

# Inference of Expressive Declassification Policies

Jeff Vaughan

Stephen Chong



IEEE Security and Privacy

May 23, 2011

```

class Client {
    public static void main(String args[]){
        String uname;
        String pwd;
        String key;
        try{
            uname = fakeReadingArg(0);
            pwd = fakeReadingArg(1);
            key = fakeReadingArg(2);
        } catch (Exception e){
            System.out.println("Usage :java Client <uname> <pwd> <key>"); 
            return;
        }
        LoginInit();
        LogIn r = LogIn.getLogin(uname, pwd);
        if(r == null){
            "Login Failed".System.out.println();
            return;
        }
        Database db = new Database();
        if (!authorizedToRead(key, db)){
            db.secretData.System.out.println();
            return;
        }
        "Access not authorized".System.out.println();
    }
}

private static boolean authorizedToRead(Role r, String key, Database db){
    Role required = db.readerOf(key);
    if (Role.superRequired(r)){
        }
    class Database {
        private SimpleArrayList database;
        public Database () {
            database = new SimpleArrayList();
            database.add(new DbEntry("A", Role.SPY, "AA"));
        }
        public void add(DbEntry db) {
            database.add(db);
        }
        public void remove(DbEntry db) {
            database.remove(db);
        }
        public void clear() {
            database.clear();
        }
        public int size() {
            return database.size();
        }
        public boolean isEmpty() {
            return database.isEmpty();
        }
        public void set(int i, Object o) {
            database.set(i, o);
        }
        public Object get(int i) {
            return database.get(i);
        }
        public Iterator iterator() {
            return database.iterator();
        }
        public void addAll(SimpleArrayList list) {
            database.addAll(list);
        }
        public void removeAll(SimpleArrayList list) {
            database.removeAll(list);
        }
        public void retainAll(SimpleArrayList list) {
            database.retainAll(list);
        }
        public void clearAll() {
            database.clearAll();
        }
        public void trimToSize() {
            database.trimToSize();
        }
    }
    if (required.equals(r))
        return true;
    }
}

private static String fakeReadingArgs(int i)
throws java.util.NoSuchElementException {
if (i<42) {
    return "bork bork";
} else {
    throw new java.util.NoSuchElementException();
}
}

public static void main(String args[]){
    String uname;
    String pwd;
    String key;
    try{
        uname = fakeReadingArg(0);
        pwd = fakeReadingArg(1);
        key = fakeReadingArg(2);
    } catch (Exception e){
        System.out.println("Usage :java Client <uname> <pwd> <key>"); 
        return;
    }
    LoginInit();
    LogIn r = LogIn.getLogin(uname, pwd);
    if(r == null){
        "Login Failed".System.out.println();
        return;
    }
    Database db = new Database();
    if (!authorizedToRead(key, db)){
        db.secretData.System.out.println();
        return;
    }
    "Access not authorized".System.out.println();
}
}

private static class SimpleArrayList implements Iterable {
    List contents;
    public void set(int i, Object o) {
        if (i<0) {
            throw new IllegalArgumentException();
        }
        if (i>contents.size()){
            contents.add(o);
        } else {
            this.contents.set(i, o);
        }
    }
    public Object get(int i) {
        return this.contents.get(i);
    }
    public void add(Object o) {
        this.contents.add(o);
    }
    public void remove(int i) {
        return this.contents.remove(i);
    }
    public int size() {
        return this.contents.size();
    }
    public boolean isEmpty() {
        return this.contents.isEmpty();
    }
    void clear(){
        this.contents = new ArrayList();
    }
    private static abstract class List {
        protected List contents;
        abstract int size();
        abstract boolean isEmpty();
        public abstract Object get(int i);
        public abstract List retainAll(SimpleArrayList list);
        public abstract void addAll(SimpleArrayList list);
        public abstract List remove(SimpleArrayList list);
        public abstract List trimToSize();
        public abstract void clear();
    }
    private static class Nil extends List {
        public Nil(){}
        public void add(Object o) {
            throw new NoSuchElementException();
        }
        public boolean isEmpty() {
            return true;
        }
        public Object get(int i) {
            throw new NoSuchElementException();
        }
        public List removePersistent(int i) {
            throw new NoSuchElementException();
        }
        private void set(int i, Object o) {
            throw new NoSuchElementException();
        }
        public void addAll(SimpleArrayList list) {
            throw new NoSuchElementException();
        }
        public void remove(int i) {
            throw new NoSuchElementException();
        }
        public void set(int i, Object o) {
            throw new NoSuchElementException();
        }
        public void add(SimpleArrayList list) {
            this.contents = list;
        }
        public void remove(SimpleArrayList list) {
            this.contents = this.contents.removePersistent(list);
        }
        public void set(int i, Object o) {
            this.contents.set(i, o);
        }
        public void addAll(List list) {
            Iterator i = list.iterator();
            while(i.hasNext()){
                this.add(i.next());
            }
        }
    }
    private class SimpleArrayListIterator implements Iterator {
        private List list;
        public SimpleArrayListIterator(SimpleArrayList simpleArrayList) {
            list = simpleArrayList.iterator();
        }
        public void next() {
            if (list.hasNext())
                list.next();
        }
        public void previous() {
            if (list.hasPrevious())
                list.previous();
        }
        public void remove() {
            list.remove();
        }
        public void add(Object o) {
            list.add(o);
        }
        public void clear() {
            list.clear();
        }
        public void retainAll(SimpleArrayList simpleArrayList) {
            list.retainAll(simpleArrayList);
        }
        public void removeAll(SimpleArrayList simpleArrayList) {
            list.removeAll(simpleArrayList);
        }
        public void trimToSize() {
            list.trimToSize();
        }
        public void removePersistent(int index) {
            list.removePersistent(index);
        }
        public void removePersistent() {
            list.remove();
        }
        public boolean hasNext() {
            return list.hasNext();
        }
        public boolean hasPrevious() {
            return list.hasPrevious();
        }
        public Object next() {
            return list.next();
        }
        public Object previous() {
            return list.previous();
        }
        public void set(int index, Object value) {
            list.set(index, value);
        }
        public void addAll(SimpleArrayList list) {
            list.addAll();
        }
        public void removeAll(SimpleArrayList list) {
            list.removeAll();
        }
        public void retainAll(SimpleArrayList list) {
            list.retainAll();
        }
        public void removePersistent() {
            list.remove();
        }
    }
}
}

private static void LoginInit(){
    Role.MOLE.setReader();
    Role.SUPER_SPY.setReader();
    Role.TECH_GUY.setReader();
    Role.GUY.setReader();
    Role.BOND.setReader();
    Role.AGENCY.setReader();
    Role.HOT_SPY.setReader();
    Role.CHEF.setReader();
}

private static void fakeReadingArg(int i) throws NoSuchElementException {
    if (i<42) {
        return "bork bork";
    } else {
        throw new NoSuchElementException();
    }
}

private static String fakeReadingArg(int i) {
    if (i<42) {
        return "bork bork";
    } else {
        throw new NoSuchElementException();
    }
}

private static String fakeReadingArg(int i) {
    if (i<42) {
        return "bork bork";
    } else {
        throw new NoSuchElementException();
    }
}
}

private static class Role {
    private String id;
    private Role parent;
    public Role(String id) {
        this.id = id;
    }
    public final String getId() {
        return id;
    }
    public final Role getParent() {
        return parent;
    }
    public final void setParent(Role parent) {
        this.parent = parent;
    }
    public static final Role SPY = new Role("SPY");
    public static final Role SUPER_SPY = new Role("SPY07");
    public static final Role MOLE = new Role("MOLE");
    public static final Role TECH_GUY = new Role("Tech");
    private final String id;
    private Role parent;
    this.id = id;
    }

    public static boolean isRole(Role r, Role s){
        if (r == null || s == null){
            return false;
        }
        if (r.equals(s)) {
            return true;
        }
        Role entry = r;
        while(entry != null) {
            if (entry.equals(s)) {
                return true;
            }
            entry = entry.getParent();
        }
        return false;
    }
}

