
    CALLGRAPH(invo, callerCtx, meth, calleeCtx),
    ACTUALRETURN(invo, to), FORMALRETURN(meth, from).
```

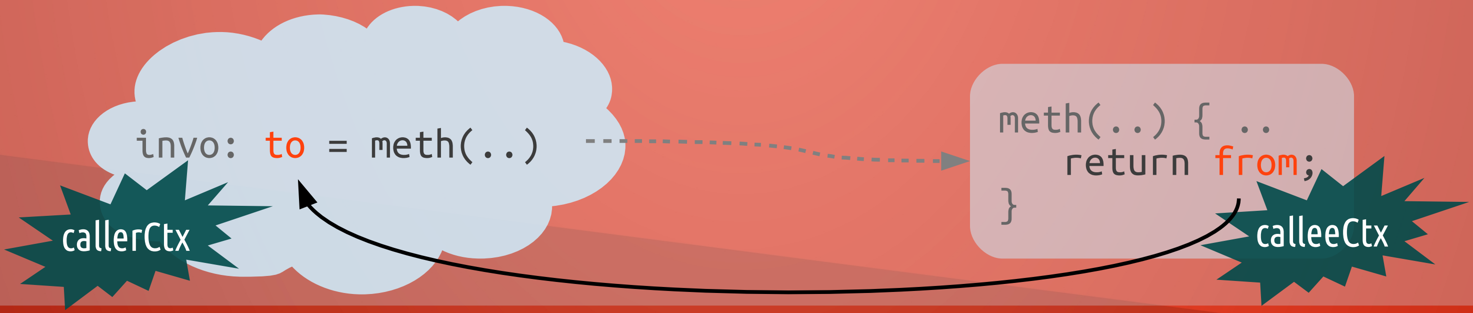


```
meth(..) { ..
   return from;
}
```

#2 Datalog Rules

to = meth(..)

```
INTERPROCASSIGN(to, callerCtx, from, calleeCtx) ←
    CALLGRAPH(invo, callerCtx, meth, calleeCtx),
    ACTUALRETURN(invo, to), FORMALRETURN(meth, from).
```



#3

Datalog Rules

var = new A()

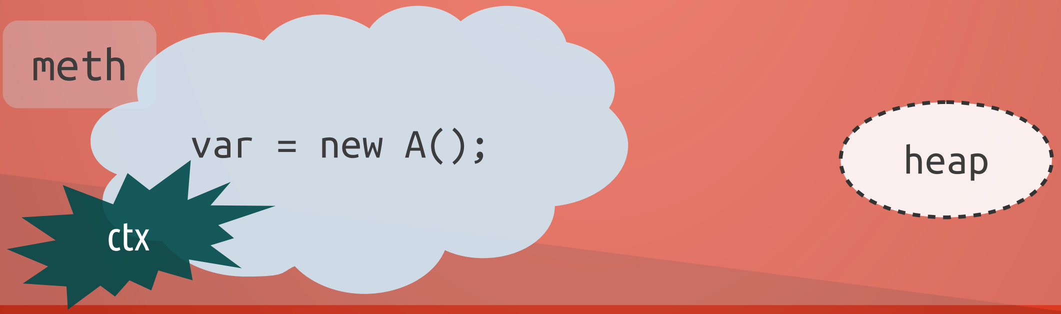
```
RECORD(heap, ctx) = hctx,
VARPOINTSTO(var, ctx, heap, hctx) ←
    REACHABLE(meth, ctx), ALLOC(var, heap, meth).
```

#3

Datalog Rules

var = new A()

```
RECORD(heap, ctx) = hctx,
VARPOINTSTO(var, ctx, heap, hctx) ←
    REACHABLE(meth, ctx), ALLOC(var, heap, meth).
```



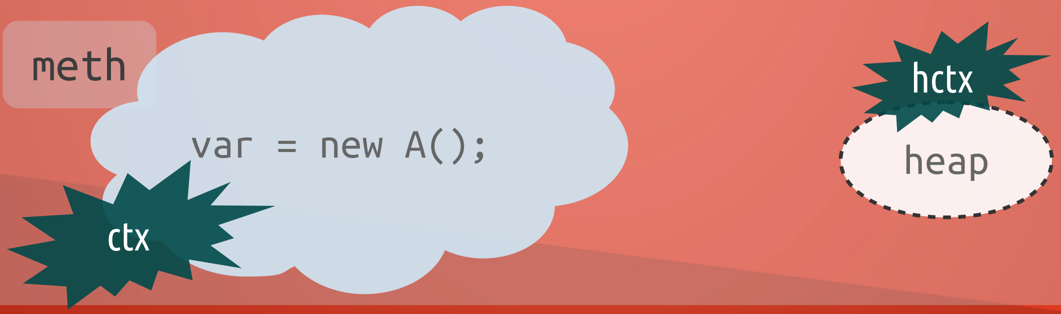
#3

Datalog Rules

var = new A()

new context!

```
RECORD(heap, ctx) = hctx,
VARPOINTSTO(var, ctx, heap, hctx) ←
    REACHABLE(meth, ctx), ALLOC(var, heap, meth).
```

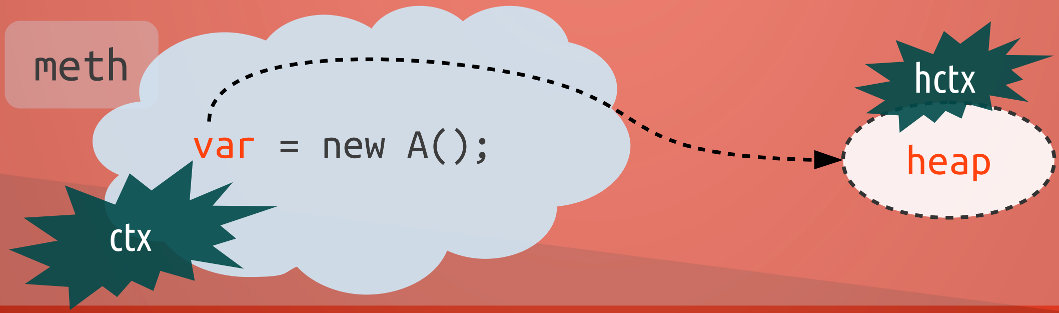


#3

Datalog Rules

var = new A()

```
RECORD(heap, ctx) = hctx,
VARPOINTSTO(var, ctx, heap, hctx) ←
    REACHABLE(meth, ctx), ALLOC(var, heap, meth).
```



#4

Datalog Rules

to = from

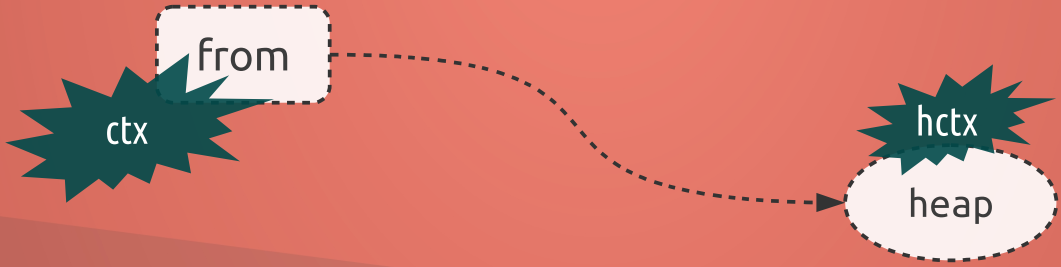
```
VARPOINTSTO(to, ctx, heap, hctx) ←
    VARPOINTSTO(from, ctx, heap, hctx), MOVE(to, from).
```

#4

Datalog Rules

to = from

