

# **Modelling System Invariants**

## **Part 1: Java-reachable States**

**Bernhard Beckert**



**Universität Koblenz-Landau**

**KeY Symposium**  
**Speyer, June 2006**

# Motivation

$o \neq \text{null}, \quad o.\text{next}.\text{created} = \text{false} \quad \vdash \quad \dots$

# Motivation

$o \neq \text{null}, \quad o.\text{next}.\text{created} = \text{false} \quad \vdash \quad \dots$

**Impossible in Java**

# Motivation

$o \neq \text{null}, \quad o.\text{next}.\text{created} = \text{false} \quad \vdash \quad \dots$

**Impossible in Java**

**Theorefore**

**Would like to close that proof branch**

# Motivation

$o \neq \text{null}, \quad o.\text{next}.\text{created} = \text{false} \quad \vdash \quad \dots$

**Impossible in Java**

**Theorefore**

Would like to close that proof branch

**But**

A valid state in Java DL

# Motivation

## Observation

**There are more states in Java DL than in Java**

# Motivation

## Observation

There are more states in Java DL than in Java

## Consequences

- Does **not** imply incompleteness of KeY w.r.t. Java

# Motivation

## Observation

There are more states in Java DL than in Java

## Consequences

- Does **not** imply incompleteness of KeY w.r.t. Java
- Provable that a particular program does not reach such states

# Motivation

## Observation

There are more states in Java DL than in Java

## Consequences

- Does **not** imply incompleteness of KeY w.r.t. Java
- Provable that a particular program does not reach such states
- But bad in practice:  
We do not want to prove this again and again

# States in Java DL

## Set of states depends on signature

- States assign values to all symbols
- Signature in general infinite (but that is not a problem)

# States in Java DL

## Set of states depends on signature

- States assign values to all symbols
- Signature in general infinite (but that is not a problem)

## Set of states is not enumerable

# States in Java DL

## Set of states depends on signature

- States assign values to all symbols
- Signature in general infinite (but that is not a problem)

## Set of states is not enumerable

- Non-rigid functions with infinite domain  
(all objects exist whether created or not)

# States in Java DL

## Set of states depends on signature

- States assign values to all symbols
- Signature in general infinite (but that is not a problem)

## Set of states is not enumerable

- Non-rigid functions with infinite domain  
(all objects exist whether created or not)
- Consequence of constant domain assumption  
(important for smooth handling of quantifiers)

# States in Java DL

## Set of states depends on signature

- States assign values to all symbols
- Signature in general infinite (but that is not a problem)

## Set of states is not enumerable

- Non-rigid functions with infinite domain  
(all objects exist whether created or not)
- Consequence of constant domain assumption  
(important for smooth handling of quantifiers)
- Therefore: Not even quantified updates can reach all states

# Java-reachable States: Empty States

## Empty states

States in which no object is created

# Java-reachable States: Empty States

## Empty states

States in which no object is created

### Note

- Not unique

# Java-reachable States: Empty States

## Empty states

States in which no object is created

### Note

- Not unique
- Differ on interpretation of
  - program variables
  - attributes of non-created objects

# Java-reachable States: Definition

## Definition: Java-reachable state (JRS)

State  $s$  such that

- there is a Java program  $p$
- there is an empty state  $e$

with

$p$  terminates in  $s$  when started in  $e$

# Java-reachable States: Properties

## Invariant

**“Java-reachable state” is an invariant for all Java programs**

# Java-reachable States: Properties

## Invariant

“Java-reachable state” is an invariant for all Java programs

## Note

**Not** every JRS is Java-reachable from every other JRS

- Objects cannot be deleted
- (Infinitely many) differences in attributes of non-created objects

# Java-reachable States: Declarative Definition

## Properties of Java-reachable states

# Java-reachable States: Declarative Definition

## Properties of Java-reachable states

- only finitely many objects created

# Java-reachable States: Declarative Definition

## Properties of Java-reachable states

- only finitely many objects created
- created objects are initialised

# Java-reachable States: Declarative Definition

## Properties of Java-reachable states

- **only finitely many objects created**
- **created objects are initialised**
- **attributes of created objects point to created objects  
(heap separation)**

# Java-reachable States: Declarative Definition

## Properties of Java-reachable states

- only finitely many objects created
- created objects are initialised
- attributes of created objects point to created objects  
(heap separation)