```

```

class Client {
    public static String fakeReadingArgs(int i)
        throws java.util.NoSuchElementException {
        if(i<42) {
            return "bork bork";
        } else {
            throw new java.util.NoSuchElementException();
        }
    }

    public void main(String args[])
        String uname;
        String pwd;
        String key;
        try{
            uname = fakeReadingArg(0);
            pwd = fakeReadingArg(1);
            key = fakeReadingArg(2);
        } catch (Exception e){
            System.out.println(
                "Usage :java Client <uname> <pwd> <key> ");
            return;
        }
    }

    LoginInit();
    Role r = Login.logIn(uname, pwd);

    If (r == null)
        System.out.println("User not found");
    else {
        If (authorizedToRead(r, key, db))
            db.secret();
        else{
            "Access not authorized".println();
        }
    }

    private static boolean authorizedToRead(Role r, String key, Database db){
        Role required = db.readerOf(key);

        for (Role.head(requires(r)))
            class Database {
                private SimpleArrayList data;
                public Database() {
                    data = new SimpleArrayList();
                    data.add(new DbEntry("A", Role.SPY,
                        "AA"));
                }
                public Database(String uname, String pwd) {
                    for(int i=0; i<data.size(); i++)
                        if (pwd.equals(data.get(i).pwd))
                            data.get(i).username = uname;
                    return this;
                }
                public void doLogin(String userName, String pwd){
                    for (int i=0; i<data.size(); i++)
                        if (pwd.equals(data.get(i).pwd))
                            if ((data.get(i).instanceof PwdTableEntry) &&
                                data.get(i).username.equals(userName) &&
                                data.get(i).pwd.equals(pwd))
                                    return data.get(i).username;
                }
                return null;
            }
        }
        class Role {
            public static final Role SPY = new Role("SPY");
            public static final Role SUPER_SPY = new Role("007");
            public static final Role MOLE = new Role("mole");
            public static final Role TECH_GUY = new Role("tech");

            private final String id;
            private Role(String id){
                this.id = id;
            }

            public static boolean isEq(Role r, Role s){
                if (r == null || s == null)
                    return false;
                else
                    return r.id.equals(s.id);
            }
        }

        return null; If no such entry
        private DbEntry lookup(String key) {
            for (int i=0; i<data.size(); i++)
                if (key.equals(data.get(i).key)) {
                    DbEntry e = new DbEntry(data.get(i));
                    if (e.getKey().equals(key))
                        return e;
                }
            }
        }

        return e;
    }

    return null;
}

public Role readerOf(String key) {
    if (lookup(key) != null)
        return lookup(key).getReader();
    return null;
}

public String secretOf(String key) {
    if (lookup(key) != null)
        return lookup(key).getData();
    return null;
}

class DBEntry{
    private String key;
    private Role reader;
    private String data;
    this.key = key;
    this.reader = reader;
    this.data = data;
}

public String getKey(){
    return this.key;
}
public Role getReader(){
    return this.reader;
}
public String getData(){
    return this.data;
}

class Login {
    private static class PwdTableEntry{
        String uname;
        String pwd;
        Role role;
        private PwdTableEntry(String uname, String pwd, Role role){
            this.uname = uname;
            this.pwd = pwd;
            this.role = role;
        }
    }

    private static SimpleArrayList peds;
    public static void init(){
        peds = new SimpleArrayList();
        peds.add(new PwdTableEntry("Bond", "007", Role.SUPER_SPY));
        peds.add(new PwdTableEntry("M", "0", Role.SPY));
        peds.add(new PwdTableEntry("Q", "1337", Role.TECH_GUY));
    }

    public static Role doLogin(String userName, String pwd){
        for (int i=0; i<peds.size(); i++)
            if (peds.get(i).instanceof PwdTableEntry) {
                PwdTableEntry e = PwdTableEntry(peds.get(i));
                if (e.username.equals(userName) && e.pwd.equals(pwd))
                    return e.role;
            }
        }
        return null;
    }

    class Role {
        public static final Role SPY = new Role("SPY");
        public static final Role SUPER_SPY = new Role("007");
        public static final Role MOLE = new Role("mole");
        public static final Role TECH_GUY = new Role("tech");

        private final String id;
        private Role(String id){
            this.id = id;
        }

        public static boolean isEq(Role r, Role s){
            if (r == null || s == null)
                return false;
            else
                return r.id.equals(s.id);
        }
    }

    return true;
}

if (r == MOLE) {
    return true;
}

return (s == r);
}

import java.util.iterator;
}

public void add(int i, Object o) {
    if (i < 0)
        throw new IllegalArgumentException();
}

public void set(int i, Object o) {
    contents.set(i, o);
}

public Object get(int i) {
    return contents[i];
}

public List removePersistent(int i) {
    if (i < 0)
        throw new IllegalArgumentException();
}

public void addAll(SimpleArrayList list) {
    Iterator i = list.iterator();
    while(i.hasNext()){
        this.add(i.next());
    }
}

private class SimpleArrayListIterator implements Iterator{

    private List list;
    public SimpleArrayListIterator(SimpleArrayList simpleArrayList) {
        list = simpleArrayList.comments;
    }

    public boolean hasNext(){
        return !list.isEmpty();
    }

    public Object next(){
        if (!list.hasNext())
            throw new NoSuchElementException();
        Object c = list.remove();
        list.add(c);
        return c;
    }

    public void remove(){
        list.remove();
    }
}

public Iterator isEmpty(){
    return SimpleArrayList.iterator();
}
}

throw new IllegalArgumentException();
}
}

return this.next;
}

return next.removePersistent(i-1);
}

public void set(int i, Object o) {
    if (i < 0)
        throw new IllegalArgumentException();
}

public Object get(int i) {
    return contents[i];
}

public List removePersistent(int i) {
    if (i < 0)
        throw new IllegalArgumentException();
}
}

throw new IllegalArgumentException();
}
}
}

Inferred policy

out  $\mapsto$  if (authCheckOk[0])  

then Reveal(secret[0+]),  

authCheckOk[1+])


