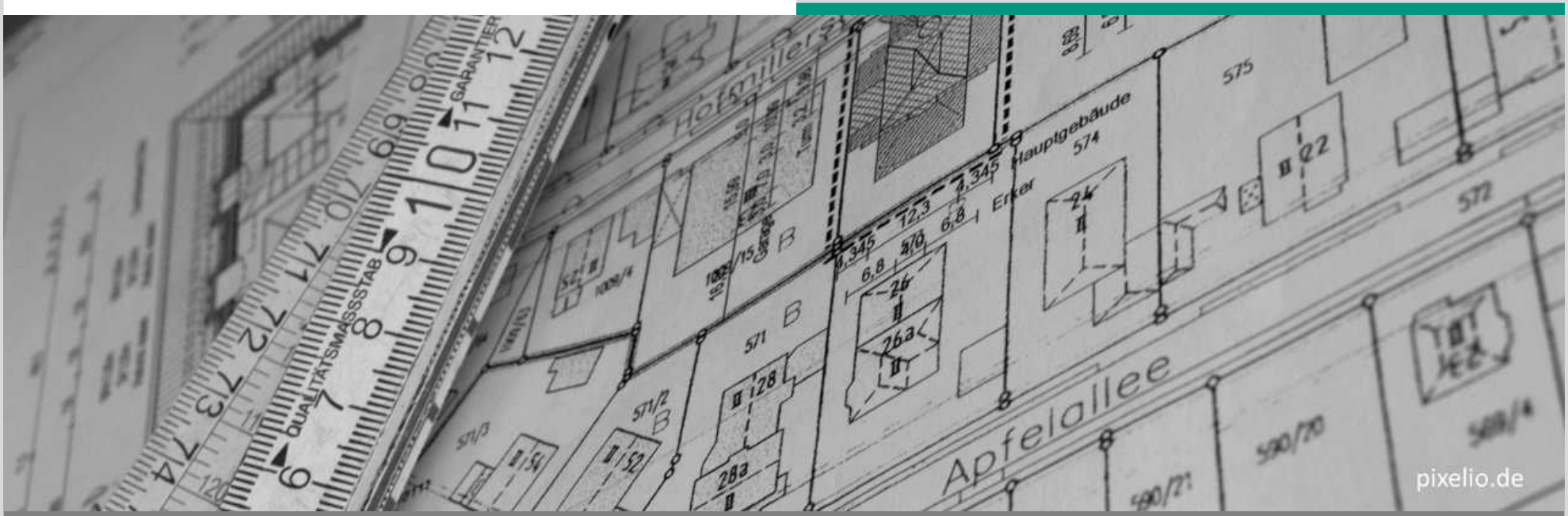
Karlsruhe Institute of Technology

Semantics for the Java Modeling Language

Daniel Bruns

Forschungszentrum Karlsruhe
in der Helmholtz-Gemeinschaft

Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825



Outlook

The Java Modeling Language (JML)

- is a specification language especially for Java.
- is implemented by KeY.

But...

- There are no "official" formal semantics, only verbal descriptions.
- The Reference Manual still is incomplete or ambiguous on some occasions.
- The language is constantly evolving.

The Java Modeling Language (JML)

Karlsruhe Institute of Technology

- is a specification language especially for Java.
- is implemented by KeY.

But...

- There are no “official” formal semantics, only verbal descriptions.
- The Reference Manual still is incomplete or ambiguous on some occasions.
- The language is constantly evolving.

