
Guided Refactoring

Christoph Reichenbach, Amer Diwan
University of Colorado at Boulder, January 9, 2007

PROGRAMMING IS HARD

- Specifications change
- Our understanding increases
- Thus, software evolves

We use the following mechanisms for evolution:

- Manual editing
- Refactoring

PROGRAMMING IS HARD

- Specifications change
- Our understanding increases
- Thus, software evolves

We use the following mechanisms for evolution:

- Manual editing
- Refactoring

REFACTORING...

Martin Fowler (Fowler, 1999) distinguishes “*refactoring*”:

- *as noun*: Program transformation,
preserves behaviour
- *as verb*: Process of transforming programs,
preserves behaviour

Difference in practice?

REFACTORING...

Martin Fowler (Fowler, 1999) distinguishes “*refactoring*”:

- as *noun*: Program transformation,
preserves behaviour *atomically*
- as *verb*: Process of transforming programs,
preserves behaviour *holistically*

Difference in practice!

EXISTING REFACTORING SYSTEMS

Eclipse, HaRe, IntelliJ ...

- Refactorings are atomically behaviour-preserving
- Sometimes disallow convenient (but incorrect) transformations
- Sometimes perform additional (implicit) transformations

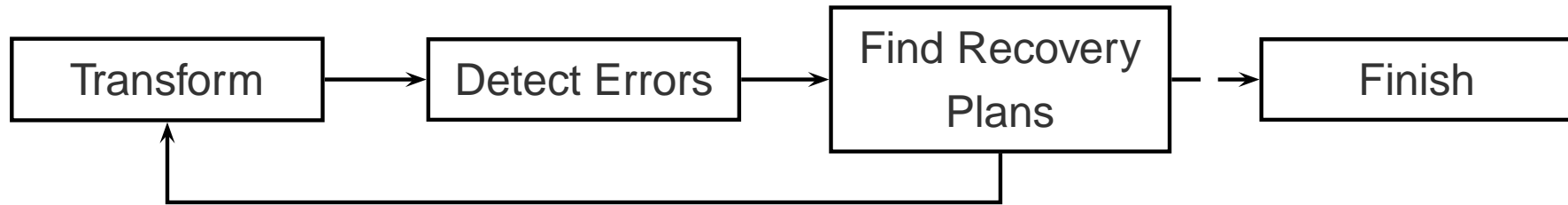
EXISTING REFACTORING SYSTEMS

Eclipse, HaRe, IntelliJ ...

- Refactorings are atomically behaviour-preserving
- Sometimes disallow convenient (but incorrect) transformations
- Sometimes perform additional (implicit) transformations

Alternative: “Guided Refactoring”

GUIDED REFACTORING: OVERVIEW



GUIDED REFACTORING: AN EXAMPLE

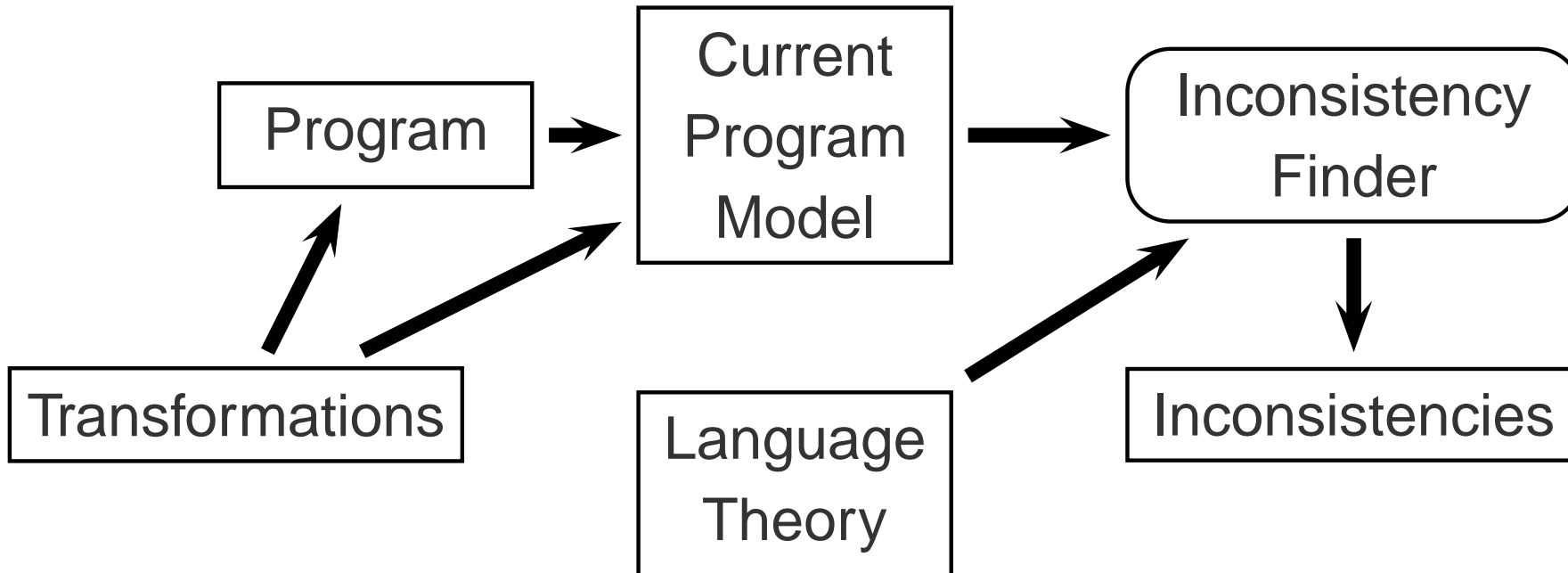
```
val result = 1746
fun f(x) = let val y = x * x
           in  if y = result
              then 0
              else sign (y - result)
           end
```

