

# Dynamic Logic with Non-rigid Functions

## A Basis for Object-oriented Program Verification

Bernhard Beckert<sup>1</sup>   André Platzer<sup>2</sup>

`www.key-project.org`

<sup>1</sup>University of Koblenz-Landau, Department of Computer Science  
`beckert@uni-koblenz.de`

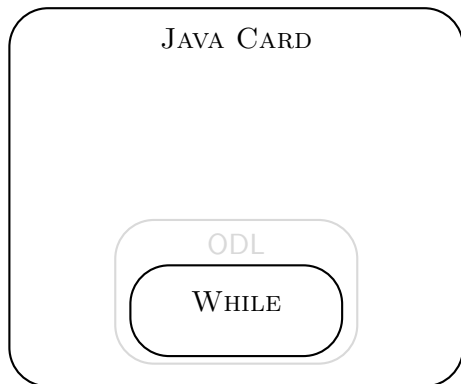
<sup>2</sup>University of Oldenburg, Department of Computing Science  
`platzer@informatik.uni-oldenburg.de`

ESF Exploratory Workshop

*Challenges in Java Program Verification*

Nijmegen, 16–18. October 2006

# Object-oriented Dynamic Logic



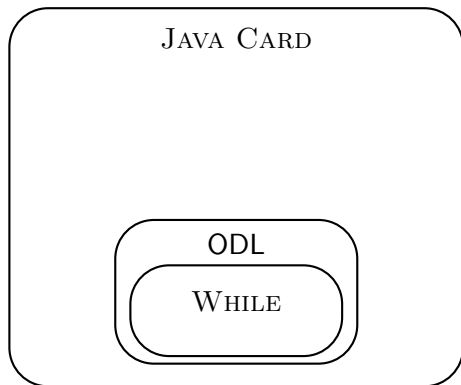
## Purpose

- ✓ Theoretical investigations
- ✓ Program verification

## Essential features of ODL

- Object type system
- Object creation
- Non-rigid functions

# Object-oriented Dynamic Logic



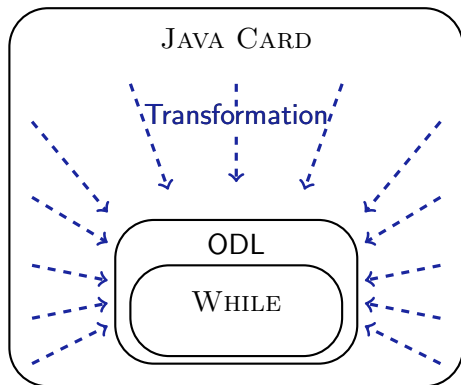
## Purpose

- ✓ Theoretical investigations
- ✓ Program verification

## Essential features of ODL

- Object type system
- Object creation
- Non-rigid functions

# Object-oriented Dynamic Logic



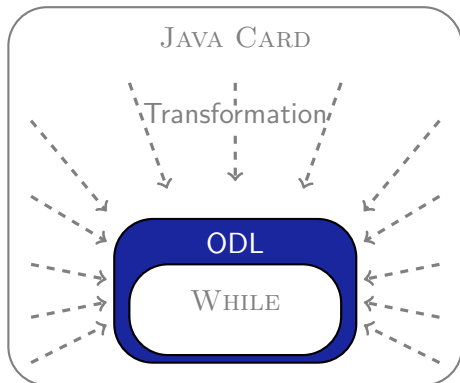
## Purpose

- ✓ Theoretical investigations
- ✓ Program verification

## Essential features of ODL

- Object type system
- Object creation
- Non-rigid functions

# Object-oriented Dynamic Logic



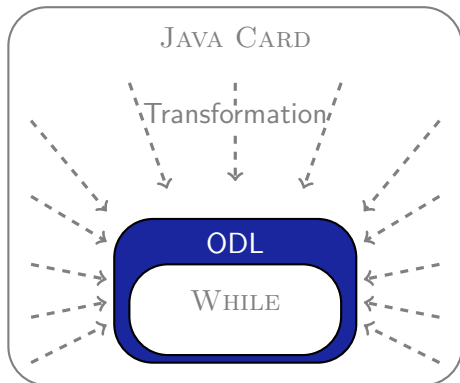
## Purpose

- ✓ Theoretical investigations
- ✓ Program verification

## Essential features of ODL

- Object type system
- Object creation
- Non-rigid functions

# Object-oriented Dynamic Logic



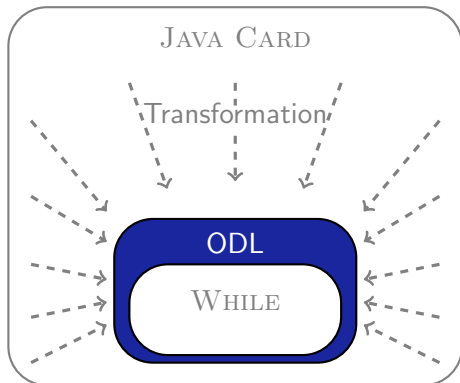
## Purpose

- ✓ Theoretical investigations
- ✓ Program verification

## Essential features of ODL

- Object type system
- Object creation
- Non-rigid functions

# Object-oriented Dynamic Logic



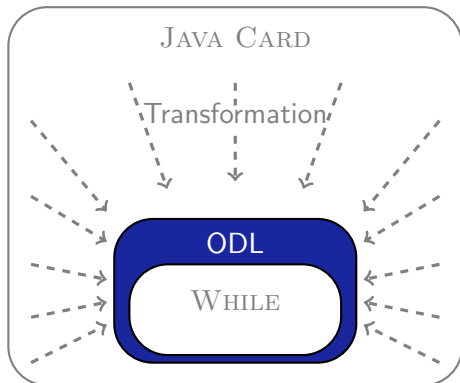
## Purpose

- ✓ Theoretical investigations
- ✓ Program verification

## Essential features of ODL

- Object type system
- **Object creation**
- Non-rigid functions

# Object-oriented Dynamic Logic



## Purpose

- ✓ Theoretical investigations
- ✓ Program verification

## Essential features of ODL

- Object type system
- Object creation
- **Non-rigid functions**

# Outline

- 1 Motivation
- 2 Dynamic Logic
- 3 ODL and its Essential Features
- 4 Java  $\rightsquigarrow$  ODL
- 5 Calculus
- 6 Conclusions

# Outline

- 1 Motivation
- 2 Dynamic Logic**
- 3 ODL and its Essential Features
- 4 JAVA  $\rightsquigarrow$  ODL
- 5 Calculus
- 6 Conclusions

# Dynamic Logic: Syntax and Semantics

## Syntax

- Basis: (Typed) first-order predicate logic
- Modal operators  $\langle p \rangle$  and  $[p]$  for each program  $p$

## Semantics

- Operators refer to the final state of  $p$
- $[p]F$ : If  $p$  terminates, then  $F$  holds in the final state  
(partial correctness)
- $\langle p \rangle F$ :  $p$  terminates and  $F$  holds in the final state  
(total correctness)

Formulas contain unaltered program source code

# Dynamic Logic: Syntax and Semantics

## Syntax

- Basis: (Typed) first-order predicate logic
- Modal operators  $\langle p \rangle$  and  $[p]$  for each program  $p$

## Semantics

- Operators refer to the final state of  $p$
- $[p]F$ : If  $p$  terminates, then  $F$  holds in the final state  