Goal and approach of my thesis

Karlsruhe Institute of Technology

- Provide rigorously formal semantics
- Build on simple mathematical notion
- Independent of logical system/ implementing tool
- See JML as a proper extension to Java (no interference with execution)

Foundations

- Universe $\mathcal{U}$ of semantical objects and primitive values
- System state $s = (h, \sigma, \chi)$
  - Heap $h : \mathcal{U} \times \mathbb{I} \rightarrow \mathcal{U}$ maps locations to values
  - Stack $\sigma : \mathbb{I} \rightarrow \mathcal{U}$ evaluates local variables
  - Call stack χ contains methods and receivers
- Program behavior through Black Box
  - Input: pre-state s_0, program π
  - Output: run R, return condition Ω, return value ρ

Foundations

- Universe $\mathcal{U}$ of semantical objects and primitive values
- System state $s = (h, \sigma, \chi)$
  - Heap $h : \mathcal{U} \times \mathbb{I} \rightarrow \mathcal{U}$ maps locations to values
  - Stack $\sigma : \mathbb{I} \rightarrow \mathcal{U}$ evaluates local variables
  - Call stack χ contains methods and receivers
- Program behavior through Black Box
  - Input: pre-state s_0, program π
  - Output: run R, return condition Ω, return value ρ

Foundations

- Universe $\mathcal{U}$ of semantical objects and primitive values
- System state $s = (h, \sigma, \chi)$
  - Heap $h : \mathcal{U} \times \mathbb{I} \rightarrow \mathcal{U}$ maps locations to values
  - Stack $\sigma : \mathbb{I} \rightarrow \mathcal{U}$ evaluates local variables
  - Call stack χ contains methods and receivers
- Program behavior through Black Box
  - Input: pre-state s_0, program π
  - Output: run R, return condition Ω, return value ρ

Expression evaluation

Evaluation function $val_s : Expr \rightarrow \mathcal{U}$ for state $s = (h, \sigma, \chi)$

Atomic expressions

- $val_s(\mathrm{v}) = \sigma(\mathrm{v})$ for a local variable
- $val_s(\mathrm{r.x}) = h(val_s(\mathrm{r}), \mathrm{x})$ for a field
- $val_s(\mathrm{this}) = \mathrm{top}(\chi)$

Compound expressions

- $val_s(\mathrm{A} \mid\mid \mathrm{B}) = true$ iff $val_s(\mathrm{A}) = true$ or $val_s(\mathrm{B}) = true$
- $val_s(\mathrm{x} == \mathrm{y}) = true$ iff $val_s(\mathrm{x}) = val_s(\mathrm{y})$

Expression evaluation

Evaluation function $val_s : Expr \rightarrow \mathcal{U}$ for state $s = (h, \sigma, \chi)$

Atomic expressions

- $val_s(\mathrm{v}) = \sigma(\mathrm{v})$ for a local variable
- $val_s(\mathrm{r.x}) = h(val_s(\mathrm{r}), \mathrm{x})$ for a field
- $val_s(\mathrm{this}) = \mathrm{top}(\chi)$

Compound expressions

- $val_s(\mathrm{A} \mid\mid \mathrm{B}) = true$ iff $val_s(\mathrm{A}) = true$ or $val_s(\mathrm{B}) = true$
- $val_s(\mathrm{x} == \mathrm{y}) = true$ iff $val_s(\mathrm{x}) = val_s(\mathrm{y})$

Expression evaluation

Evaluation function $val_s : Expr \rightarrow \mathcal{U}$ for state $s = (h, \sigma, \chi)$

Atomic expressions

- $val_s(\mathrm{v}) = \sigma(\mathrm{v})$ for a local variable
- $val_s(\mathrm{r.x}) = h(val_s(\mathrm{r}), \mathrm{x})$ for a field
- $val_s(\mathrm{this}) = \mathrm{top}(\chi)$

Compound expressions

- $val_s(\mathrm{A} \mid\mid \mathrm{B}) = true$ iff $val_s(\mathrm{A}) = true$ or $val_s(\mathrm{B}) = true$
- $val_s(\mathrm{x} == \mathrm{y}) = true$ iff $val_s(\mathrm{x}) = val_s(\mathrm{y})$

Problems with expressions

- may have side-effects, e.g. new Object() == o
- may "throw exceptions" in JML, e.g. upon division by zero
- may refer to another state, e.g. through \old

