Backstage Java

Making a Difference in Metaprogramming

Zachary Palmer and Scott F. Smith

The Johns Hopkins University

October 27, 2011

Difference-Based Metaprogramming

- Introduction to Metaprogramming

Difference-Based Metaprogramming

- Introduction to Metaprogramming
- Compile-Time Metaprogramming in Java

Difference-Based Metaprogramming

- Introduction to Metaprogramming
- Compile-Time Metaprogramming in Java
- Traditional Metaprogramming Model

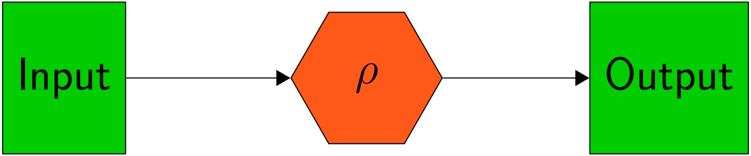
Difference-Based Metaprogramming

- Introduction to Metaprogramming
- Compile-Time Metaprogramming in Java
- Traditional Metaprogramming Model
- Difference-Based Metaprogramming Model

What is metaprogramming?

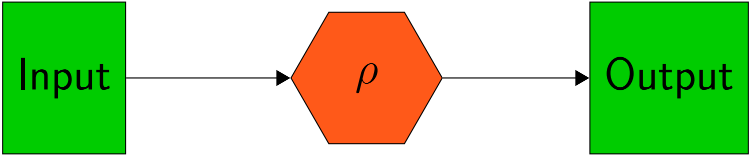
Metaprogramming

- Programs input data and output data.

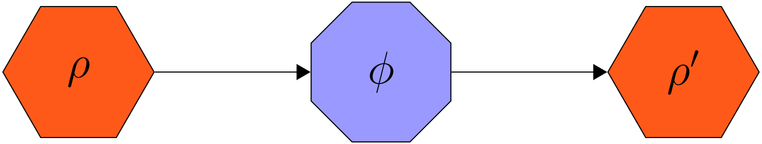


Metaprogramming

- Programs input data and output data.



- Metaprograms input programs (or program fragments) and output the same.



Examples of Metaprogramming

- C Macros
- C++ Templates
- LISP Macros
- Template Haskell
- MetaOCaml
- Stratego
- Groovy
- etc. etc.

Classifying Metaprogramming

- When is it run?
    - Compile-time (static)
    - Runtime (dynamic)
- How are programs represented?
    - Textually (strings)
    - Lexically (tokens)
    - Structurally (ASTs)
    - Semantically (various structures)
- Which language is used to metaprogram?
    - Same language (homogenous)
    - Different language (heterogeneous)

from *Accomplishments and Research Challenges in Metaprogramming* (Tim Sheard)

Classifying Metaprogramming

- When is it run?
  - **Compile-time (static)**
  - Runtime (dynamic)
- How are programs represented?
  - Textually (strings)
  - Lexically (tokens)
  - **Structurally (ASTs)**
  - Semantically (various structures)
- Which language is used to metaprogram?
  - **Same language (homogenous)**
  - Different language (heterogeneous)

from *Accomplishments and Research Challenges in Metaprogramming* (Tim Sheard)

Template Haskell Example

```
$(
  let mkExp n v =
        if n == 0
          then [| 1 |]
          else [| $(v) * $(mkExp (n-1) v) |]
  in
  let f n =
        let funNm = mkName ("exp" ++ (show n)) in
        let params = [varP (mkName "x")] in
        funD funNm $ [clause params
          (normalB $ mkExp n (varE $ mkName "x")) []]
  in
 mapM f [2..50]
 )
```

Template Haskell Example

```
exp2 x = x * x
exp3 x = x * x * x
exp4 x = x * x * x * x
exp5 x = x * x * x * x * x
exp6 x = x * x * x * x * x * x
```

$$\vdots \qquad\qquad\qquad\qquad \vdots \qquad\qquad\qquad\qquad \vdots$$

$$\texttt{exp50 x = }\overbrace{\texttt{x * x * ... * x * x}}^{50}$$

Template Haskell

- Programmatic code generation

Template Haskell

- Programmatic code generation
- Literal syntax for AST construction

Template Haskell

- Programmatic code generation
- Literal syntax for AST construction
- Functional programming style

Template Haskell

- Programmatic code generation
- Literal syntax for AST construction
- Functional programming style
- Very limited ability to inspect environment

Why not Template Java?

Template Java?

```
public class Location {
  private int x;
  public int getX() { return this.x; }
  public void setX(int x) { this.x = x; }
  private int y;
  public int getY() { return this.y; }
  public void setY(int y) { this.y = y; }
  public Location(int x, int y) {
    this.x = x;
    this.y = y;
  }
  public String toString() {
    return "("+this.x+","+this.y+")";
  }
}
```

Template Java?

```
public class Location {
  private int x;
  public int getX() { return this.x; }
  public void setX(int x) { this.x = x; }
  private int y;
  public int getY() { return this.y; }
  public void setY(int y) { this.y = y; }
  public Location(int x, int y) {
    this.x = x;
    this.y = y;
  }
  public String toString() {
    return "("+this.x+","+this.y+")";
  }
}
```

Template Java?

```
public class Location {

  $( property( [|private int x|] )

  private int y;
  public int getY() { return this.y; }
  public void setY(int y) { this.y = y; }
  public Location(int x, int y) {
    this.x = x;
    this.y = y;
  }
  public String toString() {
    return "("+this.x+","+this.y+")";
  }
}
```

Template Java?

```
public class Location {

  $( property( [|private int x|] )

  private int y;
  public int getY() { return this.y; }
  public void setY(int y) { this.y = y; }
  public Location(int x, int y) {
    this.x = x;
    this.y = y;
  }
  public String toString() {
    return "("+this.x+","+this.y+")";
  }
}
```

Template Java?

```
public class Location {

  $( property( [|private int x|] )


  $( property( [|private int y|] )

  public Location(int x, int y) {
    this.x = x;
    this.y = y;
  }
  public String toString() {
    return "("+this.x+","+this.y+")";
  }
}
```

Template Java?

```
public class Location {

  $( property( [|private int x|] )


  $( property( [|private int y|] )

  public Location(int x, int y) {
    this.x = x;
    this.y = y;
  }
  public String toString() {
    return "("+this.x+","+this.y+")";
  }
}
```

Template Java?

```
public class Location {

  $( property( [|private int x|] )


  $( property( [|private int y|] )


  $( makePropertyConstructor(
       [|int x|], [|int y|]) )

  public String toString() {
    return "("+this.x+","+this.y+")";
  }
}
```

Template Java?

```
public class Location {
  $( property( [|private int x|] )
  $( property( [|private int y|] )
  $( makePropertyConstructor(
        [|int x|], [|int y|]) )
  public String toString() {
    return "("+this.x+","+this.y+")";
  }
}
```

Template Java?

```
public class Location {
  $( property( [|private int x|] )
  $( property( [|private int y|] )
  $( makePropertyConstructor(
       [|int x|], [|int y|]) )
  public String toString() {
    return "("+this.x+","+this.y+")";
  }
}
```

- Metaprograms can't react to surrounding code

Template Java?

```
public class Location {
  $( property( [|private int x|] )
  $( property( [|private int y|] )
  $( makePropertyConstructor(
        [|int x|], [|int y|]) )
  public String toString() {
    return "("+this.x+","+this.y+")";
  }
}
```

- Metaprograms can't react to surrounding code
- Metaprogrammers compensate by duplicating information

Template Java?

```
public class Location {
  $( property( [|private int x|] )
  $( property( [|private int y|] )
  $( makePropertyConstructor(
       [|int x|], [|int y|]) )
  public String toString() {
    return "("+this.x+","+this.y+")";
  }
}
```

- Metaprograms can't react to surrounding code
- Metaprogrammers compensate by duplicating information
- Functional metaprogramming in a declarative object-oriented language

What Do We Want?

- Object-oriented, declarative metaprogramming style

What Do We Want?

- Object-oriented, declarative metaprogramming style
- Awareness of surrounding code

What Do We Want?

- Object-oriented, declarative metaprogramming style
- Awareness of surrounding code
- Modular, independent metaprograms

Backstage Java

How about some of this?

```
@@GenerateConstructorFromProperties
public class Location {
  @@Property private int x;
  @@Property private int y;
  public String toString() {
    return "("+this.x+","+this.y+")";
  }
}
```

Backstage Java

How about some of this?

```
@@GenerateConstructorFromProperties
public class Location {
  @@Property private int x;
  @@Property private int y;
  public String toString() {
    return "("+this.x+","+this.y+")";
  }
}
```

Harder than it looks...

Traditional Metaprogramming

Traditional Metaprogramming

- Metaprograms are a series of program transformations

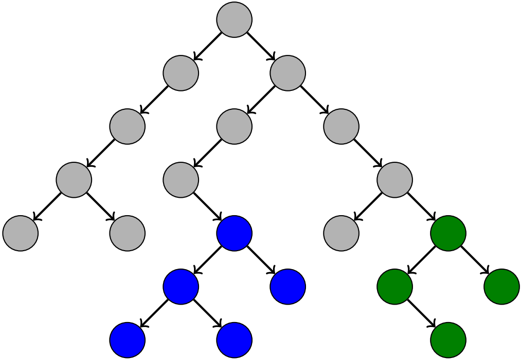
Traditional Metaprogramming

- Metaprograms are a series of program transformations
- Each available transformation occurs exactly once

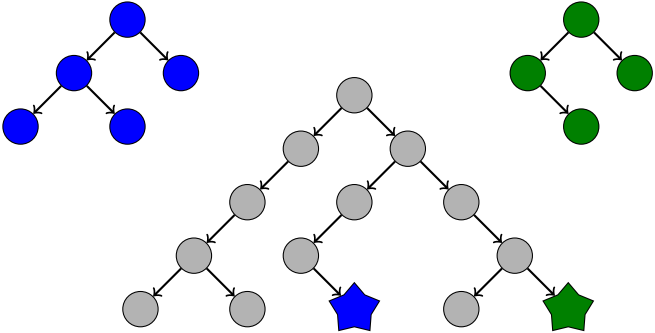
Traditional Metaprogramming

- Metaprograms are a series of program transformations
- Each available transformation occurs exactly once
- True even for embedded syntax

