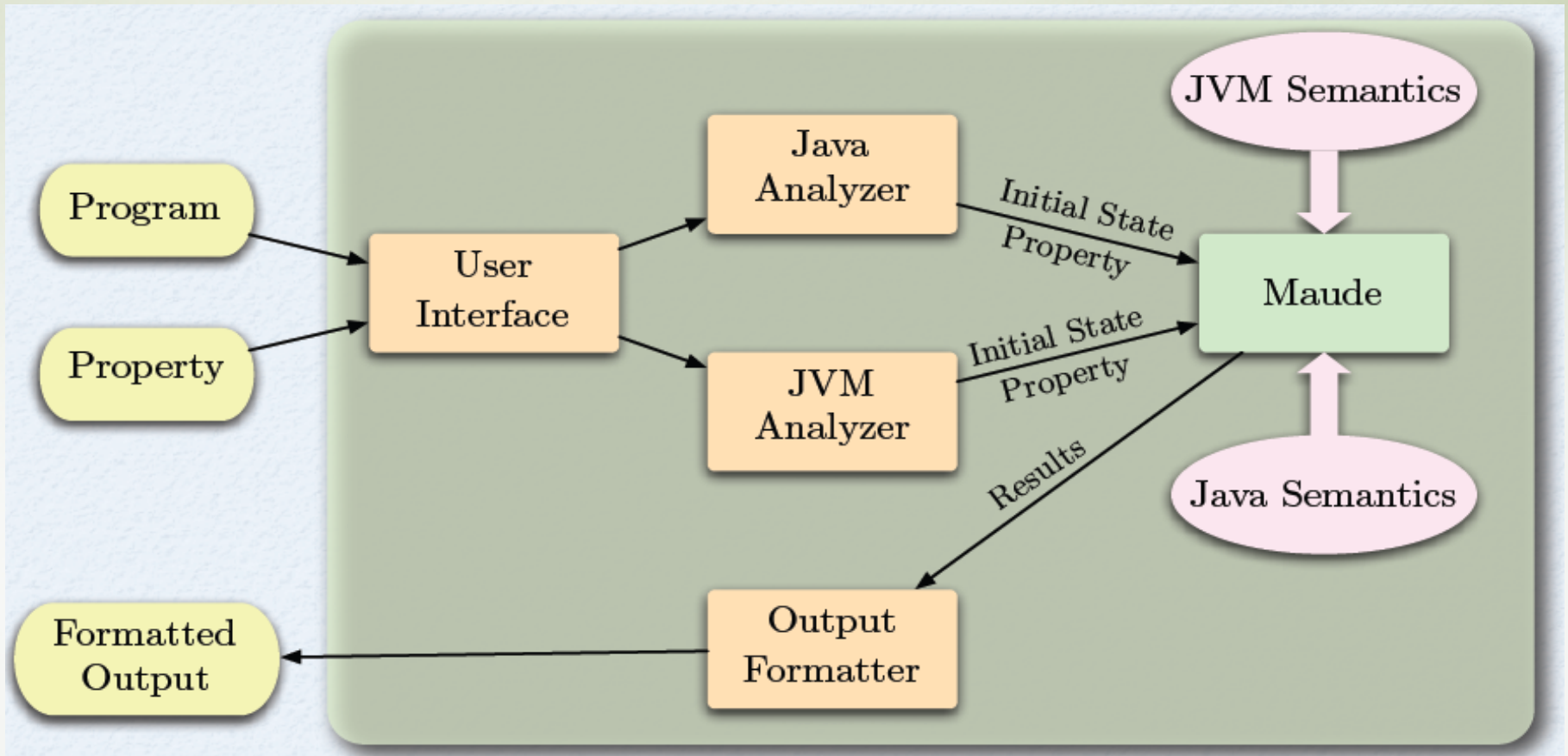


Analyzing Java Programs Using JavaFAN

Feng Chen

Joint work with Azadeh Farzan, Jose Meseguer and Grigore Rosu

JavaFAN (Java Formal Analyzer)



