

JAIL: Firewall Analysis of Java Card by Abstract Interpretation

Pietro Ferrara

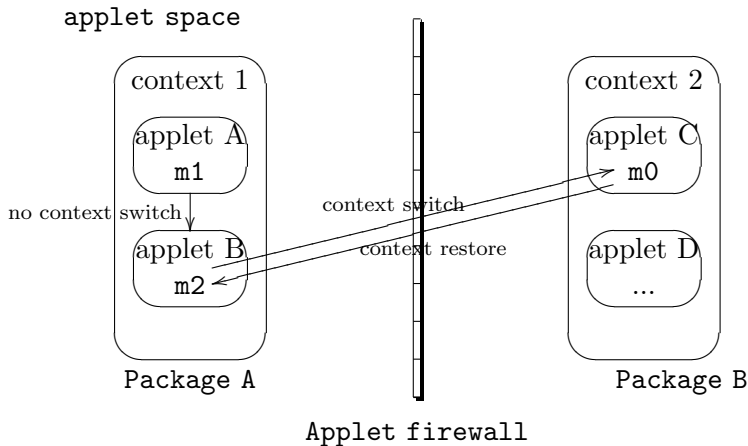
Università Ca' Foscari di Venezia, Italy
Ecole Polytechnique, Paris

Vienna, 26th March 2006

Java Card

- *"Java Card technology provides a secure environment for applications that run on smart cards and other devices with very limited memory and processing capabilities.*
- *Applications written in the Java programming language can be executed securely on cards from different vendors"*

from: <http://java.sun.com/products/javacard/>



An example (1/2)

```
package smartcard_reader;
public interface User extends Shareable {
    public void use();
}
package smartcard_reader;
public interface Administrator extends Shareable {
    public void use();
    public void manage();
}
package smartcard_reader;
public class AppletUser extends Applet implements User {
    public void use(){...}
    public void manage(){...}
}
package smartcard_reader;
public class AppletAdministrator extends AppletUser implements
Administrator {
    public void use(){...}
    public void manage(){...}
}
```

An example(2/2)

```
package smartcard;
public class AppletService extends Applet {
    AppletUser app=null;
    public AppletService(int code) throws Exception {
        if(code==123)
            app=(AppletAdministrator) JCSYSTEM.getShareableInterfaceObject(0);
        else app=(AppletUser) JCSYSTEM.getShareableInterfaceObject(1);
        register();
    }
    public void use(int i) {
        if(i==0) app.manage();
        else app.use();
    }
}
```

The statement `app.manage()` may violate the applet isolation property because:

- `app` may be an instance of `AppletAdministrator` or `AppletUser` class
- `AppletUser` does not declare the method `manage` as `Shareable`

Our idea

- This property is checked by Java Card at run-time
- We want to check statically if an applet respects this property...
- ... our approach is based on abstract interpretation
- Given the source code of an applet, we want to analyse if it respects this property, for all possible contexts of interaction with this applet

Outline

- 1 Concrete and abstract domains
- 2 JAIL and its instances
- 3 Conclusions

Concrete domain

We want to define a semantics that supports object aliasing and that allows each object to have its own environment

- Environment: partial function that, given the name of a variable, returns its address ($E : N \rightarrow A$);
- Store: partial function that, given an address, returns the value contained ($S : A \rightarrow V$);
- Value: it can be an object (whose status is contained in an environment) or a number ($V = E \cup PV$);
- Exceptions' function: function that, given the name of an exception, returns the state in which it has been thrown ($EX : N \rightarrow ST$);
- State: it is triple composed by an environment, a store, and an exceptions' function ($ST = E \times S \times EX$)

Abstract domain - Addresses

- in the concrete semantics an address is a numerical value
- in the abstract semantics we can not use a numerical counter, that is incremented each time that we create a new location in the store. In this way the convergence of the analysis is not guaranteed.