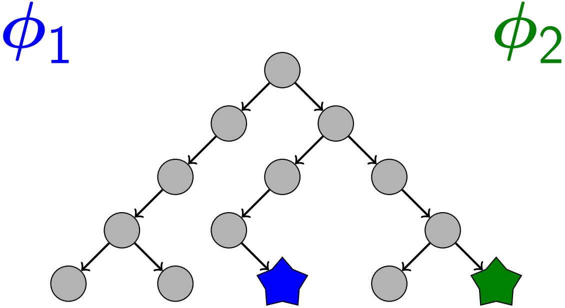
Embedded Metaprogram Semantics



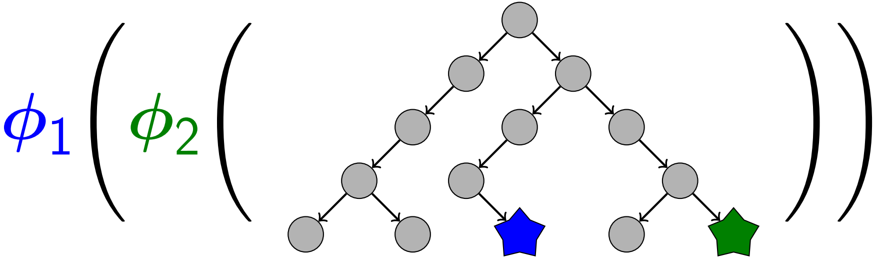
Embedded Metaprogram Semantics



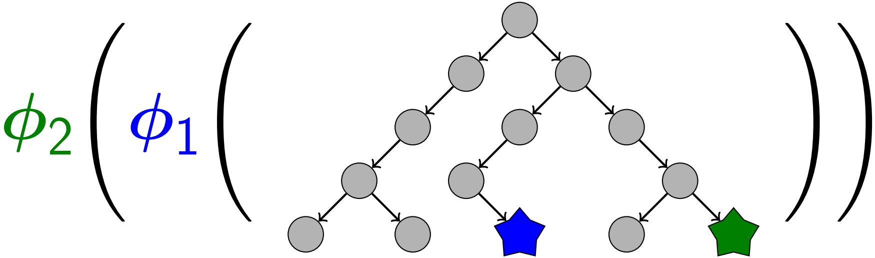
Embedded Metaprogram Semantics



Embedded Metaprogram Semantics

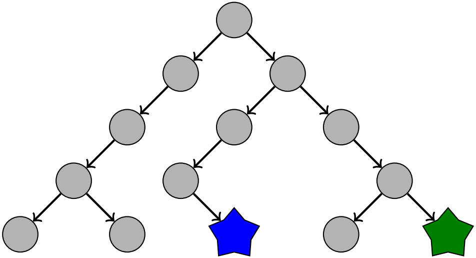


Embedded Metaprogram Semantics



Embedded Metaprogram Semantics

$\phi_2\Bigg(\phi_1\Bigg($

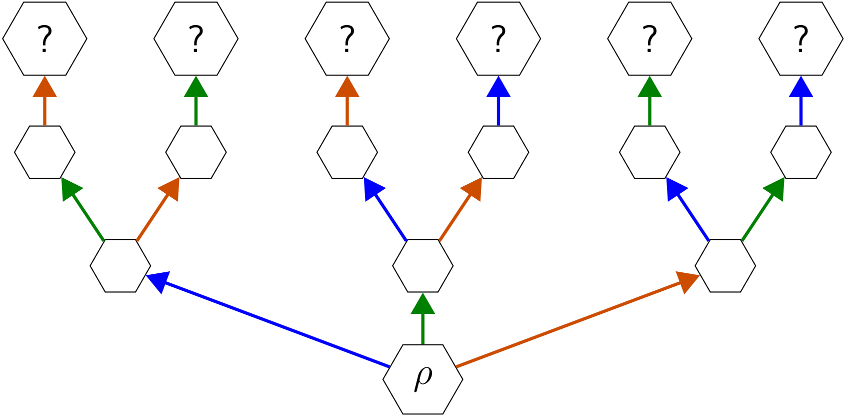


$\Bigg)\Bigg)$

How do we pick?

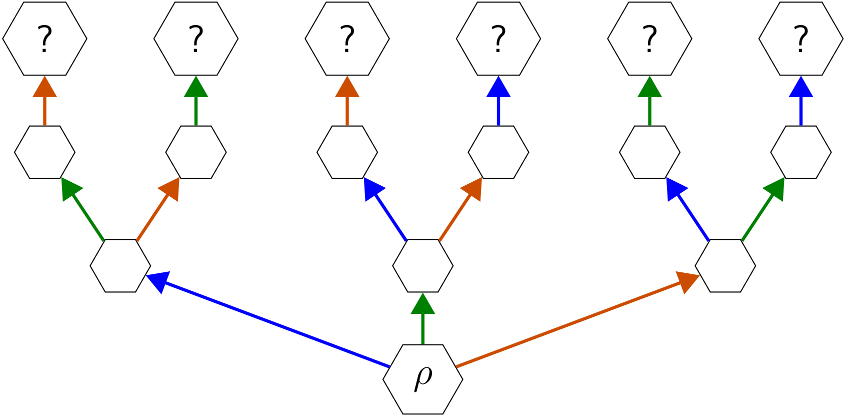
Ambiguity in Metaprogramming

Assuming three transformations ϕ_1, ϕ_2, ϕ_3...



Ambiguity in Metaprogramming

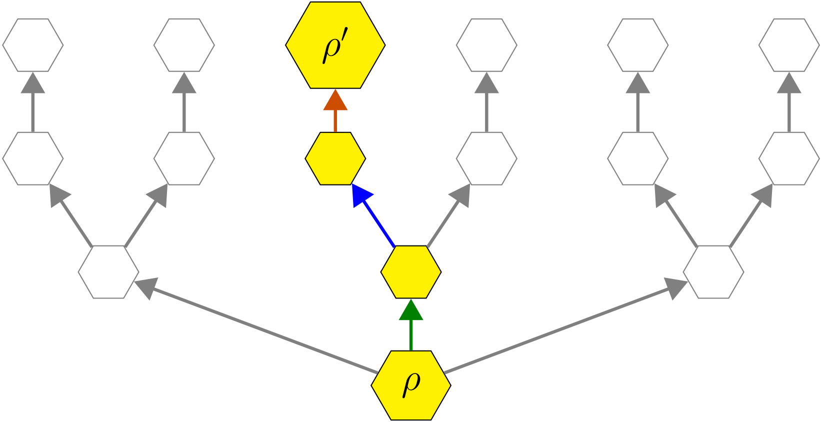
Assuming three transformations ϕ_1, ϕ_2, ϕ_3...



OpenJava, Groovy

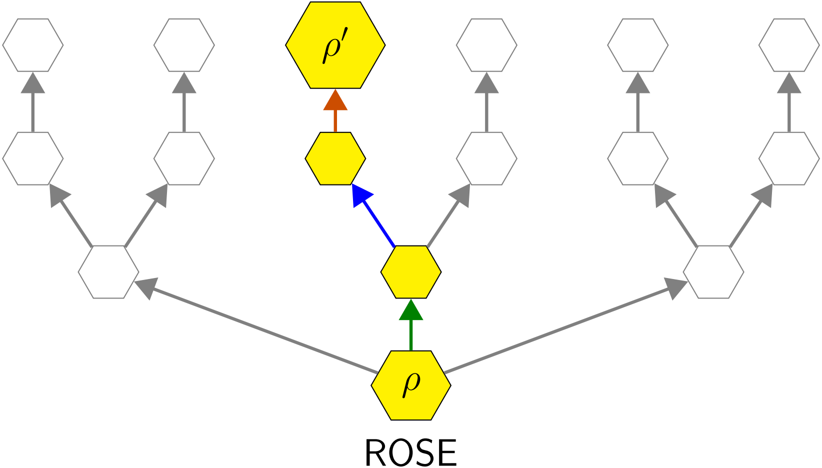
Total Ordering Solution

Declaring ϕ_2 before ϕ_1, which is before ϕ_3.



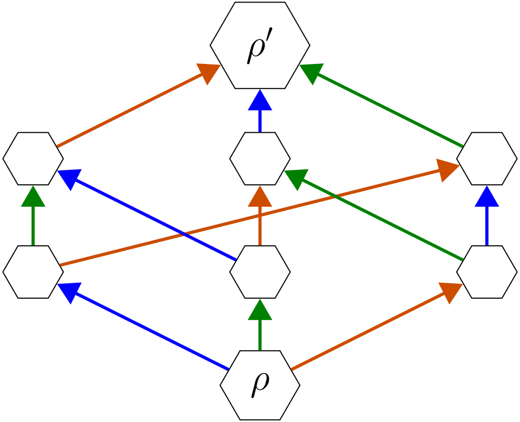
Total Ordering Solution

Declaring ϕ_2 before ϕ_1, which is before ϕ_3.



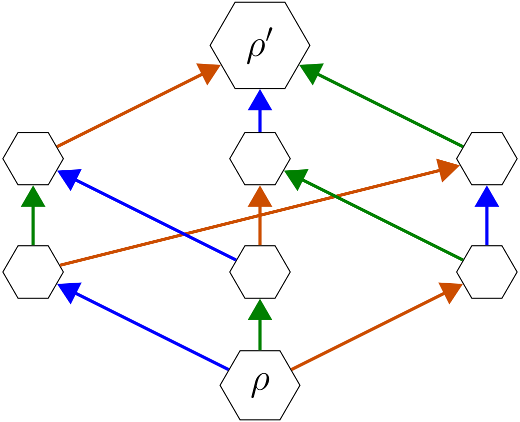
Necessary Commutation Solution

Requiring that $\phi_i \circ \phi_j = \phi_j \circ \phi_i$ for all i and j



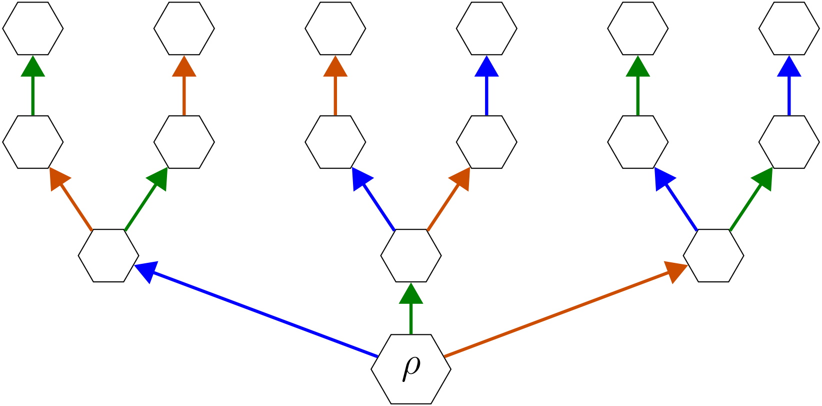
Necessary Commutation Solution

Requiring that $\phi_i \circ \phi_j = \phi_j \circ \phi_i$ for all i and j



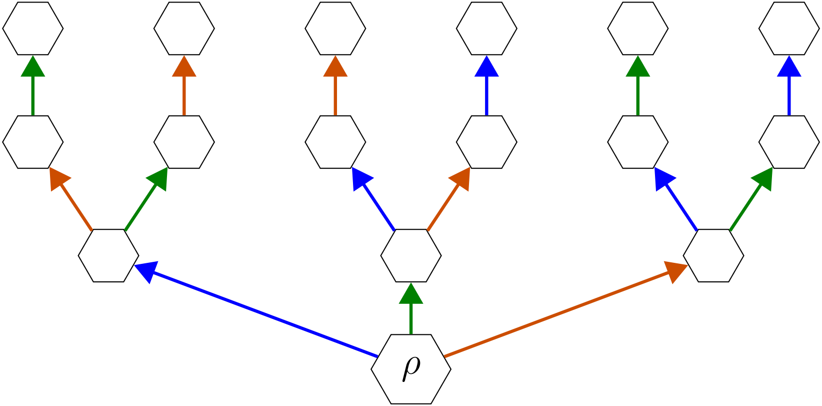
Template Haskell, MetaOCaml, LISP, ...