**Allows non-procedural definition of Java-reachable states**

# Java-reachable States: Declarative Definition

## Properties of Java-reachable states

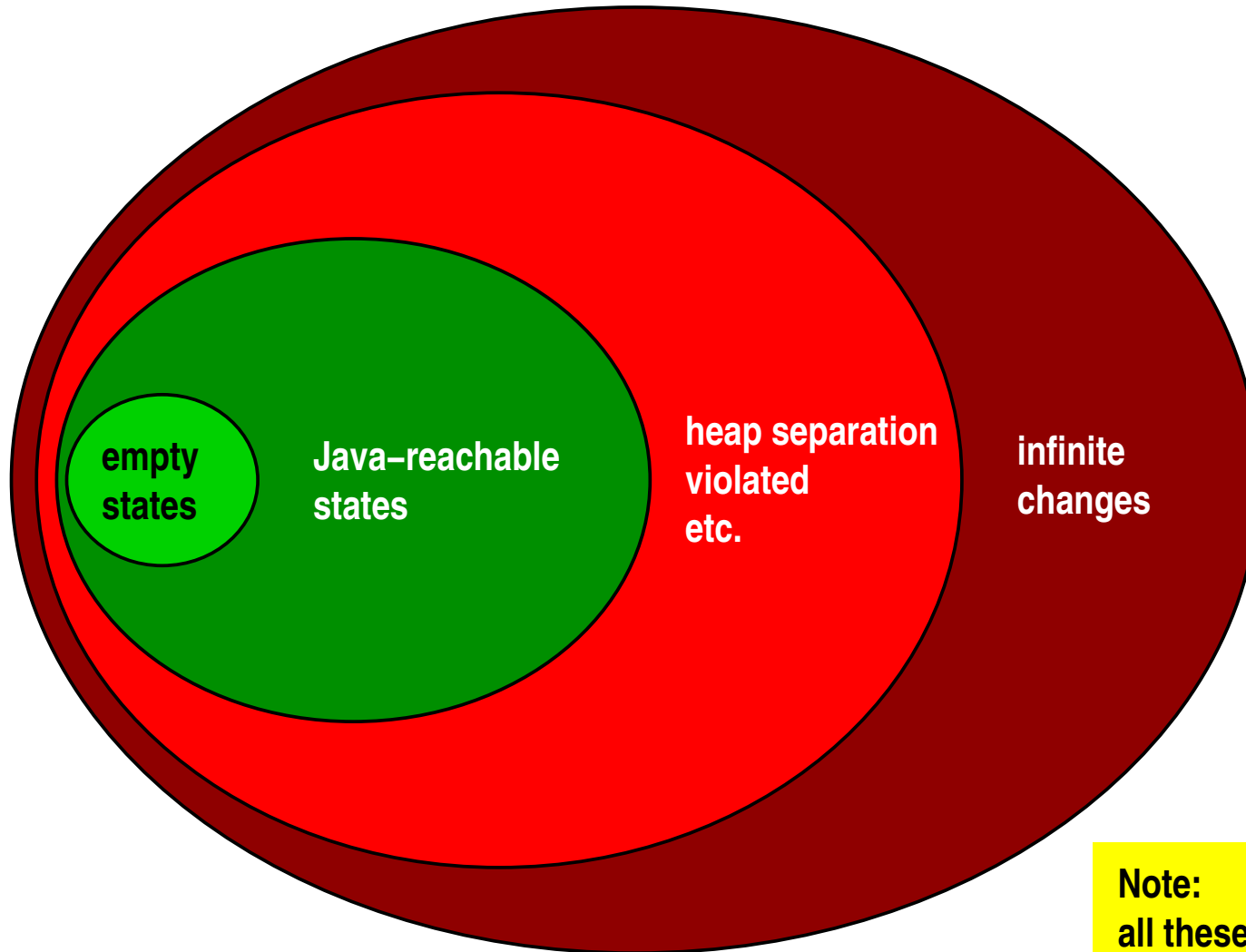
- only finitely many objects created
- created objects are initialised
- attributes of created objects point to created objects (heap separation)