Inside Java Semantics

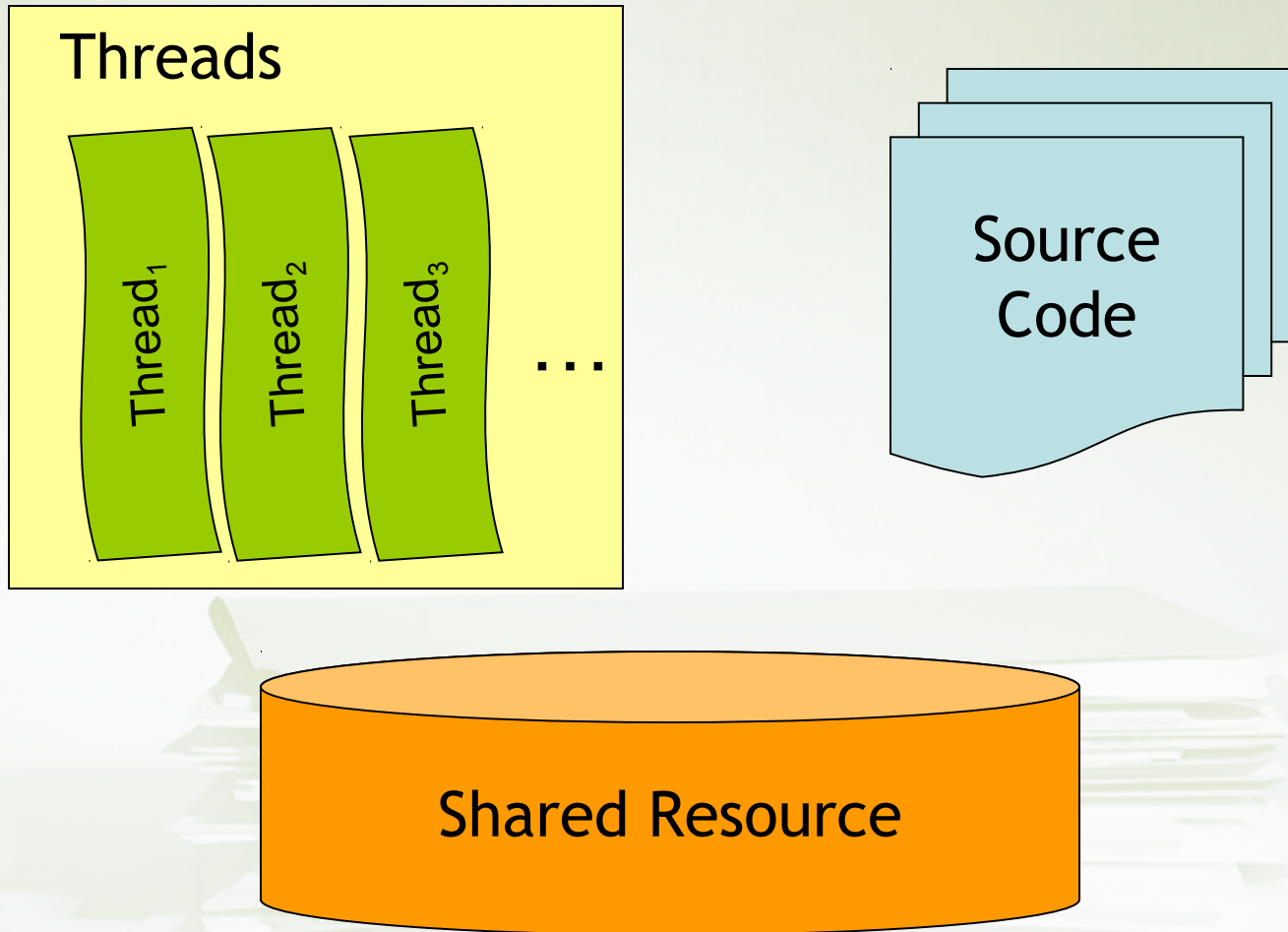


JavaRL (Java Rewrite Logic) Semantics

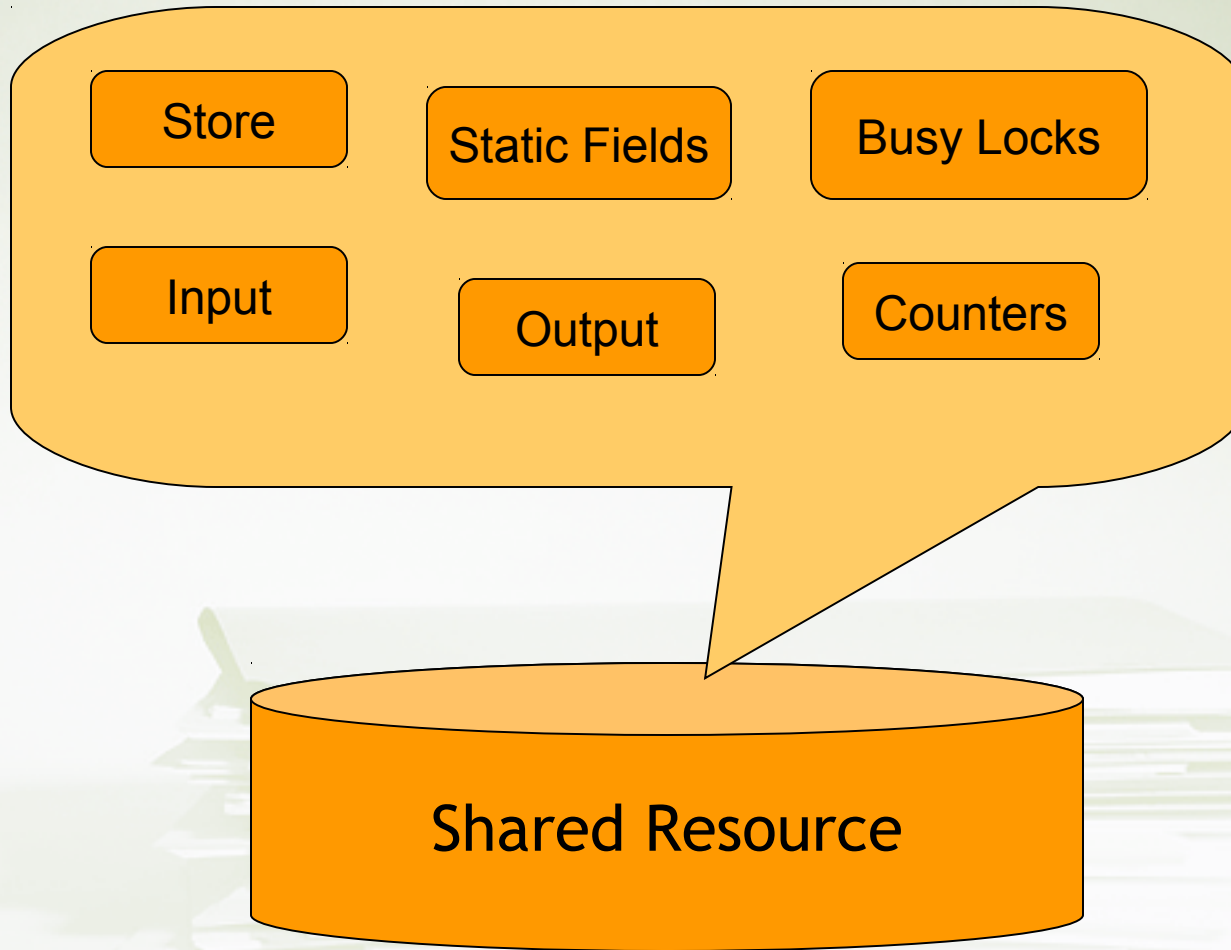
- An operational semantics
- An execution = a sequence of state transitions
 - State transitions
 - Equations: deterministic sequential state transitions
 - Rewrite rules: nondeterministic concurrent state transitions
 - State definition



JavaState



JavaState - Shared Resource



JavaState - Definition

sort JavaState .