```

```
class Client {
    public static String fakeReadingArgs(int i)
        throws java.util.NoSuchElementException {
        if(i<42) {
            return "bork bork";
        } else {
            throw new java.util.NoSuchElementException();
        }
    }

    public static void main(String args[])
        String uname;
        String pwd;
        String key;
        System.out.println("@output \"out\"");
        System.out.println("Usage (>java Client <uname> <pwd> <key>)");
        return;
    }
}

class Database {
    private String uname;
    private Role reader;
    private String data;
    class PwdTableEntry{
        private String key;
        private Role reader;
        private String data;
        PwdTableEntry(String key, Role reader, String data) {
            this.key = key;
            this.reader = reader;
            this.data = data;
        }
        String getKey(){return this.key;}
        Role getReader(){return this.reader;}
        String getData(){return this.data;}
        void setKey(String key){this.key = key;}
        void setReader(Role reader){this.reader = reader;}
        void setData(String data){this.data = data;}
    }
    static class PwdTableEntry{
        private String uname;
        private String pwd;
        private Role role;
        PwdTableEntry(String uname, String pwd, Role role){
            this.uname = uname;
            this.pwd = pwd;
            this.role = role;
        }
        boolean equals(PwdTableEntry p) {
            return this.uname.equals(p.uname) && this.pwd.equals(p.pwd) && this.role.equals(p.role);
        }
    }
    public static void init() {
        PwdTableEntry p1 = new PwdTableEntry("Bond", "007", Role.SUPER_SPY);
        PwdTableEntry p2 = new PwdTableEntry("M", "0", Role.SPY);
        PwdTableEntry p3 = new PwdTableEntry("Q", "1337", Role.TECH_GUY);
        HashMap<String, PwdTableEntry> pwdMap = new HashMap<String, PwdTableEntry>();
        pwdMap.put("Bond", p1);
        pwdMap.put("M", p2);
        pwdMap.put("Q", p3);
        login(String.userName, String.string);
    }
    void login(String.userName, String.string) {
        PwdTableEntry p = pwdMap.get(String.string);
        if(p != null) {
            if(p.equals(PwdTableEntry.pwd)) {
                System.out.println("Success");
            } else {
                System.out.println("Failure");
            }
        }
    }
}

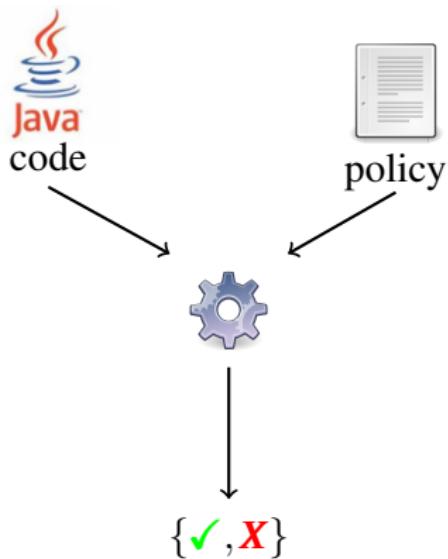
class Role {
    private String id;
    private Role(String id) {
        this.id = id;
    }
    public static boolean isEqual(Role r, Role s) {
        if(r == null || s == null) {
            return false;
        }
        RoleEntry r1 = (RoleEntry) r;
        RoleEntry s1 = (RoleEntry) s;
        if(r.getKey().equals(s.getKey())) {
            return true;
        }
    }
}

class RoleEntry {
    private String key;
    private Role reader;
    private String data;
    RoleEntry(String key, Role reader, String data) {
        this.key = key;
        this.reader = reader;
        this.data = data;
    }
    String getKey(){return this.key;}
    Role getReader(){return this.reader;}
    String getData(){return this.data;}
    void setKey(String key){this.key = key;}
    void setReader(Role reader){this.reader = reader;}
    void setData(String data){this.data = data;}
}
```

## Inferred policy

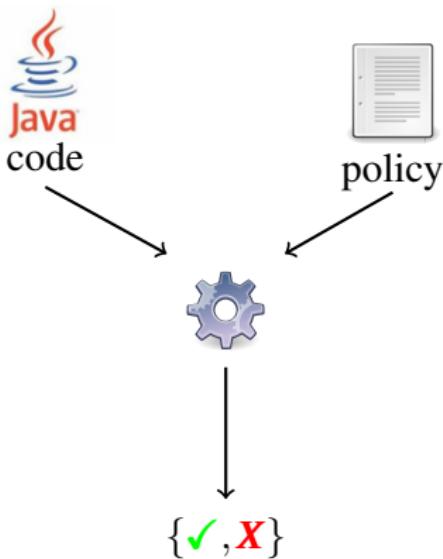
out  $\mapsto$  if (authCheckOk[0])  
then Reveal(secret[0+]), authCheckOk[1+])

# Goal: Workflow of iterative policy refinement via inference

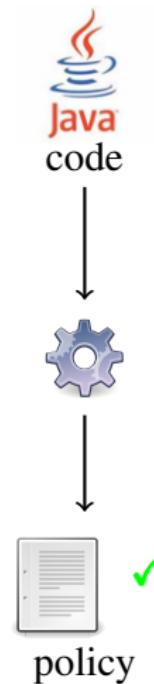


JIF [Myers et. al. '03],  
FlowCaml [Simonet '03], ...

# Goal: Workflow of iterative policy refinement via inference



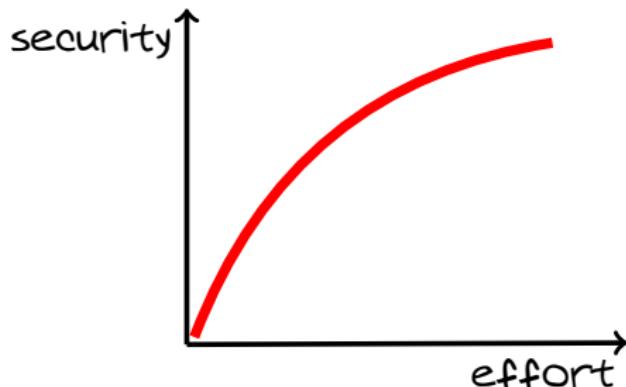
JIF [Myers et. al. '03],  
FlowCaml [Simonet '03], ...