GUIDED REFACTORING: AN EXAMPLE

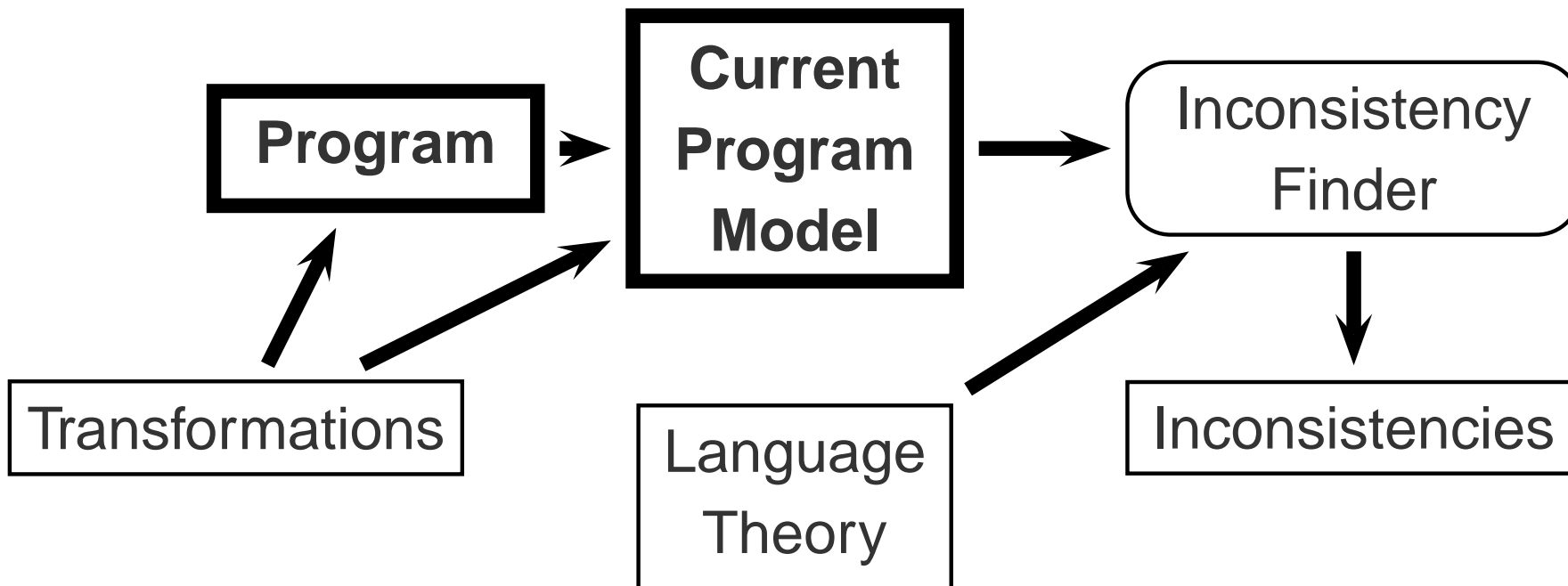
```
val result = 1746
fun f(val) = let val y = val * val
            in  if y = result
               then 0
               else sign (y - result)
            end
```

Invalid Name “val” (reserved keyword)

IMPLEMENTING GUIDED REFACTORING



IMPLEMENTING GUIDED REFACTORING



REPRESENTING THE PROGRAM (1/2)

```
val result = 1746
fun f(x) = let val y = x * x
           in  if y = result
              then 0
              else sign (y - result)
           end
```