Expressions yield state transitions

- Successor function $\omega : \mathit{Expr} \to (\mathcal{S} \to \mathcal{S})$
- E.g. $\omega(\mathrm{A} \mid\mid \mathrm{B})(s) = \begin{cases} \omega(\mathrm{A})(s) & \mathit{val}_s(\mathrm{A}) = \mathit{true} \\ \omega(\mathrm{B})(\omega(\mathrm{A})(s)) & \text{otherwise} \end{cases}$
- Result: order of evaluation matters

Problems with expressions

- may have side-effects, e.g. new Object() == o
- may "throw exceptions" in JML, e.g. upon division by zero
- may refer to another state, e.g. through \old

Expressions yield state transitions

- Successor function $\omega : \mathit{Expr} \to (\mathcal{S} \to \mathcal{S})$
- E.g. $\omega(\mathrm{A} \mid\mid \mathrm{B})(s) = \begin{cases} \omega(\mathrm{A})(s) & \mathit{val}_s(\mathrm{A}) = \mathit{true} \\ \omega(\mathrm{B})(\omega(\mathrm{A})(s)) & \text{otherwise} \end{cases}$
- Result: order of evaluation matters

Expression evaluation

Evaluation function $val_s : Expr \rightarrow \mathcal{U}$ for state $s = (h, \sigma, \chi)$

Atomic expressions

- $val_s(\mathrm{v}) = \sigma(\mathrm{v})$ for a local variable
- $val_s(\mathrm{r.x}) = h'(val_s(\mathrm{r}), \mathrm{x})$ where $\omega(\mathrm{r})(s) = (h', \sigma', \chi')$
- $val_s(\mathrm{this}) = \mathrm{top}(\chi)$

Compound expressions

- $val_s(\mathrm{A} \mid\mid \mathrm{B}) = true$ iff $val_s(\mathrm{A}) = true$ or $val_{\omega(A)(s)}(\mathrm{B}) = true$
- $val_s(\mathrm{x} == \mathrm{y}) = true$ iff $val_s(\mathrm{x}) = val_{\omega(x)(s)}(\mathrm{y})$

Problems with expressions

- may have side-effects, e.g. new Object() == o
- may "throw exceptions" in JML, e.g. upon division by zero
- may refer to another state, e.g. through \old

Well-definition of expressions

- E.g. $wd_s(\mathrm{r.x})$ iff $val_s(\mathrm{r}) \neq null$
- On the top level, expressions are valid if they valuate to true and are well-defined
- $val(\mathrm{A} \mid\mid \mathrm{B}) = true$ iff
  - $val(\mathrm{A}) = true$ and $wd(\mathrm{A})$ or
  - $val(\mathrm{B}) = true$ and $wd(\mathrm{A})$ and $wd(\mathrm{B})$

Problems with expressions

- may have side-effects, e.g. new Object() == o
- may "throw exceptions" in JML, e.g. upon division by zero
- may refer to another state, e.g. through \old

Well-definition of expressions

- E.g. $wd_s(\mathrm{r.x})$ iff $val_s(\mathrm{r}) \neq null$
- On the top level, expressions are valid if they valuate to true and are well-defined
- $val(\mathrm{A} \mid\mid \mathrm{B}) = true$ iff
  - $val(\mathrm{A}) = true$ and $wd(\mathrm{A})$ or
  - $val(\mathrm{B}) = true$ and $wd(\mathrm{A})$ and $wd(\mathrm{B})$

Problems with expressions

- may have side-effects, e.g. new Object() == o
- may "throw exceptions" in JML, e.g. upon division by zero
- may refer to another state, e.g. through \old

Two-state evaluation

- Evaluate with two states: s_0 (pre-state), s_1 (post-state)
- $val_{s_0,s_1}(\backslash\text{old}(e)) = val_{s_0,s_0}(e)$

Type specifications

- A method invoked in a legal pre-state s_0 yields a run R.
- I_C instance invariant of class C

Invariant

If I_C holds in s_0, then I_C also holds in every state $s \in R$ which is visible for an instance of C.

