Motivating Example

Java Program

```
class C {

    int x;
    int y;

    void m() {
        x = x+y;
    }

}
```

Motivating Example

Java Program

```
class C {

    int x;
    int y;

    void m() {
        x = x+y;
    }

}
```

Specification

```
pre:    true
post:   x = x@pre + y
```

Motivating Example

Java Program

```
class C {

    int x;
    int y;

    void m() {
        x = x+y;
    }

}
```

Specification

pre: true
post: x = x@pre + y

Proof Goal

$$\mathbf{o.x} > 1,\ \mathbf{o.y} > 1 \vdash \langle \mathbf{o.m();} \rangle (\mathbf{o.x} * \mathbf{o.y} > 1)$$

Rule for Method Invocation

Rule

$$\frac{\Gamma \vdash PRE \qquad \Gamma@pre, POST \vdash \phi}{\Gamma \vdash \langle \texttt{m();} \rangle \phi}$$

Rule for Method Invocation

Rule

$$\frac{\Gamma \vdash PRE \qquad \Gamma@pre, POST \vdash \phi}{\Gamma \vdash \langle \textbf{m();} \rangle \phi}$$

Application to Example

pre: **true**
post: **x = x@pre + y**

$$\textbf{o.x} > 1,\ \textbf{o.y} > 1 \vdash \langle \textbf{o.m();} \rangle (\textbf{o.x} * \textbf{o.y} > 1)$$

Rule for Method Invocation

Rule

$$\frac{\Gamma \vdash PRE \qquad \Gamma@pre, POST \vdash \phi}{\Gamma \vdash \langle \textbf{m();} \rangle \phi}$$

Application to Example

pre: true
post: x = x@pre + y

$$\frac{\ldots \vdash \textbf{true} \qquad \begin{array}{l}\textbf{o.x}@pre > 1, \textbf{o.y}@pre > 1,\\ \textbf{o.x} = \textbf{o.x}@pre + \textbf{o.y}\end{array} \qquad \vdash \textbf{o.x} * \textbf{o.y} > 1}{\textbf{o.x} > 1, \textbf{o.y} > 1 \vdash \langle \textbf{o.m();} \rangle (\textbf{o.x} * \textbf{o.y} > 1)}$$

Improved Specification

Specification

pre: true

post: x = x@pre + y and
y = y@pre

Improved Specification

Specification

pre: true
post: x = x@pre + y **and**
y = y@pre

Rule Application

$$\mathbf{o.x} > 1,\ \mathbf{o.y} > 1 \vdash \langle \mathbf{o.m();} \rangle (\mathbf{o.x} * \mathbf{o.y} > 1)$$

Improved Specification

Specification

pre: true
post: x = x@pre + y **and**
y = y@pre

Rule Application

$$\frac{\begin{array}{ll} & \mathbf{o.x}@pre > 1,\ \mathbf{o.y}@pre > 1, \\ & \mathbf{o.x} = \mathbf{o.x}@pre + \mathbf{o.y}, \\ \ldots \vdash \mathbf{true} & \mathbf{o.y} = \mathbf{o.y}@pre \qquad\qquad \vdash \mathbf{o.x} * \mathbf{o.y} > 1 \end{array}}{\mathbf{o.x} > 1,\ \mathbf{o.y} > 1 \vdash \langle \mathbf{o.m();} \rangle (\mathbf{o.x} * \mathbf{o.y} > 1)}$$

But . . . No Solution in General

Java Program

```
class C {

        int x;
        int y;

        C next;

        void m() {
                x = x+y;
        }

}
```

But . . . No Solution in General

Java Program

```
class C {

    int x;
    int y;

    C next;

    void m() {
        x = x+y;
    }

}
```

Specification

```
pre:    true
post:   x = x@pre + y   and
        y = y@pre   and
        next = next@pre   and
        next.next = next.next@pre   and
        next.next.next = next.next.next@pre
        and . . .  and . . .
```

Modification Conditions

Java Program

```
class C {

    int x;
    int y;

    void m() {
        x = x+y;
    }

}
```

Modification Conditions

Java Program

```
class C {

    int x;
    int y;

    void m() {
        x = x+y;
    }

}
```

Specification

pre:	true
post:	x = x@pre + y
modifies:	x

Modification Conditions

What is needed?

- Syntax
- Precise semantics
- Methods for checking modification conditions
- Methods for using modification conditions
  (rules for verification calculus)

Syntax

List of expressions of the form

expr.attr

where expr evaluates to a single object

What If Location Changes?

What if location (not its value) changes?

```
x   = y;
x.a = 1;
```

modifies: x, $\underbrace{\text{x.a, y.a}}_{?}$

What If Location Changes?

What if location (not its value) changes?

```
x   = y;
x.a = 1;
```

modifies: x, $\underbrace{\text{x.a, y.a}}_{?}$

Answer

modifies: x, y.a (and x.a if x=y in pre-state)

Modification conditions talk about the location that an expression refers to in the pre-state

Further Questions

What if the location is not allocated in the pre-state?

It is still modified

Further Questions

What if the location is not allocated in the pre-state?

It is still modified

What if the method does not terminate?

Nothing is modified

Further Questions

What if the location is not allocated in the pre-state?