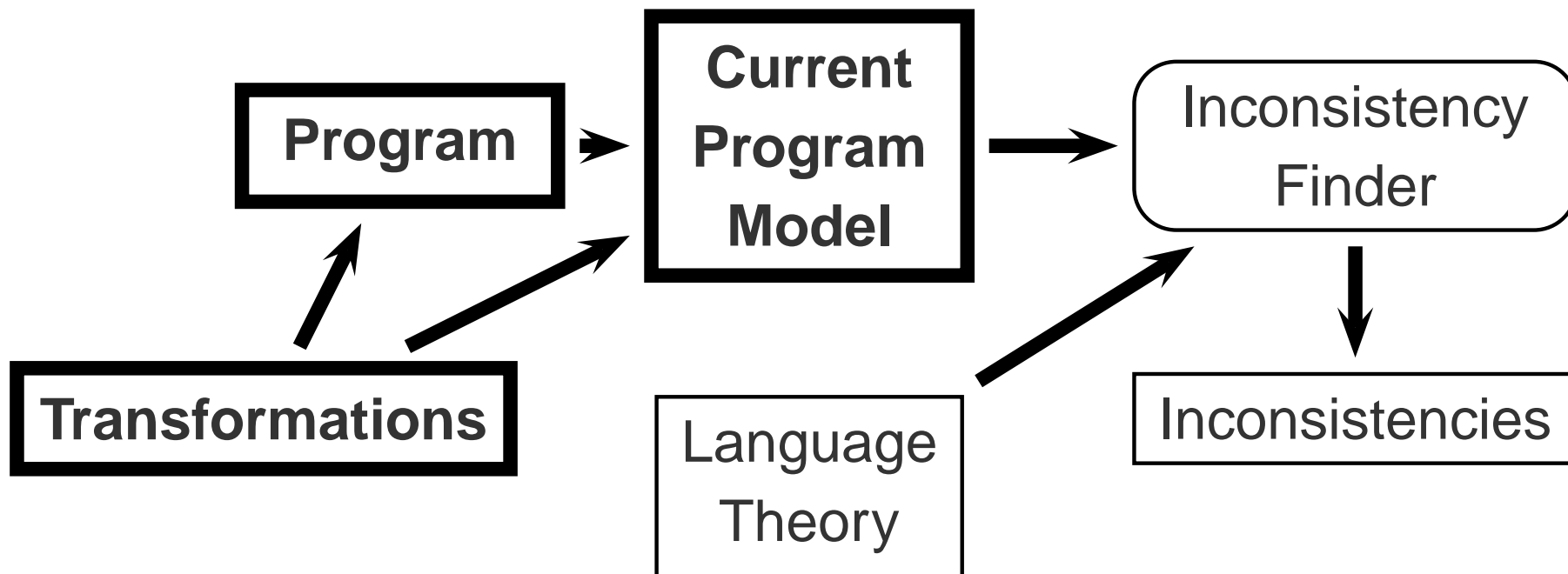
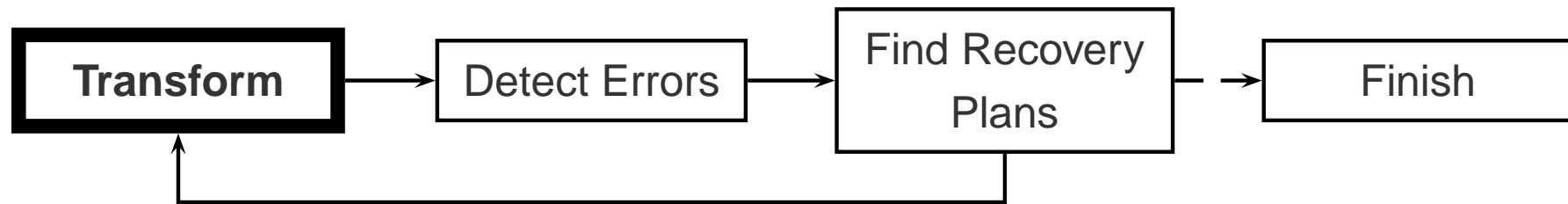
REPRESENTING THE PROGRAM (2/2)

```
val i0 = 1746
fun i1(i2) = let val i3 = i2 * i2
               in  if i3 = i0
                   then 0
                   else sign (i3 - i0)
               end
```

Current Program Model

Identifiers	Defining Locations	Scope structure
$i_0 \xrightarrow{r} \text{"result"}$
$i_1 \xrightarrow{r} \text{"f"}$...
$i_2 \xrightarrow{r} \text{"x"}$		
$i_3 \xrightarrow{r} \text{"y"}$		

IMPLEMENTING GUIDED REFACTORING



THE RENAMING TRANSFORMATION

Program Transformation:

rename (i : id , n : name, p : program) = ...

Model Transformation:

Elim: $i \xrightarrow{r} n'$

Intro: $i \xrightarrow{r} n$

APPLYING THE RENAMING TRANSFORMATION

Program Transformation:

rename (i_2 , "val", p : program) = ...