Abstract domain - Addresses

- in the concrete semantics an address is a numerical value
 - in the abstract semantics we can not use a numerical counter, that is incremented each time that we create a new location in the store. In this way the convergence of the analysis is not guaranteed.
 - $A^\# = P \times Int \times Int \times Stack$ where
 - the first value identifies the point of the source code (P is the set of the identifier of the points of program)...
 - the second value contains the number of iterations of a possible loop..
 - the third value contains the number of iterations of the property computation (we will discuss about it)...
 - the fourth value is the stack of method calls...
- ... in which the abstract memory address has been allocated

Abstract domain - Environments

$$E^\# : N \rightarrow \wp(A^\#)$$

- Partial ordering relation:

$$\hat{e}_1 \leq_{E^\#} \hat{e}_2 \iff \text{dom}(\hat{e}_1) \subseteq \text{dom}(\hat{e}_2) \wedge \\ \forall n \in \text{dom}(\hat{e}_1) : \hat{e}_1(n) \subseteq \hat{e}_2(n)$$

- Abstraction function:

$$\alpha'_E(e) = \hat{e} : N \rightarrow \wp(A)^\# \text{ such that} \\ \text{dom}(\hat{e}) = \text{dom}(e) \wedge \forall n \in \text{dom}(e) : \hat{e}(n) = \alpha'_A(e(n))$$

- Concretization function:

$$\gamma_E(\hat{e}) = \{e_i : N \rightarrow A \mid i \in [1..n]\} \text{ such that} \\ \forall i \in [1..n] : \text{dom}(e_i) = \text{dom}(\hat{e}) \wedge \forall n \in \text{dom}(\hat{e}) : e_i(n) \in \gamma_A(\hat{e}(n)) \\ \wedge \forall n \in \text{dom}(\hat{e}) : \bigcup_{i \in [1..n]} e_i(n) = \gamma_A(\hat{e}(n))$$

Abstract domain - Stores

$$S^\# : A^\# \rightarrow V^\#$$

- Partial ordering:

$$\hat{s}_1 \leq_{S^\#} \hat{s}_2 \iff \text{dom}(\hat{s}_1) \subseteq \text{dom}(\hat{s}_2) \\ \forall \hat{a} \in \text{dom}(\hat{s}_1) : \hat{s}_1(\hat{a}) \leq_{V^\#} \hat{s}_2(\hat{a})$$

- Abstraction function:

$$\alpha'_S(s) = \hat{s} : A^\# \rightarrow V^\# \text{ such that} \\ \text{dom}(\hat{s}) = \bigcup_{a \in \text{dom}(s)} \alpha'_A(a) \\ \wedge \forall a \in \text{dom}(s) : \hat{s}(\alpha'_A(a)) = \alpha'_V(s(a))$$

- Concretization function:

$$\gamma_S(\hat{s}) = \{s_i : A \rightarrow V \mid i \in [1..n]\} \text{ such that} \\ \forall i \in [1..n] : \text{dom}(s_i) = \bigcup_{\hat{a} \in \text{dom}(\hat{s})} \gamma_A(\hat{a}) \\ \wedge \forall \hat{a} \in \text{dom}(\hat{s}), a \in \gamma_A(\hat{a}) : s_i(a) \in \gamma_V(\hat{s}(\hat{a})) \\ \wedge \forall \hat{a} \in \text{dom}(\hat{s}) : \bigcup_{i \in [1..n], a \in \gamma_A(\hat{a})} s_i(a) = \gamma_V(\hat{s}(\hat{a}))$$

Abstract domain - Exceptions' function

$$EX^\# : N \rightarrow ST^\#$$

- Partial ordering:

$$\begin{aligned} \hat{ex}_1 \leq_{EX^\#} \hat{ex}_2 &\iff dom(\hat{ex}_1) \subseteq dom(\hat{ex}_2) \\ &\wedge \forall n \in dom(\hat{ex}_1) : \hat{ex}_1(n) \leq_{ST^\#} \hat{ex}_2(n) \end{aligned}$$

- Abstraction function:

$$\begin{aligned} \alpha'_{EX}(ex) = \hat{ex} : N \rightarrow ST^\# \text{ such that} \\ dom(\hat{ex}) = dom(ex) \wedge \forall n \in dom(ex) : \hat{ex}(n) = \alpha'_{ST}(ex(n)) \end{aligned}$$