(partial correctness)
- $\langle p \rangle F$ :  $p$  terminates and  $F$  holds in the final state  
(total correctness)

Formulas contain unaltered program source code

# Dynamic Logic: Syntax and Semantics

## Syntax

- Basis: (Typed) first-order predicate logic
- Modal operators  $\langle p \rangle$  and  $[p]$  for each program  $p$

## Semantics

- Operators refer to the final state of  $p$
- $[p]F$ : If  $p$  terminates, then  $F$  holds in the final state  
(partial correctness)
- $\langle p \rangle F$ :  $p$  terminates and  $F$  holds in the final state  
(total correctness)

Formulas contain unaltered program source code

# Dynamic Logic: Syntax and Semantics

## Syntax

- Basis: (Typed) first-order predicate logic
- Modal operators  $\langle p \rangle$  and  $[p]$  for each program  $p$

## Semantics

- Operators refer to the final state of  $p$
- $[p]F$ : If  $p$  terminates, then  $F$  holds in the final state  
(partial correctness)
- $\langle p \rangle F$ :  $p$  terminates and  $F$  holds in the final state  
(total correctness)

Formulas contain unaltered program source code

# Dynamic Logic: Syntax and Semantics

## Syntax

- Basis: (Typed) first-order predicate logic
- Modal operators  $\langle p \rangle$  and  $[p]$  for each program  $p$

## Semantics

- Operators refer to the final state of  $p$
- $[p]F$ : If  $p$  terminates, then  $F$  holds in the final state  
(partial correctness)
- $\langle p \rangle F$ :  $p$  terminates and  $F$  holds in the final state  
(total correctness)

Formulas contain unaltered program source code

# Dynamic Logic Example Formulas

$$(balance > 1 \wedge amount > 1) \rightarrow \langle charge(amount); \rangle (balance > 1)$$
$$\langle x := 1; \rangle ([while(x \doteq 1);] false)$$

- Program formulas can appear nested

$$\forall val ((\langle p \rangle x \doteq val) \leftrightarrow (\langle q \rangle x \doteq val))$$

- $p, q$  **equivalent** relative to computation state restricted to  $x$

# Dynamic Logic Example Formulas

$$(balance > 1 \wedge amount > 1) \rightarrow \langle charge(amount); \rangle (balance > 1)$$
$$\langle x := 1; \rangle ([while(x \doteq 1);] false)$$

- Program formulas can appear nested

$$\forall val ((\langle p \rangle x \doteq val) \leftrightarrow (\langle q \rangle x \doteq val))$$

- $p, q$  **equivalent** relative to computation state restricted to  $x$

# Dynamic Logic Example Formulas

$$(balance > 1 \wedge amount > 1) \rightarrow \langle charge(amount); \rangle (balance > 1)$$
$$\langle x := 1; \rangle ([while(x \doteq 1);] false)$$

- Program formulas can appear nested

$$\forall val ((\langle p \rangle x \doteq val) \leftrightarrow (\langle q \rangle x \doteq val))$$

- $p, q$  **equivalent** relative to computation state restricted to  $x$

# Why Dynamic Logic?

Application-specific

Universal

Type system

Static Analysis

Dynamic Logic

Hoare Logic

Logical Framework

HOL

- Transparency wrt target programming language, no encoding
- More expressive and flexible than Hoare logic
- Rule for each program construct in calculus
- Symbolic execution is a natural interactive proof paradigm
- Proven technology that scales up