```
VARPOINTSTO(to, ctx, heap, hctx) ←
    VARPOINTSTO(from, ctx, heap, hctx), MOVE(to, from).
```

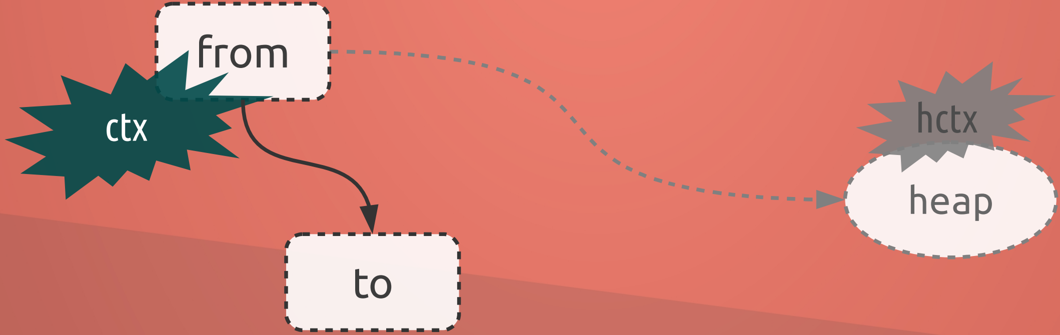


#4

Datalog Rules

to = from

```
VARPOINTSTO(to, ctx, heap, hctx) ←
    VARPOINTSTO(from, ctx, heap, hctx), MOVE(to, from).
```

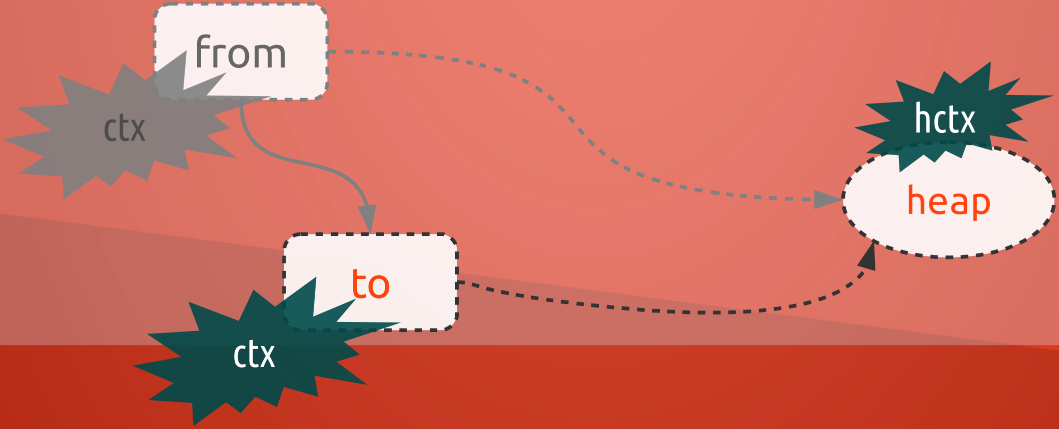


#4

Datalog Rules

to = from

```
VARPOINTSTO(to, ctx, heap, hctx) ←
    VARPOINTSTO(from, ctx, heap, hctx), MOVE(to, from).
```



#4

Datalog Rules

to = from

```
VARPOINTSTO(to, ctx, heap, hctx) ←
    VARPOINTSTO(from, ctx, heap, hctx), MOVE(to, from).
```

Recursion!

#5

Datalog Rules

to ≈ from

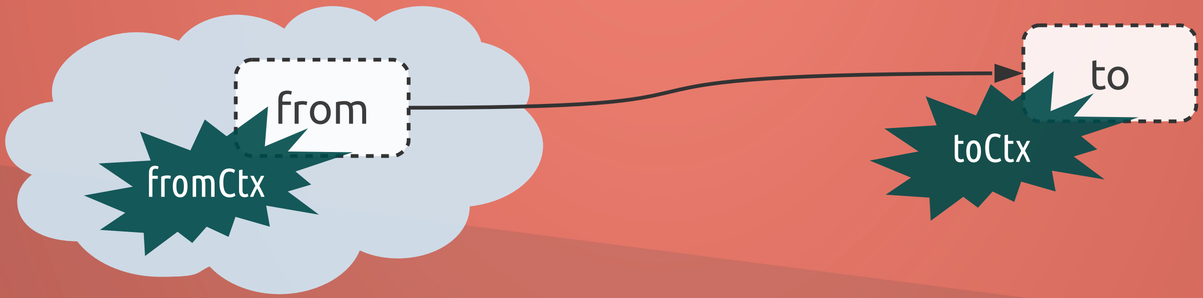
```
VARPOINTSTO(to, toCtx, heap, hctx) ←
    INTERPROCASSIGN(to, toCtx, from, fromCtx),
    VARPOINTSTO(from, fromCtx, heap, hctx).
```

#5

Datalog Rules

to ≈ from

```
VARPOINTSTO(to, toCtx, heap, hctx) ←
    INTERPROCASSIGN(to, toCtx, from, fromCtx),
    VARPOINTSTO(from, fromCtx, heap, hctx).
```

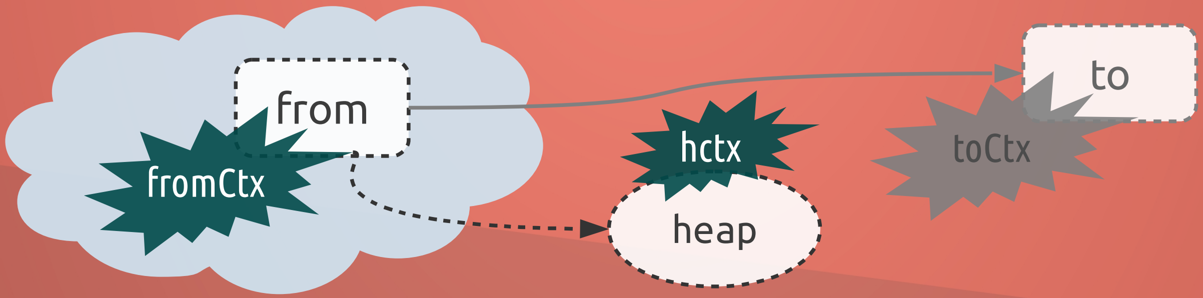


#5

Datalog Rules

to ≈ from

```
VARPOINTSTO(to, toCtx, heap, hctx) ←
    INTERPROCASSIGN(to, toCtx, from, fromCtx),
    VARPOINTSTO(from, fromCtx, heap, hctx).
```

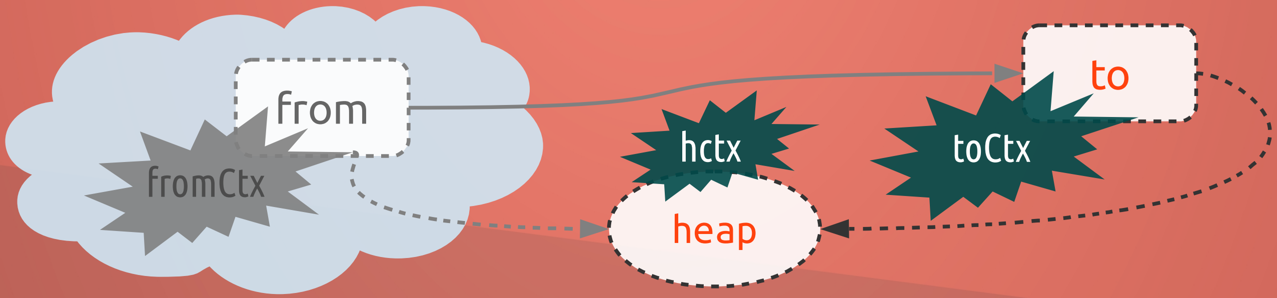


#5

Datalog Rules

to ≈ from

```
VARPOINTSTO(to, toCtx, heap, hctx) ←
    INTERPROCASSIGN(to, toCtx, from, fromCtx),
    VARPOINTSTO(from, fromCtx, heap, hctx).
```



#6

Datalog Rules

base.fld = from

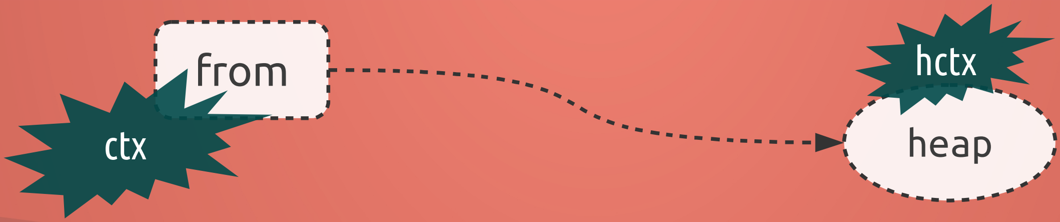
```
FLDPOINTSTO(baseH, baseHCtx, fld, heap, hctx) ←
    VARPOINTSTO(from, ctx, heap, hctx),
    STORE(base, fld, from), VARPOINTSTO(base, ctx, baseH, baseHCtx).
```

#6

Datalog Rules

base.fld = from

