Verifying JAVA CARD Programs with KeY

Wojciech Mostowski
http://www.cs.ru.nl/~woj/

Radboud University Nijmegen

Overview

Goals

- Explain only really the necessary things, if hungry for more:
  - Reiner Hähnle's tutorial from COST Winter School 2009: http://www.key-project.org/costws09/
  - The KeY book

Overview

Goals

- Explain only really the necessary things, if hungry for more:
  - Reiner Hähnle's tutorial from COST Winter School 2009: http://www.key-project.org/costws09/
  - The KeY book
- Get your hands dirty with some verification work!

Overview

Goals

- Explain only really the necessary things, if hungry for more:
  - Reiner Hähnle's tutorial from COST Winter School 2009: http://www.key-project.org/costws09/
  - The KeY book
- Get your hands dirty with some verification work!

Outline

- What is KeY?
- What is JAVA CARD?
- How does KeY deal with JAVA CARD?
- Tasks for the rest of the session

The KeY System

- First Order Java Dynamic Logic prover
  - $< p > \varphi$ – total correctness
  - $[\, p \,] \varphi$ – partial correctness
  - ($[\![p]\!] \varphi$ – strong invariant correctness)
  - Hoare triple in DL is $pre \rightarrow [\, p \,] post$

The KeY System

- First Order Java Dynamic Logic prover
  - $< p > \varphi$ – total correctness
  - $[\, p \,] \varphi$ – partial correctness
  - ($[\![p]\!] \varphi$ – strong invariant correctness)
  - Hoare triple in DL is $pre \rightarrow [\, p \,] post$
- Automated, interactive if necessary
- Self-contained prover, but can use external SMT solvers...

The KeY System

- First Order Java Dynamic Logic prover
  - $< p > \varphi$ – total correctness
  - $[\, p \,]\ \varphi$ – partial correctness
  - $([\![p]\!]\ \varphi$ – strong invariant correctness)
  - Hoare triple in DL is $pre \rightarrow [\, p \,]\ post$
- Automated, interactive if necessary
- Self-contained prover, but can use external SMT solvers. . .

  . . . but not today 
- Dynamic and customisable rule base – taclets
- Rules can be fine tuned with taclet options
- Proofs can be guided by fiddling with prover options

The KeY System

- First Order Java Dynamic Logic prover
  - $< p > \varphi$ – total correctness
  - $[\, p \,]\ \varphi$ – partial correctness
  - ($[\![p]\!]\ \varphi$ – strong invariant correctness)
  - Hoare triple in DL is $pre \rightarrow [\, p \,]\ post$
- Automated, interactive if necessary
- Self-contained prover, but can use external SMT solvers. . .

  . . . but not today 
- Dynamic and customisable rule base – taclets
- Rules can be fine tuned with taclet options
- Proofs can be guided by fiddling with prover options
- Verification paradigm: symbolic execution with state updates (implicit heap representation)

Symbolic Execution

```
\programVariables { int a, a@pre, b; }

\problem {

  b > 0 ->
  { a@pre := a }
    \<{ if(a == b) { a += 2; } }\>
  ( (a@pre != b & a = a@pre) | a = b + 2 )

}
```

Demo

example1.key

KeY Specification Languages

- The core of KeY is Java Dynamic Logic
- DL Proof Obligations (PO-s) can be generated from program contracts written in:
  - Object Constraint Language (OCL) – legacy
  - (Subset of) JML
  - Dynamic Logic itself
- The last option is most attractive:
  - (Almost) no translation overhead
  - JML semantics moving target, no need to worry about it
  - KeY semantics is different anyhow
  - Direct access to the logic enables more precise specifications
- KeY DL is easy!

Program Specifications – JML vs. KeY DL

JML

```
class MyClass {
  int i; //@invariant i > 0;

  //@requires o != null;
  //@ensures
      this.i == \old(o.i);
  //@assignable this.i;
  void setFrom(MyClass o) {
    this.i = o.i;
  }
}
```

Program Specifications – JML vs. KeY DL

JML

```
class MyClass {
  int i; //@invariant i > 0;

  //@requires o != null;
  //@ensures
      this.i == \old(o.i);
  //@assignable this.i;
  void setFrom(MyClass o) {
    this.i = o.i;
  }
}
```

KeY DL

```
MyClass_setFrom {
  \programVariables {
      MyClass self, o; }
  o != null -> \<{
    self.setFrom(o)@MyClass;
  }\> (self.i = o@pre.i)

  \modifies { self.i }
};

\invariants (MyClass self) {
  MyClass_inv { self.i > 0 };
}
```