# Why Dynamic Logic?

Application-specific

Universal

Type system

Static Analysis

Dynamic Logic

Hoare Logic

Logical Framework

HOL

- Transparency wrt target programming language, no encoding
- More expressive and flexible than Hoare logic
- Rule for each program construct in calculus
- Symbolic execution is a natural **interactive** proof paradigm
- Proven technology that scales up

# Why Dynamic Logic?

Application-specific

Universal

Type system

Dynamic Logic

Logical Framework

Static Analysis

Hoare Logic

HOL

- Transparency wrt target programming language, no encoding
- **More expressive and flexible than Hoare logic**
- Rule for each program construct in calculus
- Symbolic execution is a natural **interactive** proof paradigm
- Proven technology that scales up

# Why Dynamic Logic?

Application-specific

Universal

Type system

Dynamic Logic

Logical Framework

Static Analysis

Hoare Logic

HOL

- Transparency wrt target programming language, no encoding
- More expressive and flexible than Hoare logic
- **Rule for each program construct in calculus**
- Symbolic execution is a natural **interactive** proof paradigm
- Proven technology that scales up

# Why Dynamic Logic?

Application-specific

Universal

Type system

Dynamic Logic

Logical Framework

Static Analysis

Hoare Logic

HOL

- Transparency wrt target programming language, no encoding
- More expressive and flexible than Hoare logic
- Rule for each program construct in calculus
- **Symbolic execution is a natural interactive proof paradigm**
- Proven technology that scales up

# Why Dynamic Logic?

Application-specific

Universal

Type system

Dynamic Logic

Logical Framework

Static Analysis

Hoare Logic

HOL

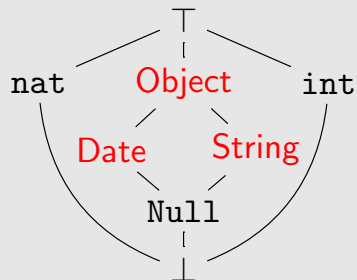
- Transparency wrt target programming language, no encoding
- More expressive and flexible than Hoare logic
- Rule for each program construct in calculus
- Symbolic execution is a natural **interactive** proof paradigm
- **Proven technology that scales up**

# Outline

- 1 Motivation
- 2 Dynamic Logic
- 3 ODL and its Essential Features**
- 4 JAVA  $\rightsquigarrow$  ODL
- 5 Calculus
- 6 Conclusions

# Essential ODL Feature 1: Type System

## Type system



- built-in types
- object types (finitely many)

# Essential ODL Feature 2: Object Creation

## Constant domain assumption

All objects “exist” from the beginning

## Object enumerator symbols

$\text{obj}_C : \text{nat} \rightarrow C$  (disjoint bijections for object-types  $C$ )

## Implicit static field $\text{next}_C$

$\text{next}_C : \rightarrow \text{nat}$  (index of next object to be created)

# Essential ODL Feature 2: Object Creation

## Constant domain assumption

All objects “exist” from the beginning

## Object enumerator symbols

$\text{obj}_C : \text{nat} \rightarrow C$  (disjoint bijections for object-types  $C$ )

## Implicit static field $\text{next}_C$

$\text{next}_C : \rightarrow \text{nat}$  (index of next object to be created)

# Essential ODL Feature 2: Object Creation

## Constant domain assumption

All objects “exist” from the beginning

## Object enumerator symbols

$\text{obj}_C : \text{nat} \rightarrow C$  (disjoint bijections for object-types  $C$ )

## Implicit static field $\text{next}_C$

$\text{next}_C : \rightarrow \text{nat}$  (index of next object to be created)

# Essential ODL Feature 3: Non-rigid Symbols

## Non-rigid function symbols

- Value can change during program execution
- Represent object attributes

## Example

$$x.a = 3 \rightsquigarrow a(x) := 3$$

# Essential ODL Feature 3: Non-rigid Symbols

## Non-rigid function symbols

- Value can change during program execution
- Represent object attributes

## Example

$$\boxed{x.a = 3} \rightsquigarrow \boxed{a(x) := 3}$$

# The ODL Programming Language

## Programming constructs

$\alpha; \gamma$	concatenation
$\text{if}(\phi) \alpha \text{ else } \gamma$	conditional
$\text{while}(\phi) \alpha$	loop
$f(t) := t', g(r) := r'$	atomic (parallel) state update

Also: Extension for method invocation

[▶ Details](#)

# Outline

- 1 Motivation
- 2 Dynamic Logic
- 3 ODL and its Essential Features
- 4 Java  $\rightsquigarrow$  ODL**
- 5 Calculus
- 6 Conclusions

## Simple & natural translation of

- Object creation
- Dynamic dispatch
- Side-effects
- Exception handling



# Java $\rightsquigarrow$ ODL: Object Creation

## Transforming “new”

$$x = \text{new } C()$$
 $\rightsquigarrow$ 
$$\begin{aligned} x &:= \text{obj}_C(\text{next}_C), \\ \text{next}_C &:= \text{next}_C + 1 \end{aligned}$$

## Transforming “instanceof”

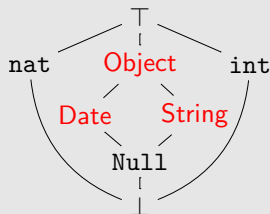
$$t \text{ instanceof String}$$
 $\Downarrow$ 
$$\exists n : \text{nat } t \doteq \text{obj}_{\text{String}}(n)$$