```
FLDPOINTSTO(baseH, baseHCtx, fld, heap, hctx) ←
     VARPOINTSTO(from, ctx, heap, hctx),
     STORE(base, fld, from), VARPOINTSTO(base, ctx, baseH, baseHCtx).
```

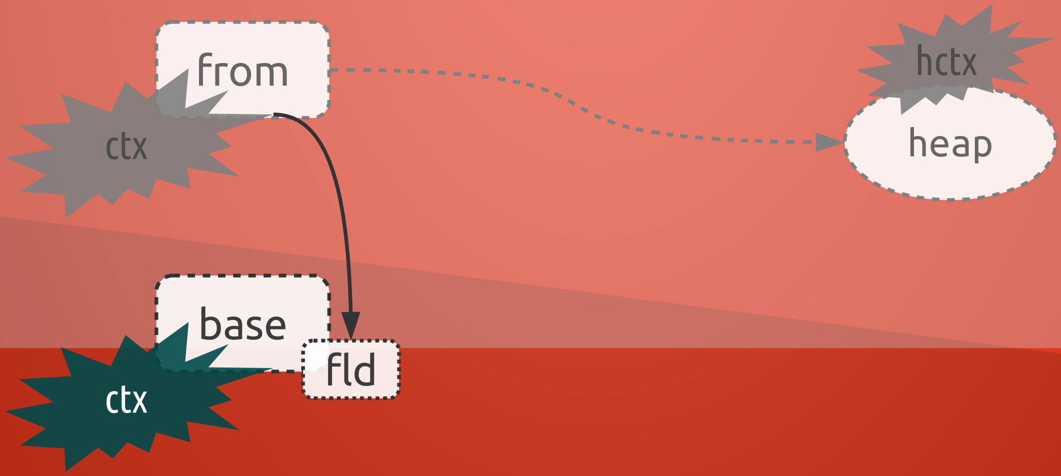


#6

Datalog Rules

base.fld = from

```
FLDPOINTSTO(baseH, baseHCtx, fld, heap, hctx) ←
    VARPOINTSTO(from, ctx, heap, hctx),
    STORE(base, fld, from), VARPOINTSTO(base, ctx, baseH, baseHCtx).
```

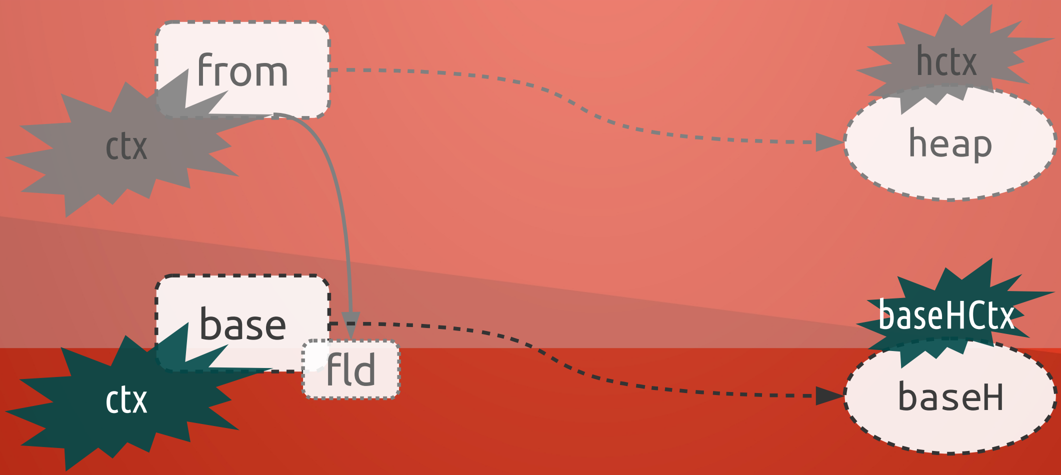


#6

Datalog Rules

base.fld = from

```
FLDPOINTSTO(baseH, baseHCtx, fld, heap, hctx) ←
    VARPOINTSTO(from, ctx, heap, hctx),
    STORE(base, fld, from), VARPOINTSTO(base, ctx, baseH, baseHCtx).
```

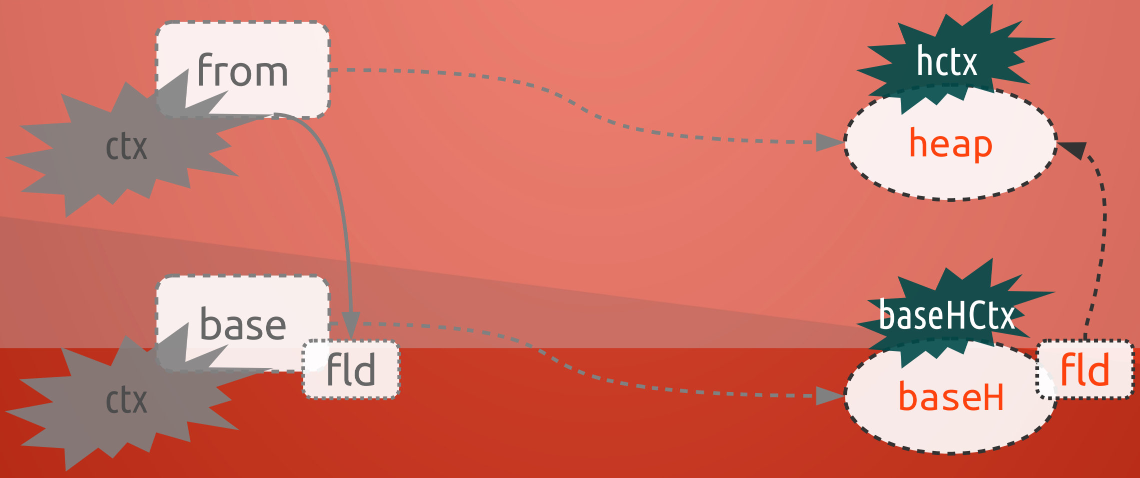


#6

Datalog Rules

base.fld = from

```
FLDPOINTSTO(baseH, baseHCtx, fld, heap, hctx) ←
    VARPOINTSTO(from, ctx, heap, hctx),
    STORE(base, fld, from), VARPOINTSTO(base, ctx, baseH, baseHCtx).
```



#7

Datalog Rules

to = base.fld

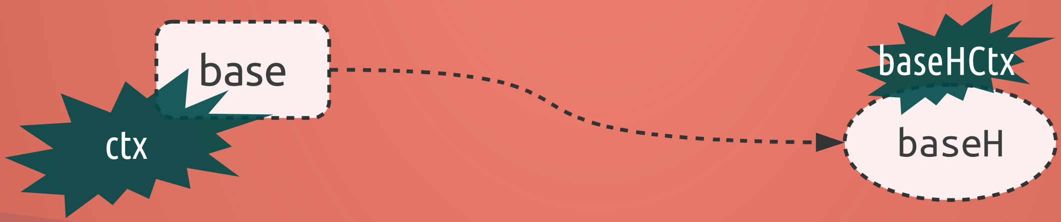
```
VARPOINTSTO(to, ctx, heap, hctx) ←
    VARPOINTSTO(base, ctx, baseH, baseHCtx),
    FLDPOINTSTO(baseH, baseHCtx, fld, heap, hctx), LOAD(to, base, fld).
```

#7

Datalog Rules

to = base.fld