Type specifications

- A method invoked in a legal pre-state s_0 yields a run R.
- I_C instance invariant of class C

Invariant

If I_C holds in s_0, then I_C also holds in every state $s \in R$ which is visible for an instance of C.

Which states are visible?

Karlsruhe Institute of Technology

A state is visible for an object *o* if

- it is reached at the beginning or end of an invocation on *o*,
- it is reached at the beginning or end of a static invocation on *o*'s dynamic type or a super-type or
- none of the above is in progress.

Different invariant semantics

```
public class Invariants {
    private int z = 1;
    //@ private invariant z > 0;

    public void a() {
        z++= 0;
    }

    public void b() {
        z = 0;
        a();
    }
}
```

Different invariant semantics

```
public class Invariants {
    private int z = 1;
    //@ private invariant z > 0;

    public void a() {
        z++= 0;
    }

    public void b() {
        z = 0;
        a();
    }
}
```

Different invariant semantics

```
public class Invariants {
    private int z = 1;
    //@ private invariant z > 0;

    public void a() {
        z++= 0;
    }

    public void b() {
        z = 0;
        a();
    }
}
```

Different invariant semantics

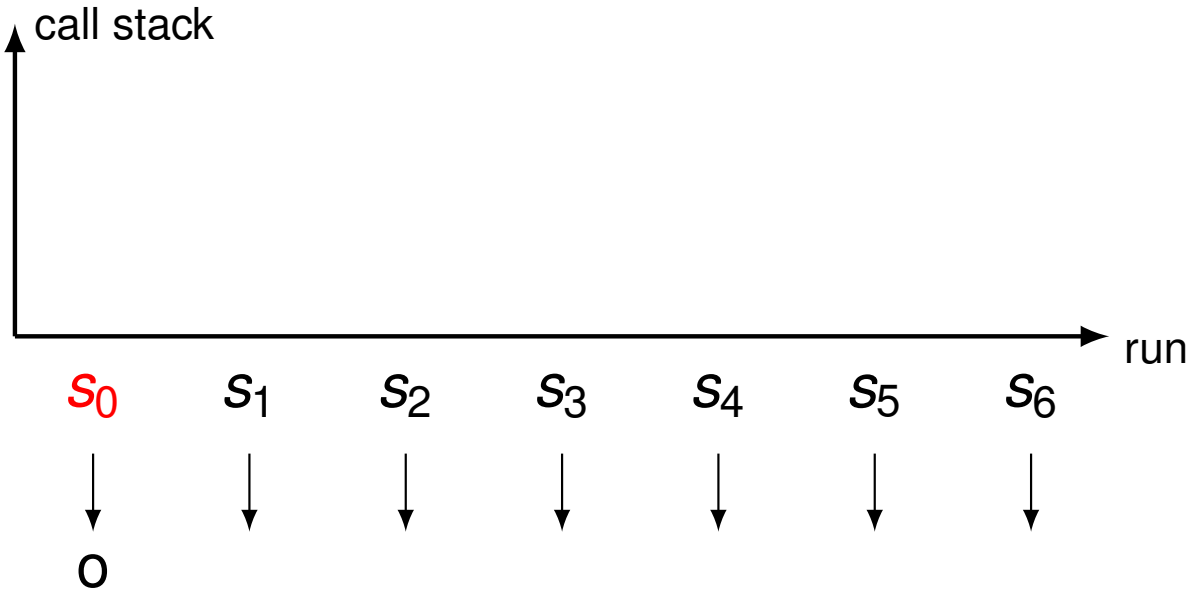
```
public class Invariants {
    private int z = 1;
    //@ private invariant z > 0;

    public void a() {
        z++= 0;
    }

    public void b() {
        z = 0;
        a();
    }
}
```