# Java $\rightsquigarrow$ ODL: Object Creation

## Transforming “new”

$$x = \text{new } C()$$
 $\rightsquigarrow$ 
$$x := \text{obj}_C(\text{next}_C), \\ \text{next}_C := \text{next}_C + 1$$

## Transforming “instanceof”


$$t \text{ instanceof String}$$
 $\Downarrow$ 
$$\exists n : \text{nat } t \doteq \text{obj}_{\text{String}}(n)$$

# Outline

- 1 Motivation
- 2 Dynamic Logic
- 3 ODL and its Essential Features
- 4 JAVA  $\rightsquigarrow$  ODL
- 5 Calculus**
- 6 Conclusions

# Calculus: State Updates

## Merge updates

$$\langle \mathcal{U} \rangle \langle f(t) := t' \rangle \phi \rightsquigarrow \langle \mathcal{U}, f(\langle \mathcal{U} \rangle t) := \langle \mathcal{U} \rangle t' \rangle \phi$$

Example:

$$\langle g(a) := t \rangle \langle f(g(a)) := h(g(a)) \rangle \phi \rightsquigarrow \langle g(a) := t, f(t) := h(t) \rangle \phi$$

## Apply updates

# Calculus: State Updates

## Merge updates

$$\langle \mathcal{U} \rangle \langle f(t) := t' \rangle \phi \rightsquigarrow \langle \mathcal{U}, f(\langle \mathcal{U} \rangle t) := \langle \mathcal{U} \rangle t' \rangle \phi$$

Example:

$$\langle g(a) := t \rangle \langle f(g(a)) := h(g(a)) \rangle \phi \rightsquigarrow \langle g(a) := t, f(t) := h(t) \rangle \phi$$

## Apply updates

$$\langle f(t) := t', g(t) := c, f(s) := s' \rangle f(a)$$

# Calculus: State Updates

## Merge updates

$$\langle \mathcal{U} \rangle \langle f(t) := t' \rangle \phi \rightsquigarrow \langle \mathcal{U}, f(\langle \mathcal{U} \rangle t) := \langle \mathcal{U} \rangle t' \rangle \phi$$

Example:

$$\langle g(a) := t \rangle \langle f(g(a)) := h(g(a)) \rangle \phi \rightsquigarrow \langle g(a) := t, f(t) := h(t) \rangle \phi$$

## Apply updates

$$\frac{\langle f(t) := t', g(t) := c, f(s) := s' \rangle f(a)}{f}$$
$$\langle f(t) := t', f(s) := s' \rangle$$

# Calculus: State Updates

## Merge updates

$$\langle \mathcal{U} \rangle \langle f(t) := t' \rangle \phi \rightsquigarrow \langle \mathcal{U}, f(\langle \mathcal{U} \rangle t) := \langle \mathcal{U} \rangle t' \rangle \phi$$

Example:

$$\langle g(a) := t \rangle \langle f(g(a)) := h(g(a)) \rangle \phi \rightsquigarrow \langle g(a) := t, f(t) := h(t) \rangle \phi$$

## Apply updates

$$\frac{\langle f(t) := t', g(t) := c, f(s) := s' \rangle f(a)}{f} \langle f(t) := t', f(s) := s' \rangle f(a)$$

# Calculus: State Updates

## Merge updates

$$\langle \mathcal{U} \rangle \langle f(t) := t' \rangle \phi \rightsquigarrow \langle \mathcal{U}, f(\langle \mathcal{U} \rangle t) := \langle \mathcal{U} \rangle t' \rangle \phi$$

Example:

$$\langle g(a) := t \rangle \langle f(g(a)) := h(g(a)) \rangle \phi \rightsquigarrow \langle g(a) := t, f(t) := h(t) \rangle \phi$$

## Apply updates

$$\langle f(t) := t', g(t) := c, f(s) := s' \rangle f(a)$$

$$\begin{array}{c} \langle f(t) := t', g(t) := c, f(s) := s' \rangle f(a) \\ \text{--- } f \text{ ---} \\ \langle f(t) := t', f(s) := s' \rangle f(a) \end{array}$$

$if\ s \doteq a\ then\ s' \ else \ \dots \ fi$

# Calculus: State Updates

## Merge updates

$$\langle \mathcal{U} \rangle \langle f(t) := t' \rangle \phi \rightsquigarrow \langle \mathcal{U}, f(\langle \mathcal{U} \rangle t) := \langle \mathcal{U} \rangle t' \rangle \phi$$

Example:

$$\langle g(a) := t \rangle \langle f(g(a)) := h(g(a)) \rangle \phi \rightsquigarrow \langle g(a) := t, f(t) := h(t) \rangle \phi$$

## Apply updates

$$\langle f(t) := t', g(t) := c, f(s) := s' \rangle f(a)$$

$$\begin{array}{c} \langle f(t) := t', g(t) := c, f(s) := s' \rangle f(a) \\ \hline f \\ \langle f(t) := t', f(s) := s' \rangle f(a) \end{array}$$

$$\text{if } s \doteq a \text{ then } s' \text{ else if } t \doteq a \text{ then } t' \text{ else } f(a) \text{ fi fi}$$

# Calculus: State Updates

## Merge updates

$$\langle \mathcal{U} \rangle \langle f(t) := t' \rangle \phi \rightsquigarrow \langle \mathcal{U}, f(\langle \mathcal{U} \rangle t) := \langle \mathcal{U} \rangle t' \rangle \phi$$