op code : Classes -> JavaState . --- source code

op t : ThreadCtrl -> JavaState . --- thread contexts

op out : Output -> JavaState . --- output

op in : ValueList -> JavaState . --- input

op store : Store -> JavaState . --- memory

op static : ObjEnv -> JavaState . --- the static fields of classes

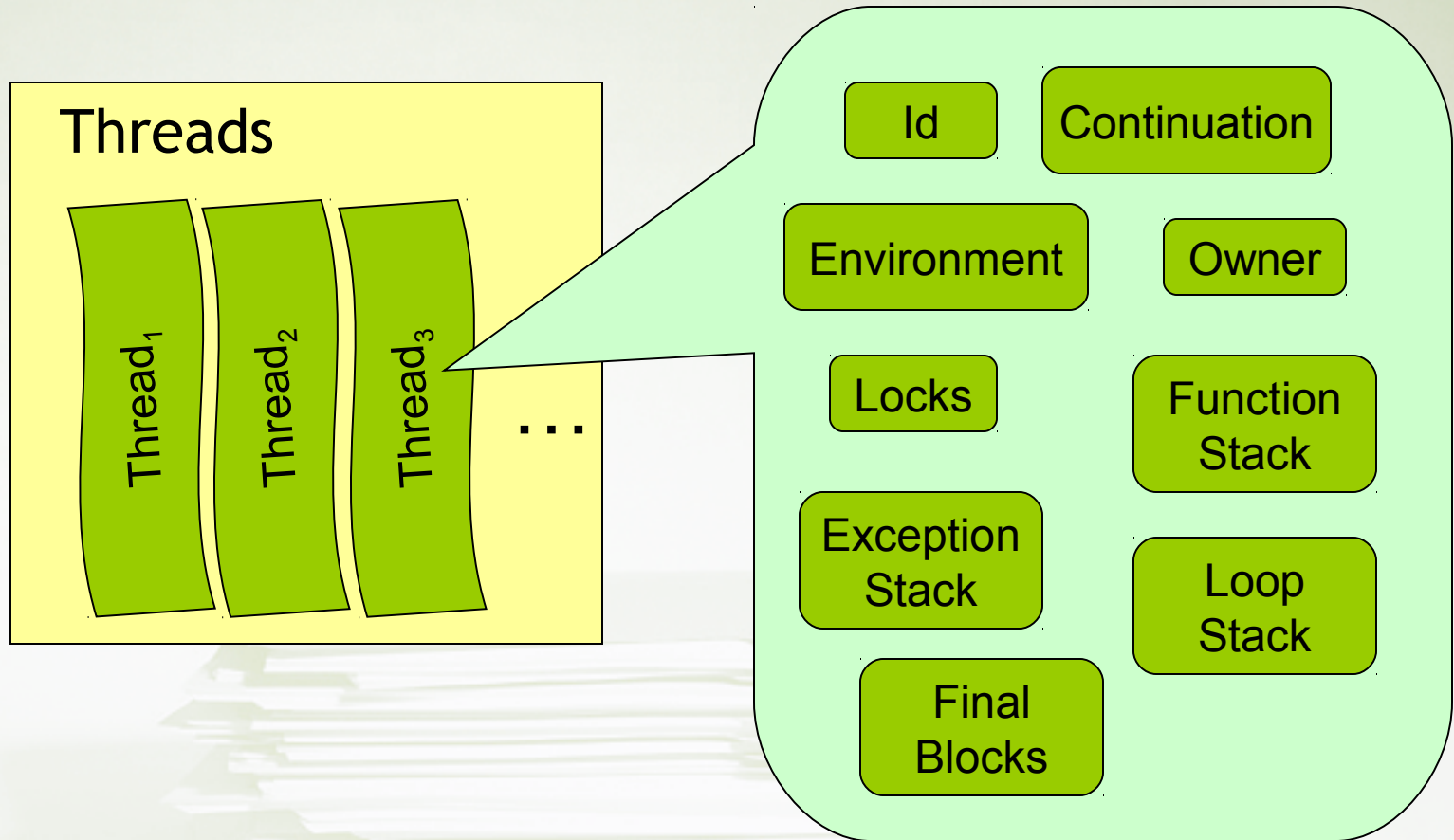
op busy : ObjectSet -> JavaState . --- the locks are in use

op nextLoc : Int -> JavaState . --- the next available location

op nextTid : Int -> JavaState . --- the next thread id

op __ : JavaState JavaState -> JavaState [comm assoc] .

ThreadCtrl



FlowStack - Method Return

eq $k(\text{return } E ; \rightarrow K) = k(E \rightarrow \text{return} \rightarrow K)$.

eq $t(k(V \rightarrow \text{return} \rightarrow K) \text{fstack}(\text{fsi}(K', (\text{holds}(LI) \text{env}(\text{Env}) \text{tc})) \text{fstack}) \text{holds}(LI') \text{env}(\text{Env}') \text{tc}')$
 $= t(k(\text{releaseEnv}(\text{Env}') \rightarrow \text{release}(LI, LI') \rightarrow (V \rightarrow K')) \text{fstack}(\text{fstack}) \text{holds}(LI) \text{env}(\text{Env}) \text{tc})$.