- Concretization function:

$$\begin{aligned} \gamma_{EX}(\hat{ex}) = \{ex_i : N \rightarrow ST \mid i \in [1..n]\} \text{ such that} \\ \forall i \in [1..n] : dom(ex_i) = dom(\hat{ex}) \\ \wedge \forall n \in dom(\hat{ex}) : ex_i(n) \in \gamma_{ST}(\hat{ex}(n)) \\ \wedge \forall n \in dom(\hat{ex}) : \bigcup_{i \in [1..n]} ex_i(n) = \gamma_{ST}(\hat{ex}(n)) \end{aligned}$$

Abstract domain - States

$$ST^\# = E^\# \times S^\# \times EX^\#$$

- Partial ordering:

$$\hat{st}_1 \leq_{ST^\#} \hat{st}_2 \iff \hat{e}_1 \leq_{E^\#} \hat{e}_2 \wedge \hat{s}_1 \leq_{S^\#} \hat{s}_2 \wedge \hat{ex}_1 \leq_{EX^\#} \hat{ex}_2$$

- Abstraction function:

$$\alpha'_{ST}((e, s, ex)) = (\alpha'_E(e), \alpha'_S(s), \alpha'_{EX}(ex))$$

- Concretization function:

$$\begin{aligned} \gamma_{ST}((\hat{e}, \hat{s}, \hat{ex})) &= (e, s, ex) \text{ such that} \\ &e \in \gamma_E(\hat{e}), s \in \gamma_S(\hat{s}), ex \in \gamma_{EX}(\hat{ex}) \wedge \\ &\bigcup_{st=(e,s,ex) \in \gamma_{ST}((\hat{e}, \hat{s}, \hat{ex}))} e = \gamma_E(\hat{e}), \\ &\bigcup_{st=(e,s,ex) \in \gamma_{ST}((\hat{e}, \hat{s}, \hat{ex}))} s = \gamma_S(\hat{s}), \\ &\bigcup_{st=(e,s,ex) \in \gamma_{ST}((\hat{e}, \hat{s}, \hat{ex}))} ex = \gamma_{EX}(\hat{ex}) \end{aligned}$$

Least upper bound operator (1/2)

- Environments:

$$\hat{e}_1 \sqcup_{E\#} \hat{e}_2 = \{ \hat{e} : n \mapsto \hat{a} \mid \hat{a} = \begin{cases} \hat{e}_1(n) & \text{if } n \in \text{dom}(\hat{e}_1) \setminus \text{dom}(\hat{e}_2) \\ \hat{e}_2(n) & \text{if } n \in \text{dom}(\hat{e}_2) \setminus \text{dom}(\hat{e}_1) \\ \hat{e}_1(n) \cup \hat{e}_2(n) & \text{if } n \in \text{dom}(\hat{e}_1) \cap \text{dom}(\hat{e}_2) \end{cases} \}$$

- Stores:

$$\hat{s}_1 \sqcup_{S\#} \hat{s}_2 = \{ \hat{s} : \hat{a} \mapsto \hat{v} \mid \hat{v} = \begin{cases} \hat{s}_1(\hat{a}) & \text{if } \hat{a} \in \text{dom}(\hat{s}_1) \setminus \text{dom}(\hat{s}_2) \\ \hat{s}_2(\hat{a}) & \text{if } \hat{a} \in \text{dom}(\hat{s}_2) \setminus \text{dom}(\hat{s}_1) \\ \hat{s}_1(\hat{a}) \sqcup_{V\#} \hat{s}_2(\hat{a}) & \text{if } \hat{a} \in \text{dom}(\hat{s}_1) \cap \text{dom}(\hat{s}_2) \end{cases} \}$$

Least upper bound operator (2/2)