Example:

$$\langle g(a) := t \rangle \langle f(g(a)) := h(g(a)) \rangle \phi \rightsquigarrow \langle g(a) := t, f(t) := h(t) \rangle \phi$$

## Apply updates

$$\begin{array}{c} \mathcal{U} \\ \overline{\langle f(t) := t', g(t) := c, f(s) := s' \rangle} \quad f(a) \\ \underbrace{\hspace{10em}}_f \\ \langle f(t) := t', f(s) := s' \rangle \quad f(a) \\ \underbrace{\hspace{10em}} \end{array}$$

*if*  $s \doteq \langle \mathcal{U} \rangle a$  *then*  $s'$  *else if*  $t \doteq \langle \mathcal{U} \rangle a$  *then*  $t'$  *else*  $f(\langle \mathcal{U} \rangle a)$  *fi fi*

# Calculus: First-order Part

## 18 rules

$$\frac{\vdash A}{\neg A \vdash}$$

$$\frac{A, B \vdash}{A \wedge B \vdash}$$

$$\frac{A \vdash \quad B \vdash}{A \vee B \vdash}$$

$$\frac{\vdash A \quad B \vdash}{A \rightarrow B \vdash}$$

$$\frac{A \vdash}{\vdash \neg A}$$

$$\frac{\vdash A \quad \vdash B}{\vdash A \wedge B}$$

$$\frac{\vdash A, B}{\vdash A \vee B}$$

$$\frac{A \vdash B}{\vdash A \rightarrow B}$$

$$\frac{\vdash A_x^X}{\vdash \forall x A}$$

$$\frac{A_x^t, \forall x A \vdash}{\forall x A \vdash}$$

$$\frac{A_x^X \vdash}{\exists x A \vdash}$$

$$\frac{\vdash A_x^t, \exists x A}{\vdash \exists x A}$$

$$\frac{}{A \vdash A}$$

$$\frac{\Gamma_t', t \doteq t' \vdash \Delta_t'}{\Gamma, t \doteq t' \vdash \Delta}$$

$$\frac{}{\vdash t \doteq t}$$

$$\frac{A \vdash \quad \vdash A}{\vdash}$$

$$\frac{\Gamma_t', t' \doteq t \vdash \Delta_t'}{\Gamma, t' \doteq t \vdash \Delta}$$

$$\frac{\vdash \phi(0) \quad \phi(X) \vdash \phi(X+1)}{\vdash \forall n \phi(n)}$$

# Calculus: Program Logic Part

## 12 rules

$$\frac{\langle \alpha \rangle \langle \gamma \rangle \phi}{\langle \alpha; \gamma \rangle \phi}$$

$$\frac{e \rightarrow \langle \alpha \rangle \phi \wedge (\neg e \rightarrow \langle \gamma \rangle \phi)}{\langle \text{if}(e) \alpha \text{ else } \gamma \rangle \phi}$$

$$\frac{e \rightarrow \phi(t) \wedge (\neg e \rightarrow \phi(t'))}{\phi(\text{if } e \text{ then } t \text{ else } t')}$$

$$\frac{\langle \text{if}(e) \{ \alpha; \text{while}(e) \alpha \} \rangle \phi}{\langle \text{while}(e) \alpha \rangle \phi}$$

$$\frac{A \vdash B}{\exists x A \vdash \exists x B}$$

$$\langle \mathcal{U} \rangle f(u) \rightsquigarrow \text{if } s_{i_r} \doteq \langle \mathcal{U} \rangle u \text{ then } t_{i_r} \text{ else } \dots \text{if } s_{i_1} \doteq \langle \mathcal{U} \rangle u \text{ then } t_{i_1} \text{ else } f(\langle \mathcal{U} \rangle u) \text{ fi } \dots \text{fi}$$

$$\langle \vec{\mathcal{U}} \rangle \langle \mathcal{U} \rangle \phi \rightsquigarrow \langle \vec{\mathcal{U}}, f_1(\langle \vec{\mathcal{U}} \rangle s_1) := \langle \vec{\mathcal{U}} \rangle t_1, \dots, f_n(\langle \vec{\mathcal{U}} \rangle s_n) := \langle \vec{\mathcal{U}} \rangle t_n \rangle \phi$$

$$\frac{}{\vdash \text{obj}_C(i) \doteq \text{obj}_C(j) \rightarrow i \doteq j}$$

$$\frac{}{\vdash \neg(\text{obj}_C(i) \doteq \text{obj}_D(j))}$$

$$\frac{}{\vdash \forall o : C (o \text{ instance of } C \vee o \doteq \text{null})}$$

$$\frac{\Gamma \vdash \langle \mathcal{U} \rangle p, \Delta \quad p, e \vdash [\alpha] p \quad p, \neg e \vdash \phi}{\Gamma \vdash \langle \mathcal{U} \rangle [\text{while}(e) \alpha] \phi, \Delta}$$