```
VARPOINTSTO(to, ctx, heap, hctx) ←
    VARPOINTSTO(base, ctx, baseH, baseHCtx),
    FLDPOINTSTO(baseH, baseHCtx, fld, heap, hctx), LOAD(to, base, fld).
```

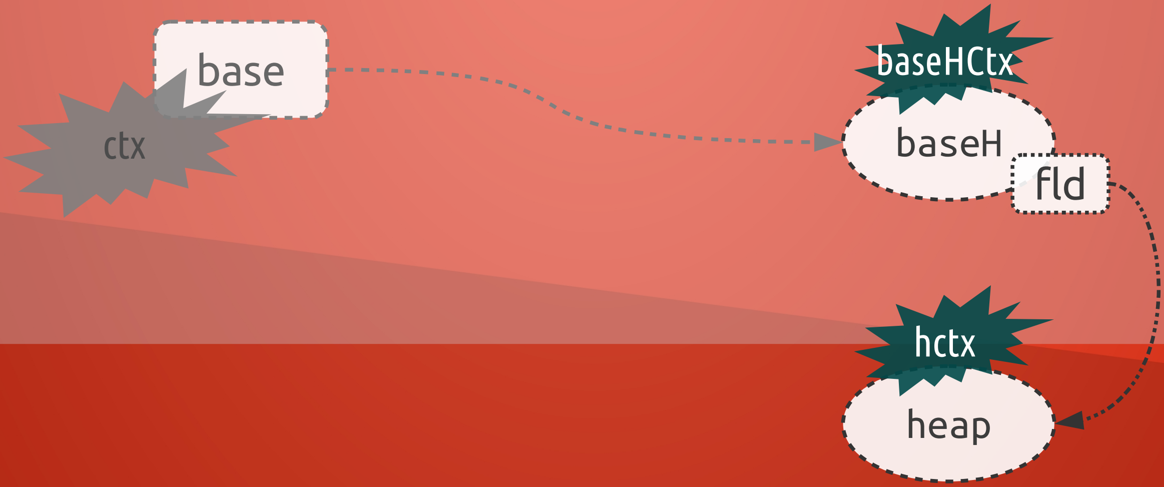


#7

Datalog Rules

to = base.fld

```
VarPointsTo(to, ctx, heap, hctx) ←
    VarPointsTo(base, ctx, baseH, baseHCtx),
    FldPointsTo(baseH, baseHCtx, fld, heap, hctx), Load(to, base, fld).
```

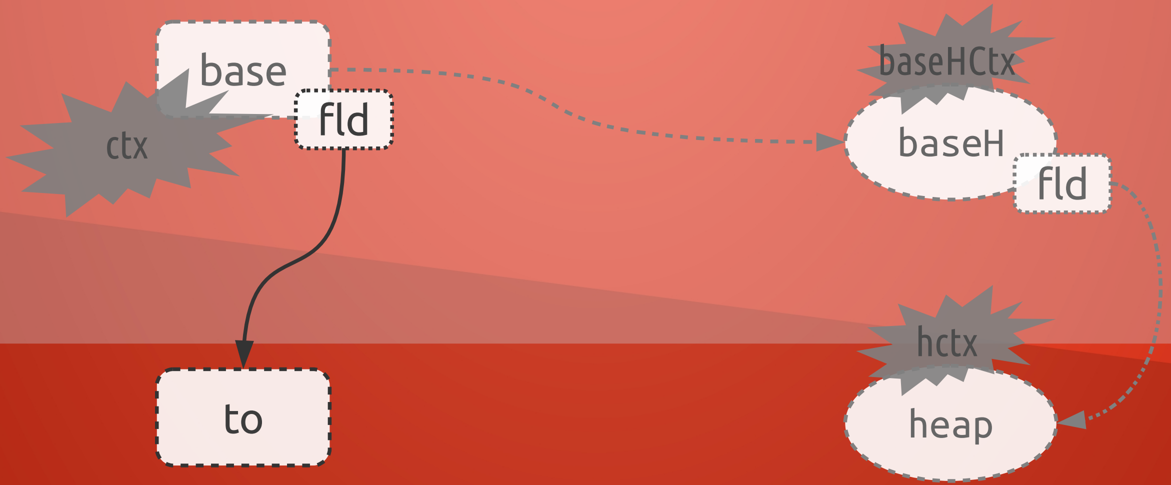


#7

Datalog Rules

to = base.fld

```
VARPOINTSTO(to, ctx, heap, hctx) ←
    VARPOINTSTO(base, ctx, baseH, baseHCtx),
    FLDPOINTSTO(baseH, baseHCtx, fld, heap, hctx), LOAD(to, base, fld).
```

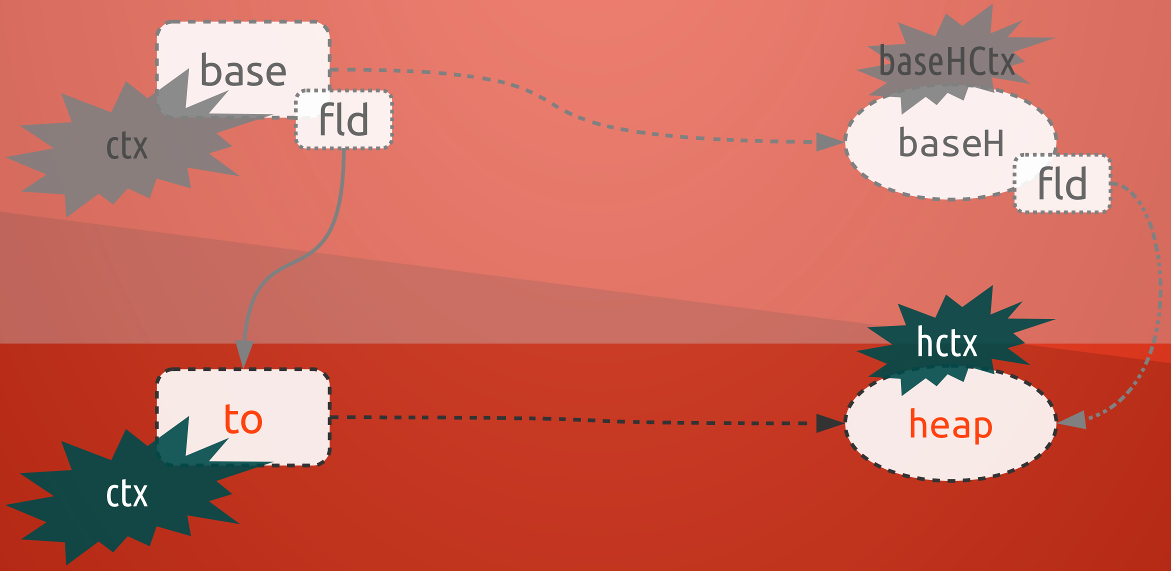


#7

Datalog Rules

to = base.fld

```
VARPOINTSTO(to, ctx, heap, hctx) ←
    VARPOINTSTO(base, ctx, baseH, baseHCtx),
    FLDPOINTSTO(baseH, baseHCtx, fld, heap, hctx), LOAD(to, base, fld).
```



#8

Datalog Rules

A::toMeth()

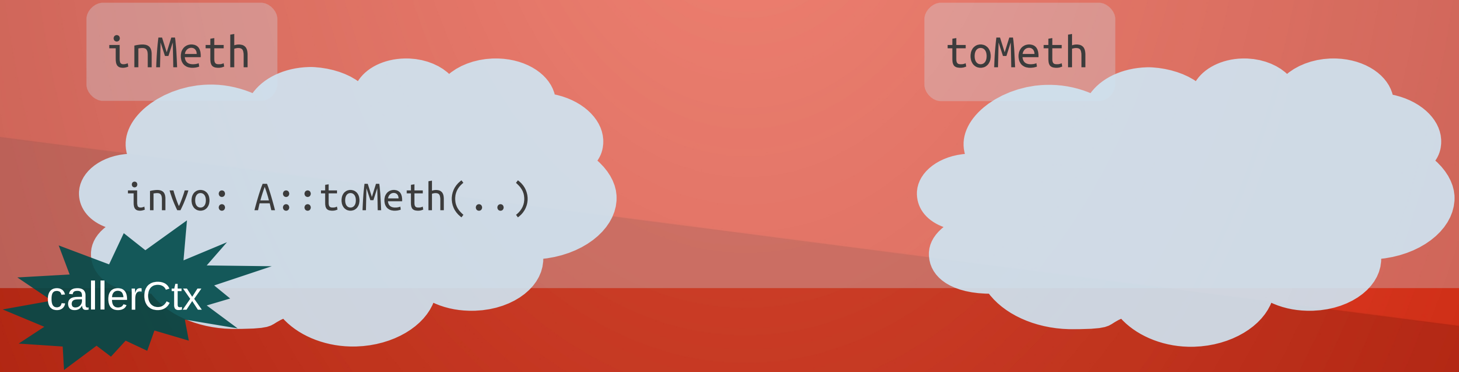
```
MERGESTATIC(invo, callerCtx) = calleeCtx,
REACHABLE(toMeth, calleeCtx),
CALLGRAPH(invo, callerCtx, toMeth, calleeCtx) ←
    SCALL(toMeth, invo, inMeth), REACHABLE(inMeth, callerCtx).
```

#8

Datalog Rules

A::toMeth()