ThreadCtrl - Definition

```
sorts ThreadCtrl FlowStackItem FlowStack .
op id : Int -> ThreadCtrl . --- the thread id
op k : Continuation -> ThreadCtrl .
op obj : Object -> ThreadCtrl .
op fstack : FlowStack -> ThreadCtrl . --- function stack
op xstack : FlowStack -> ThreadCtrl . --- exception stack
op lstack : FlowStack -> ThreadCtrl . --- loop stack
op finalblocks : FlowStack -> ThreadCtrl . --- final blocks
op env : Env -> ThreadCtrl .
op holds : LockSet -> ThreadCtrl . --- the locks held by the thread
op __ : ThreadCtrl ThreadCtrl -> ThreadCtrl [comm assoc] .

subsort FlowStackItem < FlowStack .
op noltem : -> FlowStackItem .
op fsi : Continuation ThreadCtrl -> FlowStackItem .
op __ : FlowStack FlowStack -> FlowStack [assoc id: noltem] .
```

Continuation

Continuation:

```
sort Continuation .  
op stop : -> Continuation .  
op noop : -> Continuation .  
op _->_ : Continuation Continuation -> Continuation [assoc] .  
  
var K : Continuation .  
eq noop -> K = K .
```

Addition:

```
eq k((E + E') -> K) = k((E, E') -> (+ -> K)) .  
eq k((int(l), int(l')) -> (+ -> K)) = k(int(l + l') -> K) .
```

Environment and Store

Environment: mapping from names to locations

```
sort Env .  
op noEnv : -> Env .  
op [_,_] : Qid Location -> Env .  
op __ : Env Env -> Env [assoc comm id: noEnv] .
```

Store: mapping from locations to values

```
sort Store .  
op noStore : -> Store .  
--- the last bit is used to store the thread id  
--- needed for fine grained concurrency and reclaiming locations  
op [_,_,_] : Location Value Int -> Store .  
op __ : Store Store -> Store [assoc comm id: noStore] .
```


Example: Store Accesses

Fetch a value:

eq $t(k(\#(L) \rightarrow K) \text{ id}(I) \text{ tc}) \text{ store}([L, V, I] \text{ st}) = t(k(V \rightarrow K) \text{ id}(I) \text{ tc}) \text{ store}([L, V, I] \text{ st}) .$

eq $t(k(\#(L) \rightarrow K) \text{ id}(I) \text{ tc}) \text{ store}([L, V, -2] \text{ st}) = t(k(V \rightarrow K) \text{ id}(I) \text{ tc}) \text{ store}([L, V, -2] \text{ st}) .$

rl $t(k(\#(L) \rightarrow K) \text{ id}(I) \text{ tc}) \text{ store}([L, V, -1] \text{ st}) \Rightarrow t(k(V \rightarrow K) \text{ id}(I) \text{ tc}) \text{ store}([L, V, -1] \text{ st}) .$