It is still modified

What if the method does not terminate?

Nothing is modified

What about temporal/intermediate changes?

Do not matter

Modifies Conditions: Formal Definition

Pre-state

**After binding parameters and self/this,
before execution of the method body**

Modifies Conditions: Formal Definition

Pre-state

**After binding parameters and self/this,
before execution of the method body**

Post-state

After (abrupt or normal) termination of the method

Modifies Conditions: Formal Definition

Pre-state

**After binding parameters and self/this,
before execution of the method body**

Post-state

After (abrupt or normal) termination of the method

Modified location

A location L is modified by running the method in a state s if

- **the method terminates when started in s**
- **there is an expression that refers to L in the pre-state**
  (not a local variable)
- **the values of L in the pre- and the post-state differ**

Modifies Conditions: Formal Definition

Modified expression

An expression is modified by starting the method in a state s if it refers in s to a location that is modified

Modifies Conditions: Formal Definition

Modified expression

An expression is modified by starting the method in a state s if it refers in s to a location that is modified

Modification condition

If there is a state s such that some location L is modified by starting the method in s,

then the modification clause must contain an expression referring to L in s.

A New Rule for Method Invocation

New rule (2nd version)

$$\frac{\Gamma \vdash PRE \qquad \Gamma, \mathcal{U}\mathcal{POST} \vdash \mathcal{U}\phi}{\Gamma \vdash \langle \texttt{m();} \rangle \phi}$$

$\mathcal{U}$:

$$\{ \begin{array}{ll} a@pre & := a \\ mod & := \underbrace{mod@post}_{\text{new var}} \end{array} \}$$

A New Rule for Method Invocation

New rule (2nd version)

$$\frac{\Gamma \vdash PRE \qquad \Gamma, \mathcal{U}\mathcal{POST} \vdash \mathcal{U}\phi}{\Gamma \vdash \langle \textbf{m();} \rangle \phi}$$

$\mathcal{U}$:

$$\{ \begin{array}{ll} a@pre & := a \\ mod & := \underbrace{mod@post}_{\textbf{new var}} \end{array} \}$$

Application to Example

pre:	**true**
post:	**x = x@pre + y**
modifies:	**x**

$$\textbf{o.x} > 1,\ \textbf{o.y} > 1 \vdash \langle \textbf{o.m();} \rangle (\textbf{o.x} * \textbf{o.y} > 1)$$

A New Rule for Method Invocation

New rule (2nd version)

$$\frac{\Gamma \vdash PRE \qquad \Gamma, \mathcal{U}\mathcal{POST} \vdash \mathcal{U}\phi}{\Gamma \vdash \langle \textbf{m();} \rangle \phi}$$

$\mathcal{U}$:

$$\{ \begin{array}{ll} a@pre & := a \\ mod & := \underbrace{mod@post}_{\text{new var}} \end{array} \}$$

Application to Example

pre:	**true**
post:	**x = x@pre + y**
modifies:	**x**

$$\{ \begin{array}{ll} x@pre & := x \\ x & := x@post \end{array} \}$$

$$\textbf{o.x} > 1,\ \textbf{o.y} > 1 \vdash \langle \textbf{o.m();} \rangle (\textbf{o.x} * \textbf{o.y} > 1)$$

A New Rule for Method Invocation

New rule (2nd version)

$$\frac{\Gamma \vdash PRE \qquad \Gamma, \mathcal{U}\mathcal{POST} \vdash \mathcal{U}\phi}{\Gamma \vdash \langle \textbf{m();} \rangle \phi}$$

$\mathcal{U}$:

$$\{\ a@pre := a \quad mod := \underbrace{mod@post}_{\text{new var}}\ \}$$

Application to Example

pre:	**true**
post:	**x = x@pre + y**
modifies:	**x**

$$\{\ x@pre := x \quad x := x@post\ \}$$

$$\frac{\ldots \vdash \textbf{true} \qquad \begin{array}{l}\textbf{o.x} > 1, \textbf{o.y} > 1, \\ \textbf{o.x}@post = \textbf{o.x} + \textbf{o.y} \vdash \textbf{o.x}@post * \textbf{o.y} > 1\end{array}}{\textbf{o.x} > 1,\ \textbf{o.y} > 1 \vdash \langle \textbf{o.m();} \rangle (\textbf{o.x} * \textbf{o.y} > 1)}$$

Checking Modification Conditions

Problematic programs

`x = y; x.a = 1;`	modifies: x, y.a
`n++; a[n] = 1;`	modifies: n, a[n+1]

Checking Modification Conditions

Problematic programs

`x = y; x.a = 1;`	modifies: x, y.a
`n++; a[n] = 1;`	modifies: n, a[n+1]

CHASE static checker

- Extends ESC/Java to check modification clauses in JML
- Neither sound nor complete

More Expressive Modification Conditions

Extended language for modification conditions

- o.∗ — all attributes of o
- reachable(o) — all objects reachable from o
- reachable(o,next) — all objects reachable from o via next
- etc.

More Expressive Modification Conditions

Extended language for modification conditions

o.*	all attributes of o
reachable(o)	all objects reachable from o
reachable(o,next)	all objects reachable from o via next
etc.	

\+ more expressivity

– harder to check

– harder to define rule