Hybrid Solution



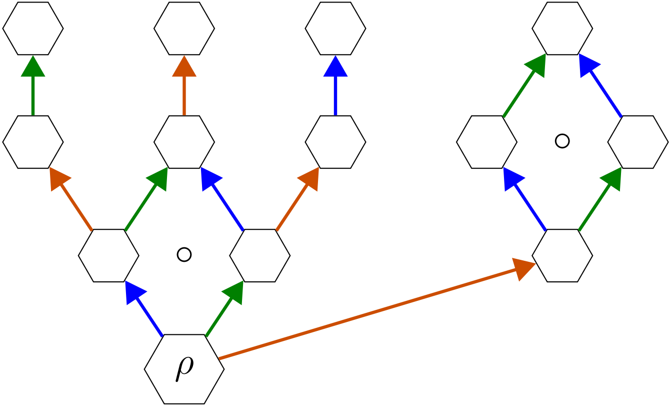
Hybrid Solution

Suppose that ϕ_1 and ϕ_2 commute.



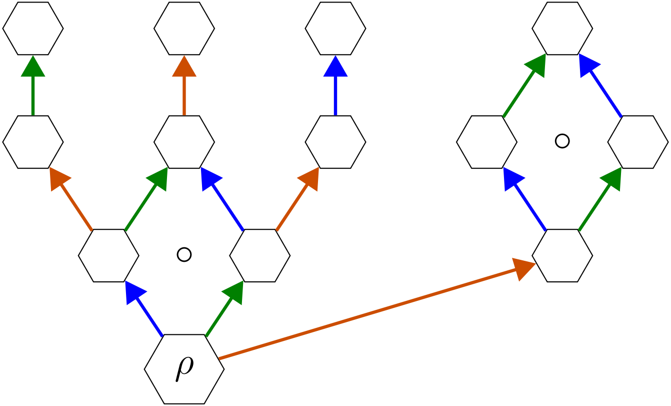
Hybrid Solution

Suppose that ϕ_1 and ϕ_2 commute.



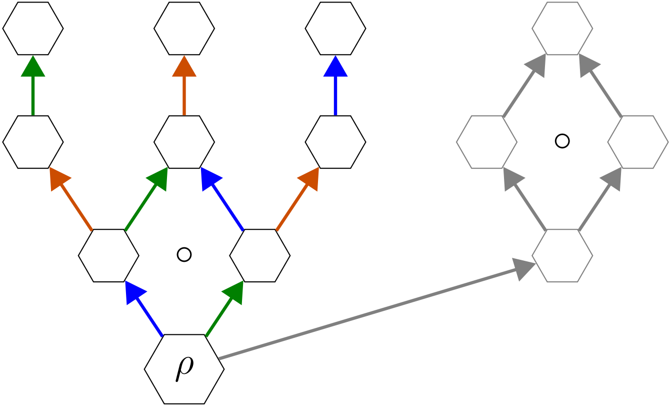
Hybrid Solution

Suppose that ϕ_1 and ϕ_2 commute.
Suppose that ϕ_3 must occur after them.



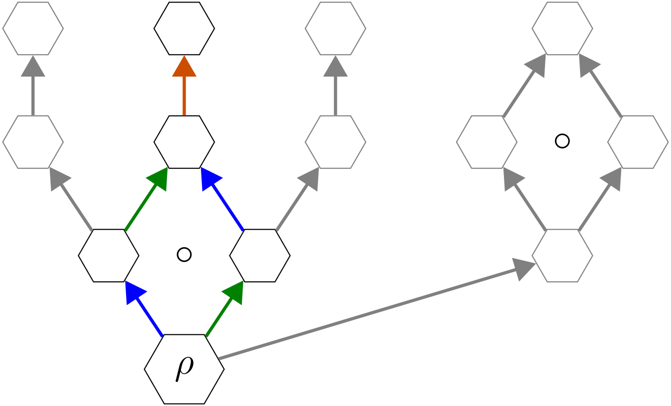
Hybrid Solution

Suppose that ϕ_1 and ϕ_2 commute.
Suppose that ϕ_3 must occur after them.



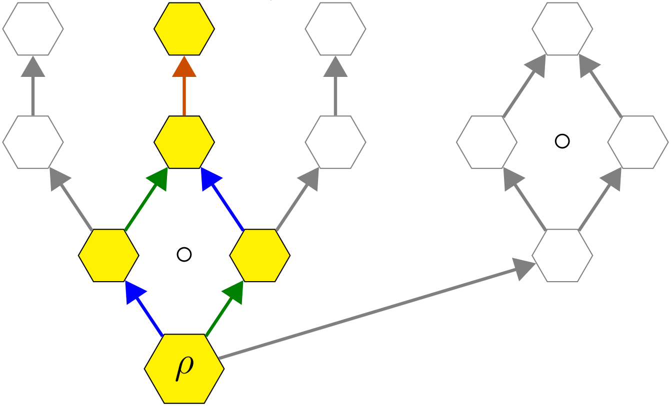
Hybrid Solution

Suppose that ϕ_1 and ϕ_2 commute.
Suppose that ϕ_3 must occur after them.



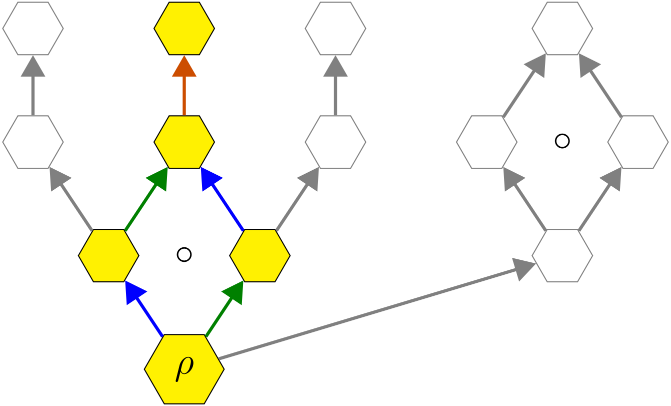
Hybrid Solution

Suppose that ϕ_1 and ϕ_2 commute.
Suppose that ϕ_3 must occur after them.



Hybrid Solution

Suppose that ϕ_1 and ϕ_2 commute.
Suppose that ϕ_3 must occur after them.



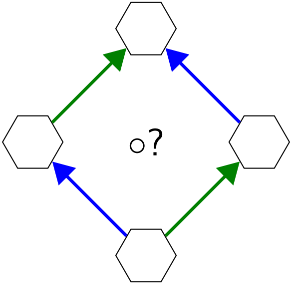
Backstage Java*

Commuting Transformations

How do we tell if ϕ_1 and ϕ_2 commute?

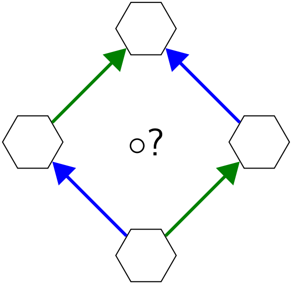
Commuting Transformations

How do we tell if ϕ_1 and ϕ_2 commute?



Commuting Transformations

How do we tell if ϕ_1 and ϕ_2 commute?



Determining whether or not two arbitrary transformations commute is *undecidable!*

Difference-Based Metaprogramming

Difference-Based Metaprogramming

Treat metaprograms as transformation *generators*:

Difference-Based Metaprogramming

Treat metaprograms as transformation *generators*:

$$\phi(\rho) \quad = \bar{\delta}$$

Difference-Based Metaprogramming

Treat metaprograms as transformation *generators*:

$$\begin{aligned} \phi(\rho) &= \bar{\delta} \\ [\![\bar{\delta}]\!](\rho) &= \rho' \end{aligned}$$

Difference-Based Metaprogramming

Treat metaprograms as transformation *generators*:

$$\begin{aligned} \phi(\rho) &= \bar{\delta} \\ [\![\bar{\delta}]\!](\rho) &= \rho' \end{aligned}$$

- Language of $\bar{\delta}$ is not Turing-complete

Difference-Based Metaprogramming

Treat metaprograms as transformation *generators*:

$$\begin{aligned} \phi(\rho) &= \bar{\delta} \\ [\![\bar{\delta}]\!](\rho) &= \rho' \end{aligned}$$

- Language of $\bar{\delta}$ is not Turing-complete
- Each $\bar{\delta}$ is generated on a case-by-case basis

Difference-Based Metaprogramming

Treat metaprograms as transformation *generators*:

$$\begin{aligned} \phi(\rho) &= \bar{\delta} \\ [\![\bar{\delta}]\!](\rho) &= \rho' \end{aligned}$$

- Language of $\bar{\delta}$ is not Turing-complete
- Each $\bar{\delta}$ is generated on a case-by-case basis
- No practically significant loss of expressiveness

Difference-Based Metaprogramming

Treat metaprograms as transformation *generators*:

$$\begin{aligned} \phi(\rho) &= \bar{\delta} \\ [\![\bar{\delta}]\!](\rho) &= \rho' \end{aligned}$$

$\bar{\delta}$ can express:

- Creation of a node

Difference-Based Metaprogramming

Treat metaprograms as transformation *generators*:

$$\begin{aligned} \phi(\rho) &= \bar{\delta} \\ [\![\bar{\delta}]\!](\rho) &= \rho' \end{aligned}$$

$\bar{\delta}$ can express:

- Creation of a node
- Assignment to a node property

Difference-Based Metaprogramming

Treat metaprograms as transformation *generators*:

$$\begin{aligned} \phi(\rho) &= \bar{\delta} \\ [\![\bar{\delta}]\!](\rho) &= \rho' \end{aligned}$$

$\bar{\delta}$ can express:

- Creation of a node
- Assignment to a node property
- Additions before or after an element in a list

Difference-Based Metaprogramming

Treat metaprograms as transformation *generators*:

$$\begin{aligned} \phi(\rho) &= \bar{\delta} \\ [\![\bar{\delta}]\!](\rho) &= \rho' \end{aligned}$$

$\bar{\delta}$ can express:

- Creation of a node
- Assignment to a node property
- Additions before or after an element in a list
- Removal of an element from a list

Difference-Based Metaprogramming

Treat metaprograms as transformation *generators*:

$$\begin{aligned} \phi(\rho) &= \bar{\delta} \\ [\![\bar{\delta}]\!](\rho) &= \rho' \end{aligned}$$

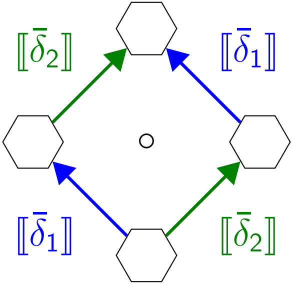
Now, prove commutation over pairs of $[\![\bar{\delta}]\!]$.

