Formal Verification of Memory Performance Contracts

Christian Engel

Universität Karlsruhe (TH)

6th International KeY Symposium

Outline

Motivation

Is formal verification of performance constraints really necessary?

Motivation

Is formal verification of performance constraints really necessary?

No

Motivation

Is formal verification of performance constraints really necessary?

No

Usually bad performance is not an issue of software correctness.

Motivation

Is formal verification of performance constraints really necessary?

No

Usually bad performance is not an issue of software correctness.

But ...

Real-time applications have to meet certain performance constraints, otherwise they are erroneous.

RTSJ – The Real-Time Specification for Java

RTSJ – The Real-Time Specification for Java

One obstacle for writting real-time Java applications: Garbage Collection

RTSJ – The Real-Time Specification for Java

One obstacle for writting real-time Java applications: Garbage Collection

New Types of Heap Memory . . .

- Immortal Memory

RTSJ – The Real-Time Specification for Java

One obstacle for writting real-time Java applications: Garbage Collection

New Types of Heap Memory . . .

- Immortal Memory
- Scoped Memory

RTSJ – The Real-Time Specification for Java

One obstacle for writting real-time Java applications: Garbage Collection

New Types of Heap Memory . . .

- Immortal Memory
- Scoped Memory
- Both are not subject to garbage collection

RTSJ – The Real-Time Specification for Java

One obstacle for writting real-time Java applications: Garbage Collection

New Types of Heap Memory . . .

- Immortal Memory
- Scoped Memory
- Both are not subject to garbage collection

. . . and Threads

- Realtime Thread

RTSJ – The Real-Time Specification for Java

One obstacle for writting real-time Java applications: Garbage Collection

New Types of Heap Memory . . .

- Immortal Memory
- Scoped Memory
- Both are not subject to garbage collection

. . . and Threads

- Realtime Thread
- No-Heap Realtime Thread

JML Performance Specifications

Worst Case Execution Time (WCET)

JML Performance Specifications

Worst Case Execution Time (WCET)

- **`duration`** clause: part of the method contract

JML Performance Specifications

Worst Case Execution Time (WCET)

- **`duration`** clause: part of the method contract
- **`\duration`** function: **`\duration`**`(o.m())`

JML Performance Specifications

Worst Case Execution Time (WCET)

- **`duration`** clause: part of the method contract
- **`\duration`** function: **`\duration`**`(o.m())`

Worst Case Memory Usage (WCMU)

- **`working_space`** clause:
  - part of the method contract
  - specifies the WCMU of a method

JML Performance Specifications

Worst Case Execution Time (WCET)

- **`duration`** clause: part of the method contract
- **`\duration`** function: **`\duration`**`(o.m())`

Worst Case Memory Usage (WCMU)

- **`working_space`** clause:
  - part of the method contract
  - specifies the WCMU of a method
- **`\working_space`** function: **`\working_space`**`(o.m())`

JML Performance Specifications

Worst Case Execution Time (WCET)

- **`duration`** clause: part of the method contract
- **`\duration`** function: **`\duration`**`(o.m())`

Worst Case Memory Usage (WCMU)

- **`working_space`** clause:
  - part of the method contract
  - specifies the WCMU of a method
- **`\working_space`** function: **`\working_space`**`(o.m())`
- **`\space`** function: **`\space`**`(`**`new int`**`[3])`

Shortcomings of JML Memory Specs

This specification can be incorrect:

JAVA + JML

```
 static SomeClass instance;

/*@ working_space \working_space(clear()) +
  @               \working_space(getInstance());   @*/
 public SomeClass freshInstance(){
     clear();
     return getInstance();
 }
```

JAVA + JML

Shortcomings of JML Memory Specs

This specification can be incorrect:

JAVA + JML

```
 static SomeClass instance;

/*@ working_space \working_space(clear()) +
  @               \working_space(getInstance());   @*/
 public SomeClass freshInstance(){
     clear();
     return getInstance();
 }
```