Program Specifications – JML vs. KeY DL

JML

```
class MyClass {
  int i; //@invariant i > 0;

  //@requires o != null;
  //@ensures
      this.i == \old(o.i);
  //@assignable this.i;
  void setFrom(MyClass o) {
    this.i = o.i;
  }
}
```

KeY DL

```
MyClass_setFrom {
  \programVariables {
      MyClass self, o; }
  o != null -> \<{
    self.setFrom(o)@MyClass;
  }\> (self.i = o@pre.i)

  \modifies { self.i }
};

\invariants (MyClass self) {
  MyClass_inv { self.i > 0 };
}
```

Program Specifications

Exceptions

```
MyClass_setFrom {
  \programVariables { MyClass self, o; }

  true -> \<{
    #catchAll(java.lang.Exception e) {
      self.setFrom(o)@MyClass;
    }
  }\>(( e = null -> self.i = o@pre.i )
    & ( e != null ->
      java.lang.NullPointerException::instance(e) = TRUE ) )

  \modifies { self.i }
};
```

Class Invariants in KeY

- JML adopts visible state semantics of invariants,
- KeY adopts observable state semantics,

Class Invariants in KeY

- JML adopts visible state semantics of invariants,
- KeY adopts observable state semantics,
- For KeY soundness you need to prove that:
  - `EnsuresPost`
    Each method ensures its postcondition.
    You are free to choose which invariants you need for that,
  - `RespectsModifies`
    Each method respects its modifies clause.
    Same w.r.t. invariant use,
  - `PreservesInv`
    On a per-contract application base, each method preserves invariants that were chosen during that contract application.

Class Invariants in KeY

- JML adopts visible state semantics of invariants,
- KeY adopts observable state semantics,
- For KeY soundness you need to prove that:
  - `EnsuresPost`
    Each method ensures its postcondition.
    You are free to choose which invariants you need for that,
  - `RespectsModifies`
    Each method respects its modifies clause.
    Same w.r.t. invariant use,
  - `PreservesInv`
    On a per-contract application base, each method preserves invariants that were chosen during that contract application.
- KeY will tell you what you need to prove to make it happy,
- Best explained with `example2.key`.

Example 2 – Code

```
public class MyClass {
  int i, j;

  void setIFrom(/*@ nullable @*/ MyClass other) {
    this.i = other.i;
  }

  void setJFrom(/*@ nullable @*/ MyClass other) {
    this.j = other.j;
  }

  void setFrom(/*@ nullable @*/ MyClass other) {
    setIFrom(other);
    setJFrom(other);
  }
}
```

Example 2 – Specification

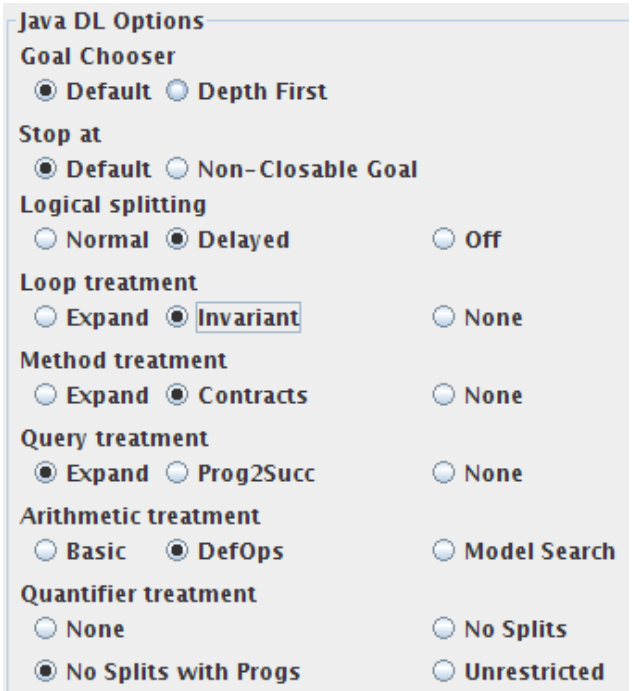
```
MyClass_setIFrom { \programVariables { MyClass self, other; }
  other != null -> \<{ self.setIFrom(other)@MyClass; }\>
    (self.i = other@pre.i)
  \modifies { self.i } };

MyClass_setJFrom { \programVariables { MyClass self, other; }
  other != null -> \<{ self.setJFrom(other)@MyClass; }\>
    (self.j = other@pre.j)
  \modifies { self.j } };

MyClass_setFrom { \programVariables { MyClass self, other; }
  other != null -> \<{ self.setFrom(other)@MyClass; }\>
    (self.i = other@pre.i & self.j = other@pre.j)
  \modifies { self.i, self.j } };

\invariants (MyClass self) {
  inv_i { self.i >= 0 };
  inv_j { self.j >= 0 };
}
```

Let's Play!