This work

## Goal: Analysis suitable for mostly unmodified, legacy Java programs

- Low annotation burden
- Deep analysis of practical Java features: exceptions, heap allocated objects, etc.
- Security guarantees scale with effort



## The central tension:



# Outline

1 Policies

2 Inference

3 Implementation and Validation

Reveal policies describe *what* information is released

```
x =           readln();  
y = x;  
println(      y);
```

Reveal policies describe *what* information is released

```
x = @input "stdin" readln();  
y = x;  
println(@output "stdout" y);
```

Reveal policies describe *what* information is released

```
x = @input "stdin" readln();  
y = x;  
println(@output "stdout" y);
```

Inferred policy

```
stdout  $\mapsto$  Reveal( stdin[0] )
```

## *Precise input expressions* name inputs and derived data exactly

```
String read(){ return           readln(); }  
  
x = read();  
y = read();  
println(           (x == null));
```

*Precise input expressions* name inputs and derived data exactly

```
String read(){ return @input "H" readln(); }
```

```
x = read();
y = read();
println(@output "stdout" (x == null));
```

*Precise input expressions* name inputs and derived data exactly

```
String read(){ return @input "H" readln(); }
```

```
x = read();
y = read();
println(@output "stdout" (x == null));
```

## Inferred policy

```
stdout  $\mapsto$  Reveal( H[1] == null )
```

$H[0]$  — the most recent  $H$  input

$H[1]$  — the second most recent  $H$  input

⋮

## Imprecise expressions name multiple inputs sharing a label

```
int i = 0;  
while (i <= 100)  
    i += parse(@input "in" readln());  
}  
println(@output "out" i);
```

# Imprecise expressions name multiple inputs sharing a label

```
int i = 0;  
while (i <= 100)  
    i += parse(@input "in" readln());  
}  
println(@output "out" i);
```

## Inferred policy

out  $\mapsto$  **Reveal**( in[0+] )

in[0+] — information from any in input

in[1+] — info. from the second most recent and older in inputs

⋮

Conditional policies describe  
*when* information is released

```
secret = @input "secret" ...
password = @input "password" ...
guess = @input "guess" ...
```

```
if (guess == password) {
    println(@output "stdout" secret);
}
```

Policy?

```
stdout ↢ Reveal( secret[0], guess[0], password[0] )
```

Conditional policies describe  
*when* information is released

```
secret = @input "secret" ...
password = @input "password" ...
guess = @input "guess" ...
```

```
if (guess == password) {
    println(@output "stdout" secret);
}
```

### Inferred policy

```
stdout ↢ if ( guess[0] == password[0] ) then
    Reveal( secret[0] )
```

## Track policies describe *where* information is released

```
String safe_release(String x) {  
    if /* auth check */ {  
        return x;  
    } else {  
        return "";  
    }  
}  
  
x = @input "in" readIn();  
y = safe_release(x);  
println(@output "out" y);
```

Track policies describe *where* information is released

```
String safe_release(String x) @track {  
    if /* auth check */ {  
        return x;  
    } else {  
        return "";  
    }  
}  
x = @input "in" readIn();  
y = safe_release(x);  
println(@output "out" y);
```

Track policies describe *where* information is released

```
String safe_release(String x) @track {  
    if /* auth check */ {  
        return x;  
    } else {  
        return "";  
    }  
}  
x = @input "in" readIn();  
y = safe_release(x);  
println(@output "out" y);
```

Inferred policy

out  $\mapsto$  **if-executed** safe\_release(String) **then**  
**Reveal**(in[0])

# Inference

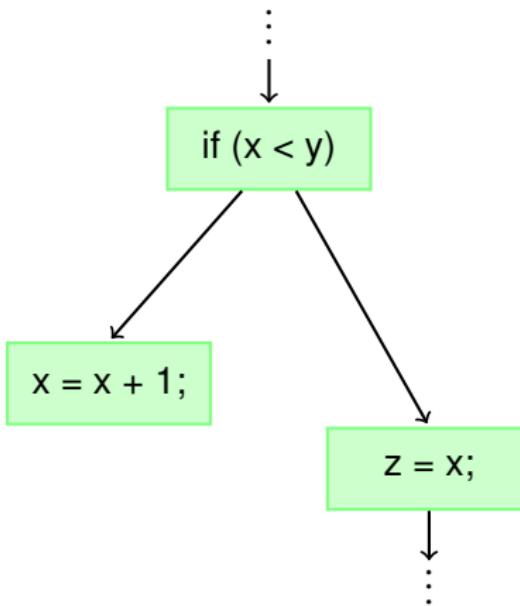
# Inference algorithm is based on dataflow analysis

```
if (x < y){  
    x = x + 1;  
}  
z = x;
```

# Inference algorithm is based on dataflow analysis

```
if (x < y){  
    x = x + 1;  
}  
z = x;
```

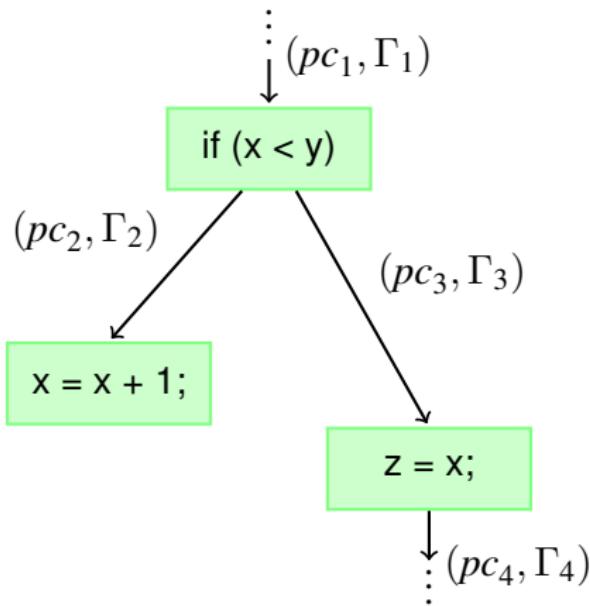
$\Rightarrow$



# Inference algorithm is based on dataflow analysis

```
if (x < y){  
    x = x + 1;  
}  
z = x;
```

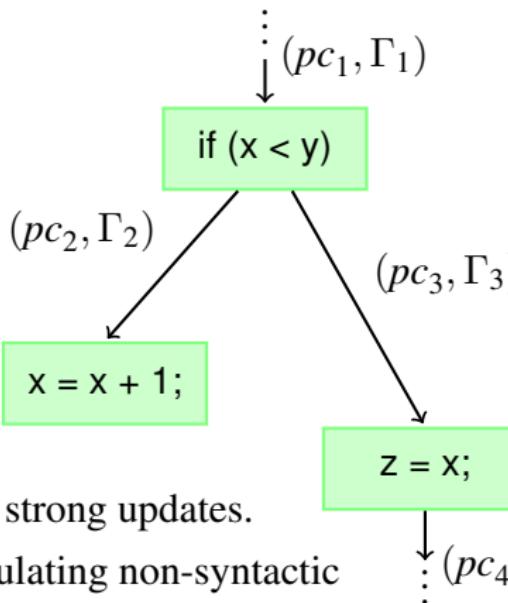
$\Rightarrow$



# Inference algorithm is based on dataflow analysis

```
if (x < y){  
    x = x + 1;  
}  
z = x;
```

$\Rightarrow$



- ✓ Natural handling of strong updates.
- ✓ Convenient for calculating non-syntactic control dependencies.