$$\frac{A \vdash B}{\langle \alpha \rangle A \vdash \langle \alpha \rangle B}$$

# Verification Example

```
class E { static int g; int id;  
  E generate() {E r=new E(); r.id=g;g=g+5; return r;}}
```

$$\forall x:E (x.id < g \rightarrow [\text{generate}] (x.id < r.id))$$

# Verification Example

```
class E { static int g; int id;
  E generate() {E r=new E(); r.id=g;g=g+5; return r;}}
```

$$\forall x:E (x.id < g \rightarrow [\text{generate}] (x.id < r.id))$$

$*$	$\dots$
$\frac{X.id < g, \neg o(n) \doteq X}{\vdash X.id < g}$	$\frac{X.id < g, o(n) \doteq X}{\vdash g < g}$
$\frac{X.id < g \quad \vdash (\neg o(n) \doteq X \rightarrow X.id < g) \wedge (o(n) \doteq X \rightarrow g < g)}{X.id < g \quad \vdash (ifo(n) \doteq X \text{ then } g \text{ else } X.id \text{ fi}) < g}$	
$\frac{X.id < g \quad \vdash \langle r := o(n), n := n+1, o(n).id := g, g := g+5 \rangle (X.id < r.id)}{X.id < g \quad \vdash \langle r := o(n), n := n+1, o(n).id := g \rangle [g := g+5] (X.id < r.id)}$	
$\frac{X.id < g \quad \vdash \langle r := o(n), n := n+1 \rangle [r.id := g] [g := g+5] (X.id < r.id)}{X.id < g \quad \vdash \langle r := o(n), n := n+1 \rangle [r.id := g; g := g+5] (X.id < r.id)}$	
$\frac{X.id < g \quad \vdash [\alpha] (X.id < r.id)}{\vdash X.id < g \rightarrow [\alpha] (X.id < r.id)}$	
$\frac{\vdash X.id < g \rightarrow [\alpha] (X.id < r.id)}{\vdash \forall x:E (x.id < g \rightarrow [\alpha] (x.id < r.id))}$	

# Soundness and Relative Completeness

## Theorem (Soundness)

*The ODL calculus is sound:*

$$\vdash \phi \text{ implies } \models \phi$$

## Theorem (Relative completeness)

*The ODL calculus is complete wrt first-order arithmetic:*

$$\models \phi \text{ implies } \text{Taut}_{\text{Arith}} \vdash \phi$$

► Proof Outline

# Soundness and Relative Completeness

## Theorem (Soundness)

*The ODL calculus is sound:*

$$\vdash \phi \text{ implies } \models \phi$$

## Theorem (Relative completeness)

*The ODL calculus is complete wrt first-order arithmetic:*

$$\models \phi \text{ implies } \text{Taut}_{\text{Arith}} \vdash \phi$$

▶ Proof Outline

# Outline

- 1 Motivation
- 2 Dynamic Logic
- 3 ODL and its Essential Features
- 4 JAVA  $\rightsquigarrow$  ODL
- 5 Calculus
- 6 Conclusions**

# Conclusions

- ODL is essentials-only object-oriented dynamic logic:
  - 1 object type system
  - 2 object enumerators
  - 3 non-rigid functions
- Natural translation  $JAVA \rightsquigarrow ODL$
- Calculus proven sound & relatively complete wrt arithmetic
- Small number of rules
  - ✓ Theoretical investigations
  - ✓ Program verification

# Conclusions

- ODL is essentials-only object-oriented dynamic logic:
  - 1 object type system
  - 2 object enumerators
  - 3 non-rigid functions
- Natural translation  $JAVA \rightsquigarrow ODL$
- Calculus proven sound & relatively complete wrt arithmetic
- Small number of rules
  - ✓ Theoretical investigations
  - ✓ Program verification

# Conclusions

- ODL is essentials-only object-oriented dynamic logic:
  - 1 object type system
  - 2 object enumerators
  - 3 non-rigid functions
- Natural translation  $JAVA \rightsquigarrow ODL$
- Calculus proven sound & relatively complete wrt arithmetic
- Small number of rules
  - ✓ Theoretical investigations
  - ✓ Program verification

# Conclusions

- ODL is essentials-only object-oriented dynamic logic:
  - 1 object type system
  - 2 object enumerators
  - 3 non-rigid functions
- Natural translation  $JAVA \rightsquigarrow ODL$
- Calculus proven sound & relatively complete wrt arithmetic
- Small number of rules
  - ✓ Theoretical investigations
  - ✓ Program verification

# Conclusions

- ODL is essentials-only object-oriented dynamic logic:
  - 1 object type system
  - 2 object enumerators
  - 3 **non-rigid functions**
- Natural translation  $JAVA \rightsquigarrow ODL$
- Calculus proven sound & relatively complete wrt arithmetic
- **Small number of rules**
  - ✓ Theoretical investigations
  - ✓ Program verification

# Conclusions

- ODL is essentials-only object-oriented dynamic logic:
  - 1 object type system
  - 2 object enumerators
  - 3 **non-rigid functions**
- Natural translation  $JAVA \rightsquigarrow ODL$
- Calculus proven sound & relatively complete wrt arithmetic
- Small number of rules
  - ✓ Theoretical investigations
  - ✓ Program verification



# Outline

## 7 The Logic ODL (Details)

- Syntax
- Semantics

## 8 Transformation (Details)

- Object Creation
- Side-effects
- Exception Handling
- Dynamic Dispatch

## 9 Calculus (Details)

- Calculus of Object Creation - Details
- Calculus of State Updates - Details
- Completeness Proof
- Misc