- Exceptions' functions:

$$\widehat{ex}_1 \sqcup_{EX\#} \widehat{ex}_2 = \{ \widehat{ex} : n \mapsto \widehat{st} \mid$$

$$\widehat{st} = \left\{ \begin{array}{ll} \widehat{ex}_1(n) & \text{if } n \in \text{dom}(\widehat{ex}_1) \setminus \text{dom}(\widehat{ex}_2) \\ \widehat{ex}_2(n) & \text{if } n \in \text{dom}(\widehat{ex}_2) \setminus \text{dom}(\widehat{ex}_1) \\ \widehat{ex}_1(n) \sqcup_{ST\#} \widehat{ex}_2(n) & \text{if } n \in \text{dom}(\widehat{ex}_1) \cap \text{dom}(\widehat{ex}_2) \end{array} \right\}$$

- States:

$$\widehat{st}_1 \sqcup_{ST\#} \widehat{st}_2 = (\widehat{e}_1 \sqcup_{E\#} \widehat{e}_2, \widehat{s}_1 \sqcup_{S\#} \widehat{s}_2, \widehat{ex}_1 \sqcup_{EX\#} \widehat{ex}_2)$$

Greatest lower bound operator

- Environments:

$$\hat{e}_1 \sqcap_{E\#} \hat{e}_2 = \{\hat{e} : n \mapsto \hat{a} \mid \hat{a} = \hat{e}_1(n) \cap \hat{e}_2(n) \text{ if } n \in \text{dom}(\hat{e}_1) \cap \text{dom}(\hat{e}_2)\}$$

- Stores:

$$\hat{s}_1 \sqcap_{S\#} \hat{s}_2 = \{\hat{s} : \hat{a} \mapsto \hat{v} \mid \hat{v} = \hat{s}_1(\hat{a}) \sqcap_{V\#} \hat{s}_2(\hat{a}) \text{ if } \hat{a} \in \text{dom}(\hat{s}_1) \cap \text{dom}(\hat{s}_2)\}$$

- Exceptions' functions:

$$\hat{e}x_1 \sqcap_{EX\#} \hat{e}x_2 = \{\hat{e}x : n \mapsto \hat{st} \mid \hat{st} = \begin{array}{l} \hat{e}x_1(n) \sqcap_{ST\#} \hat{e}x_2(n) \\ \text{if } n \in \text{dom}(\hat{e}x_1) \cap \text{dom}(\hat{e}x_2) \end{array}\}$$

- States:

$$\hat{st}_1 \sqcap_{ST\#} \hat{st}_2 = (\hat{e}_1 \sqcap_{E\#} \hat{e}_2, \hat{s}_1 \sqcap_{S\#} \hat{s}_2, \hat{e}x_1 \sqcap_{EX\#} \hat{e}x_2)$$

Widening

- Environments: $\hat{e}_1 \nabla_{E\#} \hat{e}_2 = \hat{e}_1 \sqcup_{E\#} \hat{e}_2$
- Stores:

$$\hat{s}_1 \nabla_{S\#} \hat{s}_2 = \{ \hat{s} : \hat{a} \mapsto \hat{v} \mid \hat{v} = \begin{cases} \hat{s}_1(\hat{a}) & \text{if } \hat{a} \in \text{dom}(\hat{s}_1) \setminus \text{dom}(\hat{s}_2) \\ \hat{s}_2(\hat{a}) & \text{if } \hat{a} \in \text{dom}(\hat{s}_2) \setminus \text{dom}(\hat{s}_1) \\ \hat{s}_1(\hat{a}) \nabla_{V\#} \hat{s}_2(\hat{a}) & \text{if } \hat{a} \in \text{dom}(\hat{s}_1) \cap \text{dom}(\hat{s}_2) \end{cases} \}$$

- Exceptions' functions:

$$\hat{e}x_1 \nabla_{EX\#} \hat{e}x_2 = \{ \hat{e}x : n \mapsto \hat{st} \mid \hat{st} = \begin{cases} \hat{e}x_1(n) & \text{if } n \in \text{dom}(\hat{e}x_1) \setminus \text{dom}(\hat{e}x_2) \\ \hat{e}x_2(n) & \text{if } n \in \text{dom}(\hat{e}x_2) \setminus \text{dom}(\hat{e}x_1) \\ \hat{e}x_1(n) \nabla_{ST\#} \hat{e}x_2(n) & \text{if } n \in \text{dom}(\hat{e}x_1) \cap \text{dom}(\hat{e}x_2) \end{cases} \}$$