- Preferred JDK is Sun/Oracle 1.6
- Install KeY 1.6
  `http://www.key-project.org/download/`
  - Permanent Installation
  - Java Webstart
- Switch off:
  - KiKi The Proof Assistant
  - Sound
- Load and prove `example1.key`
  - Change integer number semantics, try again & fix
- Play around with `example2.key`

Let's Play!

- Preferred JDK is Sun/Oracle 1.6
- Install KeY 1.6
  `http://www.key-project.org/download/`
  - Permanent Installation
  - Java Webstart
- Switch off:
  - KiKi The Proof Assistant
  - Sound
- Load and prove `example1.key`
  - Change integer number semantics, try again & fix
- Play around with `example2.key`



JAVA CARD

- A smart card running a cut-down JAVA VM
- Smart card specific cut-down JAVA API:
  - Smart card specific functionality
  - Cryptographic libraries
- No "disk" storage, only two types of memory:
  - RAM – session memory, fast, 0.5–4K
  - E^2PROM – storage memory, slow (write), 16–72K
  - Memory destination of a variable/field is determined by the declaration place and/or allocation method

JAVA CARD and Verification

Why important?

- Smart card applications highly security sensitive
- Roll-out and replacement expensive and time consuming:
  - Really need to be bug free
  - Patches and fixes are out of question

JAVA CARD and Verification

Why important?

- Smart card applications highly security sensitive
- Roll-out and replacement expensive and time consuming:
  - Really need to be bug free
  - Patches and fixes are out of question

Why we like JAVA CARD?

- Much smaller, smart card tailored API, and **NO**:

JAVA CARD and Verification

Why important?

- Smart card applications highly security sensitive
- Roll-out and replacement expensive and time consuming:
  - Really need to be bug free
  - Patches and fixes are out of question

Why we like JAVA CARD?

- Much smaller, smart card tailored API, and **NO**:
  (1) Concurrency

JAVA CARD and Verification

Why important?

- Smart card applications highly security sensitive
- Roll-out and replacement expensive and time consuming:
  - Really need to be bug free
  - Patches and fixes are out of question

Why we like JAVA CARD?

- Much smaller, smart card tailored API, and **NO**:
  (1) Concurrency (2) Garbage collection

JAVA CARD and Verification

Why important?

- Smart card applications highly security sensitive
- Roll-out and replacement expensive and time consuming:
  - Really need to be bug free
  - Patches and fixes are out of question

Why we like JAVA CARD?

- Much smaller, smart card tailored API, and **NO**:
  (1) Concurrency (2) Garbage collection (3) GUI

JAVA CARD and Verification

Why important?

- Smart card applications highly security sensitive
- Roll-out and replacement expensive and time consuming:
  - Really need to be bug free
  - Patches and fixes are out of question

Why we like JAVA CARD?

- Much smaller, smart card tailored API, and **NO**:
  (1) Concurrency (2) Garbage collection (3) GUI
  (4) Strings

JAVA CARD and Verification

Why important?

- Smart card applications highly security sensitive
- Roll-out and replacement expensive and time consuming:
  - Really need to be bug free
  - Patches and fixes are out of question

Why we like JAVA CARD?

- Much smaller, smart card tailored API, and **NO**:
  (1) Concurrency (2) Garbage collection (3) GUI
  (4) Strings (5) Large integers

JAVA CARD and Verification

Why important?

- Smart card applications highly security sensitive
- Roll-out and replacement expensive and time consuming:
  - Really need to be bug free
  - Patches and fixes are out of question

Why we like JAVA CARD?

- Much smaller, smart card tailored API, and **NO**:
  (1) Concurrency (2) Garbage collection (3) GUI
  (4) Strings (5) Large integers (6) Floats

JAVA CARD and Verification

Why important?

- Smart card applications highly security sensitive
- Roll-out and replacement expensive and time consuming:
  - Really need to be bug free
  - Patches and fixes are out of question

Why we like JAVA CARD?