JAVA + JML

- **working_space** clauses are evaluated in the post state.
- no access to intermediate program states in **\working_space** expressions

Shortcomings of JML Memory Specs

This specification can be incorrect:

JAVA + JML

```
static SomeClass instance;

public static clear(){ instance = null; }

public static getInstance(){
    if(instance==null) instance = new SomeClass();
    return instance;
}
```

JAVA + JML

- **working_space** clauses are evaluated in the post state.
- no access to intermediate program states in **\working_space** expressions

Shortcomings of JML Memory Specs

This specification can be incorrect:

JAVA + JML

```
 static SomeClass instance;

/*@ working_space \working_space(clear()) +
  @               \working_space(getInstance());   @*/
 public SomeClass freshInstance(){
     clear();
     return getInstance();
 }
```

JAVA + JML

- **working_space** clauses are evaluated in the post state.
- no access to intermediate program states in **\working_space** expressions

Shortcomings of JML Memory Specs

This specification can be incorrect:

JAVA + JML

```
 static SomeClass instance;

/*@ working_space \working_space(clear()) +
  @               \working_space(getInstance());   @*/
 public SomeClass freshInstance(){
     clear();  // instance == null
     return getInstance();
 }
```

JAVA + JML

- **working_space** clauses are evaluated in the post state.
- no access to intermediate program states in **\working_space** expressions

Shortcomings of JML Memory Specs

This specification can be incorrect:

JAVA + JML

```
 static SomeClass instance;

/*@ working_space \working_space(clear()) +
  @               \working_space(getInstance());   @*/
 public SomeClass freshInstance(){
     clear();  // instance == null
     return getInstance(); // instance != null
 }
```

JAVA + JML

- **working_space** clauses are evaluated in the post state.
- no access to intermediate program states in **\working_space** expressions

`working_space` in KeYJML

Alternative Approach

State in which the target method is executed is specified within `\working_space` expressions: `\working_space(method, cond)`

`working_space` in KeYJML

Alternative Approach

State in which the target method is executed is specified within `\working_space` expressions: `\working_space(method, cond)`
`\working_space(method, cond)` is a rigid expression.

working_space in KeYJML

Alternative Approach

State in which the target method is executed is specified within `\working_space` expressions: `\working_space(method, cond)`
`\working_space(method, cond)` is a rigid expression.

JAVA + JML

```
static SomeClass instance;

/*@ working_space \working_space(clear(), true) +
  @     \working_space(getInstance(), instance==null);@*/
public SomeClass freshInstance(){
    clear();
    return getInstance();
}
```

JAVA + JML

Loop Specs in KeYJML

```
/*@  requires a!=null;
  @  working_space a.length*\space(new Object()) +
  @  \working_space(new ArrayStoreException(), true);
  @*/
 public void initArr(Object[] a){
     int i=0;
     /*@ loop_invariant i>=0;
       @ assignable a[*];
       @ decreasing a.length-i;
       @ working_space_single_iteration \space(new Object());
       @*/
     while(i<a.length){
         a[i++] = new Object();
     }
 }
```

Proof Obligations

JAVA + JML

```
/*@ public normal_behavior
  @   requires PRE;
  @   working_space S;
  @*/
 public void doSth(){ ...
```

JAVA + JML

Proof Obligations

JAVA + JML

```
/*@ public normal_behavior
  @   requires PRE;
  @   working_space S;
  @*/
 public void doSth(){ ...
```

JAVA + JML

Idea

Use a program variable to log the memory allocation of Java programs.

$$PRE \rightarrow \{\mathbf{h}_{old} := \mathbf{h}\}\langle \texttt{doSth();}\rangle \mathbf{h} \leq \mathbf{h}_{old} + S$$

Object Creation Rule

Symbolic execution of constructors increases **h** by the heap space consumed by the created object.

