Oracle-based Differential Operational Semantics

(Or explaining program differences with programs)

Thibaut Girka [1,2] David Mentré [1] Yann Régis-Gianas [2]

[1]Mitsubishi Electric R&D Centre Europe, F-35708 Rennes, France
[2]Univ Paris Diderot, Sorbonne Paris Cité, IRIF, INRIA PiR2

November 8, 2016

What is the meaning of a program evolution?

P_0

```
sum = 0;
x = -x;
y = 0;
while (sum < x) {
  y = y + 1;
  sum += 1 + 2 * y;
}
sum = 0;
```

P_2

```
x = -x;
sum = 0;
count = 0;
while (sum < x) {
  count = count + 1;
  sum += 1 + 2 * count;
}
sum = 0;
```

How to express changes between two close programs?

```
From: Thibaut Girka <thibaut.girka@irif.fr>
Subject: [PATCH] Change stuff
```

```diff
-sum = 0;
 x = -x;
-y = 0;
+sum = 0;
+count = 0;
 while (sum < x) {
-   y = y + 1;
-   sum += 1 + 2 * y;
+   count = count + 1;
+   sum += 1 + 2 * count;
 }
 sum = 0;
```

How to express changes between two close programs?

```
From: Thibaut Girka <thibaut.girka@irif.fr>
Subject: [PATCH] Rename y to count ;
  Change variable initialization order
```

```
-sum = 0;
 x = -x;
-y = 0;
+sum = 0;
+count = 0;
 while (sum < x) {
-  y = y + 1;
-  sum += 1 + 2 * y;
+  count = count + 1;
+  sum += 1 + 2 * count;
 }
 sum = 0;
```

What could be
a formal **difference language**?

A formal difference

```
From: Thibaut Girka <thibaut.girka@irif.fr>
Subject: [PATCH] Rename y → count ;
                 SwapAssign@0
```

```
-sum = 0;
 x = -x;
-y = 0;
+sum = 0;
+count = 0;
 while (sum < x) {
-   y = y + 1;
-   sum += 1 + 2 * y;
+   count = count + 1;
+   sum += 1 + 2 * count;
 }
 sum = 0;
```

Difference languages
(as wanted by programmers)

Syntax

- Readable change descriptions to express **intent**
- Sufficiently **declarative** to be understood by programmers

Semantics

- **Mechanically verifiable**
- Reason about **program evolution**
- Ease code review
- Formal ground to build incremental development tools

Difference languages
(as wanted by semanticists)

A general framework to compare program behaviors

- Plenty of frameworks to reason about program equivalence
- Not that much to **compare** (inequivalent) programs
- To justify **differential static analysis**
- To specify transformations that **do not preserve semantics**

Difference languages
(as wanted by semanticists)

A general framework to compare program behaviors

- Plenty of frameworks to reason about program equivalence
- Not that much to **compare** (inequivalent) programs
- To justify **differential static analysis**
- To specify transformations that **do not preserve semantics**

How should we relate the reduction of two close programs?

Difference languages
(as wanted by semanticists)

A general framework to compare program behaviors

- Plenty of frameworks to reason about program equivalence
- Not that much to **compare** (inequivalent) programs
- To justify **differential static analysis**
- To specify transformations that **do not preserve semantics**

How should we relate the reduction of two close programs?

Using a program!

Oracle-based Differential Operational Semantics

(The idea)

P_1 $\qquad$ $c_1^1 \longrightarrow$

P_2

Oracle-based Differential Operational Semantics

(The idea)

$$
\begin{tikzcd}
P_1 & c_1^1 \arrow[r] \arrow[d, dash, dashed, "\sim"'] \arrow[dr, phantom, "\|\ \mathcal{O}(\delta)\ \Downarrow"] & {} \\
P_2 & c_2^1 \arrow[r, dashed, two heads] & {}
\end{tikzcd}
$$

Oracle-based Differential Operational Semantics

(The idea)

$$\begin{tikzcd}
P_1 & c_1^1 \arrow[r] \arrow[d, dash, "\sim"] \arrow[r, phantom, shift right=6, "\mathcal{O}(\delta)"] & c_1^2 \arrow[d, dash, "\sim"] \\
P_2 & c_2^1 \arrow[r, dashed, two heads] & c_2^2
\end{tikzcd}$$

Oracle-based Differential Operational Semantics

(The idea)

$$
\begin{tikzcd}
P_1 & c_1^1 \arrow[r] \arrow[d, dash, "\sim"] \arrow[dr, phantom, "\mathcal{O}(\delta)"] & c_1^2 \arrow[d, dash, "\sim"] & \\
P_2 & c_2^1 \arrow[r, dashed, two heads] & c_2^2 \arrow[r] &
\end{tikzcd}
$$

Oracle-based Differential Operational Semantics

(The idea)

$$
\begin{tikzcd}
P_1 & c_1^1 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\Downarrow\,\mathcal{O}(\delta)"] & c_1^2 \arrow[r, dashed, two heads] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\Uparrow\,\mathcal{O}(\delta)"] & c_1^3 \arrow[d, dash, "\sim" description] \\
P_2 & c_2^1 \arrow[r, dashed, two heads] & c_2^2 \arrow[r] & c_2^3
\end{tikzcd}
$$

Oracle-based Differential Operational Semantics

(The idea)

$$\begin{tikzcd}
P_1 & c_1^1 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\mathcal{O}(\delta)\Downarrow"] & c_1^2 \arrow[r, dashed, two heads] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\Uparrow\mathcal{O}(\delta)"] & c_1^3 \arrow[d, dash, "\sim" description] & \cdots \\
P_2 & c_2^1 \arrow[r, dashed, two heads] & c_2^2 \arrow[r] & c_2^3 & \cdots
\end{tikzcd}$$

Oracle-based Differential Operational Semantics

(by analogy)

An **interpreter** maps a **program** to its **reduction trace**.