- Much smaller, smart card tailored API, and **NO**:
  (1) Concurrency (2) Garbage collection (3) GUI
  (4) Strings (5) Large integers (6) Floats (7) I/O

JAVA CARD and Verification

Why important?

- Smart card applications highly security sensitive
- Roll-out and replacement expensive and time consuming:
  - Really need to be bug free
  - Patches and fixes are out of question

Why we like JAVA CARD?

- Much smaller, smart card tailored API, and **NO**:
  (1) Concurrency (2) Garbage collection (3) GUI
  (4) Strings (5) Large integers (6) Floats (7) I/O
  (8) External storage

JAVA CARD and Verification

Why important?

- Smart card applications highly security sensitive
- Roll-out and replacement expensive and time consuming:
  - Really need to be bug free
  - Patches and fixes are out of question

Why we like JAVA CARD?

- Much smaller, smart card tailored API, and **NO**:
  (1) Concurrency (2) Garbage collection (3) GUI
  (4) Strings (5) Large integers (6) Floats (7) I/O
  (8) External storage (9) Dynamic class loading

JAVA CARD and Verification

Why important?

- Smart card applications highly security sensitive
- Roll-out and replacement expensive and time consuming:
  - Really need to be bug free
  - Patches and fixes are out of question

Why we like JAVA CARD?

- Much smaller, smart card tailored API, and **NO**:
  (1) Concurrency (2) Garbage collection (3) GUI
  (4) Strings (5) Large integers (6) Floats (7) I/O
  (8) External storage (9) Dynamic class loading
- Not much left really, seems to be an easy verification target for most tools...

JAVA CARD Verification Difficulties

Smart card infrastructure

- Two kinds of memory (RAM & E^2PROM), interplaying with

JAVA CARD Verification Difficulties

Smart card infrastructure

- Two kinds of memory (RAM & E^2PROM), interplaying with
- The transaction mechanism to support safe atomic updates:
  - Really nasty roll-back semantics
  - Built-in the JAVA CARD VM

JAVA CARD Verification Difficulties

Smart card infrastructure

- Two kinds of memory (RAM & E^2PROM), interplaying with
- The transaction mechanism to support safe atomic updates:
  - Really nasty roll-back semantics
  - Built-in the JAVA CARD VM
- Cryptography libraries

JAVA CARD Verification Difficulties

Smart card infrastructure

- Two kinds of memory (RAM & E^2PROM), interplaying with
- The transaction mechanism to support safe atomic updates:
  - Really nasty roll-back semantics
  - Built-in the JAVA CARD VM
- Cryptography libraries
- Data is mostly byte arrays – reference aliasing issues

JAVA CARD Verification Difficulties

Smart card infrastructure

- Two kinds of memory (RAM & E^2PROM), interplaying with
- The transaction mechanism to support safe atomic updates:
  - Really nasty roll-back semantics
  - Built-in the JAVA CARD VM
- Cryptography libraries
- Data is mostly byte arrays – reference aliasing issues
- Some API calls have very deep semantics:
  - Change the internal behaviour of the JAVA CARD VM
  - Hardly possible to model with a pre/-post contract
  - Most notably transaction mechanism calls

JAVA CARD Verification Difficulties

Applications

- Need to follow a few design and coding principles:
  - What is in RAM, what is in E^2PROM (session vs. stored data)
  - No (re-)allocation of memory after initialisation (prevent DOS)
  - Small code and memory footprint (limited resources)

JAVA CARD Verification Difficulties

Applications

- Need to follow a few design and coding principles:
  - What is in RAM, what is in E^2PROM (session vs. stored data)
  - No (re-)allocation of memory after initialisation (prevent DOS)
  - Small code and memory footprint (limited resources)
- Applications can to be very complex at the same time:
  - Domains heavy on cryptography
  - The new JAVA CARD 3.0 technology – smart cards will run embedded web-servers and similar applications...