Allows non-procedural definition of Java-reachable states

### Note

(Quantified) updates can destroy all these properties

# States in Java DL



**Note:**  
all these sets  
not enumerable

# Java-reachable States: Expressibility in Java DL

## Expressing “Java-reachable state” in Java DL

- The property “Java-reachable state” is expressible in Java DL
- See Part 2 of the talk

# Java-reachable States: Expressibility in Java DL

## Expressing “Java-reachable state” in Java DL

- The property “Java-reachable state” is expressible in Java DL
- See Part 2 of the talk

There is a Java DL formula  $JRS$  that is

- true in all Java-reachable states
- false in all other states

# Java-reachable States: Expressibility in Java DL

## Expressing “Java-reachable state” in Java DL

- The property “Java-reachable state” is expressible in Java DL
- See Part 2 of the talk

There is a Java DL formula  $JRS$  that is

- true in all Java-reachable states
- false in all other states

### Note

$JRS$  is an invariant of all Java programs

# Using the Formula *JRS*: (Bad) Idea 1

## Idea 1

- Eliminate non-JRS states from the models (by definition)

# Using the Formula $JRS$ : (Bad) Idea 1

## Idea 1

- Eliminate non- $JRS$  states from the models (by definition)
- Add a closure rule that allows to close branches containing  $\neg JRS$

# Using the Formula $JRS$ : (Bad) Idea 1

## Idea 1

- Eliminate non-JRS states from the models (by definition)
- Add a closure rule that allows to close branches containing  $\neg JRS$

## But

What about updates going to non-JRS states?

# Using the Formula $JRS$ : (Bad) Idea 1

## Idea 1

- Eliminate non-JRS states from the models (by definition)
- Add a closure rule that allows to close branches containing  $\neg JRS$

## But

What about updates going to non-JRS states?

- Cannot be checked syntactically

# Using the Formula $JRS$ : (Bad) Idea 1

## Idea 1

- Eliminate non-JRS states from the models (by definition)
- Add a closure rule that allows to close branches containing  $\neg JRS$

## But

What about updates going to non-JRS states?

- Cannot be checked syntactically
- updates may non terminate
- **BAD**

# Using the Formula $JRS$ : (Bad) Idea 1

## However

- Idea 1 can be used partially
- If the partial restriction cannot be circumvented by updates

# Using the Formula *JRS*: (Bad) Idea 1

## However

- Idea 1 can be used partially
- If the partial restriction cannot be circumvented by updates

## Restriction to finite number of created objects

- **Implicit class attribute** `nextToCreate`

# Using the Formula *JRS*: (Bad) Idea 1

## However

- Idea 1 can be used partially
- If the partial restriction cannot be circumvented by updates

## Restriction to finite number of created objects

- Implicit class attribute `nextToCreate`
- Now: Even updates cannot create infinite number of objects

# Using the Formula *JRS*: (Bad) Idea 1

## However

- Idea 1 can be used partially
- If the partial restriction cannot be circumvented by updates

## Restriction to finite number of created objects

- Implicit class attribute `nextToCreate`
- Now: Even updates cannot create infinite number of objects
- That part of *JRS* now preserved automatically

# Using the Formula $JRS$ : (Good) Idea 2

## Idea 2

Use formula  $JRS$  as an invariant

# Using the Formula $JRS$ : (Good) Idea 2

## Idea 2

Use formula  $JRS$  as an invariant

- can be used in specifications
- can be used in proofs

# Using the Formula $JRS$ : (Good) Idea 2

## Idea 2

Use formula  $JRS$  as an invariant

- can be used in specifications
- can be used in proofs
- Additional proof obligation:  $JRS$  is in fact an invariant

# Using the Formula *JRS*: (Good) Idea 2

## Good idea because

- semantics of Java DL unchanged

# Using the Formula *JRS*: (Good) Idea 2

## Good idea because

- semantics of Java DL unchanged
- calculus unchanged

(additional rule needed expressing that *JRS* is preserved by anonymous updates representing Java programs)

# Using the Formula *JRS*: (Good) Idea 2

## Good idea because

- semantics of Java DL unchanged
- calculus unchanged  
(additional rule needed expressing that *JRS* is preserved by anonymous updates representing Java programs)
- only in proofs where *JRS* is needed, does it have to be considered  
(possibility of non-terminating proofs would affect every proof)