ESC/Java 2

Extended Static Checking for Java

Joe Kiniry

University of Nijmegen

What is Extended Static Checking?

Take a nicely-constructed program that (perhaps) has a number of well-thought out assertions scattered throughout.

What is Extended Static Checking?

Take a nicely-constructed program that (perhaps) has a number of well-thought out assertions scattered throughout. Now, imagine that you had a tool capable of:

What is Extended Static Checking?

Take a nicely-constructed program that (perhaps) has a number of well-thought out assertions scattered throughout. Now, imagine that you had a tool capable of:

- **automatically *checking* that nearly all the assertions in the program are *always true***

What is Extended Static Checking?

Take a nicely-constructed program that (perhaps) has a number of well-thought out assertions scattered throughout. Now, imagine that you had a tool capable of:

- **automatically *checking* that nearly all the assertions in the program are *always true***
- **performing this analysis *statically*. That is, it works just like your compiler does and runs without any user or test input whatsoever.**

What is Extended Static Checking?

Take a nicely-constructed program that (perhaps) has a number of well-thought out assertions scattered throughout. Now, imagine that you had a tool capable of:

- **automatically *checking* that nearly all the assertions in the program are *always true***
- **performing this analysis *statically*. That is, it works just like your compiler does and runs without any user or test input whatsoever.**
- **reasoning about non-trivial properties of the system beyond, e.g., whether the code type-checks or not. In other words, an *extended* (beyond type-correctness) set of properties are checked.**

What is Extended Static Checking?

Take a nicely-constructed program that (perhaps) has a number of well-thought out assertions scattered throughout. Now, imagine that you had a tool capable of:

- **automatically *checking* that nearly all the assertions in the program are *always true***
- **performing this analysis *statically*. That is, it works just like your compiler does and runs without any user or test input whatsoever.**
- **reasoning about non-trivial properties of the system beyond, e.g., whether the code type-checks or not. In other words, an *extended* (beyond type-correctness) set of properties are checked.**

Such a tool is called an extended static checker.

What is Extended Static Checking?

Three main Extended Static Checkers have been developed.

What is Extended Static Checking?

Three main Extended Static Checkers have been developed.

- **The original research and tool were accomplished by Rustan Leino et. al at the DEC SRC. The two tools they developed were ESC/Modula-III and SRC ESC/Java (version 1).**

What is Extended Static Checking?

Three main Extended Static Checkers have been developed.

- **The original research and tool were accomplished by Rustan Leino et. al at the DEC SRC. The two tools they developed were ESC/Modula-III and SRC ESC/Java (version 1).**
- **ESC/Java originally used its own, JML-like annotation language.**

What is Extended Static Checking?

Three main Extended Static Checkers have been developed.

- **The original research and tool were accomplished by Rustan Leino et. al at the DEC SRC. The two tools they developed were ESC/Modula-III and SRC ESC/Java (version 1).**
- **ESC/Java originally used its own, JML-like annotation language.**
- **Last year, H.P. decided to Open Source SRC ESC/Java, so ESC/Java2 was born.**

What is Extended Static Checking?

Three main Extended Static Checkers have been developed.

- **The original research and tool were accomplished by Rustan Leino et. al at the DEC SRC. The two tools they developed were ESC/Modula-III and SRC ESC/Java (version 1).**
- **ESC/Java originally used its own, JML-like annotation language.**
- **Last year, H.P. decided to Open Source SRC ESC/Java, so ESC/Java2 was born.**
- **ESC/Java2 is the work of David Cok and Joe Kiniry (that's me!).**

What is Extended Static Checking?

Three main Extended Static Checkers have been developed.