Model Transformation:

Elim: $i_2 \xrightarrow{r} n'$

Intro: $i_2 \xrightarrow{r} \text{"val"}$

RENAMING: CHANGING THE CURRENT PROGRAM MODEL

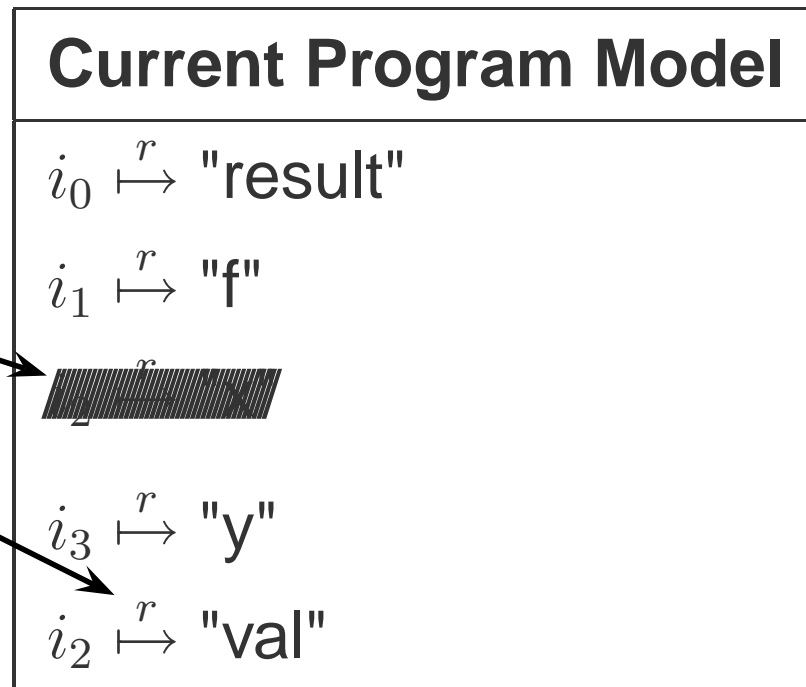
Program Transformation:

rename (i_3 , "val", p : program) = ...

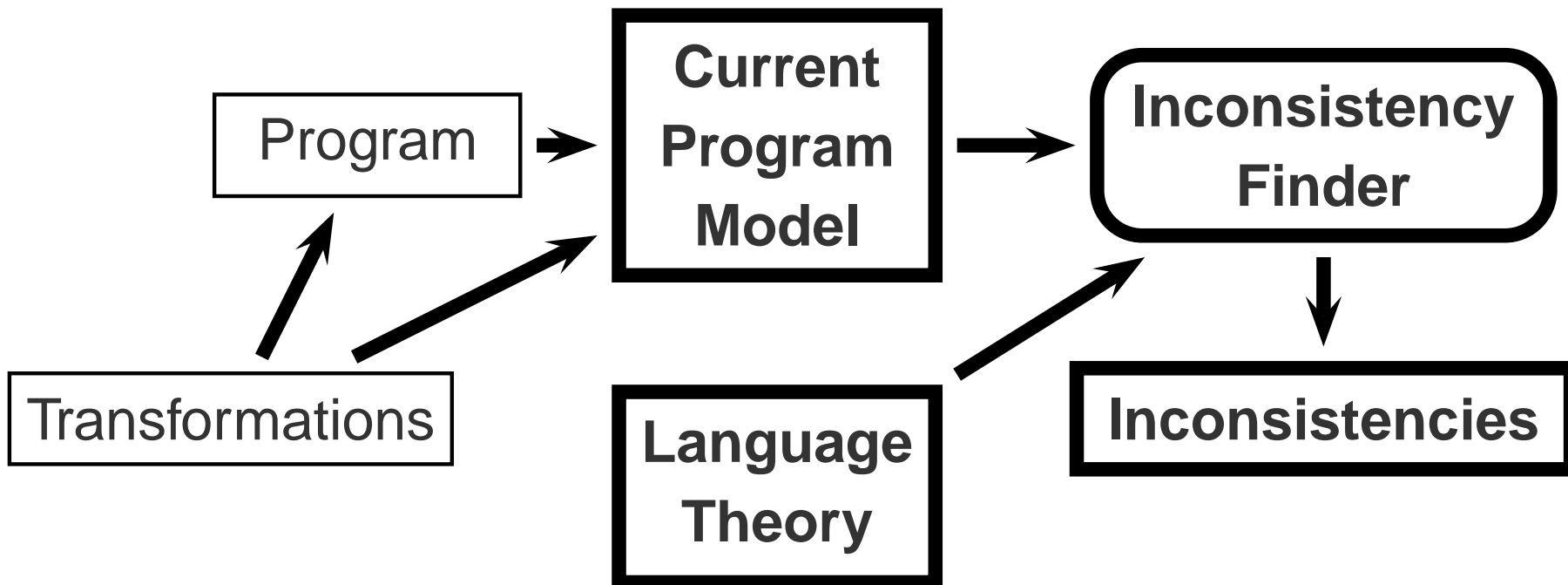
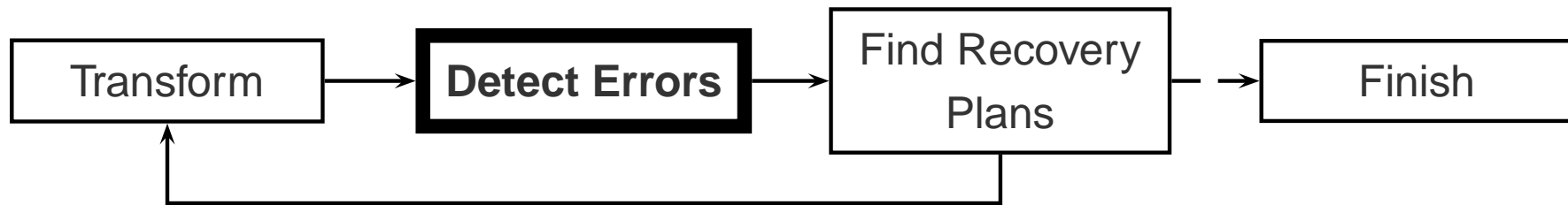
Model Transformation:

Elim: $i_2 \xrightarrow{r} n'$

Intro: $i_2 \xrightarrow{r} \text{"val"}$



IMPLEMENTING GUIDED REFACTORING



DETECTING INCONSISTENCIES (SIMPLE)

Language Theory, error conditions:

- HasInvalidName
- IsNotReachable
- ...

Language Theory, inference rules:

$$\frac{i \xrightarrow{r} \text{"let"}}{\text{HasInvalidName}(i)} \quad \frac{i \xrightarrow{r} \text{"val"}}{\text{HasInvalidName}(i)} \quad \dots$$

AN EXAMPLE

```
val result = 1746
fun f(val) = let val y = val * val
            in  if y = result
                then 0
                else sign (y - result)
            end
```

Invalid Name “val” (reserved keyword)

AN EXAMPLE

```
val result = 1746
fun f(val) = let val result = val * val
              in if result = result
                  then 0
                  else sign (result - result)
              end
```

Name Capture of “result”
*Invalid Name “**val**” (reserved keyword)*

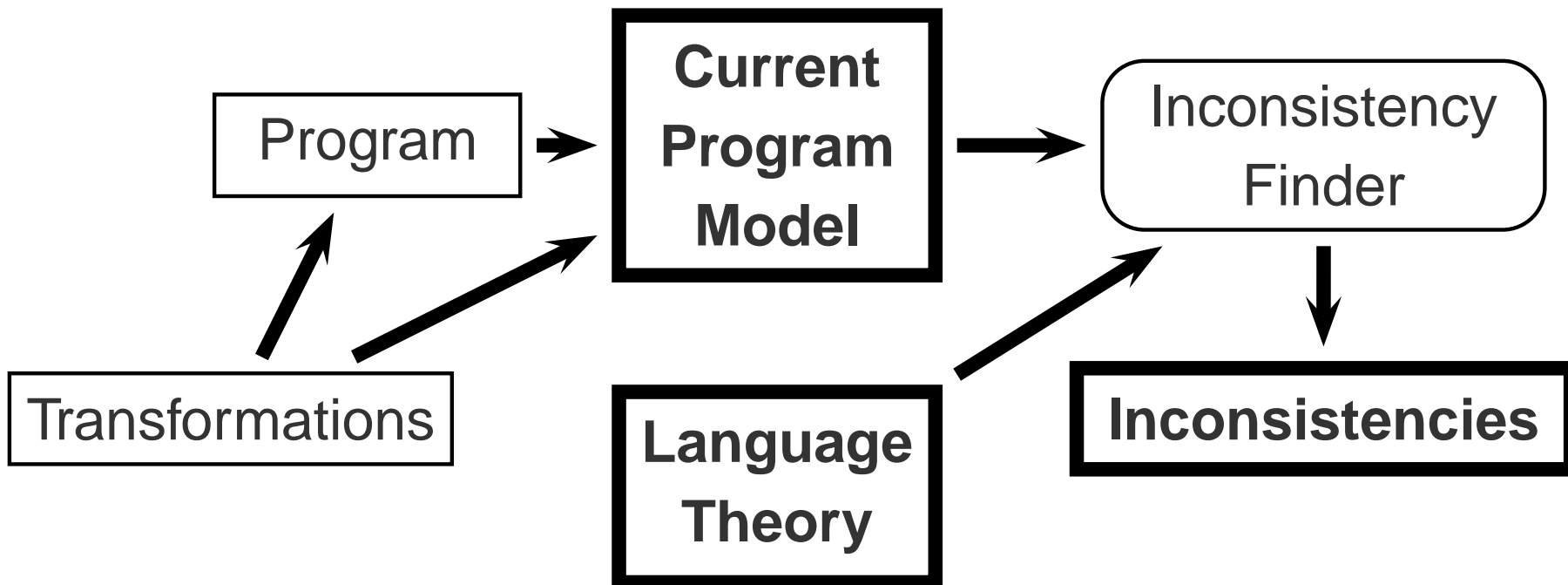
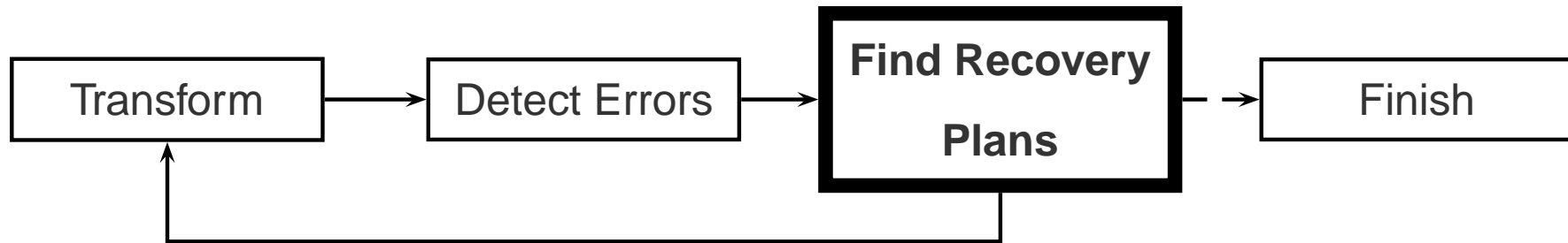
AN EXAMPLE

```
val result = 1746
```