Difference-Based Metaprogramming

Treat metaprograms as transformation *generators*:

$$\begin{aligned} \phi(\rho) &= \bar{\delta} \\ \llbracket \bar{\delta} \rrbracket(\rho) &= \rho' \end{aligned}$$

Now, prove commutation over pairs of $\llbracket \bar{\delta} \rrbracket$.



A Simple BSJ Example

```
public class Example {
  public static void main(String[] arg) {
    [:
      BlockStatementListNode list = context.getAnchor().
        getNearestAncestorOfType(
            BlockStatementListNode.class);
      list.addFirst(
          <:System.out.println("Hello, world!");:>);
    :]
    [:
      BlockStatementListNode list = context.getAnchor().
        getNearestAncestorOfType(
            BlockStatementListNode.class);
      list.addLast(
          <:System.out.println("How are you?");:>);
    :]
  }
}
```

A Simple Example

```
public class Example {
  public static void main(String[] arg) {

     $\mathcal{M}_1$
     $\mathcal{M}_2$

  }
}
```

- Replace metaprograms with anchors

A Simple Example

```
public class Example {
  public static void main(String[] arg) {

     $\mathcal{M}_1$
     $\mathcal{M}_2$

  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes

A Simple Example

```
public class Example {
  public static void main(String[] arg) {

    $\mathcal{M}_1$
    $\mathcal{M}_2$

  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes
    - $\phi_1(\rho) = \bar{\delta}_1$ ("Hello, world!" first)

A Simple Example

```
public class Example {
  public static void main(String[] arg) {

      $\mathcal{M}_1$
      $\mathcal{M}_2$

  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes
  - $\phi_1(\rho) = \bar{\delta}_1$ ("Hello, world!" first)
  - $\phi_2(\rho) = \bar{\delta}_2$ ("How are you?" last)

A Simple Example

```
public class Example {
  public static void main(String[] arg) {

      $\mathcal{M}_1$
      $\mathcal{M}_2$

  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes
    - $\phi_1(\rho) = \bar{\delta}_1$ ("Hello, world!" first)
    - $\phi_2(\rho) = \bar{\delta}_2$ ("How are you?" last)
- Prove that $\bar{\delta}_1$ and $\bar{\delta}_2$ commute.

A Simple Example

```
public class Example {
  public static void main(String[] arg) {


      M1
      M2


  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes
  - $\phi_1(\rho) = \bar{\delta}_1$ ("Hello, world!" first)
  - $\phi_2(\rho) = \bar{\delta}_2$ ("How are you?" last)
- Prove that $\bar{\delta}_1$ and $\bar{\delta}_2$ commute.
- Execute $\bar{\delta}_1$ and $\bar{\delta}_2$ in some order.

A Simple Example

```
public class Example {
  public static void main(String[] arg) {
     System.out.println("Hello, world!");
     $\mathcal{M}_1$
     $\mathcal{M}_2$

  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes
    - $\phi_1(\rho) = \bar{\delta}_1$ ("Hello, world!" first)
    - $\phi_2(\rho) = \bar{\delta}_2$ ("How are you?" last)
- Prove that $\bar{\delta}_1$ and $\bar{\delta}_2$ commute.
- Execute $\bar{\delta}_1$ and $\bar{\delta}_2$ in some order.

A Simple Example

```
public class Example {
  public static void main(String[] arg) {
     System.out.println("Hello, world!");
     $\mathcal{M}_1$
     $\mathcal{M}_2$
     System.out.println("How are you?");
  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes
  - $\phi_1(\rho) = \bar{\delta}_1$ (“Hello, world!” first)
  - $\phi_2(\rho) = \bar{\delta}_2$ (“How are you?” last)
- Prove that $\bar{\delta}_1$ and $\bar{\delta}_2$ commute.
- Execute $\bar{\delta}_1$ and $\bar{\delta}_2$ in some order.

A Simple Example

```
public class Example {
  public static void main(String[] arg) {

    $\mathcal{M}_1$
    $\mathcal{M}_2$

  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes
  - $\phi_1(\rho) = \bar{\delta}_1$ ("Hello, world!" first)
  - $\phi_2(\rho) = \bar{\delta}_2$ ("How are you?" last)
- Prove that $\bar{\delta}_1$ and $\bar{\delta}_2$ commute.
- Execute $\bar{\delta}_1$ and $\bar{\delta}_2$ in some order.

A Simple Example

```
public class Example {
  public static void main(String[] arg) {

     $\mathcal{M}_1$
     $\mathcal{M}_2$
     System.out.println("How are you?");
  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes
  - $\phi_1(\rho) = \bar{\delta}_1$ ("Hello, world!" first)
  - $\phi_2(\rho) = \bar{\delta}_2$ ("How are you?" last)
- Prove that $\bar{\delta}_1$ and $\bar{\delta}_2$ commute.
- Execute $\bar{\delta}_1$ and $\bar{\delta}_2$ in some order.

A Simple Example

```
public class Example {
  public static void main(String[] arg) {
     System.out.println("Hello, world!");
     $\mathcal{M}_1$
     $\mathcal{M}_2$
     System.out.println("How are you?");
  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes
  - $\phi_1(\rho) = \bar{\delta}_1$ ("Hello, world!" first)
  - $\phi_2(\rho) = \bar{\delta}_2$ ("How are you?" last)
- Prove that $\bar{\delta}_1$ and $\bar{\delta}_2$ commute.
- Execute $\bar{\delta}_1$ and $\bar{\delta}_2$ in some order.

A Simple Example

```
public class Example {
  public static void main(String[] arg) {
     System.out.println("Hello, world!");
     ;
     ;
     System.out.println("How are you?");
  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes
  - $\phi_1(\rho) = \bar{\delta}_1$ (“Hello, world!” first)
  - $\phi_2(\rho) = \bar{\delta}_2$ (“How are you?” last)
- Prove that $\bar{\delta}_1$ and $\bar{\delta}_2$ commute.
- Execute $\bar{\delta}_1$ and $\bar{\delta}_2$ in some order.

An Example of Conflict

```
public class Example {
 public static void main(String[] arg) {
  [:
   BlockStatementListNode list = context.getAnchor().
    getNearestAncestorOfType(
      BlockStatementListNode.class);
   list.addFirst(
     <:System.out.println("Hello, world!");:>);
  :]
  [:
   BlockStatementListNode list = context.getAnchor().
    getNearestAncestorOfType(
      BlockStatementListNode.class);
   list.addFirst(
     <:System.out.println("How are you?");:>);
  :]
 }
}
```

An Example of Conflict

```
public class Example {
  public static void main(String[] arg) {



     $\mathcal{M}_1$
     $\mathcal{M}_2$
  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes
  - $\phi_1(\rho) = \bar{\delta}_1$ (“Hello, world!” first)
  - $\phi_2(\rho) = \bar{\delta}_2$ (“How are you?” **first**)

An Example of Conflict

```
public class Example {
  public static void main(String[] arg) {



     $\mathcal{M}_1$
     $\mathcal{M}_2$
  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes
  - $\phi_1(\rho) = \bar{\delta}_1$ ("Hello, world!" first)
  - $\phi_2(\rho) = \bar{\delta}_2$ ("How are you?" **first**)
- Now, $\bar{\delta}_1$ and $\bar{\delta}_2$ do not commute!

An Example of Conflict

```
public class Example {
  public static void main(String[] arg) {

     System.out.println("How are you?");
     M1
     M2
  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes
  - $\phi_1(\rho) = \overline{\delta}_1$ ("Hello, world!" first)
  - $\phi_2(\rho) = \overline{\delta}_2$ ("How are you?" **first**)
- Now, $\overline{\delta}_1$ and $\overline{\delta}_2$ do not commute!

An Example of Conflict

```
public class Example {
  public static void main(String[] arg) {
     System.out.println("Hello, world!");
     System.out.println("How are you?");
     M1
     M2
  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes
  - $\phi_1(\rho) = \bar{\delta}_1$ (“Hello, world!” first)
  - $\phi_2(\rho) = \bar{\delta}_2$ (“How are you?” **first**)
- Now, $\bar{\delta}_1$ and $\bar{\delta}_2$ do not commute!

An Example of Conflict

```
public class Example {
  public static void main(String[] arg) {



     $\mathcal{M}_1$
     $\mathcal{M}_2$
  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes
  - $\phi_1(\rho) = \bar{\delta}_1$ (“Hello, world!” first)
  - $\phi_2(\rho) = \bar{\delta}_2$ (“How are you?” **first**)
- Now, $\bar{\delta}_1$ and $\bar{\delta}_2$ do not commute!

An Example of Conflict

```
public class Example {
  public static void main(String[] arg) {

     System.out.println("Hello, world!");
     M1
     M2
  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes
  - $\phi_1(\rho) = \overline{\delta}_1$ ("Hello, world!" first)
  - $\phi_2(\rho) = \overline{\delta}_2$ ("How are you?" **first**)
- Now, $\overline{\delta}_1$ and $\overline{\delta}_2$ do not commute!

An Example of Conflict

```
public class Example {
  public static void main(String[] arg) {
     System.out.println("How are you?");
     System.out.println("Hello, world!");
     𝓜1
     𝓜2
  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes
  - $\phi_1(\rho) = \bar{\delta}_1$ ("Hello, world!" first)
  - $\phi_2(\rho) = \bar{\delta}_2$ ("How are you?" **first**)
- Now, $\bar{\delta}_1$ and $\bar{\delta}_2$ do not commute!