```
MERGESTATIC(invo, callerCtx) = calleeCtx,
REACHABLE(toMeth, calleeCtx),
CALLGRAPH(invo, callerCtx, toMeth, calleeCtx) ←
    SCALL(toMeth, invo, inMeth), REACHABLE(inMeth, callerCtx).
```

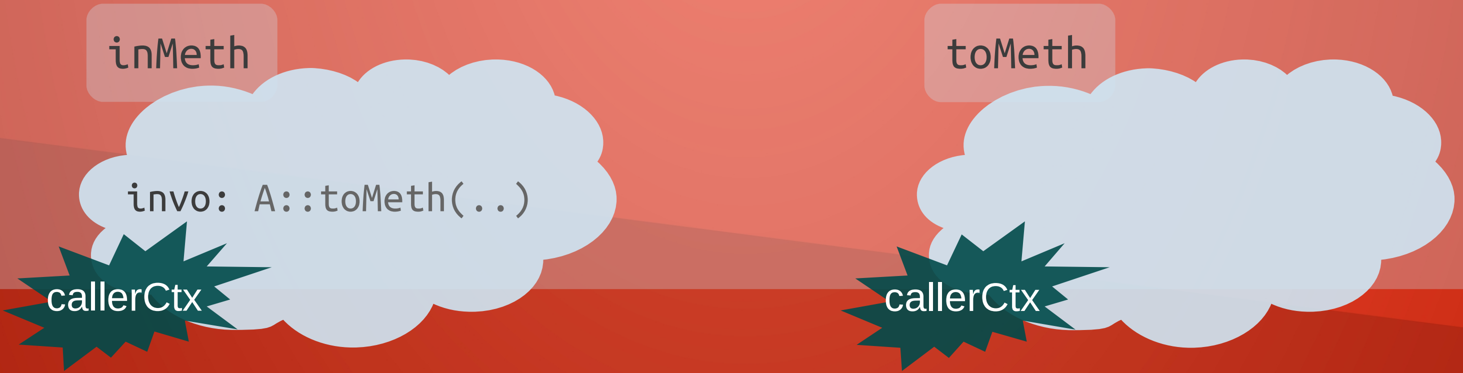


#8

Datalog Rules

A::toMeth()

```
MERGESTATIC(invo, callerCtx) = calleeCtx,   new context!
REACHABLE(toMeth, calleeCtx),
CALLGRAPH(invo, callerCtx, toMeth, calleeCtx) ←
    SCALL(toMeth, invo, inMeth), REACHABLE(inMeth, callerCtx).
```

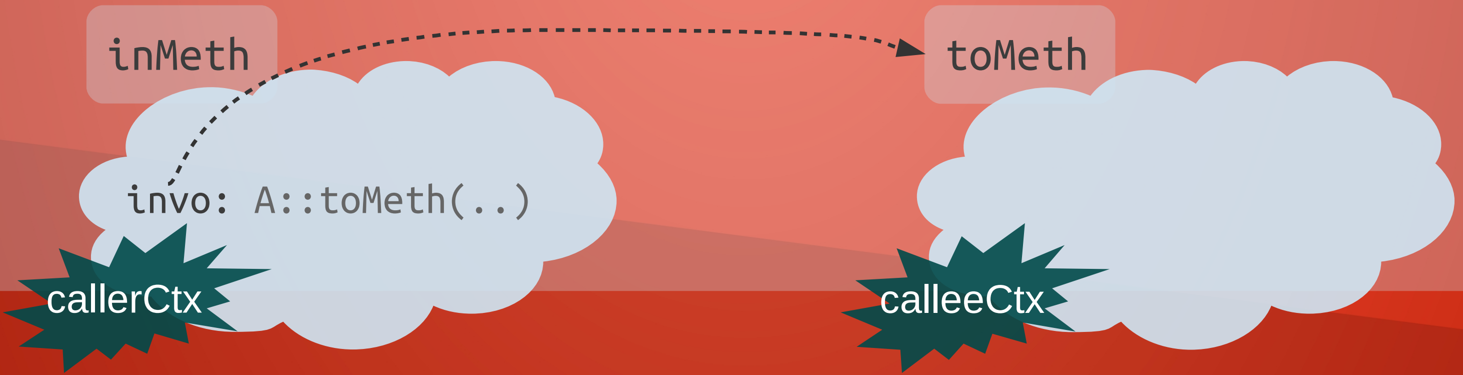


#8

Datalog Rules

A::toMeth()

```
MERGESTATIC(invo, callerCtx) = calleeCtx,
REACHABLE(toMeth, calleeCtx),
CALLGRAPH(invo, callerCtx, toMeth, calleeCtx) ←
    SCALL(toMeth, invo, inMeth), REACHABLE(inMeth, callerCtx).
```



#9

Datalog Rules

!

base.sig(..)

```
MERGE(heap, hctx, invo, callerCtx) = calleeCtx,
REACHABLE(toMeth, calleeCtx),
VARPOINTSTO(this, calleeCtx, heap, hctx),
CALLGRAPH(invo, callerCtx, toMeth, calleeCtx) ←
    REACHABLE(inMeth, callerCtx), VCALL(base, sig, invo, inMeth),
    VARPOINTSTO(base, callerCtx, heap, hctx),
    HEAPTYPE(heap, heapT), LOOKUP(heapT, sig, toMeth),
    THISVAR(toMeth, this).
```

#9

Datalog Rules

!

base.sig(..)

```
MERGE(heap, hctx, invo, callerCtx) = calleeCtx,
REACHABLE(toMeth, calleeCtx),
VARPOINTSTO(this, calleeCtx, heap, hctx),
CALLGRAPH(invo, callerCtx, toMeth, calleeCtx) ←
    REACHABLE(inMeth, callerCtx), VCALL(base, sig, invo, inMeth),
    VARPOINTSTO(base, callerCtx, heap, hctx),
    HEAPTYPE(heap, heapT), LOOKUP(heapT, sig, toMeth),
    THISVAR(toMeth, this).
```

inMeth

invo: base.sig(..)

callerCtx

#9

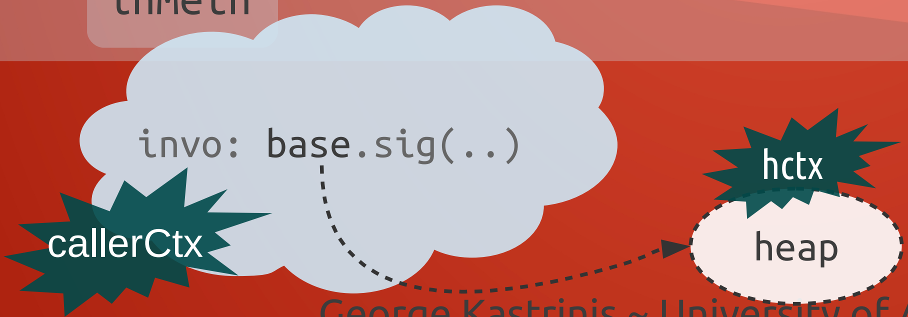
Datalog Rules

!

base.sig(..)