- States: $\hat{st}_1 \nabla_{ST\#} \hat{st}_2 = (\hat{e}_1 \nabla_{E\#} \hat{e}_2, \hat{s}_1 \nabla_{S\#} \hat{s}_2, \hat{e}x_1 \nabla_{EX\#} \hat{e}x_2)$

Outline

- 1 Concrete and abstract domains
- 2 JAIL and its instances**
- 3 Conclusions

Abstract domains for numerical values

- We approximate integer values
- So we obtain a more precise analysis
- Our analysis is parametrized on these abstract domains
- The abstract operators are based on the operators of these domains
- Thus we suppose that:
 - $\langle PV^\#, \leq_{PV^\#} \rangle$ is a partially ordered set
 - \sqcup_{PV} is a least upper bound operator
 - \sqcap_{PV} is a greatest lower bound operator
 - ∇_{PV} is a widening operator

Analysed property

In order to analyse the applet isolation property we have defined an abstract firewall. It is activated when:

- the program accedes to a field of an object
- the program calls a method of an object

The firewall analyses the current context and what we are acceding, and so it can eventually throw in the abstract analysis an exception.

Applet isolation property

For instance: during the analysis of the applet isolation property, the source code calls a method of an object. The firewall checks:

- If the package to whom belongs the program that invokes the method is the same of the one of the object accessed. If the 2 packages are different
 - it checks if the invoked method is declared shareable. If it is not declared shareable
 - it does not invoke the method and it throws a `SecurityException`.

Class properties

- We base our analysis on the work of Francesco Logozzo
 - "Modular static analysis of object-oriented languages",
15/06/2004, Ecole Polytechnique, Paris
- Concrete semantics of a class: the set of all possible states of the objects that can be instantiated from the given class.
- A sound approximation of this concrete semantics is given by the solution of the following equations' system.

$$s_0 = \bigsqcup_{init \in Constructors} M[[init]](\emptyset, \top_{init})$$

$$s_i = s_{i-1} \sqcup_{ST} \bigsqcup_{me \in Methods} M[[me]](s_{i-1}, \top_{me})$$

where:

- $M[[me]]$ represents the semantics of the method me , given an initial state and the values of the parameters
- $Constructors$ is the set of constructors of the analysed class
- $Methods$ is the set of methods of the analysed class
- \top_{me} is the list of the parameters of method me , each associated with value \top

Analysing the example (1/2)

```
package smartcard_reader;
public interface User extends Shareable {
    public void use();
}
package smartcard_reader;
public interface Administrator extends User {
    public void manage();
}
package smartcard_reader;
public class AppletUser extends Applet implements User {
    public void use(){...}
    public void manage(){...}
}
package smartcard_reader;
public class AppletAdministrator extends AppletUser implements
Administrator {
    public void use(){...}
    public void manage(){...}
}
```

Analysing the example(2/2)

```
1: package smartcard;
2: public class AppletService extends Applet {
3:     AppletUser app=null;
4:     public AppletService(int code) throws Exception {
5:         if(code==123)
6:             app=(AppletAdministrator)
JCSYSTEM.getShareableInterfaceObject(0);

7:         else app=(AppletUser) JCSYSTEM.getShareableInterfaceObject(1);

8:     }
9:     public void use(int i) {
11:         if(i==0) app.manage();
12:         else app.use();
13:     }
}
```

Analysing the example(2/2)

```
1: package smartcard;
2: public class AppletService extends Applet {
3:   AppletUser app=null;
4:   public AppletService(int code) throws Exception {
5:     if(code==123)
6:       app=(AppletAdministrator)
7:         JCSystem.getShareableInterfaceObject(0);
8:     else app=(AppletUser) JCSystem.getShareableInterfaceObject(1);
9:   }
10:  }
11:  public void use(int i) {
12:    if(i==0) app.manage();
13:    else app.use();
14:  }
15: }
```