$$\text{arrayCreation}\ \frac{\Gamma \Rightarrow \{\mathcal{U}; \mathbf{h} := \mathbf{h} + \mathit{space}^{\mathit{arr}}(e, \texttt{l1})\}\langle \pi\ \texttt{AC}\ \omega\rangle\ \phi,\ \Delta}{\Gamma \Rightarrow \{\mathcal{U}\}\langle \pi\ \texttt{v=}\mathbf{new}\ \texttt{T[l1])};\ \omega\rangle\ \phi,\ \Delta}$$

$$\text{objectCreation}\ \frac{\Gamma \Rightarrow \{\mathcal{U}; \mathbf{h} := \mathbf{h} + \mathit{space}_T\}\langle \pi\ \texttt{OC}\ \omega\rangle\ \phi,\ \Delta}{\Gamma \Rightarrow \{\mathcal{U}\}\langle \pi\ \texttt{v=}\mathbf{new}\ \texttt{T(a1,...,an)};\ \omega\rangle\ \phi,\ \Delta}$$

Contract Rule

Java + JML

```
/*@ public normal_behavior
  @   requires Pre;
  @   ensures Post;
  @   assignable Mod;
  @   working_space S;  @*/
 public void m(){ ...
```

Java + JML

$$\text{applyContract}\ \frac{\begin{array}{l}\Gamma \Rightarrow \{\mathcal{U}\}Pre,\ \Delta \\ \Gamma \Rightarrow \{\mathcal{U}\}(ws^{nr}_{m()} = \{V(Mod)\}S \rightarrow \\ \qquad \{V(Mod)||\mathbf{h} := \mathbf{h} + ws^{nr}_{m()}\}(Post \rightarrow \langle \pi\, \omega \rangle\, \phi)),\ \Delta\end{array}}{\Gamma \Rightarrow \{\mathcal{U}\}\langle \pi\, \mathtt{m()}\,;\, \omega \rangle\, \phi,\ \Delta}$$

How ws^{nr} relates to `\working_space`

ws^{nr}_m is a nonrigid constant denoting the WCMU of `m` in a certain set of states S.

How ws^{nr} relates to `\working_space`

ws^{nr}_{m} is a nonrigid constant denoting the WCMU of `m` in a certain set of states S.

$ws^{r}_{m,cond}$ is the JAVA CARD DL counterpart of the JML expression `\working_space(m, cond)`.

How ws^{nr} relates to `\working_space`

ws^{nr}_m is a nonrigid constant denoting the WCMU of `m` in a certain set of states S.

$ws^{r}_{m,cond}$ is the JAVA CARD DL counterpart of the JML expression `\working_space(m, cond)`.

If `cond` holds in every state in S, $\{\mathcal{U}\}ws^{nr}_m$ cannot exceed `\working_space(m, cond)`.

How ws^{nr} relates to `\working_space`

ws^{nr}_m is a nonrigid constant denoting the WCMU of `m` in a certain set of states S.

$ws^{r}_{m,cond}$ is the JAVA CARD DL counterpart of the JML expression `\working_space(m, cond)`.

If `cond` holds in every state in S, $\{\mathcal{U}\}ws^{nr}_m$ cannot exceed `\working_space(m, cond)`.

$$\text{wsNonRigid}\ \frac{\Gamma \Rightarrow \{\mathcal{U}\}cond,\ \Delta \qquad \Gamma,\ \{\mathcal{U}\}ws^{nr}_m \leq ws^{r}_{m,cond} \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}$$

Memory Usage Contract Rule

JAVA + JML

```
/*@ public behavior
  @  requires Pre;
  @  ensures Post;
  @  working_space t;
  @*/
public int m(){ ...
```

JAVA + JML

$$\text{wsContract1}\ \frac{\Gamma \Rightarrow \{*:=*\}(\varphi \rightarrow \mathit{Pre}),\ \Delta \qquad \Gamma,\ \{*:=*\}(\mathit{Post} \wedge \mathit{ws}^{r}_{m,\varphi} = t) \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}$$

Demo

Demo

Thank you for your Attention!

Questions

Questions?