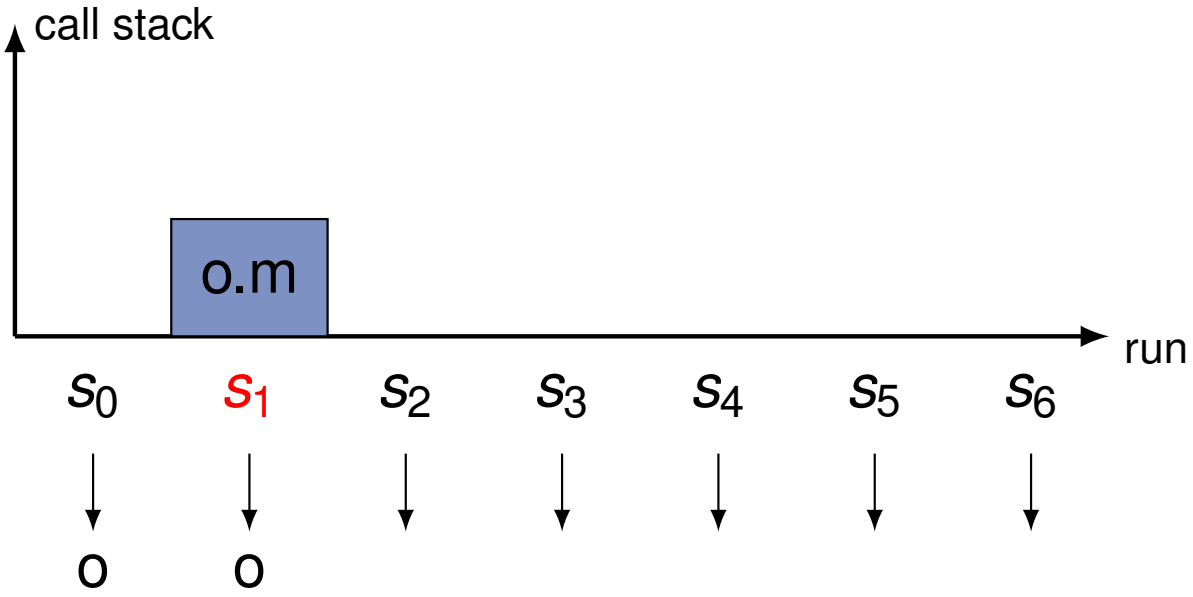
Deduce visible states from a run

```
m () { n(); q = new Q(); }
```



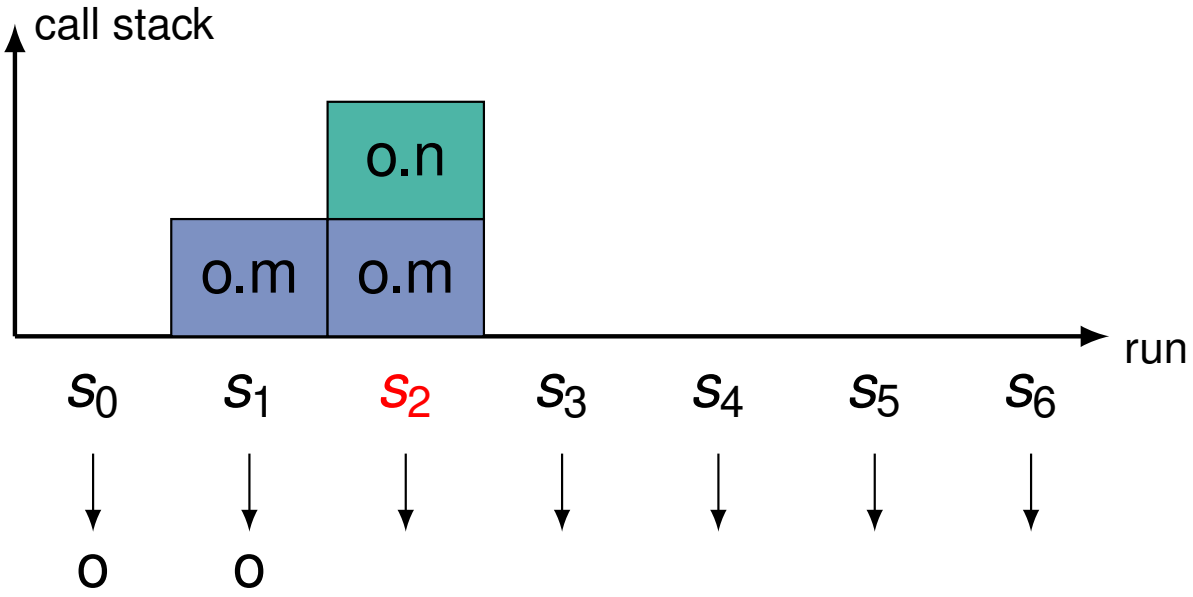
Deduce visible states from a run

```
m () { n(); q = new Q(); }
```