JAVA CARD Example

```
public class CountersApplet extends Applet implements ISO7816 {

  private final static byte GET_COUNTS = 0x10;

  private short selectCounter = 0;

  private short[] commandCounter = JCSystem.makeTransientShortArray(
        (short)1, JCSystem.CLEAR_ON_RESET);

  protected CountersApplet() {
    register();
  }

  public static void install(byte[] data, short off, short len) {
    new CountersApplet();
  }
```

JAVA CARD Example

```
public class CountersApplet extends Applet implements ISO7816 {

  private final static byte GET_COUNTS = 0x10;

  private short selectCounter = 0;

  private short[] commandCounter = JCSystem.makeTransientShortArray(
        (short)1, JCSystem.CLEAR_ON_RESET);

  protected CountersApplet() {
    register();
  }

  public static void install(byte[] data, short off, short len) {
    new CountersApplet();
  }
```

JAVA CARD Example

```
public class CountersApplet extends Applet implements ISO7816 {

  private final static byte GET_COUNTS = 0x10;

  private short selectCounter = 0; // Storage

  private short[] commandCounter = JCSystem.makeTransientShortArray(
        (short)1, JCSystem.CLEAR_ON_RESET);

  protected CountersApplet() {
    register();
  }

  public static void install(byte[] data, short off, short len) {
    new CountersApplet();
  }
```

JAVA CARD Example

```
public class CountersApplet extends Applet implements ISO7816 {

  private final static byte GET_COUNTS = 0x10;

  private short selectCounter = 0; // Storage

  private short[] commandCounter = JCSystem.makeTransientShortArray(
        (short)1, JCSystem.CLEAR_ON_RESET); // Session

  protected CountersApplet() {
    register();
  }

  public static void install(byte[] data, short off, short len) {
    new CountersApplet();
  }
```

JAVA CARD Example

```
public void process(APDU apdu) {
  if(selectingApplet()) { selectCounter++; return; }
  commandCounter[0]++;
  byte[] buffer = apdu.getBuffer();
  switch(buffer[OFFSET_INS]) {
    case GET_COUNTS:
      Util.setShort(buffer, (short)0, selectCounter);
      Util.setShort(buffer, (short)2, commandCounter[0]);
      apdu.setOutgoingAndSend((short)0, (short)4);
      return;
    default:
      ISOException.throwIt(SW_INS_NOT_SUPPORTED);
  }
}
```

JAVA CARD Example

```
public void process(APDU apdu) {
  if(selectingApplet()) { selectCounter++; return; }
  commandCounter[0]++;
  byte[] buffer = apdu.getBuffer();
  switch(buffer[OFFSET_INS]) {
    case GET_COUNTS:
      Util.setShort(buffer, (short)0, selectCounter);
      Util.setShort(buffer, (short)2, commandCounter[0]);
      apdu.setOutgoingAndSend((short)0, (short)4);
      return;
    default:
      ISOException.throwIt(SW_INS_NOT_SUPPORTED);
  }
}
```

JAVA CARD Example

```
public void process(APDU apdu) {
  if(selectingApplet()) { selectCounter++; return; }
  commandCounter[0]++;
  byte[] buffer = apdu.getBuffer();
  switch(buffer[OFFSET_INS]) {
    case GET_COUNTS:
      Util.setShort(buffer, (short)0, selectCounter);
      Util.setShort(buffer, (short)2, commandCounter[0]);
      apdu.setOutgoingAndSend((short)0, (short)4);
      return;
    default:
      ISOException.throwIt(SW_INS_NOT_SUPPORTED);
  }
}
```

JAVA CARD Example

```
public void process(APDU apdu) {
  if(selectingApplet()) { selectCounter++; return; }
  commandCounter[0]++;
  byte[] buffer = apdu.getBuffer();
  switch(buffer[OFFSET_INS]) {
    case GET_COUNTS:
      Util.setShort(buffer, (short)0, selectCounter);
      Util.setShort(buffer, (short)2, commandCounter[0]);
      apdu.setOutgoingAndSend((short)0, (short)4);
      return;
    default:
      ISOException.throwIt(SW_INS_NOT_SUPPORTED);
  }
}
```