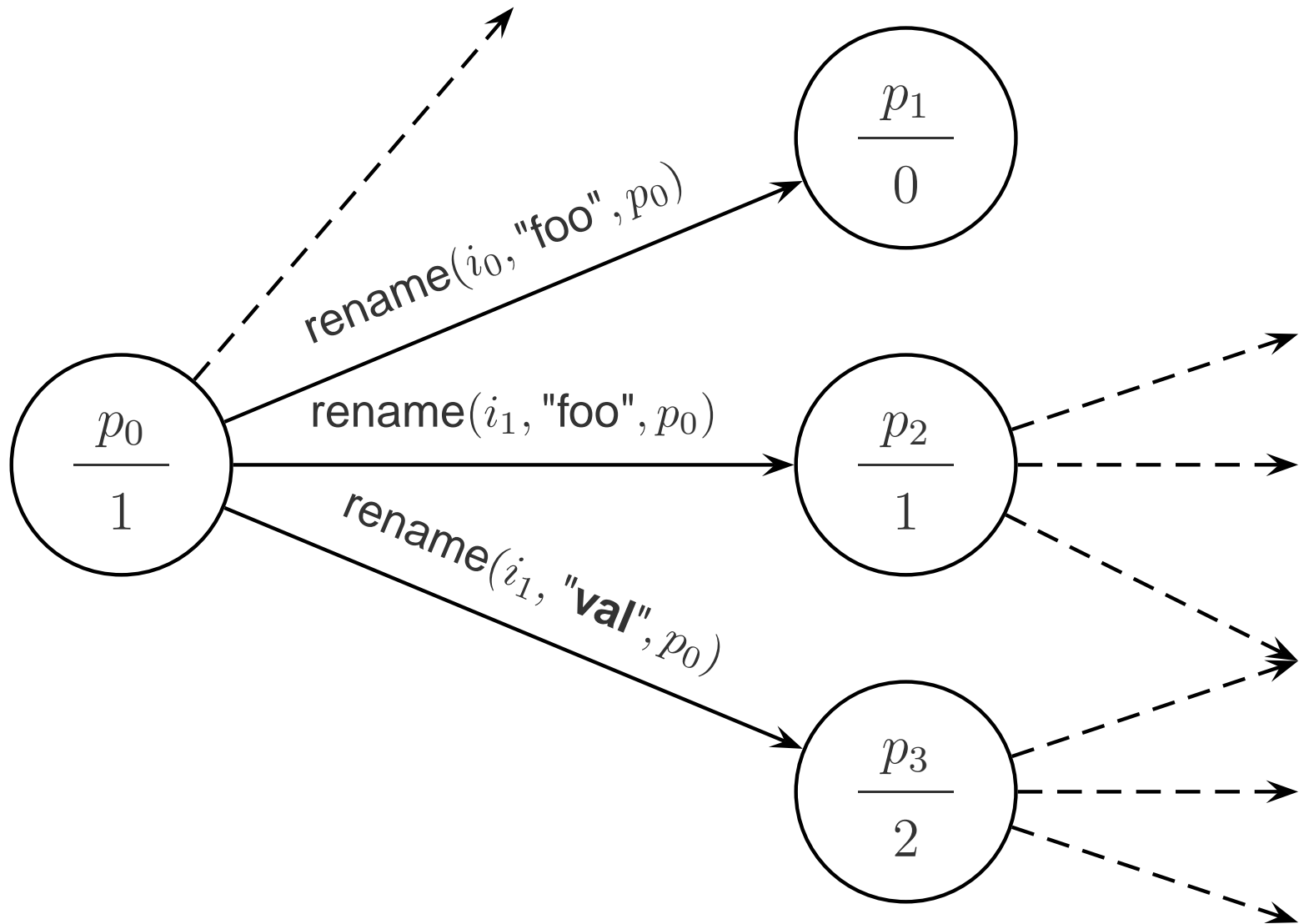
```
fun f(x) = let val result = x * x  
           in  if result = result  
              then 0  
              else sign (result - result)  
           end
```