An Example of Conflict

```
public class Example {
  public static void main(String[] arg) {
     System.out.println("How are you?");
     System.out.println("Hello, world!");
     ;
     ;
  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes
  - $\phi_1(\rho) = \overline{\delta}_1$ ("Hello, world!" first)
  - $\phi_2(\rho) = \overline{\delta}_2$ ("How are you?" **first**)
- Now, $\overline{\delta}_1$ and $\overline{\delta}_2$ do not commute!

An Example of Conflict

```
public class Example {
  public static void main(String[] arg) {
     System.out.println("How are you?");
     System.out.println("Hello, world!");
     ;
     ;
  }
}
```

- Replace metaprograms with anchors
- Run each metaprogram to collect its changes
  - $\phi_1(\rho) = \bar{\delta}_1$ (“Hello, world!” first)
  - $\phi_2(\rho) = \bar{\delta}_2$ (“How are you?” **first**)
- Now, $\bar{\delta}_1$ and $\bar{\delta}_2$ do not commute!
- But we can detect this!

Conflict Detection

Record Node Creation Rule

$$\frac{\eta \mapsto \hat{v} \notin \rho \qquad \rho[\![\eta \mapsto \{\overline{l \mapsto \hat{v}}\}]\!] \Rightarrow \rho'}{(\overset{R}{+}\eta(\overline{l = \hat{v}}))\ \rho \Rightarrow \rho'}$$

List Node Creation Rule

$$\frac{\begin{array}{c}\eta \mapsto \hat{v} \notin \rho \\ \rho[\![\eta \mapsto [(\triangleright, \mathcal{M}, \emptyset), (\triangleleft, \mathcal{M}, \emptyset)]]\!] \Rightarrow \rho'\end{array}}{(\mathcal{M} \succ \overset{L}{+}\eta)\ \rho \Rightarrow \rho'}$$

Record Assignment Rule

$$\frac{\eta \mapsto \mathcal{R} \in \rho \qquad \rho[\![\eta \mapsto \mathcal{R}[\![l \mapsto \hat{v}]\!]]\!] \Rightarrow \rho'}{(\eta.l \leftarrow \hat{v})\ \rho \Rightarrow \rho'}$$

List Add Before Rule

$$\frac{\begin{array}{c}\overset{\triangle}{\eta}_3 \neq \triangleright \qquad \eta_1 \mapsto \mathcal{L} \in \rho \\ \mathring{\eta}_3 = \Sigma(\overset{\triangle}{\eta}_3, \mathcal{M}, \mathcal{L}) \qquad \mathcal{L} = [\overline{\mathring{\eta}'}, \mathring{\eta}_3, \overline{\mathring{\eta}''}] \\ \mathcal{L}' = [\overline{\mathring{\eta}'}, (\eta_2, \mathcal{M}, \emptyset), \mathring{\eta}_3, \overline{\mathring{\eta}''}] \\ \rho[\![\eta_1 \mapsto \mathcal{L}']\!] \Rightarrow \rho'\end{array}}{(\mathcal{M} \succ \eta_1 : \eta_2 \looparrowleft \overset{\triangle}{\eta}_3)\ \rho \Rightarrow \rho'}$$

List Add After Rule

$$\frac{\begin{array}{c}\overset{\triangle}{\eta}_3 \neq \triangleleft \qquad \eta_1 \mapsto \mathcal{L} \in \rho \\ \mathring{\eta}_3 = \Sigma(\overset{\triangle}{\eta}_3, \mathcal{M}, \mathcal{L}) \qquad \mathcal{L} = [\overline{\mathring{\eta}'}, \mathring{\eta}_3, \overline{\mathring{\eta}''}] \\ \mathcal{L}' = [\overline{\mathring{\eta}'}, \mathring{\eta}_3, (\eta_2, \mathcal{M}, \emptyset), \overline{\mathring{\eta}''}] \\ \rho[\![\eta_1 \mapsto \mathcal{L}']\!] \Rightarrow \rho'\end{array}}{(\mathcal{M} \succ \eta_1 : \overset{\triangle}{\eta}_3 \looparrowright \eta_2)\ \rho \Rightarrow \rho'}$$

List Remove Rule

$$\frac{\begin{array}{c}\eta_1 \mapsto \mathcal{L} \in \rho \\ \mathring{\eta}_2 = (\eta_2, \mathcal{M}', \mathcal{S}) = \Sigma(\eta_2, \mathcal{M}, \mathcal{L}) \\ \mathcal{L} = [\overline{\mathring{\eta}'}, \mathring{\eta}_2, \overline{\mathring{\eta}''}] \\ \mathcal{L}' = [\overline{\mathring{\eta}'}, (\eta_2, \mathcal{M}', \mathcal{S} \cup \{\mathcal{M}\}), \overline{\mathring{\eta}''}] \\ \rho[\![\eta_1 \mapsto \mathcal{L}']\!] \Rightarrow \rho'\end{array}}{(\mathcal{M} \succ \eta_1 : \downarrow \eta_2)\ \rho \Rightarrow \rho'}$$

Recursive Application Rule

$$\frac{\delta'\ e \Rightarrow \rho \qquad \delta\ \rho \Rightarrow \rho'}{\delta\ (\delta'\ e) \Rightarrow \rho'}$$

Value Rule

$$\frac{}{\hat{v} \Rightarrow \hat{v}}$$

Record Assignment Conflict Rule

$$\frac{\hat{v} \neq \hat{v}'}{\eta.l \leftarrow \hat{v} \nleftrightarrow \eta.l \leftarrow \hat{v}'}$$

Add Before Conflict Rule

$$\frac{\omega(\eta_2) \qquad \omega(\eta_2')}{\eta_1 : \eta_2 \looparrowleft \overset{\triangle}{\eta}_3 \nleftrightarrow \eta_1 : \eta_2' \looparrowleft \overset{\triangle}{\eta}_3}$$

Add After Conflict Rule

$$\frac{\omega(\eta_2) \qquad \omega(\eta_2')}{\eta_1 : \overset{\triangle}{\eta}_3 \looparrowright \eta_2 \nleftrightarrow \eta_1 : \overset{\triangle}{\eta}_3 \looparrowright \eta_2'}$$

Unordered Creation Conflict Rule

$$\frac{\delta = \overset{R}{+}\eta(\overline{l \mapsto \hat{v}}) \vee \delta = \overset{L}{+}\eta \qquad \eta \in \delta'}{\delta \nleftrightarrow \delta'}$$

Conflict Detection

Record Node Creation Rule

$$\frac{\eta \mapsto \hat{v} \notin \rho \qquad \rho[\![\eta \mapsto \{\overline{l \mapsto \hat{v}}\}]\!] \Rightarrow \rho'}{(\overset{R}{+}\eta(\overline{l = \hat{v}}))\ \rho \Rightarrow \rho'}$$

List Node Creation Rule

$$\frac{\begin{array}{c}\eta \mapsto \hat{v} \notin \rho \\ \rho[\![\eta \mapsto [(\triangleright, \mathcal{M}, \emptyset), (\triangleleft, \mathcal{M}, \emptyset)]]\!] \Rightarrow \rho'\end{array}}{(\mathcal{M} \succ \overset{L}{+}\eta)\ \rho \Rightarrow \rho'}$$

Record Assignment Rule

$$\frac{\eta \mapsto \mathcal{R} \in \rho \qquad \rho[\![\eta \mapsto \mathcal{R}[\![l \mapsto \hat{v}]\!]]\!] \Rightarrow \rho'}{(\eta.l \leftarrow \hat{v})\ \rho \Rightarrow \rho'}$$

List Add Before Rule

$$\frac{\begin{array}{c}\overset{\triangle}{\eta}_3 \neq \triangleright \qquad \eta_1 \mapsto \mathcal{L} \in \rho \\ \overset{\circ}{\eta}_3 = \Sigma(\overset{\triangle}{\eta}_3, \mathcal{M}, \mathcal{L}) \qquad \mathcal{L} = [\overline{\overset{\circ}{\eta}'}, \overset{\circ}{\eta}_3, \overline{\overset{\circ}{\eta}''}] \\ \mathcal{L}' = [\overline{\overset{\circ}{\eta}'}, (\eta_2, \mathcal{M}, \emptyset), \overset{\circ}{\eta}_3, \overline{\overset{\circ}{\eta}''}] \\ \rho[\![\eta_1 \mapsto \mathcal{L}']\!] \Rightarrow \rho'\end{array}}{(\mathcal{M} \succ \eta_1 : \eta_2 \looparrowleft \overset{\triangle}{\eta}_3)\ \rho \Rightarrow \rho'}$$

List Add After Rule

$$\frac{\begin{array}{c}\overset{\triangle}{\eta}_3 \neq \triangleleft \qquad \eta_1 \mapsto \mathcal{L} \in \rho \\ \overset{\circ}{\eta}_3 = \Sigma(\overset{\triangle}{\eta}_3, \mathcal{M}, \mathcal{L}) \qquad \mathcal{L} = [\overline{\overset{\circ}{\eta}'}, \overset{\circ}{\eta}_3, \overline{\overset{\circ}{\eta}''}] \\ \mathcal{L}' = [\overline{\overset{\circ}{\eta}'}, \overset{\circ}{\eta}_3, (\eta_2, \mathcal{M}, \emptyset), \overline{\overset{\circ}{\eta}''}] \\ \rho[\![\eta_1 \mapsto \mathcal{L}']\!] \Rightarrow \rho'\end{array}}{(\mathcal{M} \succ \eta_1 : \overset{\triangle}{\eta}_3 \looparrowright \eta_2)\ \rho \Rightarrow \rho'}$$

List Remove Rule

$$\frac{\begin{array}{c}\eta_1 \mapsto \mathcal{L} \in \rho \\ \overset{\circ}{\eta}_2 = (\eta_2, \mathcal{M}', \mathcal{S}) = \Sigma(\eta_2, \mathcal{M}, \mathcal{L}) \\ \mathcal{L} = [\overline{\overset{\circ}{\eta}'}, \overset{\circ}{\eta}_2, \overline{\overset{\circ}{\eta}''}] \\ \mathcal{L}' = [\overline{\overset{\circ}{\eta}'}, (\eta_2, \mathcal{M}', \mathcal{S} \cup \{\mathcal{M}\}), \overline{\overset{\circ}{\eta}''}] \\ \rho[\![\eta_1 \mapsto \mathcal{L}']\!] \Rightarrow \rho'\end{array}}{(\mathcal{M} \succ \eta_1 : \downarrow \eta_2)\ \rho \Rightarrow \rho'}$$

Recursive Application Rule

$$\frac{\delta'\ e \Rightarrow \rho \qquad \delta\ \rho \Rightarrow \rho'}{\delta\ (\delta'\ e) \Rightarrow \rho'}$$

Value Rule

$$\frac{}{\hat{v} \Rightarrow \hat{v}}$$

Record Assignment Conflict Rule

$$\frac{\hat{v} \neq \hat{v}'}{\eta.l \leftarrow \hat{v} \nleftrightarrow \eta.l \leftarrow \hat{v}'}$$

Add Before Conflict Rule

$$\frac{\omega(\eta_2) \qquad \omega(\eta_2')}{\eta_1 : \eta_2 \looparrowleft \overset{\triangle}{\eta}_3 \nleftrightarrow \eta_1 : \eta_2' \looparrowleft \overset{\triangle}{\eta}_3}$$

Add After Conflict Rule

$$\frac{\omega(\eta_2) \qquad \omega(\eta_2')}{\eta_1 : \overset{\triangle}{\eta}_3 \looparrowright \eta_2 \nleftrightarrow \eta_1 : \overset{\triangle}{\eta}_3 \looparrowright \eta_2'}$$

Unordered Creation Conflict Rule

$$\frac{\delta = \overset{R}{+}\eta(\overline{l \mapsto \hat{v}}) \vee \delta = \overset{L}{+}\eta \qquad \eta \in \delta'}{\delta \nleftrightarrow \delta'}$$

Huzzah!

- Metaprogram conflicts are detected at compile time

Huzzah!

- Metaprogram conflicts are detected at compile time
- Metaprograms are still aware of their surroundings

Huzzah!

- Metaprogram conflicts are detected at compile time
- Metaprograms are still aware of their surroundings

☺

Dependencies

So how do we resolve the conflict?

Dependencies