Contexts  $(pc, \Gamma)$  approximates  
information flow

**Location context:** information content of fields or heap locations.

$$\Gamma : \mathbf{Location} \multimap \mathbf{Policy}$$

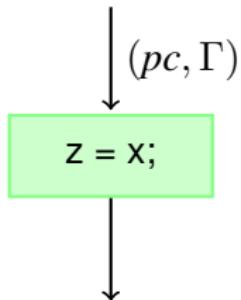
**Program counter:** what each branch taken to a program point reveals.

$$pc : \mathbf{BranchId} \multimap \mathbf{Policy} \cup \mathbf{PIE}$$

**Location** = abstract vars and refs      **PIE** = precise input exps.

**BranchId** = static branch ids      **Policy** = policies

## Example: Updating a context for an assignment node

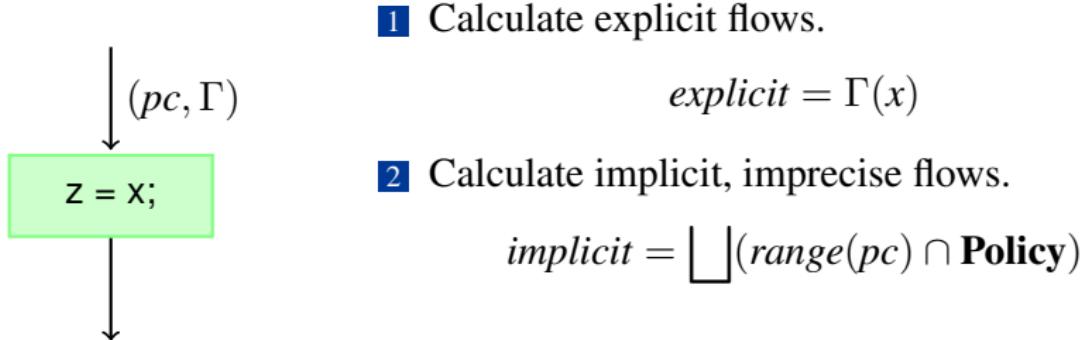


## Example: Updating a context for an assignment node

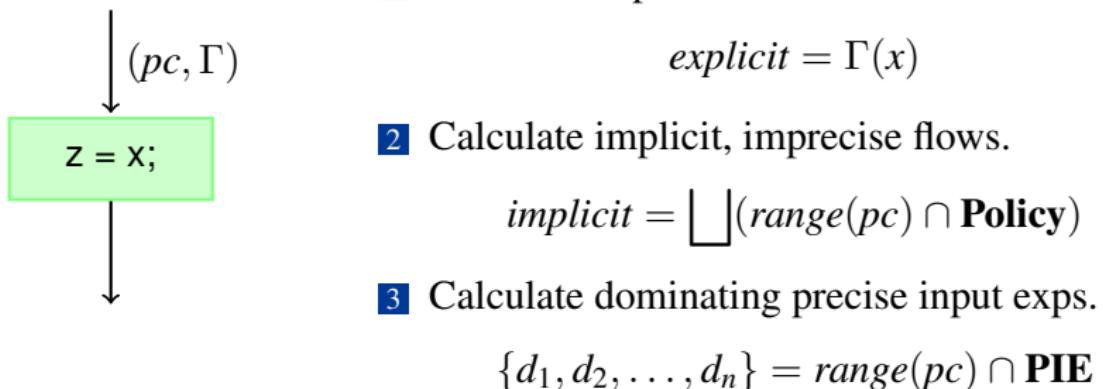
1 Calculate explicit flows.



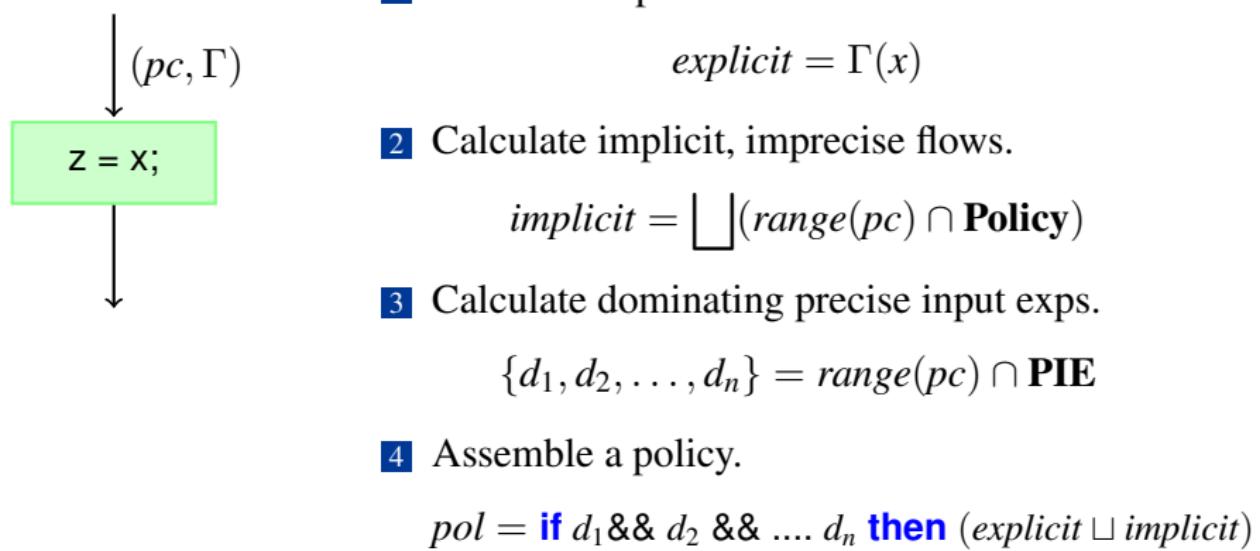
## Example: Updating a context for an assignment node



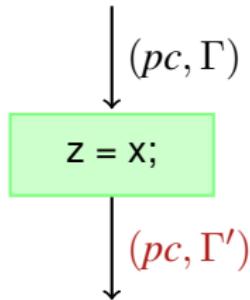
## Example: Updating a context for an assignment node



## Example: Updating a context for an assignment node



## Example: Updating a context for an assignment node



- 1 Calculate explicit flows.

$$\text{explicit} = \Gamma(x)$$

- 2 Calculate implicit, imprecise flows.