```
MERGE(heap, hctx, invo, callerCtx) = calleeCtx,
REACHABLE(toMeth, calleeCtx),
VARPOINTSTO(this, calleeCtx, heap, hctx),
CALLGRAPH(invo, callerCtx, toMeth, calleeCtx) ←
    REACHABLE(inMeth, callerCtx), VCALL(base, sig, invo, inMeth),
    VARPOINTSTO(base, callerCtx, heap, hctx),
    HEAPTYPE(heap, heapT), LOOKUP(heapT, sig, toMeth),
    THISVAR(toMeth, this).
```

inMeth



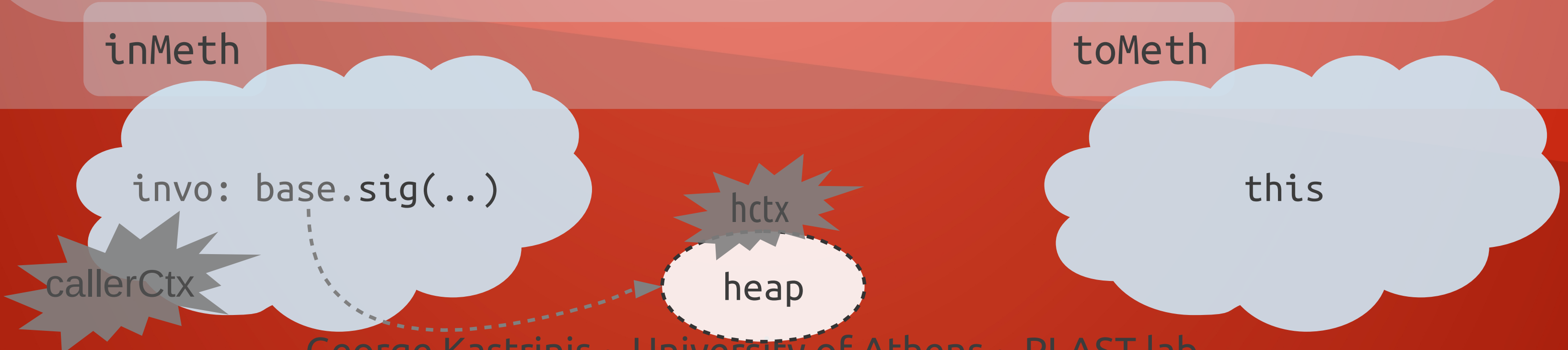
#9

Datalog Rules

!

base.sig(..)

```
MERGE(heap, hctx, invo, callerCtx) = calleeCtx,
REACHABLE(toMeth, calleeCtx),
VARPOINTSTO(this, calleeCtx, heap, hctx),
CALLGRAPH(invo, callerCtx, toMeth, calleeCtx) ←
    REACHABLE(inMeth, callerCtx), VCALL(base, sig, invo, inMeth),
    VARPOINTSTO(base, callerCtx, heap, hctx),
    HEAPTYPE(heap, heapT), LOOKUP(heapT, sig, toMeth),
    THISVAR(toMeth, this).
```



#9

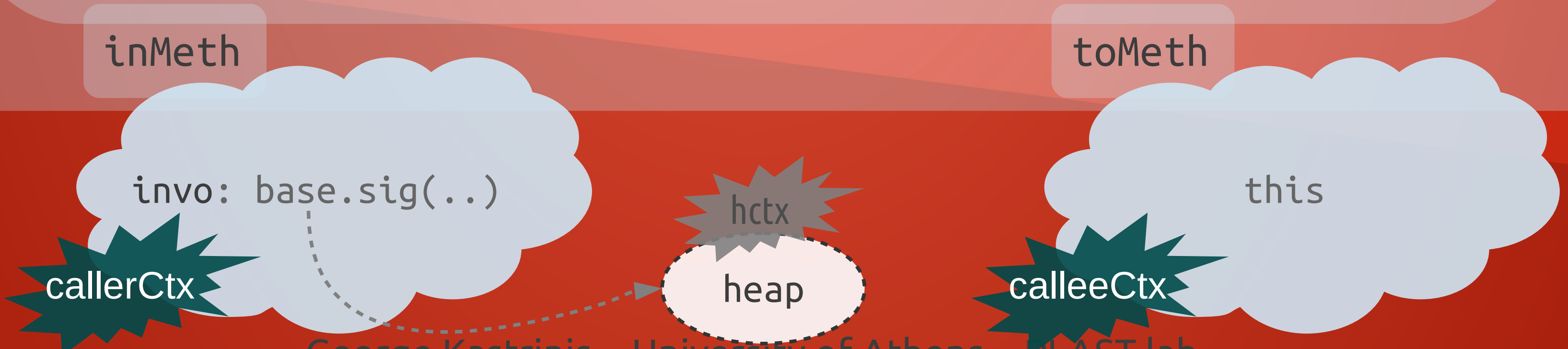
Datalog Rules

!

base.sig(..)

new context!

```
MERGE(heap, hctx, invo, callerCtx) = calleeCtx,
REACHABLE(toMeth, calleeCtx),
VARPOINTSTO(this, calleeCtx, heap, hctx),
CALLGRAPH(invo, callerCtx, toMeth, calleeCtx) ←
    REACHABLE(inMeth, callerCtx), VCALL(base, sig, invo, inMeth),
    VARPOINTSTO(base, callerCtx, heap, hctx),
    HEAPTYPE(heap, heapT), LOOKUP(heapT, sig, toMeth),
    THISVAR(toMeth, this).
```



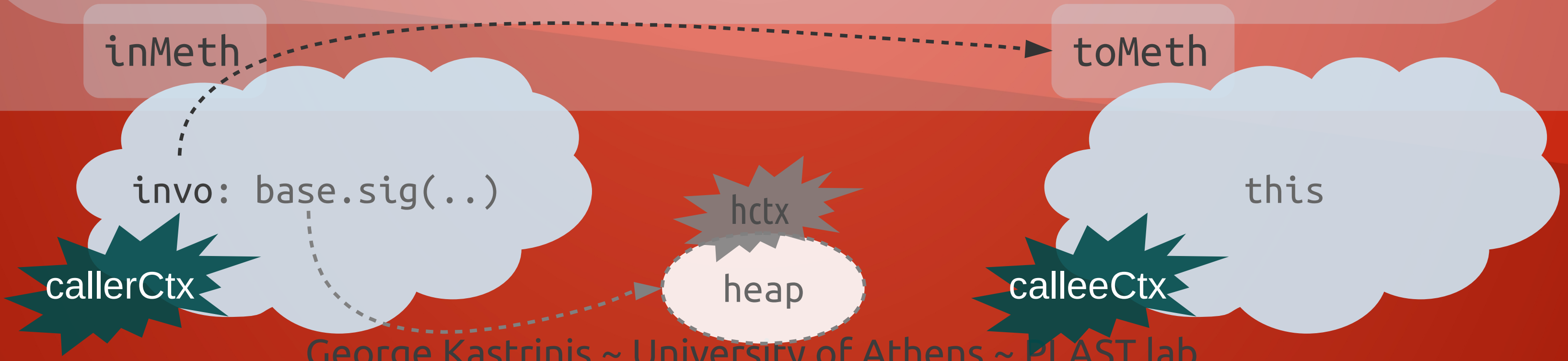
#9

Datalog Rules

!

base.sig(..)