Name Capture of “result”

IMPLEMENTING GUIDED REFACTORING



FINDING RECOVERY PLANS



AN EXAMPLE

```
val result = 1746
fun f(x) = let val result = x * x
           in if result = result
              then 0
              else sign (result - result)
           end
```

Name Capture of “result”

AN EXAMPLE

```
val result = 1746
fun f(x) = let val result = x * x
           in if result = result
              then ...
              else ...
           end
```

Name C

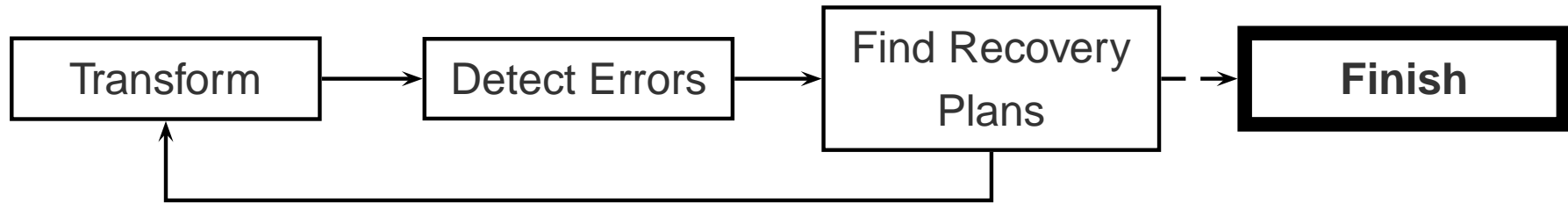
Recovery Plan Choices

- A Rename "result"
- B Rename "result"
- C Inline "result"
- D Delete "**fun f(x)**", and then...

AN EXAMPLE

```
val desired_result = 1746
fun f(x) = let val result = x * x
           in if result = desired_result
           then 0
           else sign (result - desired_result)
           end
```

GUIDED REFACTORING: OVERVIEW



GUIDED REFACTORING IN COMPARISON

Property	Atomic Refac- toring	Guided Refac- toring
Behaviour Preservation	Atomic	Holistic
Interface	Editor commands	Bimodal editor
Preconditions	Semantic	Syntactic
Transformations	Vary from requested	As requested
Implementation	Transform, Test	Transform, Model Transform

RELATED WORK

- Refactoring
 - Eclipse (Shavor et al., 2003), HaRe (Thompson and Reinke, 2001), IntelliJ
 - “Refactoring Object-Oriented Frameworks” (Opdyke, 1992)
 - “Static Composition of Refactorings” (Kniesel, Koch, 2004)
- Program Refinement
 - Munich CIP project (Bauer et al., 1985)
 - MetaWSL (Ward et al., 2005)
- Planning
 - Recent survey (Nareyek et al., 2005)
 - Planning as a CSP (Lopez, Bacchus, 2003)
 - PDDL (McDermott et al., 1998)

SUMMARY

With Guided Refactoring:

- Behaviour-preservation is checked/required only after the programmer is done
- Transformations must describe their abstract effect on the Program Model
- Transformations need not provide correctness checks (Preconditions)
- Tests for correctness are independent of concrete transformations