# The Logic ODL: Syntax

## Definition (Type system)

Type lattice with natural numbers  $\mathbf{N}$  and finite subtypes

# The Logic ODL: Semantics

## Definition (Semantics of programs)

- $(s, s') \in \rho_{I,\beta}(f_1(t_1^1, \dots, t_1^{k_1}) := t_1, \dots, f_n(t_n^1, \dots, t_n^{k_n}) := t_n) \iff$ 
  - $s = s_0, s' = s_n$ , and
  - $s_i = s_{i-1}[f_i(\text{val}_{I,\beta}(s, t_i^1), \dots, \text{val}_{I,\beta}(s, t_i^{k_i})) \mapsto \text{val}_{I,\beta}(s, t_i)]$  ( $1 \leq i \leq n$ ).
- $(s, s') \in \rho_{I,\beta}(\alpha; \gamma) \iff (s, u) \in \rho_{I,\beta}(\alpha)$  and  $(u, s') \in \rho_{I,\beta}(\gamma)$  for some state  $u$ .
- $(s, s') \in \rho_{I,\beta}(\text{if}(\phi) \alpha \text{ else } \gamma) \iff$ 
  - $\text{val}_{I,\beta}(s, \phi) = \text{true}$  and  $(s, s') \in \rho_{I,\beta}(\alpha)$ , or
  - $\text{val}_{I,\beta}(s, \phi) = \text{false}$  and  $(s, s') \in \rho_{I,\beta}(\gamma)$ .
- $(s, s') \in \rho_{I,\beta}(\text{while}(\phi) \alpha)$  iff there are  $n \in \mathbb{N}$  and  $s = s_0, \dots, s_n = s'$ 
  - for  $0 \leq i < n$ ,  $\text{val}_{I,\beta}(s_i, \phi) = \text{true}$  and  $(s_i, s_{i+1}) \in \rho_{I,\beta}(\alpha)$ , and
  - $\text{val}_{I,\beta}(s_n, \phi) = \text{false}$ .

# Outline

## 7 The Logic ODL (Details)

- Syntax
- Semantics

## 8 Transformation (Details)

- Object Creation
- Side-effects
- Exception Handling
- Dynamic Dispatch

## 9 Calculus (Details)

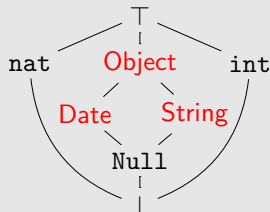
- Calculus of Object Creation - Details
- Calculus of State Updates - Details
- Completeness Proof
- Misc

# Java $\rightsquigarrow$ ODL: Object Creation

## Transforming “new”

$$x = \text{new } C()$$
 $\rightsquigarrow$ 
$$x := \text{obj}_C(\text{next}_C), \\ \text{next}_C := \text{next}_C + 1$$

## Transforming “instanceof”


$$t \text{ instanceof String}$$
 $\Downarrow$ 
$$\exists n : \text{nat } t \doteq \text{obj}_{\text{String}}(n)$$

# Java $\rightsquigarrow$ ODL: Side-effects

$$a[i++] = b - 1 + b$$
$$vi := i; i := i + 1; vb := b; b := b - 1; a(vi) := vb + b$$
$$i := i + 1, b := b - 1, a(i) := b + (b - 1)$$

Return

# Java $\rightsquigarrow$ ODL: Exception Handling

```
try { while (d != 0)
      {if (d < 0) {throw new RangeEx(d);} else {d=d-1;}}
  /* do something */
} catch (RangeEx r) {/* handle range */}
```

}  
↓

```
Exception r = null;
while (r == null && d != 0)
  {if (d < 0) {r = new RangeEx(d);} else {d=d-1;}}
if (r == null) {/* do something */}
else if (r instanceof RangeEx) {/* handle range */}
else {return r; /* pass up the call trace */}
```

Return

# Java $\rightsquigarrow$ ODL: Dynamic Dispatch

```
class Car { int follow(Car d) {...} }  
class Van extends Car { int follow(Car d) {...} }  
... return b.follow(d);
```

}

```
class Car { int Car_follow(Car d) {...} }  
class Van extends Car { int Van_follow(Car d) {...} }  
... if(b instanceof Van) {return ((Van)b).Van_follow(d);};  
else if(b instanceof Car) {return ((Car)b).Car_follow(d);};  
else {/* cannot happen when all types are known */}
```

Type-casts expressible as

$$\exists v: \text{Van } v \doteq b$$

Return

# Outline

## 7 The Logic ODL (Details)

- Syntax
- Semantics

## 8 Transformation (Details)

- Object Creation
- Side-effects
- Exception Handling
- Dynamic Dispatch

## 9 **Calculus (Details)**

- Calculus of Object Creation - Details
- Calculus of State Updates - Details
- Completeness Proof
- Misc

# Calculus: Object Creation

$$t \text{ instanceof } C := \exists n:\text{nat} \bigvee_{\text{Null} < \tau \leq C} t \doteq \text{obj}_\tau(n)$$

$$(R31) \frac{}{\vdash \forall o:C (o \text{ instanceof } C \vee o \doteq \text{null})}$$

$$(R32) \frac{}{\vdash \text{obj}_C(i) \doteq \text{obj}_C(j) \rightarrow i \doteq j}$$

# Calculus: State Updates

## Merge updates

$$\langle \mathcal{U} \rangle \langle f(t) := t' \rangle \phi \rightsquigarrow \langle \mathcal{U}, f(\langle \mathcal{U} \rangle t) := \langle \mathcal{U} \rangle t' \rangle \phi$$