JAVA CARD Example

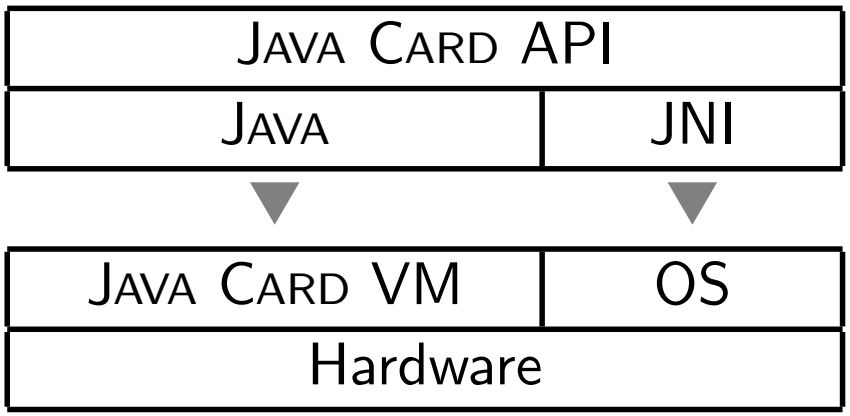
```
public void process(APDU apdu) {
  if(selectingApplet()) { selectCounter++; return; }
  commandCounter[0]++;
  byte[] buffer = apdu.getBuffer();
  switch(buffer[OFFSET_INS]) {
    case GET_COUNTS:
      Util.setShort(buffer, (short)0, selectCounter);
      Util.setShort(buffer, (short)2, commandCounter[0]);
      apdu.setOutgoingAndSend((short)0, (short)4);
      return;
    default:
      ISOException.throwIt(SW_INS_NOT_SUPPORTED);
  }
}
```

KeY and JAVA CARD

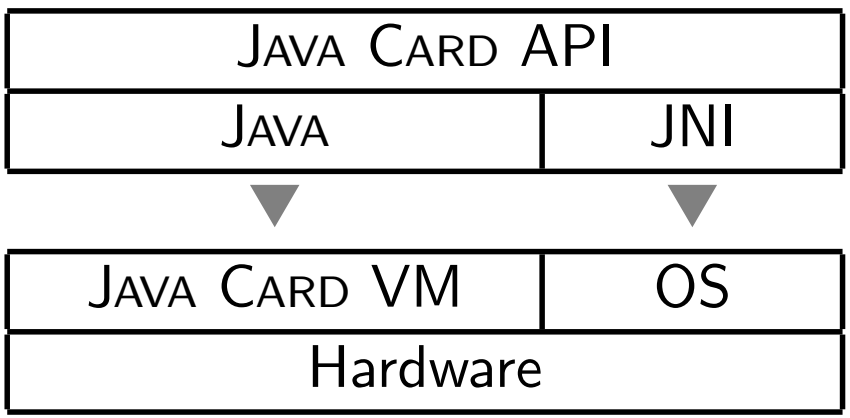
KeY's JAVA CARD VM Model

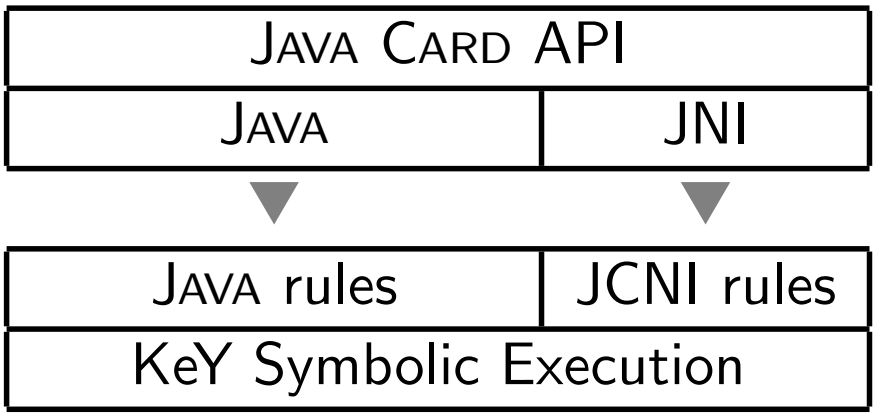
- Different kinds of memory:
  - Implicit field `<transient>` for objects (like `length` for arrays)
- Semantics of transactions, rather heavyweight inside KeY
- Relies on dedicated (built-in) API methods and tailored logic rules, rather than contracts
  I.e. built-in contracts for `KeYJCSystem.jvm*`
- Exactly as real API relies on the underlying OS through JNI

KeY and JAVA CARD



KeY and JAVA CARD





KeY and JAVA CARD

In Practice

- KeY JAVA CARD mode – taclet options
- For modular verification best use DL specified contracts
- Be aware that some API methods have a built-in meaning
  No contracts provided

What's on the Menu?

JAVA CARD API

- Specified reference implementation
  (Crypto part currently undergoes refactoring)
- DL method contracts & class invariants

What's on the Menu?

JAVA CARD API

- Specified reference implementation
  (Crypto part currently undergoes refactoring)
- DL method contracts & class invariants
- In-code JML annotations:
  - Loop invariants for automation
  - To get the default (**non_**)**null**-ness right

What's on the Menu?