$\hat{e} = \{app \mapsto (6, 0, 0, \emptyset)\}, \hat{s} = \{(6, 0, 0, \emptyset) \mapsto \langle \text{AppletAdministrator} \rangle\}$

Analysing the example(2/2)

```
1: package smartcard;
2: public class AppletService extends Applet {
3:   AppletUser app=null;
4:   public AppletService(int code) throws Exception {
5:     if(code==123)
6:       app=(AppletAdministrator)
7:         JCSystem.getShareableInterfaceObject(0);
8:     else app=(AppletUser) JCSystem.getShareableInterfaceObject(1);
9:   }
10: }
11: }
12: }
13: }
```

$\hat{e} = \{app \mapsto (6, 0, 0, \emptyset)\}, \hat{s} = \{(6, 0, 0, \emptyset) \mapsto \langle \text{AppletAdministrator} \rangle\}$

$\hat{e} = \{app \mapsto (7, 0, 0, \emptyset)\}, \hat{s} = \{(7, 0, 0, \emptyset) \mapsto \langle \text{AppletUser} \rangle\}$

Analysing the example(2/2)

```

1: package smartcard;
2: public class AppletService extends Applet {
3:   AppletUser app=null;
4:   public AppletService(int code) throws Exception {
5:     if(code==123)
6:       app=(AppletAdministrator)
JCSYSTEM.getShareableInterfaceObject(0);

7:     else app=(AppletUser) JCSYSTEM.getShareableInterfaceObject(1);

```

$$\hat{e} = \{app \mapsto \{(6, 0, 0, \emptyset), (7, 0, 0, \emptyset)\}\}$$

$$\hat{s} = \{(6, 0, 0, \emptyset) \mapsto \langle \text{AppletAdministrator} \rangle, (7, 0, 0, \emptyset) \mapsto \langle \text{AppletUser} \rangle\}$$

```

8:   }
9:   public void use(int i) {
11:     if(i==0) app.manage();
12:     else app.use();
13:   }
}

```

Analysing the example(2/2)

```
1: package smartcard;
2: public class AppletService extends Applet {
  (...)

9: public void use(int i) {
11:     if(i==0) app.manage();

12:     else app.use();
13: }
}
```

Analysing the example(2/2)

```
1: package smartcard;
2: public class AppletService extends Applet {
  (...)
 $\hat{e} = \{app \mapsto \{(6, 0, 0, \emptyset), (7, 0, 0, \emptyset)\}\}$ 
 $\hat{s} = \{(6, 0, 0, \emptyset) \mapsto \langle AppletAdministrator \rangle, (7, 0, 0, \emptyset) \mapsto \langle AppletUser \rangle\}$ 
9:   public void use(int i) {
11:       if(i==0) app.manage();

12:   else app.use();
13:   }
}
```

Analysing the example(2/2)

```

1: package smartcard;
2: public class AppletService extends Applet {
  (...)
 $\hat{e} = \{app \mapsto \{(6, 0, 0, \emptyset), (7, 0, 0, \emptyset)\}\}$ 
 $\hat{s} = \{(6, 0, 0, \emptyset) \mapsto \langle \text{AppletAdministrator} \rangle, (7, 0, 0, \emptyset) \mapsto \langle \text{AppletUser} \rangle\}$ 
9:   public void use(int i) {
11:     if(i==0) app.manage();

```

Two cases:

- $app \mapsto (6, 0, 0, \emptyset) \rightarrow OK!$
- $app \mapsto (7, 0, 0, \emptyset) \rightarrow STOP!$

```

12:     else app.use();
13:   }
}

```

Analysing the example(2/2)

```

1: package smartcard;
2: public class AppletService extends Applet {
  (...)