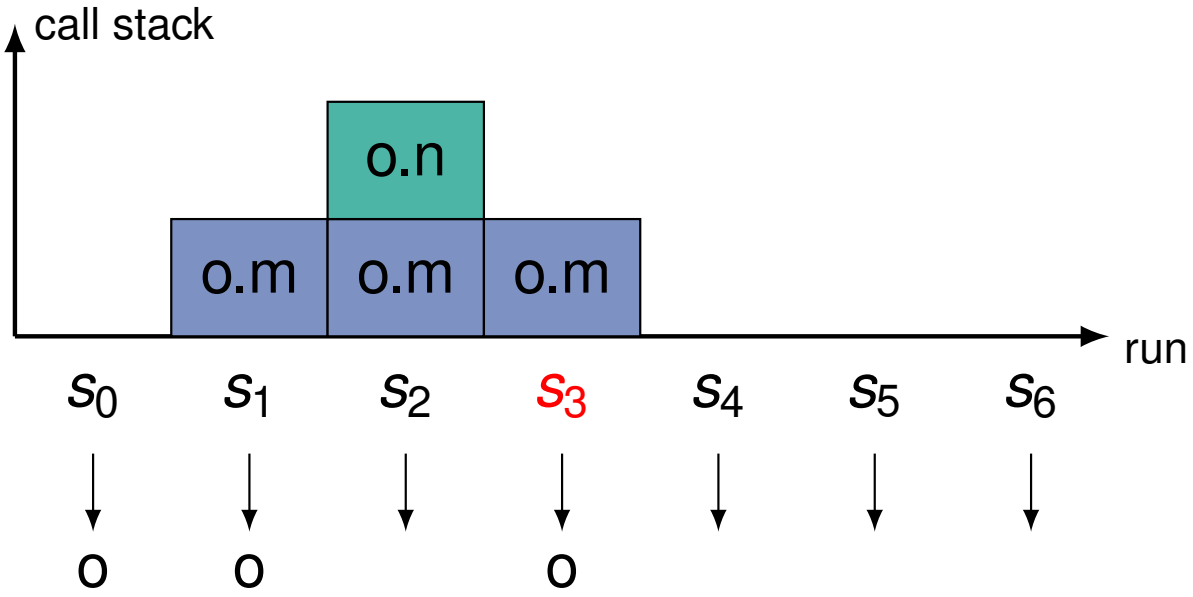
Deduce visible states from a run

```
m () { n(); q = new Q(); }
```