$$\text{implicit} = \bigsqcup(\text{range}(pc) \cap \mathbf{Policy})$$

- 3 Calculate dominating precise input exps.

$$\{d_1, d_2, \dots, d_n\} = \text{range}(pc) \cap \mathbf{PIE}$$

- 4 Assemble a policy.

$$pol = \mathbf{if } d_1 \&& d_2 \&& \dots d_n \mathbf{then } (\text{explicit} \sqcup \text{implicit})$$

- 5 Update the dataflow graph.

$$\Gamma' = \Gamma[z \mapsto pol]$$

# Implementation and Validation

## We prototyped an implementation that analyses Java 1.4 programs

- Implementation (35,100 lines of Java code) extends the Polyglot compiler framework. [Nystrom, Clarkson, & Myers '03]
- Program analyses refine security inference:
  - Object-sensitive **pointer analysis** reasons precisely about non-aliased objects, preventing label creep.
  - **Post dominance analysis** determines identifies where *pc* levels may be safely lowered.
  - Precise **type and exception analysis** simplifies control flow graph by ruling out spurious exceptions (e.g. from casts that always succeed).

# Case studies: successful inference of interesting information flows

Computer  
psychiatrist

Inputs sanitized before display.

Interactive password  
manager

Passwords only revealed via (simulated)  
cryptography.

Access-controlled  
key-value store

Information release governed by reference  
monitor.

Go Fish

The computer doesn't cheat.

Battleship game

The computer doesn't cheat. [Jif]

Mental poker

Private keys influence outputs appropriately  
[Askarov & Sabelfeld '05]

# Sample policy: Battleship

```
return lookup(key).getReader();
}
return null;
}

public String secretOf(String key) {
    if (lookup(key) != null) {
        return lookup(key).getData();
    }
    return null;
}

class DbEntry {
    private String key;
    private Role reader;
    private String data;
}

public DbEntry(String key, Role reader, String data) {
    this.key = key;
    this.reader = reader;
    this.data = data;
}

public String getKey() { return this.key; }
public Role getReader() { return this.reader; }
public String getData() { return this.data; }
}

class Login {

private static class PwdTableEntry< class Role {
    String uname;
    public static final Role SPY = new Role("Spy");
    public static final Role MOLE = new Role("Mole");
    public static final Role TECH_GUY = new Role("Tech");
}

    private PwdTableEntry(String uname) {
        this.uname = uname;
        this.pwd = pwd;
        this.role = role;
    }

    private final String id;
    private Role(role) {
        this.id = id;
    }

    public static SimpleArrayList pwds;
    public static void init() {
        pwds = new SimpleArrayList();
        pwds.add(new PwdTableEntry("Bond", "007", Role.SUPER_SPY));
        pwds.add(new PwdTableEntry("M", "0", Role.SPY));
        pwds.add(new PwdTableEntry("Q", "1837", Role.TECH_GUY));
    }

    public static Role doLogin (String userName, String pwd) {
        if (r == MOLE) {
            return true;
        }
        if (pwds.get(i) instanceof PwdTableEntry) {
            PwdTableEntry e = (PwdTableEntry) pwds.get(i);
            if (e.uname.equals(userName) && e.pwd.equals(pwd)) {
                return e.role;
            }
        }
    }

    public String getKey() { return this.key; }
    public Role getReader() { return this.reader; }
    public String getData() { return null; }
}

class Login {

private static class PwdTableEntry< class Role {
    String uname;
    public static final Role SPY = new Role("Spy");
    public static final Role MOLE = new Role("Mole");
    public static final Role TECH_GUY = new Role("Tech");
}

    private PwdTableEntry(String uname) {
        this.uname = uname;
        this.pwd = pwd;
        this.role = role;
    }

    private final String id;
    private Role(role) {
        this.id = id;
    }

    public static SimpleArrayList pwds;
    public static void init() {
        pwds = new SimpleArrayList();
        pwds.add(new PwdTableEntry("Bond", "007", Role.SUPER_SPY));
        pwds.add(new PwdTableEntry("M", "0", Role.SPY));
        pwds.add(new PwdTableEntry("Q", "1837", Role.TECH_GUY));
    }

    public static Role doLogin (String userName, String pwd) {
        if (r == MOLE) {
            return true;
        }
        if (pwds.get(i) instanceof PwdTableEntry) {
            PwdTableEntry e = (PwdTableEntry) pwds.get(i);
            if (e.uname.equals(userName) && e.pwd.equals(pwd)) {
                return e.role;
            }
        }
    }

    public String getKey() { return this.key; }
    public Role getReader() { return this.reader; }
    public String getData() { return null; }
}

private static abstract class List {
    private List() {}
    abstract int size();
    boolean isEmpty() { return contents.isEmpty(); }
    void clear() { this.contents = new Nil(); }

    private static abstract class Nil extends List {
        private Nil() {}
        public String secretOf(String key) {
            if (lookup(key) != null) {
                return lookup(key).getData();
            }
            return null;
        }

        public abstract Object get(int i);
        public abstract List removePersistent(int i);
        public abstract void set(int i, Object o);
        public abstract Object head() throws NoSuchElementException;
        public abstract List tail() throws NoSuchElementException;
        private String key;
        private Role reader;
        private String data;
    }

    public static class DbEntry extends List {
        private DbEntry(String key, Role reader, String data) {
            this.key = key;
            this.reader = reader;
            this.data = data;
        }

        public int size() { return 0; }
        public boolean isEmpty() { return true; }
        public Object get(int i) { throw new NoSuchElementException(); }
        public void set(int i, Object o) { throw new NoSuchElementException(); }
        public Object head() { throw new NoSuchElementException(); }
        public List tail() { throw new NoSuchElementException(); }
        private String getKey() { return this.key; }
        private Role getReader() { return this.reader; }
        private String getData() { return this.data; }
    }

    public static class Login extends List {
        public void set(int i, Object o) { throw new NoSuchElementException(); }
        public Object head() { throw new NoSuchElementException(); }
        public static class PwdTableEntry< class Role {
            String uname;
            public static final Role SPY = new Role("Spy");
            public static final Role MOLE = new Role("Mole");
            public static final Role TECH_GUY = new Role("Tech");
        }

        private Login() {}
        public String secretOf(String key) {
            if (lookup(key) != null) {
                return lookup(key).getData();
            }
            return null;
        }
    }
}
```