```

$$\hat{e} = \{app \mapsto \{(6, 0, 0, \emptyset), (7, 0, 0, \emptyset)\}\}$$

$$\hat{s} = \{(6, 0, 0, \emptyset) \mapsto \langle \text{AppletAdministrator} \rangle, (7, 0, 0, \emptyset) \mapsto \langle \text{AppletUser} \rangle\}$$

```

9:   public void use(int i) {
11:       if(i==0) app.manage();

```

Two cases:

- $app \mapsto (6, 0, 0, \emptyset) \rightarrow OK!$
- $app \mapsto (7, 0, 0, \emptyset) \rightarrow STOP!$

$$\hat{e} = \{app \mapsto \{(6, 0, 0, \emptyset)\}\}$$

$$\hat{s} = \{(6, 0, 0, \emptyset) \mapsto \langle \text{AppletAdministrator} \rangle, (7, 0, 0, \emptyset) \mapsto \langle \text{AppletUser} \rangle\}$$

$$\hat{e}\hat{x} = \{\text{SecurityException} \mapsto \hat{st}'\}$$

$$\hat{st}' = (\{app \mapsto \{(7, 0, 0, \emptyset)\}\}, \hat{s}, \emptyset)$$

```

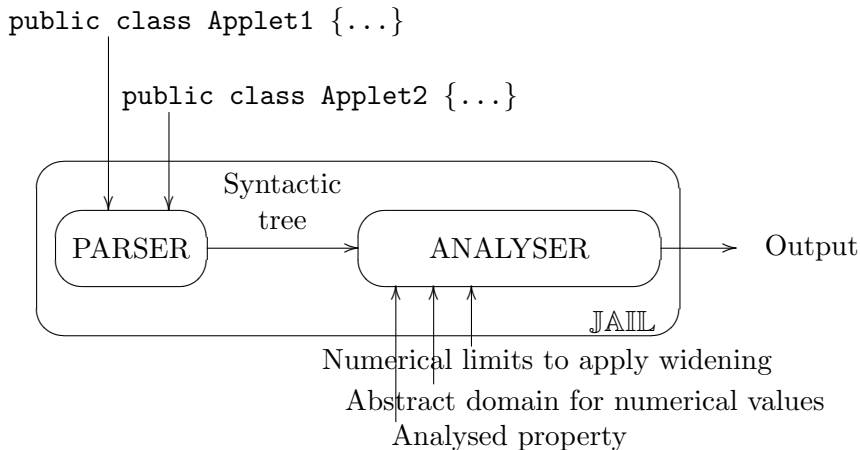
12:       else app.use();
13:   }
}

```

Outline

- 1 Concrete and abstract domains
- 2 JAIL and its instances
- 3 Conclusions**

Structure of the tool



Screenshot

The screenshot displays a software analysis tool interface. On the left, there is a small diagram of a grid with a highlighted section. The main configuration area includes:

- Abstract domain (int/double): Sign
- Abstract domain for objects: NullPointerException
- Loop limit: 20
- Recursion limit: 10
- Class invariant limit: 5
- Select an example: Airline

Below the configuration is a table with the following data:

Row	Column
11	1

The code editor shows the following Java code:

```
package smart_card;

public class Use extends Applet{
    private int user;
    private Service service;

    public Use(int i) {
        if(i>0)
            user=1;
        else user=0;
        service=JCSYSTEM.getAppletServiceInterfaceObject(Service);
        service.init(user);
    }

    public int use() {
        return service.read();
    }

    public int use(int i) {
        service.write(i);
        return 1;
    }
}

}

}

}
```

On the right side, a tree view shows the abstract results:

- Abstract result
 - service
 - Value returned by method use(int i)
 - Value returned by method use()
 - user

At the bottom of the code editor, there is a button labeled "Analyse the class".

Conclusion

- The analyser has been implemented and can be tested at url <http://www.dsi.unive.it/ferrara/jail>
- We have implemented some abstract numerical domains (like intervals, signs, ...) and the analysis of some properties (applet isolation, absence of NullPointerException)
- We applied succesfully JAIL to the analysis of some case studies (some of that are taken from others papers)

Future works

- Extension of the analyser to other properties, such the absence of nested transactions, property defined by JavaCard Runtime Environment
- Development of the complete semantics of Java Card library

The end...

That's all!