- **The original research and tool were accomplished by Rustan Leino et. al at the DEC SRC. The two tools they developed were ESC/Modula-III and SRC ESC/Java (version 1).**
- **ESC/Java originally used its own, JML-like annotation language.**
- **Last year, H.P. decided to Open Source SRC ESC/Java, so ESC/Java2 was born.**
- **ESC/Java2 is the work of David Cok and Joe Kiniry (that's me!).**
- **ESC/Java2 is SRC ESC/Java++. It understands all of JML, checks more properties, it runs on more platforms, and is more robust.**

ESC/Java

- **In general, an Extended Static Checker tries to prove the correctness of specifications, at compile-time, fully automatically, but . . .**

ESC/Java

- **In general, an Extended Static Checker tries to prove the correctness of specifications, at compile-time, fully automatically, but . . .**
- **ESC/Java is *not sound* and is *not complete*. That is, it can sometimes reject a correct spec, and at other times accept an incorrect spec.**

ESC/Java

- **In general, an Extended Static Checker tries to prove the correctness of specifications, at compile-time, fully automatically, but . . .**
- **ESC/Java is *not sound* and is *not complete*. That is, it can sometimes reject a correct spec, and at other times accept an incorrect spec.**
- **However, an ESC/Java finds *lots* of (potential) bugs *quickly*, and**

ESC/Java

- In general, an Extended Static Checker tries to prove the correctness of specifications, at compile-time, fully automatically, but . . .
- ESC/Java is *not sound* and is *not complete*. That is, it can sometimes reject a correct spec, and at other times accept an incorrect spec.
- However, an ESC/Java finds *lots* of (potential) bugs *quickly*, and
- ESC/Java is good at proving the absence of runtime exceptions like (e.g., NullPointer-, ArrayIndexOutOfBounds-, ClassCast-), and

ESC/Java

- In general, an Extended Static Checker tries to prove the correctness of specifications, at compile-time, fully automatically, but . . .
- ESC/Java is *not sound* and is *not complete*. That is, it can sometimes reject a correct spec, and at other times accept an incorrect spec.
- However, an ESC/Java finds *lots* of (potential) bugs *quickly*, and
- ESC/Java is good at proving the absence of runtime exceptions like (e.g., NullPointer-, ArrayIndexOutOfBounds-, ClassCast-), and
- ESC/Java can verify a large class of fairly complex system properties.

ESC/Java vs Runtime Checking

There are important differences:

- **ESC/Java checks specs at compile-time, while `jmlc` checks specs at run-time**

ESC/Java vs Runtime Checking

There are important differences:

- **ESC/Java checks specs at compile-time, while `jmlc` checks specs at run-time**
- **ESC/Java proves correctness of specs, `jmlc`, `jmlrac`, and `jmlunit` only test correctness of specs.**

ESC/Java vs Runtime Checking

There are important differences:

- **ESC/Java checks specs at compile-time, while `jmlc` checks specs at run-time**
- **ESC/Java proves correctness of specs, `jmlc`, `jmlrac`, and `jmlunit` only test correctness of specs.**
- **Hence, ESC/Java is independent of any test suite, while the results of runtime testing are only as good as the test suite.**

ESC/Java vs Runtime Checking

There are important differences:

- **ESC/Java checks specs at compile-time, while `jmlc` checks specs at run-time**
- **ESC/Java proves correctness of specs, `jmlc`, `jmlrac`, and `jmlunit` only test correctness of specs.**
- **Hence, ESC/Java is independent of any test suite, while the results of runtime testing are only as good as the test suite.**
- **“Testing can show the presence of errors, but not their absence.”—E. W. Dijkstra**

ESC/Java vs Runtime Checking

There are important differences:

- **ESC/Java checks specs at compile-time, while `jmlc` checks specs at run-time**
- **ESC/Java proves correctness of specs, `jmlc`, `jmlrac`, and `jmlunit` only test correctness of specs.**
- **Hence, ESC/Java is independent of any test suite, while the results of runtime testing are only as good as the test suite.**
- **“Testing can show the presence of errors, but not their absence.”—E. W. Dijkstra**
- **As a result, ESC/Java provides a much higher degree of confidence than unit testing.**

The Hierarchy of Confidence

In general, the more you think about your software, and the more you write down and use your thoughts, the higher the quality of the software.

- **“hacked out” code with no documentation**

The Hierarchy of Confidence

In general, the more you think about your software, and the more you write down and use your thoughts, the higher the quality of the software.

- **“hacked out” code with no documentation**
- **+ some (English) documentation**

The Hierarchy of Confidence

In general, the more you think about your software, and the more you write down and use your thoughts, the higher the quality of the software.