# Sample policy: Battleship

```

        return lookup(key).getReader();
    }
    return null;
}

public String secretOf(String key) {
    if (lookup(key) != null) {
        return lookup(key).getData();
    }
    return null;
}

class DbEntry {
    private String key;
    private Role reader;
    private String data;
}

public DbEntry(String key, Role reader, String data) {
    this.key = key;
    this.reader = reader;
    this.data = data;
}

public String getData() { return this.data; }

class Login {
}

private static class PwdTableEntry {
    class Role {
        String uname;
        public static final Role SUPER_SPY = new Role("super");
        public static final Role SPY = new Role("spy");
        public static final Role TECH_GUY = new Role("tech");
        public static final Role MOLE = new Role("mole");
    }
    private SimpleArrayList pwds;
    private final String id;
    private Role(id);
    this.id = id;
}

private PwdTableEntry(String uname, String pwd, Role role) {
    this.uname = uname;
    this.pwd = pwd;
    this.role = role;
}

private static SimpleArrayList pwds;
public static void init() {
    pwds = new SimpleArrayList();
    pwds.add(new PwdTableEntry("Bond", "007", Role.SUPER_SPY));
    pwds.add(new PwdTableEntry("M", "0", Role.SPY));
    pwds.add(new PwdTableEntry("O", "007", Role.TECH_GUY));
}

public static Role doLogin (String id) {
    if (pwds.size() == 0) {
        return null;
    }
    for (PwdTableEntry e : pwds) {
        if (e.uname.equals(username) && e.pwd.equals(pwd)) {
            return e.role;
        }
    }
    return null;
}

public String secretOf(String key) {
    if (lookup(key) != null) {
        return contents.size();
    }
    return null;
}

private String secretOf(String key) {
    if (lookup(key) != null) {
        return contents.size();
    }
    return null;
}

private String secretOf(String key) {
    if (lookup(key) != null) {
        return contents.size();
    }
    return null;
}

```



Human  
player, p1

@input "p1 board"

```

private String secretOf(String key) {
    if (lookup(key) != null) {
        return contents.size();
    }
    return null;
}

boolean isEmpty() { return contents.isEmpty(); }

void clear() { this.contents = new Nil(); }

private class ComputerPlayer {
    public String secretOf(String key) {
        if (lookup(key) != null) {
            return contents.size();
        }
        return null;
    }

    public abstract List removePersistent(int i);
    public abstract void set(int i, Object o);
    public abstract Object head() throws NoSuchElementException;
    public abstract List tail() throws NoSuchElementException;
    private String key;
    private Role reader;
    private String data;
}

ends List {
    public DbEntry(String key, Role reader, String data) {
        this.key = key;
        this.reader = reader;
        this.data = data;
    }
    public boolean isEmpty() { return contents.isEmpty(); }
    public Object get(int i) throws NoSuchElementException;
    public void set(int i, Object o) throws IllegalArgumentException;
    public List removePersistent(int i) {
        throw new IllegalArgumentException();
    }
    public String getKey() { return key; }
    public Role getReader() { return reader; }
    public String getData() { return data; }
}

class Login {
}

public Object head() throws NoSuchElementException {
    throw new NoSuchElementException();
}

public static class PwdTableEntry implements Iterable<Role> {
    private String uname;
}

private static class SimpleArrayList implements Iterable<Object> {
    private List contents;
}
```

Computer  
player, p2



# Sample policy: Battleship

```
return lookup(key).getReader();
}
return null;
}

public String secretOf(String key) {
    if (lookup(key) != null) {
        return lookup(key).getData();
    }
    return null;
}

class DbEntry {
    private String key;
    private Role reader;
    private String data;
}

public DbEntry(String key, Role reader, String data) {
    this.key = key;
    this.reader = reader;
    this.data = data;
}

public String getData() { return this.data; }

class Login {
    private static class PwdTableEntry {
        class Role {
            String uname;
            public static final Role SUPER_SPY = new Role("super");
            public static final Role SPY = new Role("spy");
            public static final Role TECH_GUY = new Role("tech");
            public static final Role MOLE = new Role("mole");
        }
        private SimpleArrayList pwds;
        private final String id;
        private Role(Role id) {
            this.id = id;
        }
        public static void init() {
            pwds = new SimpleArrayList();
            pwds.add(new PwdTableEntry("Bond", "007", Role.SUPER_SPY));
            pwds.add(new PwdTableEntry("M", "0", Role.SPY));
            pwds.add(new PwdTableEntry("O", "007", Role.TECH_GUY));
            pwds.add(new PwdTableEntry("R", "007", Role.MOLE));
        }
        public static Role doLogin (String id) {
            if (pwds.size() == 0) {
                return null;
            }
            for (PwdTableEntry e : PwdTableEntry.pwds) {
                if (e.uname.equals(username) && e.getPassword().equals(pwd)) {
                    return e.role;
                }
            }
            return null;
        }
    }
}
```

Human  
player, p1

@input "p1 board"

p2 query  $\mapsto$  if (p2 query ok[0])  
then Reveal(p2 query ok[1+], p2 hit p1?[0+], p1 hit p2?[0+])



Computer player, p2

```
private class SimpleArrayList implements Iterable<Object> {
    List contents;
    public int size() { return contents.size(); }
    boolean isEmpty() { return contents.isEmpty(); }
    void clear() { this.contents = new Nil(); }
    public Object get(int i) { return null; }
    public abstract List removePersistent(int i);
    public abstract void set(int i, Object o);
    public abstract Object head() throws NoSuchElementException;
    public abstract List tail() throws NoSuchElementException;
    private String key;
    private Role reader;
    private String data;
}

ends List {
```

```
public DbEntry(String key, Role reader, String data) {
    this.key = key;
    this.reader = reader;
    this.data = data;
}
public List removePersistent(int i) {
    throw new IllegalArgumentException();
}
public String getKey() { return this.key; }
public Role getReader() { return this.reader; }
public String getData() { return this.data; }
}

public void set(int i, Object o) { throw new IllegalArgumentException(); }
}

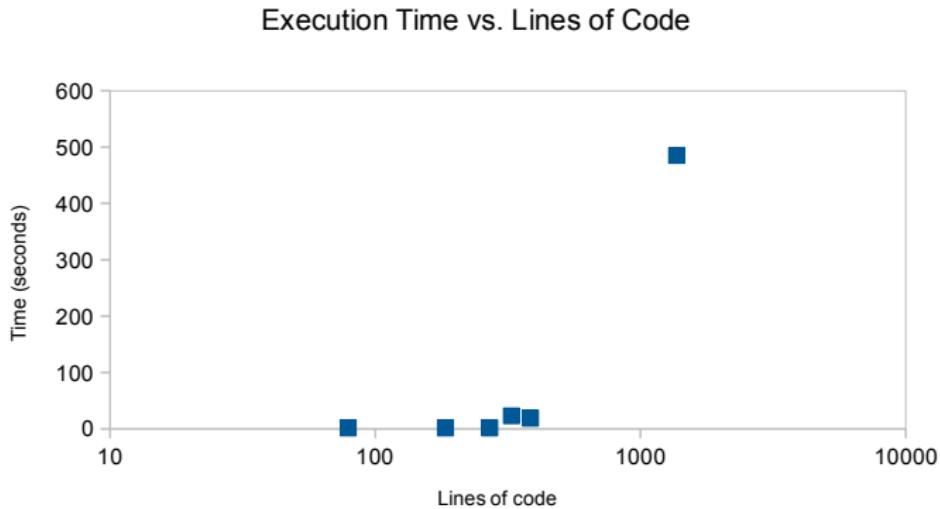
class Login {
    private static class PwdTableEntry {
        class Role {
            String uname;
```