```
public static void main(String[] arg) {
  [:

    BlockStatementListNode list = context.getAnchor().
      getNearestAncestorOfType(
          BlockStatementListNode.class);
    list.addFirst(
        <:System.out.println("Hello, world!");:>);
  :]
  [:

    BlockStatementListNode list = context.getAnchor().
      getNearestAncestorOfType(
          BlockStatementListNode.class);
    list.addFirst(
        <:System.out.println("How are you?");:>);
  :]
}
```

Dependencies

```
public static void main(String[] arg) {
  [:

    BlockStatementListNode list = context.getAnchor().
      getNearestAncestorOfType(
         BlockStatementListNode.class);
    list.addFirst(
       <:System.out.println("Hello, world!");:>);
  :]
  [:
    #target foo; ← Declare target membership
    BlockStatementListNode list = context.getAnchor().
      getNearestAncestorOfType(
         BlockStatementListNode.class);
    list.addFirst(
       <:System.out.println("How are you?");:>);
  :]
}
```

Dependencies

```
public static void main(String[] arg) {
  [:
    #depends foo; ← Declare target dependency
    BlockStatementListNode list = context.getAnchor().
      getNearestAncestorOfType(
         BlockStatementListNode.class);
    list.addFirst(
       <:System.out.println("Hello, world!");:>);
  :]
  [:
    #target foo;
    BlockStatementListNode list = context.getAnchor().
      getNearestAncestorOfType(
         BlockStatementListNode.class);
    list.addFirst(
       <:System.out.println("How are you?");:>);
  :]
}
```

Dependencies

```
public static void main(String[] arg) {
  [:
    #depends foo;
    BlockStatementListNode list = context.getAnchor().
      getNearestAncestorOfType(
          BlockStatementListNode.class);
    list.addFirst(
        <:System.out.println("Hello, world!");:>);
  :]
  [:
    #target foo;
    BlockStatementListNode list = context.getAnchor().
      getNearestAncestorOfType(
          BlockStatementListNode.class);
    list.addFirst(
        <:System.out.println("How are you?");:>);
  :]
}
```

Dependency Graph

- One node per metaprogram

Dependency Graph

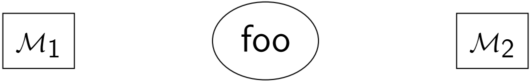
$\mathcal{M}_1$ $\mathcal{M}_2$

- One node per metaprogram

Dependency Graph

- One node per metaprogram
- $\mathcal{M}_2$ is a member of target “foo”

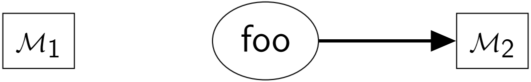
Dependency Graph



$\mathcal{M}_1$ foo $\mathcal{M}_2$

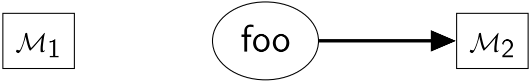
- One node per metaprogram
- $\mathcal{M}_2$ is a member of target “foo”

Dependency Graph



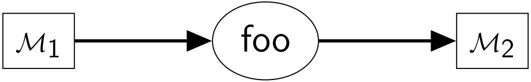
- One node per metaprogram
- $\mathcal{M}_2$ is a member of target “foo”

Dependency Graph



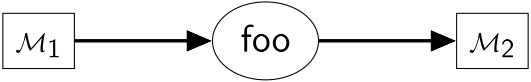
- One node per metaprogram
- $\mathcal{M}_2$ is a member of target "foo"
- $\mathcal{M}_1$ depends on the target "foo"

Dependency Graph



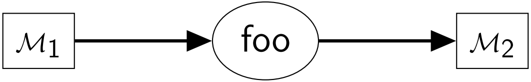
- One node per metaprogram
- $\mathcal{M}_2$ is a member of target “foo”
- $\mathcal{M}_1$ depends on the target “foo”

Dependency Graph



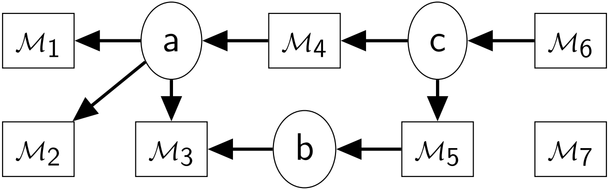
- One node per metaprogram
- $\mathcal{M}_2$ is a member of target "foo"
- $\mathcal{M}_1$ depends on the target "foo"
- Therefore, $\mathcal{M}_1$ depends on $\mathcal{M}_2$ ($\mathcal{M}_1 \rightsquigarrow \mathcal{M}_2$)

Dependency Graph



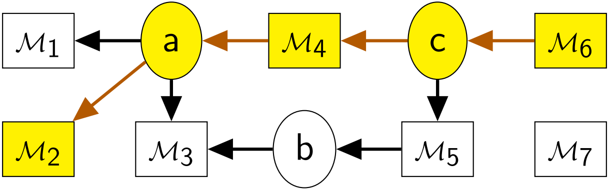
- One node per metaprogram
- $\mathcal{M}_2$ is a member of target "foo"
- $\mathcal{M}_1$ depends on the target "foo"
- Therefore, $\mathcal{M}_1$ depends on $\mathcal{M}_2$ ($\mathcal{M}_1 \leadsto \mathcal{M}_2$)
- No more requirement for them to commute!

Dependency Graph



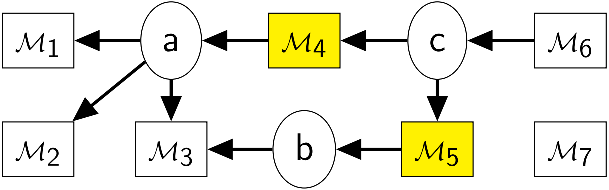
`/*` $\mathcal{M}_1$ `*/ [:  #target a; :]`
`/*` $\mathcal{M}_2$ `*/ [:  #target a; :]`
`/*` $\mathcal{M}_3$ `*/ [:  #target a, b; :]`
`/*` $\mathcal{M}_4$ `*/ [:  #target c; #depends a; :]`
`/*` $\mathcal{M}_5$ `*/ [:  #target c; #depends b; :]`
`/*` $\mathcal{M}_6$ `*/ [:  #depends c; :]`
`/*` $\mathcal{M}_7$ `*/ [:  :]`

Dependency Graph



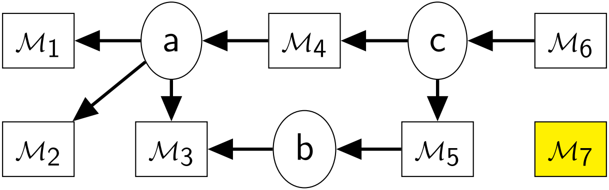
- $\mathcal{M}_6$ depends on $\mathcal{M}_2$ - no obligation to commute

Dependency Graph



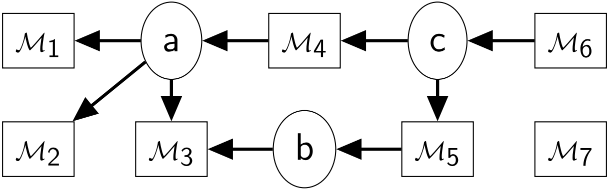
- $\mathcal{M}_6$ depends on $\mathcal{M}_2$ - no obligation to commute
- No path between $\mathcal{M}_5$ and $\mathcal{M}_4$ - must commute

Dependency Graph



- $\mathcal{M}_6$ depends on $\mathcal{M}_2$ - no obligation to commute
- No path between $\mathcal{M}_5$ and $\mathcal{M}_4$ - must commute
- No path to $\mathcal{M}_7$ - must *always* commute

Dependency Graph



- $\mathcal{M}_6$ depends on $\mathcal{M}_2$ - no obligation to commute
- No path between $\mathcal{M}_5$ and $\mathcal{M}_4$ - must commute
- No path to $\mathcal{M}_7$ - must *always* commute
- More paths means less obligation to prove commutativity

A more practical example...

A more practical example...

...but first, a new feature

Meta-Annotations

```
@@Property private int x;
```

- Declarative metaprogramming abstraction

Meta-Annotations

```
@@Property private int x;
```

- Declarative metaprogramming abstraction
- Specifies metaprogram code and dependencies

Meta-Annotations

```
@@Property private int x;
```

- Declarative metaprogramming abstraction
- Specifies metaprogram code and dependencies
- Can annotate any declaration or block statement

Meta-Annotations

```
@@Property private int x;
```

- Declarative metaprogramming abstraction
- Specifies metaprogram code and dependencies
- Can annotate any declaration or block statement
- Allows easy reuse of metaprogramming constructs

Meta-Annotations

```
@@Property private int x;
```

- Declarative metaprogramming abstraction
- Specifies metaprogram code and dependencies
- Can annotate any declaration or block statement
- Allows easy reuse of metaprogramming constructs
- Here defined by user class named `Property`

Meta-Annotation Dependencies

```
@@GenerateConstructorFromProperties
public class Location {
  @@Property private int x;
  @@Property private int y;
  public String toString() {
    return "("+this.x+","+this.y+")";
  }
}
```

Meta-Annotation Dependencies

```
@@GenerateConstructorFromProperties
public class Location {
  @@Property private int x;
  @@Property private int y;
  public String toString() {
    return "("+this.x+","+this.y+")";
  }
}
```

- Meta-annotation defns. include dependencies

Meta-Annotation Dependencies

```
@@GenerateConstructorFromProperties
public class Location {
  @@Property private int x;
  @@Property private int y;
  public String toString() {
    return "("+this.x+","+this.y+")";
  }
}
```

- Meta-annotation defns. include dependencies
- `@@Property` is a member of target “property”

Meta-Annotation Dependencies