JAVA CARD API

- Specified reference implementation
  (Crypto part currently undergoes refactoring)
- DL method contracts & class invariants
- In-code JML annotations:
  - Loop invariants for automation
  - To get the default (**non_**)**null**-ness right

Electronic Driving License JAVA CARD Implementation

- Proper JAVA CARD code (actually deployed in NL)
- Only partly specified

KeY Setup for Today

- All JAVA CARD features on
- Overflow control on
- Taclet options:

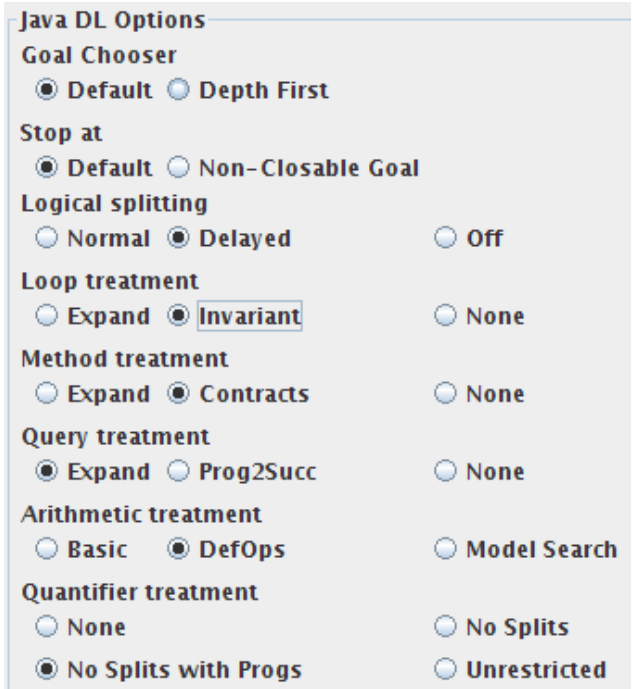
```
intRules:
  arithmeticSemanticsCheckingOF
javacard:jcOn
transactions:transactionsOn
transactionAbort:abortOn
```

- KeY project files
  http://limerick.cost-ic0701.org/home/verifying-java-card-programs-with-key

KeY Setup for Today

- All JAVA CARD features on
- Overflow control on
- Taclet options:

```
intRules:
  arithmeticSemanticsCheckingOF
javacard:jcOn
transactions:transactionsOn
transactionAbort:abortOn
```



- KeY project files
  http://limerick.cost-ic0701.org/home/verifying-java-card-programs-with-key

Possible Tasks for Today

- Unpack & browse the project files in `javacard_key.zip`

Possible Tasks for Today

- Unpack & browse the project files in `javacard_key.zip`
- Verify some of the JAVA CARD API implementation
  - Something easy (just click & enjoy KeY):
    `Util.arrayCopy`, `Util.arrayCompare`, `Util.*`
  - More difficult (small interactions & proof guiding):
    `APDU.setOutgoingAndSend`
    Ask me for Proof Strategy Settings
  - Even more difficult (careful invariant choice):
    `JCSystem.lookupAID`
  - Load file `javacardapi.key` into KeY

Possible Tasks for Today

- Unpack & browse the project files in `javacard_key.zip`
- Verify some of the JAVA CARD API implementation
  - Something easy (just click & enjoy KeY):
    `Util.arrayCopy`, `Util.arrayCompare`, `Util.*`
  - More difficult (small interactions & proof guiding):
    `APDU.setOutgoingAndSend`
    Ask me for Proof Strategy Settings
  - Even more difficult (careful invariant choice):
    `JCSystem.lookupAID`
  - Load file `javacardapi.key` into KeY
- For experts: specify and verify `FileSystem.getFileIndex`
  - Consult with me first
  - Look for hints in `FileSystem.key`
  - Load file `edl.key` into KeY

Proof Search Strategy Settings

- Goal chooser: `Depth first` is for heavy problems (never helped me...)