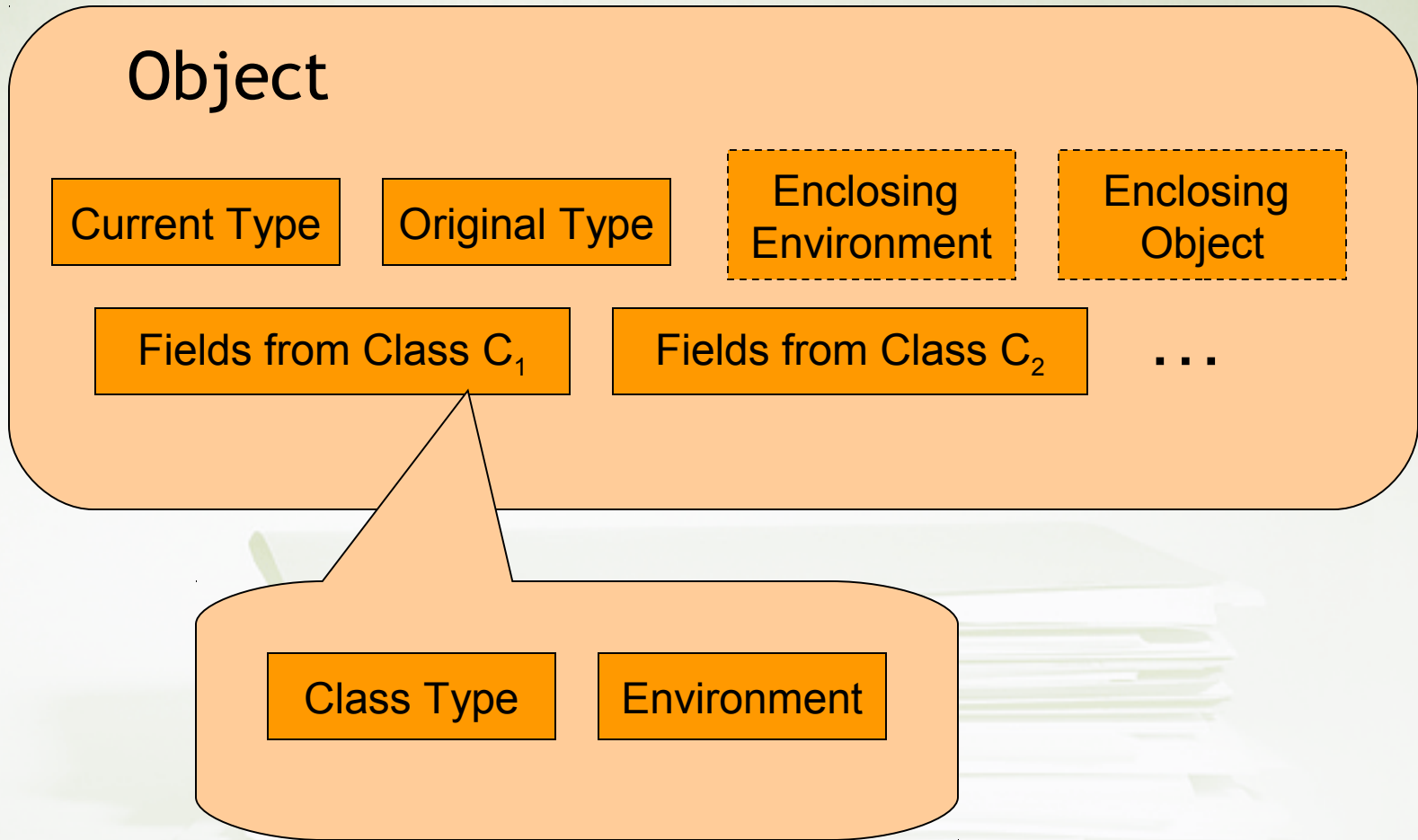
Set a value:

eq $t(k([V \rightarrow L] \rightarrow K) \text{ id}(I) \text{ tc}) \text{ store}([L, V', I] \text{ st}) = t(k(K) \text{ id}(I) \text{ tc}) \text{ store}([L, \text{setTid}(V, I), I] \text{ st}) .$

eq $t(k([V \rightarrow L] \rightarrow K) \text{ id}(I) \text{ tc}) \text{ store}([L, V', -2] \text{ st}) = t(k(K) \text{ id}(I) \text{ tc}) \text{ store}([L, V, -2] \text{ st}) .$

rl $t(k([V \rightarrow L] \rightarrow K) \text{ id}(I) \text{ tc}) \text{ store}([L, V', -1] \text{ st}) \Rightarrow t(k(K) \text{ id}(I) \text{ tc}) \text{ store}([L, \text{shared}(V), -1] \text{ st}) .$

Object



Object - Definition

```
sorts ObjEnvItem ObjEnv Object ObjectAttribute .  
subsort Object < Value .
```

```
op o : ObjectAttribute -> Object .  
op __ : ObjectAttribute ObjectAttribute -> ObjectAttribute [comm assoc id: noAttr] .  
op noAttr : -> ObjectAttribute .  
op f : ObjEnv -> ObjectAttribute .  
op curr : Type -> ObjectAttribute .  
op orig : Type -> ObjectAttribute .  
op encEnv : Env -> ObjectAttribute . --- the env of the enclosing block.  
op __ : ObjEnv ObjEnv -> ObjEnv [comm assoc id: onil] .  
op onil : -> ObjEnv .  
op [] : ObjEnvItem -> ObjEnv .  
op t : Type -> ObjEnvItem .  
op enc : Object -> ObjEnvItem . --- the enclosing object  
op f : Env -> ObjEnvItem .  
op inil : -> ObjEnvItem .  
op __, _ : ObjEnvItem ObjEnvItem -> ObjEnvItem [comm assoc id: inil] .
```