```
@@GenerateConstructorFromProperties
public class Location {
  @@Property private int x;
  @@Property private int y;
  public String toString() {
    return "("+this.x+","+this.y+")";
  }
}
```

- Meta-annotation defns. include dependencies
- `@@Property` is a member of target "property"
- `@@GenerateConstructorFromProperties` depends on "property"

BSJ Dependency Graph

- One node per metaprogram

BSJ Dependency Graph

`@@Property`

`@@Property`

`@@GenerateConstructorFromProperties`

- One node per metaprogram

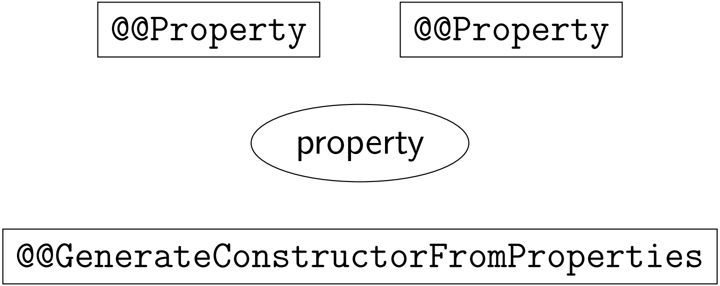
BSJ Dependency Graph

`@@Property` `@@Property`

`@@GenerateConstructorFromProperties`

- One node per metaprogram
- `@@Property` participates in “property” target

BSJ Dependency Graph



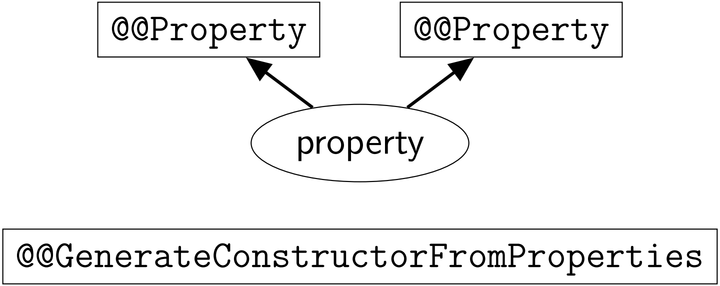
`@@Property` `@@Property`

property

`@@GenerateConstructorFromProperties`

- One node per metaprogram
- `@@Property` participates in “property” target

BSJ Dependency Graph



`@@Property`

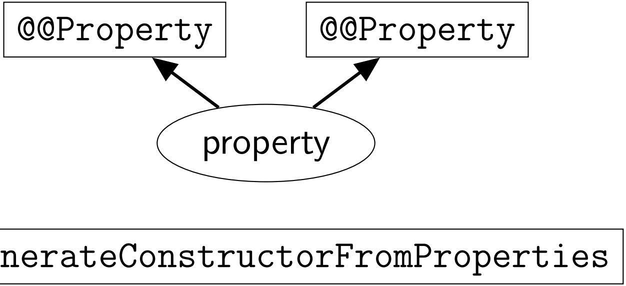
`@@Property`

property

`@@GenerateConstructorFromProperties`

- One node per metaprogram
- `@@Property` participates in “property” target

BSJ Dependency Graph



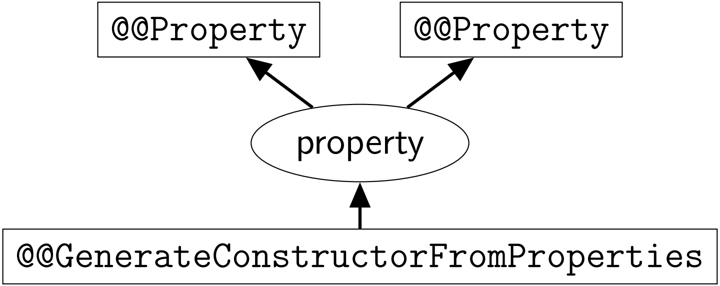
`@@Property` `@@Property`

property

`@@GenerateConstructorFromProperties`

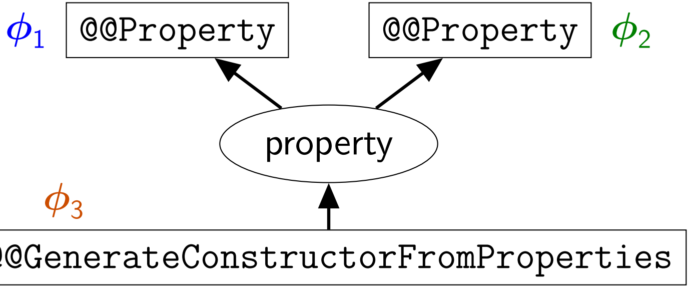
- One node per metaprogram
- `@@Property` participates in “property” target
- `@@GenerateConstructorFromProperties` depends on “property”

BSJ Dependency Graph



- One node per metaprogram
- `@@Property` participates in “property” target
- `@@GenerateConstructorFromProperties` depends on “property”

BSJ Dependency Graph



ϕ_1 `@@Property`

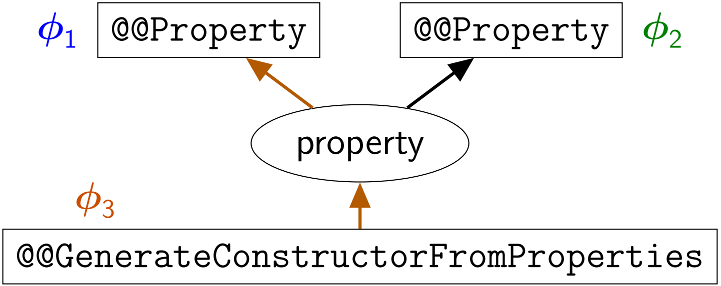
`@@Property` ϕ_2

property

ϕ_3

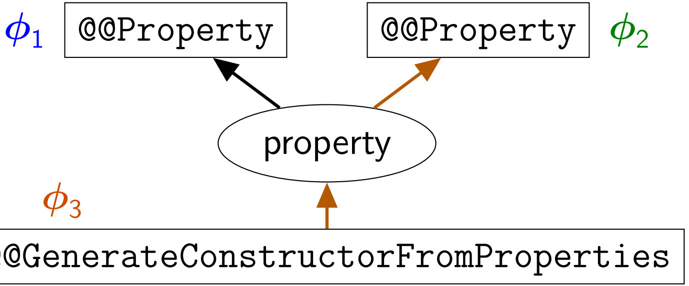
`@@GenerateConstructorFromProperties`

BSJ Dependency Graph



- ϕ_3 depends on ϕ_1

BSJ Dependency Graph



ϕ_1 `@@Property`

`@@Property` ϕ_2

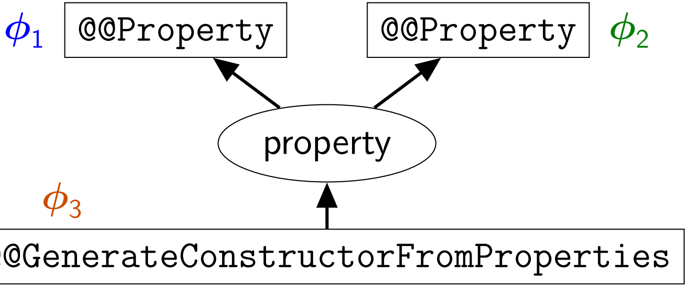
property

ϕ_3

`@@GenerateConstructorFromProperties`

- ϕ_3 depends on ϕ_1
- ϕ_3 depends on ϕ_2

BSJ Dependency Graph



ϕ_1 `@@Property`

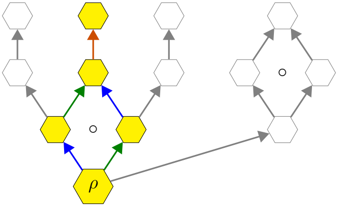
`@@Property` ϕ_2

property

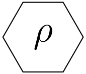
ϕ_3

`@@GenerateConstructorFromProperties`

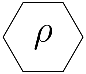
- ϕ_3 depends on ϕ_1
- ϕ_3 depends on ϕ_2



Execution Example

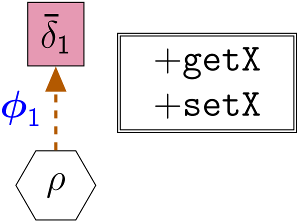


Execution Example



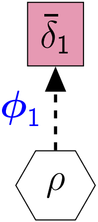
- Execute ϕ_1 obtaining $\bar{\delta}_1$.

Execution Example



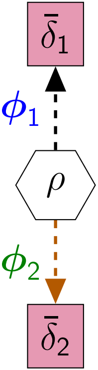
- Execute ϕ_1 obtaining $\bar{\delta}_1$.

Execution Example



- Execute ϕ_1 obtaining $\bar{\delta}_1$.
- Execute ϕ_2 obtaining $\bar{\delta}_2$.

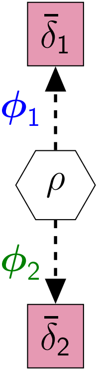
Execution Example