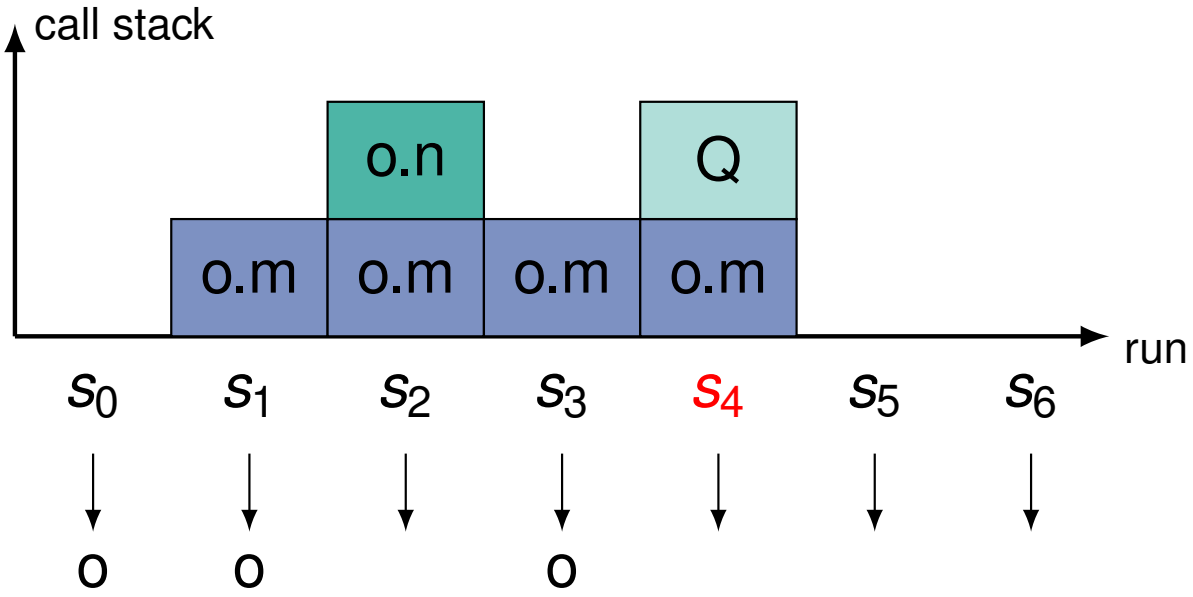
Deduce visible states from a run

```
m () { n(); q = new Q(); }
```



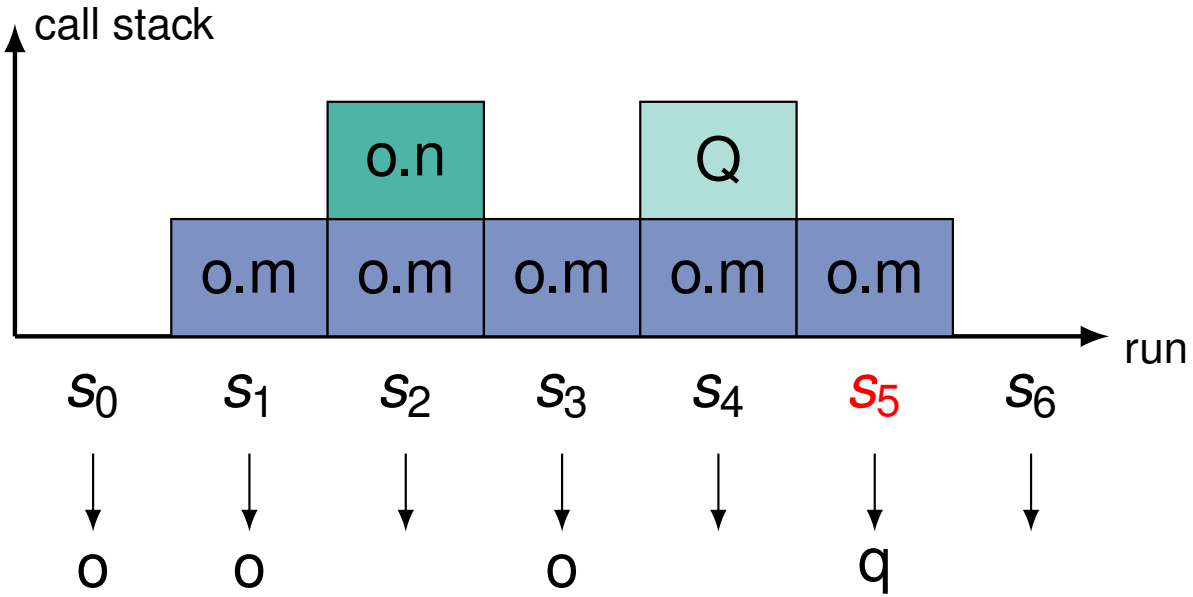
Deduce visible states from a run

```
m () { n(); q = new Q(); }
```