Access Object Fields

--- the owise option allows the system object take effect first, like system . out
eq $k((E . X) \rightarrow K) = k(E \rightarrow (.X) \rightarrow K)$ [owise] .

eq $k(o(oattr\ curr(T)) \rightarrow (.X) \rightarrow K) = k(o(oattr\ curr(T)) \rightarrow (.X, T) \rightarrow K)$.

eq $k(o(oattr\ curr(T)\ f([oeitem, t(T), f([X, L]\ env)])\ oEnv)) \rightarrow (.X, T') \rightarrow K) = k(\#(L) \rightarrow K)$.

eq $t(k(o(oattr\ curr(T)) \rightarrow (.X, T') \rightarrow K)\ tc)\ static([t(T), f([X, L]\ env)]\ oEnv) =$
 $t(k(\#(L) \rightarrow K)\ tc)\ static([t(T), f([X, L]\ env)]\ oEnv)$.

eq $t(k(o(curr(T)\ oattr) \rightarrow (.X, T') \rightarrow K)\ tc)\ code(CI) =$
 $t(k(o(curr(getSuper(T, CI))\ oattr) \rightarrow (.X, T') \rightarrow K)\ tc)\ code(CI)$ [owise] .

eq $k(o(curr(Object)\ encEnv([X, L]\ env)\ oattr) \rightarrow (.X, T) \rightarrow K) = k(\#(L) \rightarrow K)$.

eq $k(o(oattr\ curr(Object)\ f([enc(obj), t(T), oeitem]\ oEnv)) \rightarrow (.X, T) \rightarrow K) =$
 $k(obj \rightarrow (.X) \rightarrow K)$ [owise] .

Optimizations

- Minimizing the number of rewrite rules
- Tracking potential shared locations
- Reclaiming store locations
- Partial Order Reduction



Demos



Java Analyzer

java *javarl.JavaRL* *-cls* <path> [*options*] [*MainClass*]

Options: [*-maudecode*] [*-op* <path>] [*-mc* <path>] [*-s deadlock*]

- *-cls* <path>: designate the class path, can be a directory or a Java file
- *-maudecode*: generate the Maude module only
- *-op* <path>: write the output to the specified file
- *-mc* <path>: model check the program against the property specified in the property file
- *-s deadlock*: search for deadlocks
- *MainClass*: the main class in the program; if ignored, the first class with the main function will be used

Interpretation

```
java javari.JavaRL -cls <path> [MainClass]
```

```
class Sum {  
    public static void main(String[] args)  
    {  
        int sum = 0;  
        for (int i=0; i<10000; i++) sum += i;  
        System.out.println(sum);  
    }  
}
```

Search

```
java javari.JavaRL -cls <path> -s deadlock [MainClass]
```



Model Checking

```
java javarl.JavaRL -cls <path> -mc <path> [MainClass]
```



Property Specification

Syntax:

Property ::= Atom* Formula
Atom ::= "atom" <name> ":" <className> ("." | "@")
BooleanExp ("^" BooleanExp)*
BooleanExp ::= <field> (">" | ">=" | "<" | "<=" | "==") <integer>
Formula ::= "formula" ":" <LTL formula>

Example:

atom readers: data@rnum > 0
atom writers: data@wnum > 0
formula : [] (~ (readers ^ writers))

Property in Maude

```
java javarl.JavaRL -cls <path> -maudecode -op <path>
```

```
op readers : -> Prop .
```

```
ceq run(store([L, o(objAttr f([t(t('data)), f([ 'rnum, L0 ] env)] oEnv)), I]  
  [L0, int(I0), I0'] store) state) |= readers = true if (I0 > 0) .
```

```
eq output |= readers = false [otherwise] .
```

```
op writers : -> Prop .
```

```
ceq run(store([L, o(objAttr f([t(t('data)), f([ 'wnum, L0 ] env)] oEnv)), I]  
  [L0, int(I0), I0'] store) state) |= writers = true if (I0 > 0) .
```

```
eq output |= writers = false [otherwise] .
```

```
op program : -> Output .
```

```
...
```

```
red modelCheck(program, [] ( ~ ( readers ^ writers ) ) ) .
```

JavaPathFinder

- A VM-based model checker for Java
- Customizable via programming
 - Expressive but requires more effort
- <http://javapathfinder.sourceforge.net/>



Resource

- Official website: <http://javafan.cs.uiuc.edu>