- **“hacked out” code with no documentation**
- **+ some (English) documentation**
- **+ some unit tests**

The Hierarchy of Confidence

In general, the more you think about your software, and the more you write down and use your thoughts, the higher the quality of the software.

- **“hacked out” code with no documentation**
- **+ some (English) documentation**
- **+ some unit tests**
- **+ complete unit tests**

The Hierarchy of Confidence

In general, the more you think about your software, and the more you write down and use your thoughts, the higher the quality of the software.

- **“hacked out” code with no documentation**
- **+ some (English) documentation**
- **+ some unit tests**
- **+ complete unit tests**
- **+ manually written assertions**

The Hierarchy of Confidence

In general, the more you think about your software, and the more you write down and use your thoughts, the higher the quality of the software.

- **“hacked out” code with no documentation**
- **+ some (English) documentation**
- **+ some unit tests**
- **+ complete unit tests**
- **+ manually written assertions**
- **+ lightweight JML contracts + `jmlrac`**

The Hierarchy of Confidence

In general, the more you think about your software, and the more you write down and use your thoughts, the higher the quality of the software.

- **“hacked out” code with no documentation**
- **+ some (English) documentation**
- **+ some unit tests**
- **+ complete unit tests**
- **+ manually written assertions**
- **+ lightweight JML contracts + `jmlrac`**
- **+ heavyweight JML contracts + `jmlunit`**

The Hierarchy of Confidence

In general, the more you think about your software, and the more you write down and use your thoughts, the higher the quality of the software.

- **“hacked out” code with no documentation**
- **+ some (English) documentation**
- **+ some unit tests**
- **+ complete unit tests**
- **+ manually written assertions**
- **+ lightweight JML contracts + `jmlrac`**
- **+ heavyweight JML contracts + `jmlunit`**
- **+ JML models**

The Hierarchy of Confidence

In general, the more you think about your software, and the more you write down and use your thoughts, the higher the quality of the software.

- **“hacked out” code with no documentation**
- **+ some (English) documentation**
- **+ some unit tests**
- **+ complete unit tests**
- **+ manually written assertions**
- **+ lightweight JML contracts + `jmlrac`**
- **+ heavyweight JML contracts + `jmlunit`**
- **+ JML models**
- **+ ESC/Java**

The Hierarchy of Confidence

In general, the more you think about your software, and the more you write down and use your thoughts, the higher the quality of the software.

- **“hacked out” code with no documentation**
- **+ some (English) documentation**
- **+ some unit tests**
- **+ complete unit tests**
- **+ manually written assertions**
- **+ lightweight JML contracts + `jmlrac`**
- **+ heavyweight JML contracts + `jmlunit`**
- **+ JML models**
- **+ ESC/Java**
- **+ interactive verification**

(Spec|Ver)ification Trade-offs

- **When using `jmlrac` and `jmlunit` you specify any properties you like.**

(Spec|Ver)ification Trade-offs

- **When using `jmlrac` and `jmlunit` you specify any properties you like.**
- **But when using ESC/Java, you are forced to specify some properties.**

(Spec|Ver)ification Trade-offs

- **When using `jmlrac` and `jmlunit` you specify any properties you like.**
- **But when using ESC/Java, you are forced to specify some properties.**
- **If you already understand the code, then these properties are usually obvious.**

But for larger programs, this is often not the case!

(Spec|Ver)ification Trade-offs

- **When using `jmlrac` and `jmlunit` you specify any properties you like.**
- **But when using ESC/Java, you are forced to specify some properties.**
- **If you already understand the code, then these properties are usually obvious.**

  But for larger programs, this is often not the case!