Backup Slides

PROGRAM METAMORPHOSIS

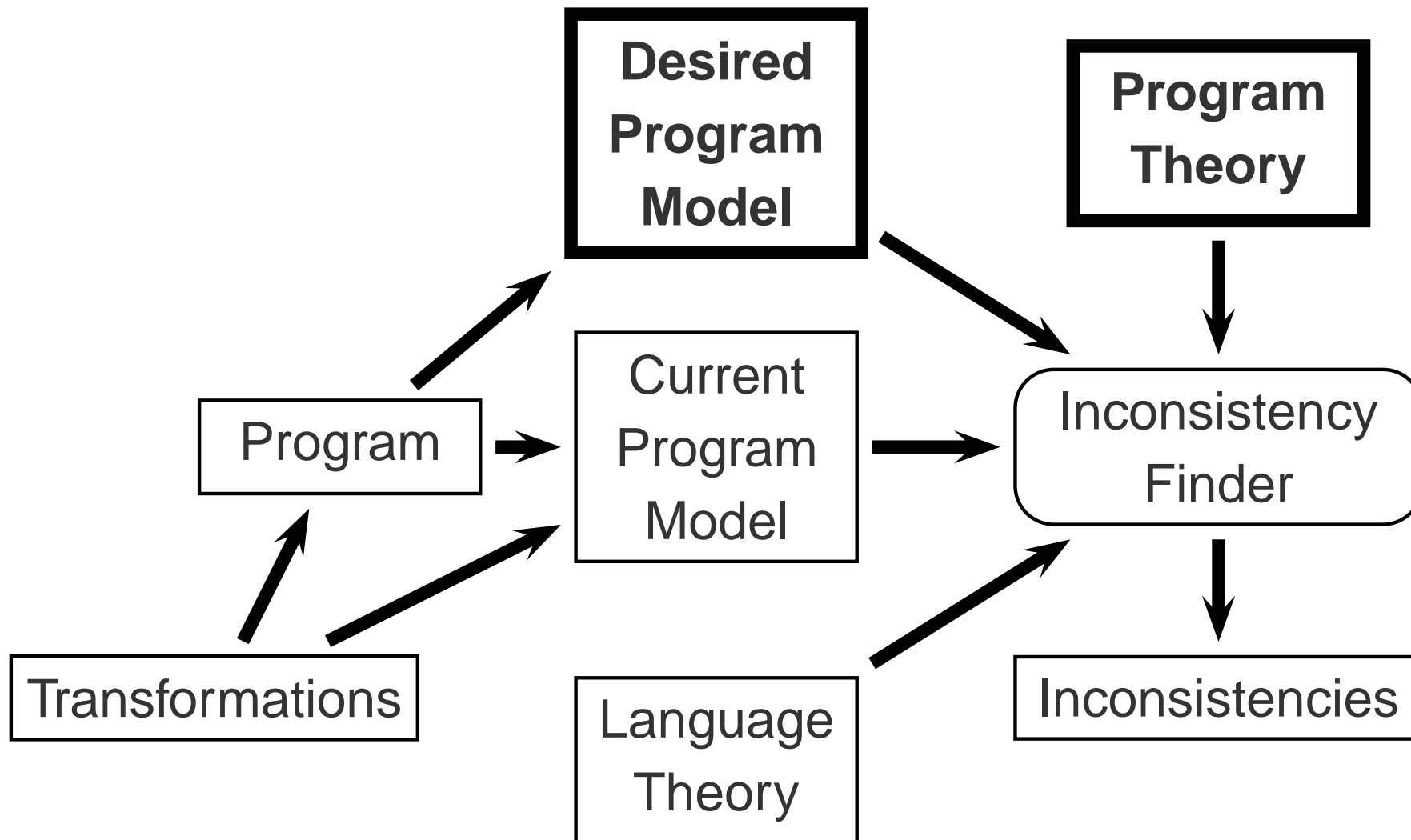
Guided Refactoring +

- Some transformations only alter program model (“accept change”)
- Transformation effects are *imprecisely* represented by program model

Program Model may be abstract:

- “print("foo") added between X and Y ”
- “print("foo") added”
- “I/O effect added”
- “side effect added”
- “side effects altered”

IMPLEMENTING PROGRAM METAMORPHOSIS



PROGRAM METAMORPHOSIS: AN EXAMPLE

```
val desired_result = 1746
fun f(x) = let val result = x * x
           in if result = desired_result
             then 0
             else sign (result - desired_result)
           end
```

PROGRAM METAMORPHOSIS: AN EXAMPLE

```
fun f(x) = let val result = x * x
           val desired_result = 1746
           in if result = desired_result
           then 0
           else sign (result - desired_result)
           end
```

Missing Export: “desired_result” no longer accessible

PROGRAM METAMORPHOSIS: AN EXAMPLE

```
fun f(x) = let val result = x * x
            val desired_result = 1746
            in if result = desired_result
               then 0
               else sign (result - desired_result)
            end
```

Recovery Plan Choices	
A	Move “desired_result” from “f” to toplevel
0	<i>Accept Change:</i> “desired_result” inaccessible