```
MERGE(heap, hctx, invo, callerCtx) = calleeCtx,
REACHABLE(toMeth, calleeCtx),
VARPOINTSTO(this, calleeCtx, heap, hctx),
CALLGRAPH(invo, callerCtx, toMeth, calleeCtx) ←
    REACHABLE(inMeth, callerCtx), VCALL(base, sig, invo, inMeth),
    VARPOINTSTO(base, callerCtx, heap, hctx),
    HEAPTYPE(heap, heapT), LOOKUP(heapT, sig, toMeth),
    THISVAR(toMeth, this).
```



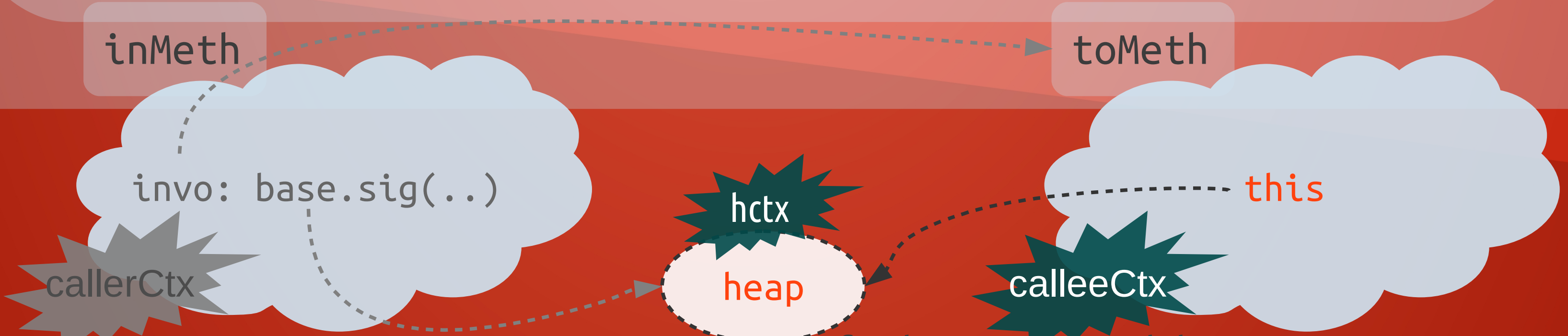
#9

Datalog Rules

!

base.sig(..)

```
MERGE(heap, hctx, invo, callerCtx) = calleeCtx,
REACHABLE(toMeth, calleeCtx),
VARPOINTSTO(this, calleeCtx, heap, hctx),
CALLGRAPH(invo, callerCtx, toMeth, calleeCtx) ←
    REACHABLE(inMeth, callerCtx), VCALL(base, sig, invo, inMeth),
    VARPOINTSTO(base, callerCtx, heap, hctx),
    HEAPTYPE(heap, heapT), LOOKUP(heapT, sig, toMeth),
    THISVAR(toMeth, this).
```



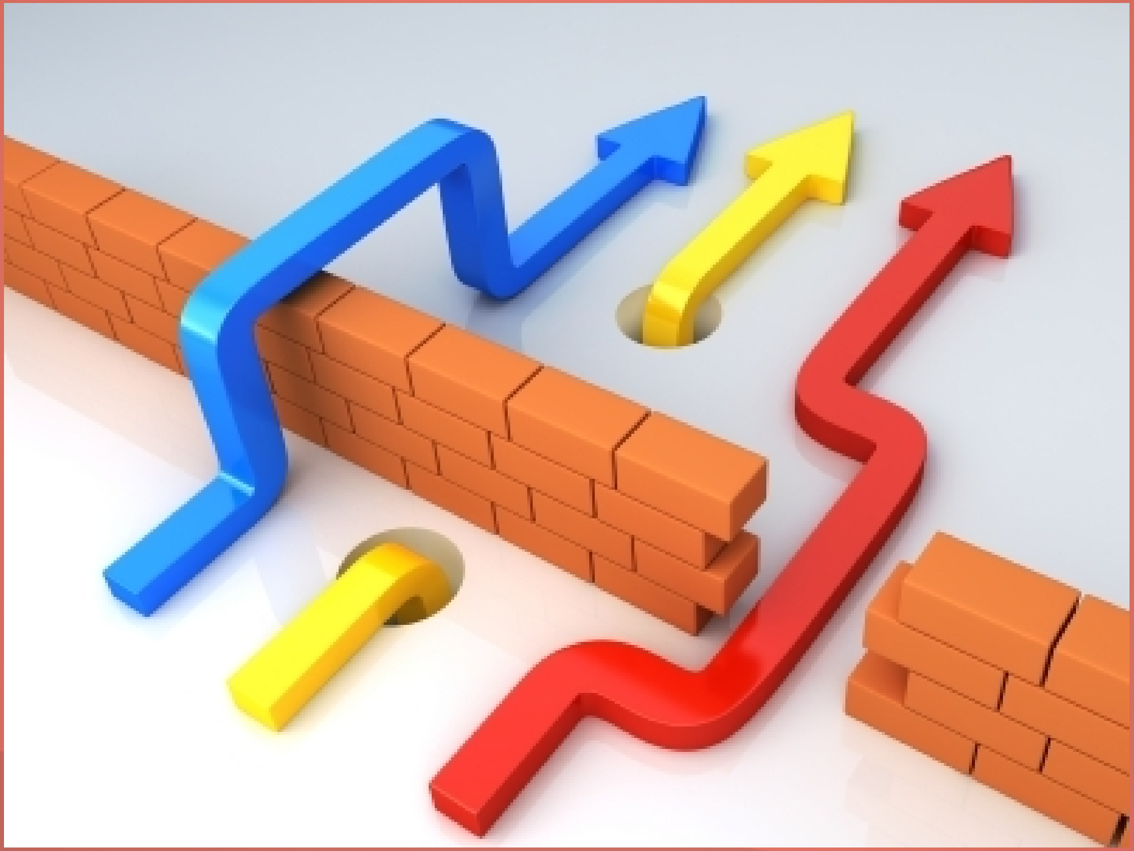
#10 Datalog Rules

#10

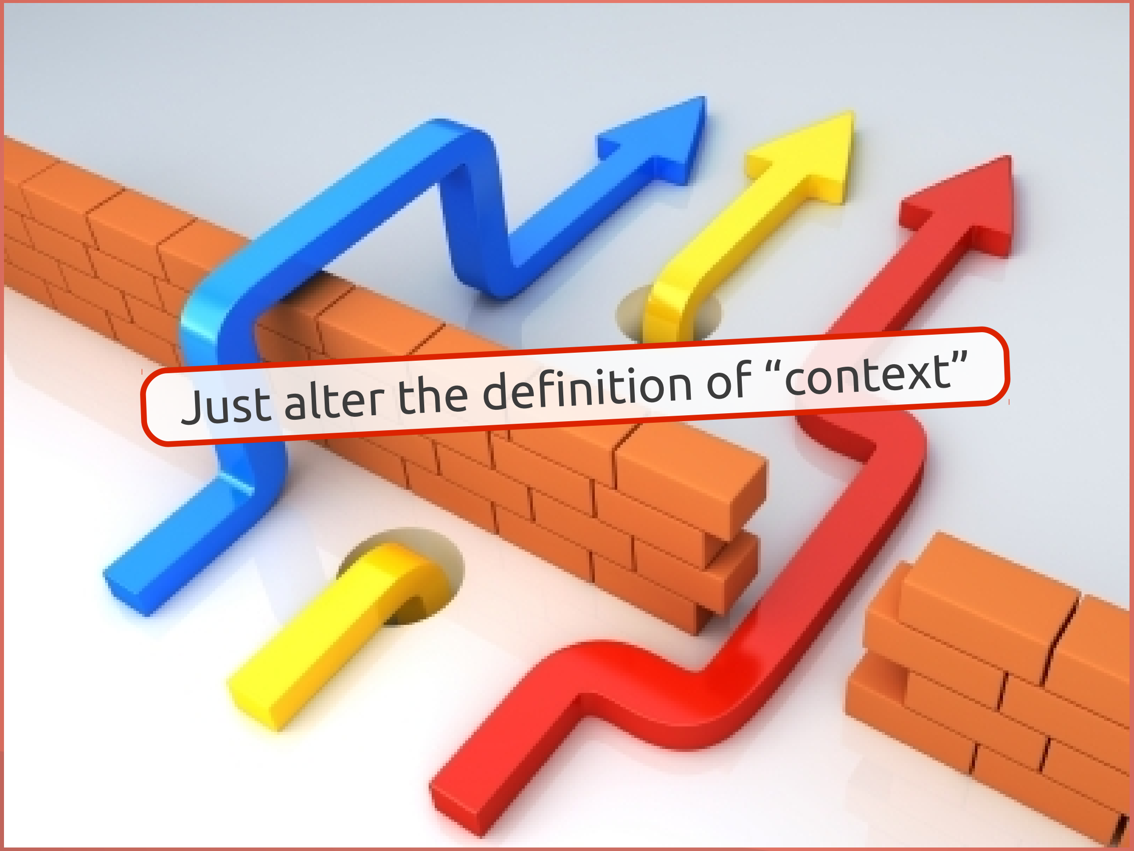
Datalog Rules

9 RULES ARE ENOUGH!

Variety of Analyses



Variety of Analyses



Just alter the definition of "context"

LET'S RECALL WHERE CONTEXTS ARE CREATED

LET'S RECALL WHERE CONTEXTS ARE CREATED