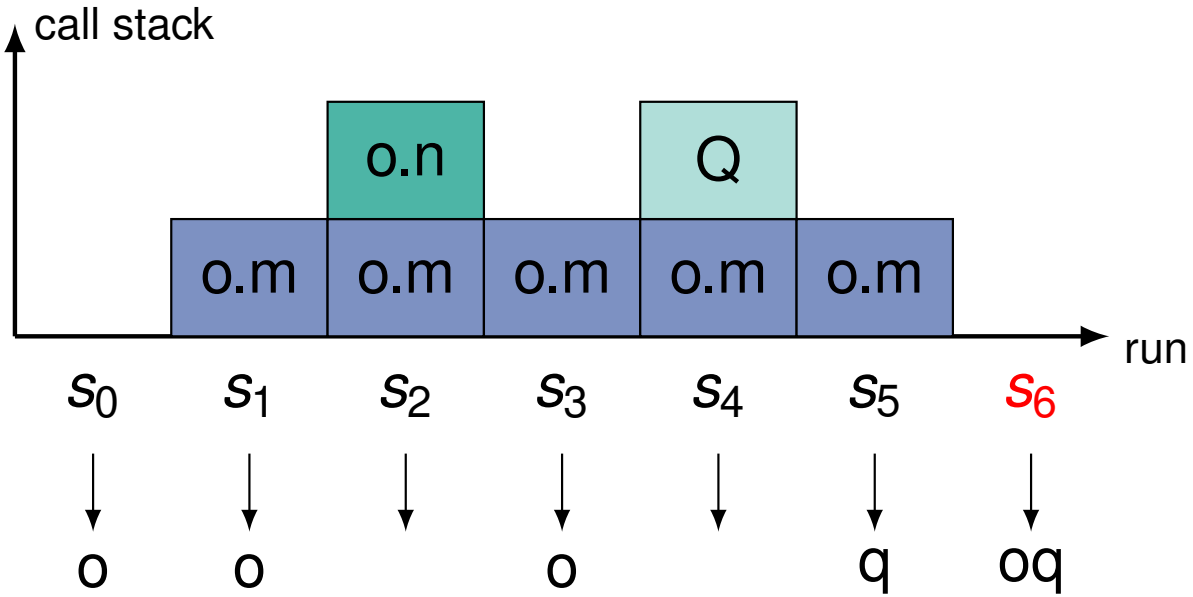
Deduce visible states from a run

```
m () { n(); q = new Q(); }
```



Deduce visible states from a run

```
m () { n(); q = new Q(); }
```



Summary

Karlsruhe Institute of Technology

My work does include

- Most expressions and specification clauses (except real-time and concurrency)
- Side-effects and well-definition of expressions
- Valuation for model and ghost fields
- Visible state semantics
- Mostly “Level 2” of JML

KeY and JML

Karlsruhe Institute of Technology

KeY does support

- Invariants, history constraints, methods specifications, loop specifications (with framing)
- Ghost fields
- Support for different integer semantics

KeY does not support

- Visible states
- Side-effects and well-definition
- Model fields

KeY and JML

Karlsruhe Institute of Technology

KeY does support

- Invariants, history constraints, methods specifications, loop specifications (with framing)
- Ghost fields
- Support for different integer semantics

KeY does not support

- Visible states
- Side-effects and well-definition
- Model fields

The end

Thank you.