## Sample policy: Computer psychiatrist

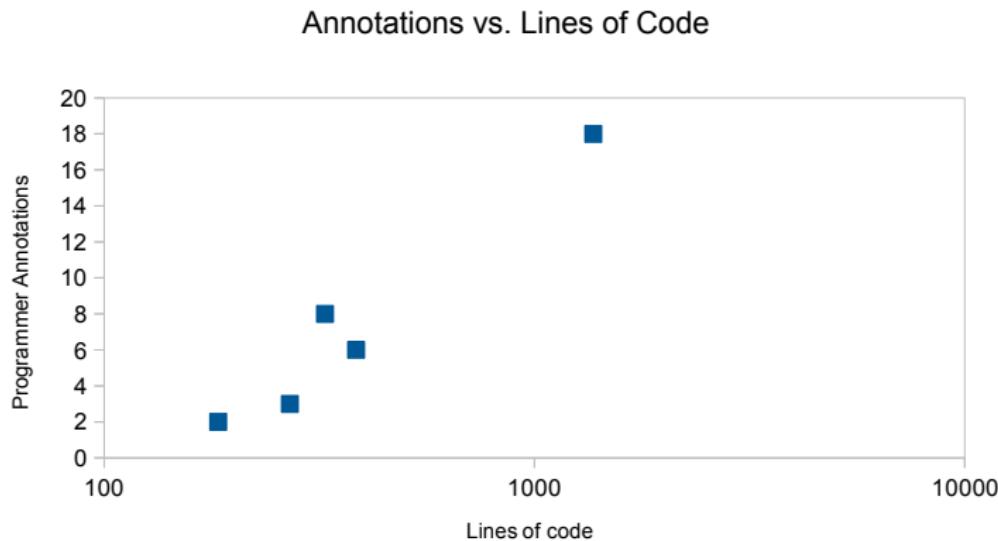
```
display ↦ if-executed (Liz.sanitize(java.lang.String))  
then Reveal(input[0], input[1+])
```

Inputs are not displayed unless `sanitize` has been called.

The prototype scales to low 1000s loc, but is not performance tuned



We succeeded in requiring few programmer annotations



# Conclusion

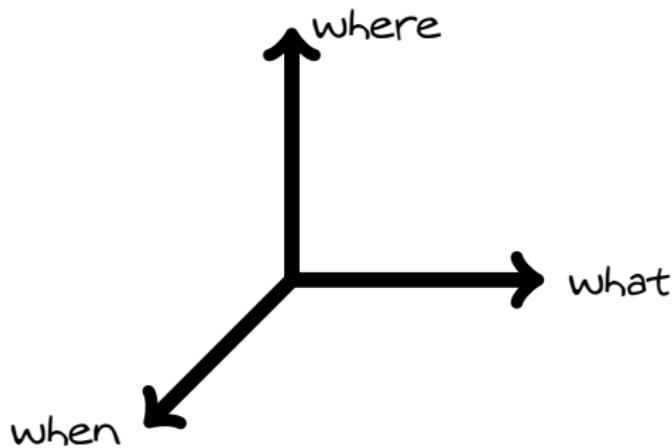
# Policy inference via dataflow is a promising info. flow analysis

This project demonstrates:

- A novel workflow of iterative policy refinement via inference
- Expressive policies that explain information flows and reflect program structure
- A precise analysis suitable for (mostly) unmodified Java programs

# Bonus Slides

Policies express how information flows from inputs to outputs



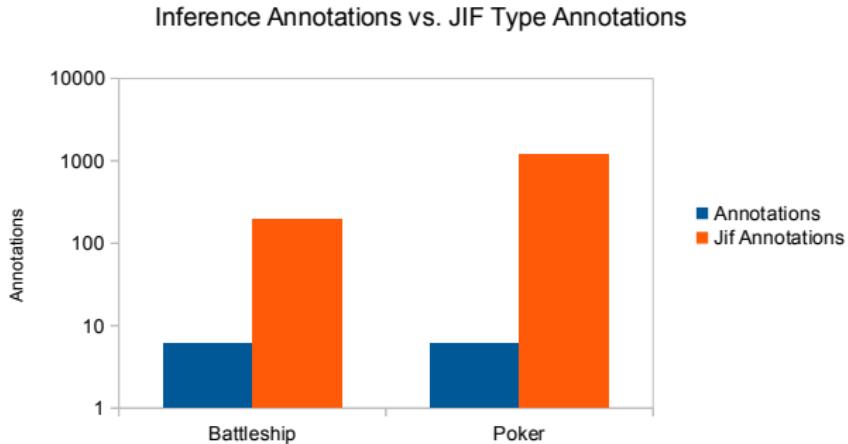
[Sabelfeld & Sands '05] NB. Our policies generalize **delimited release** [Askarov & Sabelfeld '07]

Annotation burden compares well  
with more explicit systems



Jif primary goals: modular typing, DLM policies, and no inference.  
[Myers, Zheng, Zdancewic, Chong, Nystrom '01-'09]

# Annotation burden compares well with more explicit systems



Jif primary goals: modular typing, DLM policies, and not inference.  
[Myers, Zheng, Zdancewic, Chong, Nystrom '01–'09]

# Goal: Policies that explain and reflect information flows

## Inferred policy

```
out  $\mapsto$  if (authCheckOk[0])  
      then Reveal(secret[0+], authCheckOk[1+])
```

## Policies...

- ... reflect structure of information flow within program
- ... describe flow in terms of meaningful *channel names*

Policy rewriting is needed for widening and improves readability

## Example

$$\mathbf{Reveal}(\mathsf{H}[j], \mathsf{H}[i+]) = \mathbf{Reveal}[i+] \quad (\text{provided } i \leq j)$$

$$\mathbf{Reveal}(\mathsf{H}[j], \mathsf{H}[i+]) \sqsubseteq \mathbf{Reveal}(\mathsf{H}[\min(i, j)+]) \sqsubseteq \mathbf{Reveal}(\mathsf{H}[0+])$$

Widening  $\nabla(p_1, p_2)$ :

- Let  $p := p_1 \text{ and } p_2$
- If  $|p| \leq \text{threshold}$  return  $p$
- Rewrite ( $=$ ) until  $|p| \leq \text{threshold}$  or no further rewrites possible
- Rewrite conservatively ( $\sqsubseteq$ ) until  $|p| \leq \text{threshold}$
- Result :=  $p$