```
RECORD(heap, ctx) = hctx,
VARPOINTSTO(var, ctx, heap, hctx) ←
    REACHABLE(meth, ctx), ALLOC(var, heap, meth).
```

#3

#8

```
MERGESTATIC(invo, callerCtx) = calleeCtx,
REACHABLE(toMeth, calleeCtx),
CALLGRAPH(invo, callerCtx, toMeth, calleeCtx) ←
    SCALL(toMeth, invo, inMeth), REACHABLE(inMeth, callerCtx).
```

```
MERGE(heap, hctx, invo, callerCtx) = calleeCtx,
REACHABLE(toMeth, calleeCtx),
VARPOINTSTO(this, calleeCtx, heap, hctx),
CALLGRAPH(invo, callerCtx, toMeth, calleeCtx) ←
    REACHABLE(inMeth, callerCtx), VCALL(base, sig, invo, inMeth),
    VARPOINTSTO(base, callerCtx, heap, hctx),
    HEAPTYPE(heap, heapT), LOOKUP(heapT, sig, toMeth),
    THISVAR(toMeth, this).
```

#9

LET'S RECALL WHERE CONTEXTS ARE CREATED

```
RECORD(heap, ctx) = hctx,
VARPOINTSTO(var, ctx, heap, hctx) ←
    REACHABLE(meth, ctx), ALLOC(var, heap, meth).
```

Object allocation

#3

```
MERGESTATIC(invo, callerCtx) = calleeCtx,
REACHABLE(toMeth, calleeCtx),
CALLGRAPH(invo, callerCtx, toMeth, calleeCtx) ←
    SCALL(toMeth, invo, inMeth), REACHABLE(inMeth, callerCtx).
```

#8

```
MERGE(heap, hctx, invo, callerCtx) = calleeCtx,
REACHABLE(toMeth, calleeCtx),
VARPOINTSTO(this, calleeCtx, heap, hctx),
CALLGRAPH(invo, callerCtx, toMeth, calleeCtx) ←
    REACHABLE(inMeth, callerCtx), VCALL(base, sig, invo, inMeth),
    VARPOINTSTO(base, callerCtx, heap, hctx),
    HEAPTYPE(heap, heapT), LOOKUP(heapT, sig, toMeth),
    THISVAR(toMeth, this).
```

Method invocation

#9

CONTEXT INSENSITIVE

IGNORE CONTEXT ALTOGETHER

CONTEXT INSENSITIVE

IGNORE CONTEXT ALTOGETHER

```
void foo() {
    a = new A1();
    b = id(a);
}

void bar() {
    a = new A2();
    b = id(a);
}

A id(A a) {
    return a;
}
```

```
foo::a → new A1()
bar::a → new A2()
 id::a → new A1(), new A2()
foo::b → new A1(), new A2()
bar::b → new A1(), new A2()
```

CONTEXT INSENSITIVE

IGNORE CONTEXT ALTOGETHER

```
RECORD(heap, ctx) = *
MERGE(heap, hctx, invo, callerCtx) = *
MERGESTATIC(invo, callerCtx) = *
```

Use a single context everywhere!

CALL-SITE SENSITIVITY

USE CALL-SITES AS CONTEXTS

CALL-SITE SENSITIVITY

USE CALL-SITES AS CONTEXTS

```
       void foo() {
           a = new A1();
inv1 ⇒     b = id(a);
       }

       void bar() {
           a = new A2();
inv2 ⇒     b = id(a);
       }

       A id(A a) {
           return a;
       }
```

```
foo::a → new A1()
bar::a → new A2()
 id::a (inv1) → new A1()
 id::a (inv2) → new A2()
foo::b → new A1()
bar::b → new A2()
```

1-CALL-SITE SENSITIVE

Context's depth

1-CALL-SITE SENSITIVE

```
RECORD(heap, ctx) = *
```

No context for heap abstractions

1-CALL-SITE SENSITIVE

```
RECORD(heap, ctx) = *
MERGE(heap, hctx, invo, callerCtx) = invo
MERGESTATIC(invo, callerCtx) = invo
```

1-CALL-SITE SENSITIVE+1-HEAP

Context sensitive heap abstractions

1-CALL-SITE SENSITIVE+1-HEAP

```
RECORD(heap, ctx) = ctx
```

1-CALL-SITE SENSITIVE+1-HEAP

```
RECORD(heap, ctx) = ctx
MERGE(heap, hctx, invo, callerCtx) = invo
MERGESTATIC(invo, callerCtx) = invo
```

OBJECT SENSITIVITY

USE ALLOCATION-SITES AS CONTEXTS

OBJECT SENSITIVITY

USE ALLOCATION-SITES AS CONTEXTS

Based on the receiver object in a method call

OBJECT SENSITIVITY

USE ALLOCATION-SITES AS CONTEXTS

Really good for Object-Oriented languages

OBJECT SENSITIVITY

USE ALLOCATION-SITES AS CONTEXTS

```
class C {
    void meth(Object o) { ... }
}

class Client {
    void bar(C c1, C c2) {
        ...
        c1.meth(obj1);
        ...
        c2.meth(obj2);
    }
}
```

OBJECT SENSITIVITY

USE ALLOCATION-SITES AS CONTEXTS

```
class C {
    void meth(Object o) { ... }
}

class Client {
    void bar(C c1, C c2) {
        ...
        c1.meth(obj1);
        ...
        c2.meth(obj2);
    }
}
```

#contexts for meth::o?

OBJECT SENSITIVITY

USE ALLOCATION-SITES AS CONTEXTS

```
class C {
    void meth(Object o) { ... }
}

class Client {
    void bar(C c1, C c2) {
        ...
        c1.meth(obj1);
        ...
        c2.meth(obj2);
    }
}
```

#contexts for meth::o?

#objects (and which) c1 and c2 point to?

1-OBJECT SENSITIVE

```
RECORD(heap, ctx) = *
```

No context for heap abstractions

1-OBJECT SENSITIVE

```
RECORD(heap, ctx) = *
MERGE(heap, hctx, invo, callerCtx) = heap
```

Use the allocation-site of the receiver object

1-OBJECT SENSITIVE

```
RECORD(heap, ctx) = *
MERGE(heap, hctx, invo, callerCtx) = heap
MERGESTATIC(invo, callerCtx) = ctx
```

No receiver object to use!

1-OBJECT SENSITIVE

```
RECORD(heap, ctx) = *
MERGE(heap, hctx, invo, callerCtx) = heap
MERGESTATIC(invo, callerCtx) = ctx
```

No receiver object to use!

Copy context from caller

AND NOW?

AND NOW?

- DIFFERENT CONTEXT DEPTHS

AND NOW?

- DIFFERENT CONTEXT DEPTHS
- ALTER DEFINITIONS OF **RECORD** AND **MERGE**

AND NOW?

- DIFFERENT CONTEXT DEPTHS
- ALTER DEFINITIONS OF **RECORD** AND **MERGE**
- OTHER TYPES OF CONTEXT

AND NOW?

- DIFFERENT CONTEXT DEPTHS
- ALTER DEFINITIONS OF **RECORD** AND **MERGE**
- OTHER TYPES OF CONTEXT
- COMBINE DIFFERENT CONTEXTS

AND NOW?

- DIFFERENT CONTEXT DEPTHS
- ALTER DEFINITIONS OF **RECORD** AND **MERGE**
- OTHER TYPES OF CONTEXT
- COMBINE DIFFERENT CONTEXTS

What to combine? Where? How?

RECAP

9 Rules x 3 Context Functions = ∞ Analyses

Hope you enjoyed!

George Kastrinis ● http://gkastrinis.info