Proof Search Strategy Settings

- Goal chooser: `Depth first` is for heavy problems (never helped me...)
- Logical splitting:
  - `Normal` Branch the proof "at will"
  - `Delayed` Analyse the program code first, only then branch

Proof Search Strategy Settings

- Goal chooser: `Depth first` is for heavy problems (never helped me...)
- Logical splitting:
  - `Normal` Branch the proof "at will"
  - `Delayed` Analyse the program code first, only then branch
- Arithmetic treatment:
  - `Basic` Sufficient for properties without "meaningful" arithmetic expressions
  - `DefOps` If above not sufficient use this one
  - `Model Search` If you feel this is necessary, then there is probably a bug in your code or specification 

Proof Search Strategy Settings

- Goal chooser: `Depth first` is for heavy problems (never helped me...)
- Logical splitting:
  - `Normal` Branch the proof "at will"
  - `Delayed` Analyse the program code first, only then branch
- Arithmetic treatment:
  - `Basic` Sufficient for properties without "meaningful" arithmetic expressions
  - `DefOps` If above not sufficient use this one
  - `Model Search` If you feel this is necessary, then there is probably a bug in your code or specification 
- Quantifier treatment: changes aggressiveness level for quantifier instantiation

Proof Search Strategy Settings

- Goal chooser: `Depth first` is for heavy problems (never helped me...)
- Logical splitting:
  - `Normal` Branch the proof "at will"
  - `Delayed` Analyse the program code first, only then branch
- Arithmetic treatment:
  - `Basic` Sufficient for properties without "meaningful" arithmetic expressions
  - `DefOps` If above not sufficient use this one
  - `Model Search` If you feel this is necessary, then there is probably a bug in your code or specification 
- Quantifier treatment: changes aggressiveness level for quantifier instantiation
- Leave the rest as suggested

Resources

- Tutorial page & material
  http://limerick.cost-ic0701.org/home/verifying-java-card-programs-with-key
- KeY download / webstart
  http://www.key-project.org/download/
- JAVA CARD API Documentation
  http://www.cs.ru.nl/~woj/javacardapi221/index.html
- ISO18013 Electronic Driving License project
  http://sourceforge.net/projects/isodl/
- More top quality material on KeY
  http://www.key-project.org/costws09/

APDUs

Application Protocol Data Units

- Half-duplex communication channel
- Essentially byte arrays

APDUs

Application Protocol Data Units

- Half-duplex communication channel
- Essentially byte arrays

Command APDU

CLA	INS	P1	P2	LC	0–255 LC bytes of data	LE

APDUs

Application Protocol Data Units

- Half-duplex communication channel
- Essentially byte arrays

Command APDU

CLA	INS	P1	P2	LC	0–255 LC bytes of data	LE

APDUs

Application Protocol Data Units

- Half-duplex communication channel
- Essentially byte arrays

Command APDU

CLA	INS	P1	P2	LC	0–255 LC bytes of data	LE

APDUs

Application Protocol Data Units

- Half-duplex communication channel
- Essentially byte arrays

Command APDU

CLA	INS	P1	P2	LC	0–255 LC bytes of data	LE

APDUs

Application Protocol Data Units

- Half-duplex communication channel
- Essentially byte arrays

Command APDU

CLA	INS	P1	P2	LC	0–255 LC bytes of data	LE

Response APDU

0–256 (LE) bytes of data	SW1	SW2

APDUs

Application Protocol Data Units

- Half-duplex communication channel
- Essentially byte arrays

Command APDU

CLA	INS	P1	P2	LC	0–255 LC bytes of data	LE

Response APDU

0–256 (LE) bytes of data	SW1	SW2

ISO18013 Implementation

Electronic Driving License Card

- Files, called Data Groups (DGs), with driver's data:
  - Name, Date of Birth, . . . , License Categories
  - Picture and signature, both JPEG(2000)
  - Biometric information, e.g. fingerprints
- Document Security Object: special DG with hashes of the data and issuer's digital signature
- DGs are read with APDUs:
  - `INS_SELECT_FILE`, instruction byte `0xA4`
  - `INS_READ_BINARY`, instruction byte `0xB0`
- Access to data subject to access control: BAP and EAP. . .

ISO18013 Implementation

Basic Access Protection (BAP)

- Protects all read outs from the card
- Secure messaging channel based on a static DES key (card#)
- APDUs: `GET_CHALLENGE` and `EXTERNAL_AUTHENTICATE`
- Communication that follows is encrypted and MAC-ed

ISO18013 Implementation

Basic Access Protection (BAP)

- Protects all read outs from the card
- Secure messaging channel based on a static DES key (card#)
- APDUs: `GET_CHALLENGE` and `EXTERNAL_AUTHENTICATE`
- Communication that follows is encrypted and MAC-ed

Extended Access Protection (EAP)

- Requires BAP, protects sensitive data, e.g. biometrics
- Proves the card authenticity to the terminal (Diffie-Hellman) and terminal access rights to the card (certificate verification)
- Several APDUs