```
+getY
+setY
```

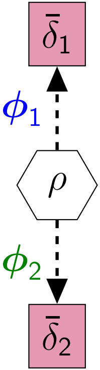
- Execute ϕ_1 obtaining $\bar{\delta}_1$.
- Execute ϕ_2 obtaining $\bar{\delta}_2$.

Execution Example



- Execute ϕ_1 obtaining $\bar{\delta}_1$.
- Execute ϕ_2 obtaining $\bar{\delta}_2$.
- Prove $[\![\bar{\delta}_1]\!]$ and $[\![\bar{\delta}_2]\!]$ commute.

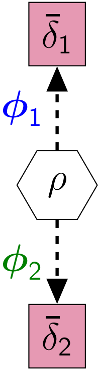
Execution Example



`+getX`
`+setX`
`+getY`
`+setY`

- Execute ϕ_1 obtaining $\bar{\delta}_1$.
- Execute ϕ_2 obtaining $\bar{\delta}_2$.
- Prove $[\![\bar{\delta}_1]\!]$ and $[\![\bar{\delta}_2]\!]$ commute.

Execution Example

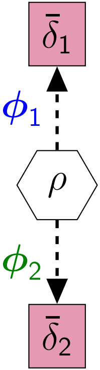


+getX
+setX
+getY
+setY

+getY
+setY
+getX
+setX

- Execute ϕ_1 obtaining $\bar{\delta}_1$.
- Execute ϕ_2 obtaining $\bar{\delta}_2$.
- Prove $[\![\bar{\delta}_1]\!]$ and $[\![\bar{\delta}_2]\!]$ commute.

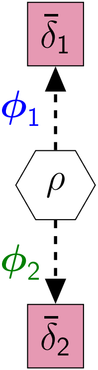
Execution Example



`+getX`		`+getY`
`+setX`	$\cong$	`+setY`
`+getY`		`+getX`
`+setY`		`+setX`

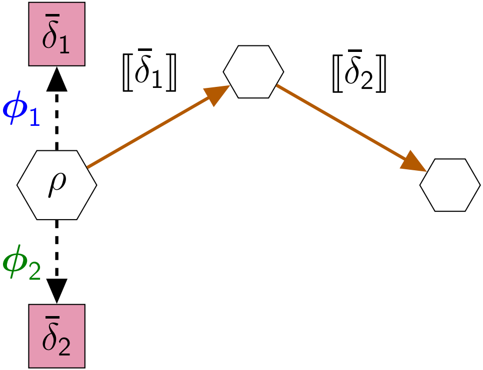
- Execute ϕ_1 obtaining $\bar{\delta}_1$.
- Execute ϕ_2 obtaining $\bar{\delta}_2$.
- Prove $[\![\bar{\delta}_1]\!]$ and $[\![\bar{\delta}_2]\!]$ commute.

Execution Example



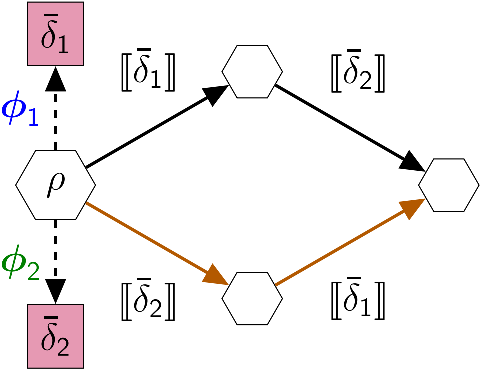
- Apply $[\![\bar{\delta}_1]\!]$ and $[\![\bar{\delta}_2]\!]$ to ρ.

Execution Example



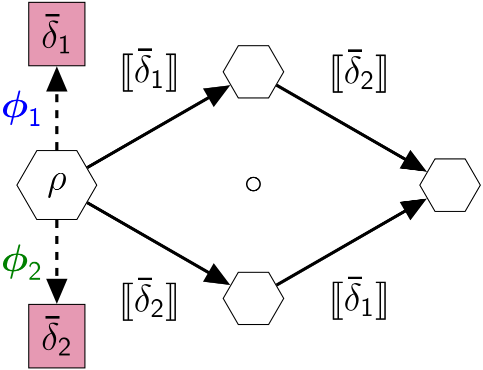
- Apply $[\![\bar{\delta}_1]\!]$ and $[\![\bar{\delta}_2]\!]$ to ρ.

Execution Example



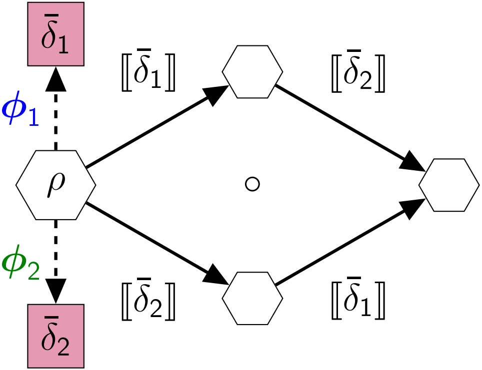
- Apply $[\![\bar{\delta}_1]\!]$ and $[\![\bar{\delta}_2]\!]$ to ρ.

Execution Example



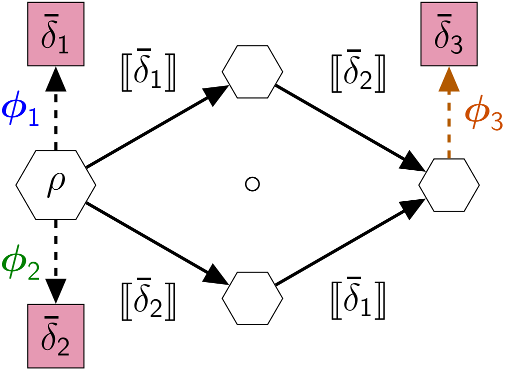
- Apply $[\![\bar{\delta}_1]\!]$ and $[\![\bar{\delta}_2]\!]$ to ρ.

Execution Example



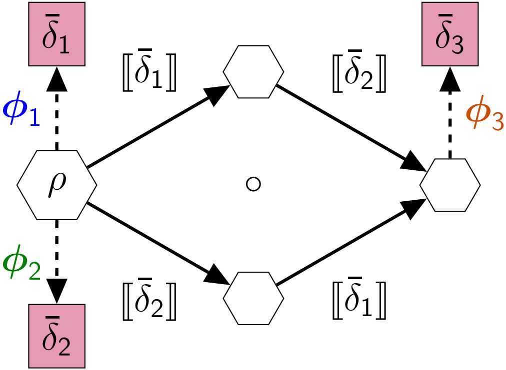
- Apply $[\![\bar{\delta}_1]\!]$ and $[\![\bar{\delta}_2]\!]$ to ρ.
- Execute ϕ_3 on the result to get $\bar{\delta}_3$.

Execution Example



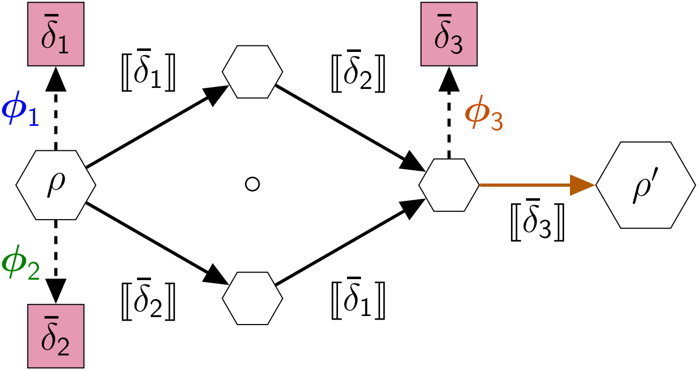
- Apply $[\![\bar{\delta}_1]\!]$ and $[\![\bar{\delta}_2]\!]$ to ρ.
- Execute ϕ_3 on the result to get $\bar{\delta}_3$.

Execution Example



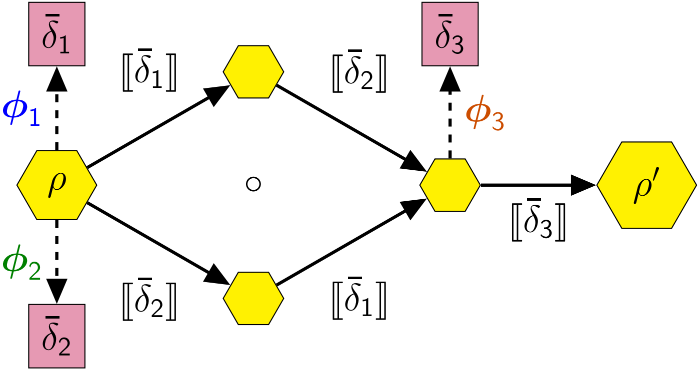
- Apply $[\![\bar{\delta}_1]\!]$ and $[\![\bar{\delta}_2]\!]$ to ρ.
- Execute ϕ_3 on the result to get $\bar{\delta}_3$.
- Apply $[\![\bar{\delta}_3]\!]$ to get the final object program.

Execution Example



- Apply $\llbracket\bar{\delta}_1\rrbracket$ and $\llbracket\bar{\delta}_2\rrbracket$ to ρ.
- Execute ϕ_3 on the result to get $\bar{\delta}_3$.
- Apply $\llbracket\bar{\delta}_3\rrbracket$ to get the final object program.

Execution Example



- Apply $[\![\bar{\delta}_1]\!]$ and $[\![\bar{\delta}_2]\!]$ to ρ.
- Execute ϕ_3 on the result to get $\bar{\delta}_3$.
- Apply $[\![\bar{\delta}_3]\!]$ to get the final object program.

Difference-Based Metaprogramming Summary

- Ambiguities detected at compile-time

Difference-Based Metaprogramming Summary

- Ambiguities detected at compile-time
- Metaprograms can inspect their environments

Difference-Based Metaprogramming Summary

- Ambiguities detected at compile-time
- Metaprograms can inspect their environments
- Modular, declarative metaprogramming style

Difference-Based Metaprogramming Summary

- Ambiguities detected at compile-time
- Metaprograms can inspect their environments
- Modular, declarative metaprogramming style
- Suitable for OO languages like Java

Backstage Java Implementation

- Working reference implementation available

Backstage Java Implementation

- Working reference implementation available
- Includes source (∼50k SLOC)

Backstage Java Implementation

- Working reference implementation available
- Includes source (∼50k SLOC)
- Full superset of Java 1.6

BSJ Standard Library – `@@Memoized`

`@@Memoized`

BSJ Standard Library – `@@Memoized`

`@@Memoized`

- Class for generating images

BSJ Standard Library – `@@Memoized`

`@@Memoized`

- Class for generating images
- Each image is generated from pair of `Colors`

BSJ Standard Library – `@@Memoized`

`@@Memoized`

- Class for generating images
- Each image is generated from pair of `Colors`
- Memoizing image generation routine for performance

BSJ Standard Library – `@@Memoized`

`@@Memoized`

- Class for generating images
- Each image is generated from pair of `Colors`
- Memoizing image generation routine for performance
- Store cached images in a private `Map` keyed by input to generation method

BSJ Standard Library – `@@Memoized`

```
public class ImageGenerator {

  public Image gen(Color a, Color b) {···}
}
```

BSJ Standard Library – @@Memoized

```
public class ImageGenerator {
  @@Memoized
  public Image gen(Color a, Color b) {···}
}
```

BSJ Standard Library – @@Memoized

```
public class ImageGenerator {



  public Image gen(Color a, Color b) {···}



}
```

BSJ Standard Library – `@@Memoized`

```
public class ImageGenerator {







  private Map<???,Image> cache = new ···
  public Image gen(Color a, Color b) {···}




}
```

BSJ Standard Library – `@@Memoized`

```
public class ImageGenerator {
  private static class Key {
    private Color a;
    private Color b;
    ...
  }
  private Map<???,Image> cache = new ···
  public Image gen(Color a, Color b) {···}



}
```

BSJ Standard Library – @@Memoized

```
public class ImageGenerator {
  private static class Key {
    private Color a;
    private Color b;
    ...
  }
  private Map<Key,Image> cache = new ···
  public Image gen(Color a, Color b) {···}



}
```

BSJ Standard Library – @@Memoized

```
public class ImageGenerator {
  private static class Key {
    private Color a;
    private Color b;
    ...
  }
  private Map<Key,Image> cache = new ···
  private Image igen(Color a, Color b) {···}



}
```

BSJ Standard Library – @@Memoized

```
public class ImageGenerator {
  private static class Key {
    private Color a;
    private Color b;
    ...
  }
  private Map<Key,Image> cache = new ···
  private Image igen(Color a, Color b) {···}
  public Image gen(Color a, Color b) {
    /* return cache value, igen as needed */
  }
}
```

BSJ Standard Library – @@Memoized

```
public class ImageGenerator {
  @@Memoized
  public Image gen(Color a, Color b) {···}
}
```

Difference-Based Metaprogramming

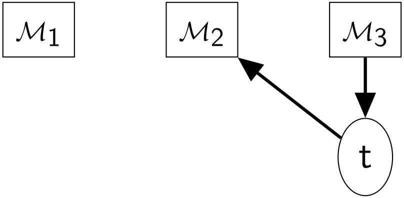
Difference-Based Metaprogramming
Questions?

Expressiveness

Difference-based metaprogramming separates analysis from modification.

```
public class Example {
    private int x = 0;
    private int y = 0;
    @@LogAndCount
    public void foo() { ⋯ }
    @@LogAndCount
    public void bar() { ⋯ }
}
```

Injection Conflicts



Injection Conflicts