- **Once you have these properties documented, then understanding the code is easier for you, and for others.**

Limitations of ESC/Java

ESC/Java is

- **not sound: it may reject a correct spec**

Limitations of ESC/Java

ESC/Java is

- **not sound: it may reject a correct spec**
- **not complete: it may accept an incorrect spec**

Limitations of ESC/Java

ESC/Java is

- **not sound: it may reject a correct spec**
- **not complete: it may accept an incorrect spec**
- **These are unavoidable concessions to main goal, that of finding many (potential) bugs, completely automatically.**

Limitations of ESC/Java

ESC/Java is

- **not sound: it may reject a correct spec**
- **not complete: it may accept an incorrect spec**
- **These are unavoidable concessions to main goal, that of finding many (potential) bugs, completely automatically.**
- **In practice, neither issue is much of a problem.**

Limitations of ESC/Java

ESC/Java is

- **not sound: it may reject a correct spec**
- **not complete: it may accept an incorrect spec**
- **These are unavoidable concessions to main goal, that of finding many (potential) bugs, completely automatically.**
- **In practice, neither issue is much of a problem.**
- **ESC/Java2 only supports statically checking a subset of full JML, but you will likely never use or learn (in this course) any of JML that ESC/Java2 does not check.**

Places where ESC/Java is not Sound

Tricky things to watch out for include:

Places where ESC/Java is not Sound

Tricky things to watch out for include:

- **very complex, interdependent invariants**

Places where ESC/Java is not Sound

Tricky things to watch out for include:

- **very complex, interdependent invariants**
- **misuse of pragmas like `assume`, `axiom`, and `nowarn`**

Places where ESC/Java is not Sound

Tricky things to watch out for include:

- **very complex, interdependent invariants**
- **misuse of pragmas like `assume`, `axiom`, and `nowarn`**
- **verification of loops**

Places where ESC/Java is not Sound

Tricky things to watch out for include:

- **very complex, interdependent invariants**
- **misuse of pragmas like `assume`, `axiom`, and `nowarn`**
- **verification of loops**
- **complex arithmetic with large numbers**

Places where ESC/Java is not Sound

Tricky things to watch out for include:

- very complex, interdependent invariants
- misuse of pragmas like `assume`, `axiom`, and `nowarn`
- verification of loops
- complex arithmetic with large numbers
- ignored exceptional conditions

Places where ESC/Java is not Sound

Tricky things to watch out for include:

- very complex, interdependent invariants
- misuse of pragmas like `assume`, `axiom`, and `nowarn`
- verification of loops
- complex arithmetic with large numbers
- ignored exceptional conditions
- aliasing and shared variables

Using ESC/Java for Java Card

- **ESC/Java can cope with Java Card-sized programs.**

Using ESC/Java for Java Card

- **ESC/Java can cope with Java Card-sized programs.**
- **First step: verify lightweight, weak (but far from trivial!) specs**

```
/*@ requires apdu != null;
    ensures  true;
    signals  (APDUException) true;
    signals  (ISOException) true;
 @*/
public void process(APDU apdu) { ..
```

That is, focus on "non-nullness" and exceptions.

Using ESC/Java for Java Card

- **ESC/Java can cope with Java Card-sized programs.**
- **First step: verify lightweight, weak (but far from trivial!) specs**

```
/*@ requires apdu != null;
    ensures  true;
    signals  (APDUException) true;
    signals  (ISOException) true;
 @*/
public void process(APDU apdu) { ..
```

  That is, focus on "non-nullness" and exceptions.

- **Next, "upgrade" lightweight contracts to heavyweight contracts, focusing on preconditions and assignable clauses.**

Using ESC/Java for Java Card

- ESC/Java can cope with Java Card-sized programs.
- First step: verify lightweight, weak (but far from trivial!) specs

```
/*@ requires apdu != null;
    ensures  true;
    signals  (APDUException) true;
    signals  (ISOException) true;
 @*/
public void process(APDU apdu) { ..
```

  That is, focus on "non-nullness" and exceptions.
- Next, "upgrade" lightweight contracts to heavyweight contracts, focusing on preconditions and assignable clauses.
- Next, add interesting invariants.

Using ESC/Java for Java Card

- ESC/Java can cope with Java Card-sized programs.
- First step: verify lightweight, weak (but far from trivial!) specs

```
/*@ requires apdu != null;
    ensures  true;
    signals  (APDUException) true;
    signals  (ISOException) true;
 @*/
public void process(APDU apdu) { ..
```

  That is, focus on "non-nullness" and exceptions.
- Next, "upgrade" lightweight contracts to heavyweight contracts, focusing on preconditions and assignable clauses.
- Next, add interesting invariants.
- Finally, focus on interesting postconditions.