Example:

$$\langle g(a) := t \rangle \langle f(g(a)) := h(g(a)) \rangle \phi \rightsquigarrow \langle g(a) := t, f(t) := h(t) \rangle \phi$$

## Apply updates

$$\begin{array}{c} \overbrace{\langle f(t) := t', g(t) := c, f(s) := s' \rangle}^{\mathcal{U}} \quad f(a) \\ \underbrace{\hspace{10em}}_f \\ \langle f(t) := t', f(s) := s' \rangle \quad f(a) \\ \underbrace{\hspace{10em}} \end{array}$$

*if*  $s \doteq \langle \mathcal{U} \rangle a$  *then*  $s'$  *else if*  $t \doteq \langle \mathcal{U} \rangle a$  *then*  $t'$  *else*  $f(\langle \mathcal{U} \rangle a)$  *fi fi*

# Calculus: State Updates

## Merge updates

$$\langle \mathcal{U} \rangle \langle f(t) := t' \rangle \phi \rightsquigarrow \langle \mathcal{U}, f(\langle \mathcal{U} \rangle t) := \langle \mathcal{U} \rangle t' \rangle \phi$$

Example:

$$\langle g(a) := t \rangle \langle f(g(a)) := h(g(a)) \rangle \phi \rightsquigarrow \langle g(a) := t, f(t) := h(t) \rangle \phi$$

## Apply updates

$$\begin{array}{l} \phi \left( \overbrace{\langle f(t) := t', g(t) := c, f(s) := s' \rangle}^{\mathcal{U}} f(a) \right) \\ \phi \left( \underbrace{\langle f(t) := t', f(s) := s' \rangle}_f f(a) \right) \\ \phi \left( \underbrace{\text{if } s \doteq \langle \mathcal{U} \rangle a \text{ then } s' \text{ else if } t \doteq \langle \mathcal{U} \rangle a \text{ then } t' \text{ else } f(\langle \mathcal{U} \rangle a)}_{f \text{ fi}} \text{ fi} \right) \end{array}$$

# Calculus: State Updates

## Merge updates

$$\langle \mathcal{U} \rangle \langle f(t) := t' \rangle \phi \rightsquigarrow \langle \mathcal{U}, f(\langle \mathcal{U} \rangle t) := \langle \mathcal{U} \rangle t' \rangle \phi$$

Example:

$$\langle g(a) := t \rangle \langle f(g(a)) := h(g(a)) \rangle \phi \rightsquigarrow \langle g(a) := t, f(t) := h(t) \rangle \phi$$

## Apply updates

$$\begin{array}{c} \phi \left( \overbrace{\langle f(t) := t', g(t) := c, f(s) := s' \rangle}^{\mathcal{U}} f(a) \right) \\ \phi \left( \underbrace{\langle f(t) := t', f(s) := s' \rangle}_f f(a) \right) \\ \phi \left( \underbrace{\text{if } s \doteq \langle \mathcal{U} \rangle a \text{ then } s' \text{ else if } t \doteq \langle \mathcal{U} \rangle a \text{ then } t' \text{ else } f(\langle \mathcal{U} \rangle a)}_{\text{if}} \text{ fi} \right) \end{array}$$

# Calculus: State Updates

$\langle \mathcal{U} \rangle f(u) \rightsquigarrow$   
 $\text{if } s_{i_r} \doteq \langle \mathcal{U} \rangle u \text{ then } t_{i_r} \text{ else } \dots \text{if } s_{i_1} \doteq \langle \mathcal{U} \rangle u \text{ then } t_{i_1} \text{ else } f(\langle \mathcal{U} \rangle u) \text{ fi } \dots \text{fi}$   
where  $i_1 < \dots < i_r$  are all those indices with  $f_{i_j} = f$ , for some  $r \geq 0$

## Example (State updates with aliasing)

$\langle f(s) := t \rangle g(f(r))$   
 $\rightsquigarrow g(\langle f(s) := t \rangle f(r))$   
 $\rightsquigarrow g(\text{if } s \doteq r \text{ then } t \text{ else } f(r) \text{ fi})$   
“ $\rightsquigarrow$ ”  $(s \doteq r \rightarrow g(t)) \wedge$   
 $(s \neq r \rightarrow g(f(r)))$

(R33)  $\langle \tilde{u} \rangle \langle \mathcal{U} \rangle \phi \rightsquigarrow \langle \tilde{u}, f_1(\langle \tilde{u} \rangle s_1) := \langle \tilde{u} \rangle t_1, \dots, f_n(\langle \tilde{u} \rangle s_n) := \langle \tilde{u} \rangle t_n \rangle \phi$

# Relative Completeness Proof

$\models \phi$  implies  $\text{Taut}_{\mathbf{N}} \vdash \phi$

## Proof.

- 1 propositionally complete
- 2 first-order complete
- 3 first-order expressible:  $\forall \phi \exists F \in \text{FOL} \models \phi \leftrightarrow F$
- 4 term rewrites are Noetherian
- 5 (rel.) complete for first-order  $F \rightarrow \langle \alpha \rangle G$
- 6 (rel.) complete for first-order dynamic typing



Return

# Use Cases in Java Card DL

- Use ODL object enumerators for object creation.
- Use quicker and rel. complete ODL within KeY in **automatic** verification scenarios. (Trafo combined with compiler construction technology)
- Import simpler ODL calculus into `JAVA CARD DL`, for formulas that are “sufficiently” ODL.