Oracle-based Differential Operational Semantics
(by analogy)

An **interpreter** maps a **program** to its **reduction trace**.

An **oracle** maps a **difference** between P_1 and P_2 to a **relation between their reduction traces**.

Oracle-based Differential Operational Semantics

Our contributions

- A formal language-agnostic framework for difference languages
- Instantiated on the `Imp` toy language
- Examples of difference languages implemented in `Coq` [1]
- What makes a difference sound?
- How to compose two differences?

This talk

- A tour through several examples of difference languages
- To highlight the expressivity of this framework

[1] Check it out! : `https://www.irif.fr/~thib/oracles`

Oracles through examples

P_0

```
sum = 0;
x = -x;
y = 0;
while (sum < x) {
  y = y + 1;
  sum += 1 + 2 * y;
}
sum = 0;
```

P_2

```
x = -x;
sum = 0;
count = 0;
while (sum < x) {
  count = count + 1;
  sum += 1 + 2 * count;
}
sum = 0;
```

Renaming

$$y \rightarrow count$$

P_0

```
sum = 0;
x = -x;
y = 0;
while (sum < x) {
  y = y + 1;
  sum += 1 + 2 * y;
}
sum = 0;
```

P_1

```
sum = 0;
x = -x;
count = 0;
while (sum < x) {
  count = count + 1;
  sum += 1 + 2 * count;
}
sum = 0;
```

Renaming: semantics (simpl.)

$$\begin{array}{ccc} c_8 & \xrightarrow{(\delta\kappa_8, [y{:=}1])} & c_9 \\ \wr & & \\ c_8' & & \end{array}$$

Renaming: semantics (simpl.)

$$\begin{tikzcd}[row sep=huge, column sep=huge]
c_8 \arrow[r, "{(\delta\kappa_8, [y{:=}1])}"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\parallel \; \mathcal{O}(\delta) \; \Downarrow"] & c_9 \arrow[d, dash, "\sim" description] \\
c'_8 \arrow[r, dashed, two heads, "{(\delta\kappa_8, [count{:=}1])}"'] & c'_9
\end{tikzcd}$$

Renaming: semantics (simpl.)

$$
\begin{tikzcd}[column sep=huge, row sep=huge]
c_8 \arrow[r, "{(\delta\kappa_8, [y{:=}1])}"] \arrow[d, dash, "\sim"'] \arrow[dr, phantom, "\begin{array}{c} \| \\ \mathcal{O}(\delta) \\ \Downarrow \end{array}"] & c_9 \arrow[r, "{(\delta\kappa_9, [sum{:=}3])}"] \arrow[d, dash, "\sim"] & c_{10} \\
c'_8 \arrow[r, dashed, twoheadrightarrow, "{(\delta\kappa_8, [count{:=}1])}"'] & c'_9 &
\end{tikzcd}
$$

Renaming: semantics (simpl.)

$$\begin{tikzcd}[column sep=huge, row sep=huge]
c_8 \arrow[r, "{(\delta\kappa_8, [y{:=}1])}"] \arrow[d, dash, dashed, "\sim" description] \arrow[dr, phantom, "\shortstack{$\|$\\ $\mathcal{O}(\delta)$\\ $\Downarrow$}"] & c_9 \arrow[r, "{(\delta\kappa_9, [sum{:=}3])}"] \arrow[d, dash, dashed, "\sim" description] \arrow[dr, phantom, "\shortstack{$\|$\\ $\mathcal{O}(\delta)$\\ $\Downarrow$}"] & c_{10} \arrow[d, dash, dashed, "\sim" description] \\
c'_8 \arrow[r, dashed, twoheadrightarrow, "{(\delta\kappa_8, [count{:=}1])}"'] & c'_9 \arrow[r, dashed, twoheadrightarrow, "{(\delta\kappa_9, [sum{:=}3])}"'] & c'_{10}
\end{tikzcd}$$

Renaming: semantics (simpl.)

$$\mathcal{O}(\mathit{y} \mapsto \mathit{count})((\delta\kappa, \delta M))$$
$$= (\delta\kappa, \delta M[\mathit{y} \mapsto \mathit{count}])$$

Renaming: semantics (simpl.)

$$
\begin{tikzcd}
c_0 \arrow[r, "\delta_0"] \arrow[d, dash, "\sim" description] & c_1 \arrow[r, dashed, two heads, "\delta_1"] \arrow[d, dash, "\sim" description] & c_2 \arrow[r, "\delta_2"] \arrow[d, dash, "\sim" description] & c_3 \arrow[d, dash, "\sim" description] \\
c'_0 \arrow[r, dashed, two heads, "\delta'_0"'] & c'_1 \arrow[r, "\delta'_1"'] & c'_2 \arrow[r, dashed, two heads, "\delta'_2"'] & c'_3
\end{tikzcd}
$$

Type of an oracle's prediction function

(As a first approximation)

$$d \times \delta c \nrightarrow \delta c$$

- $d = \{\downharpoonleft, \upharpoonleft\}$
  (direction of the prediction)
- δc: reified step

Assignment commutation

SwapAssign@0

P_1

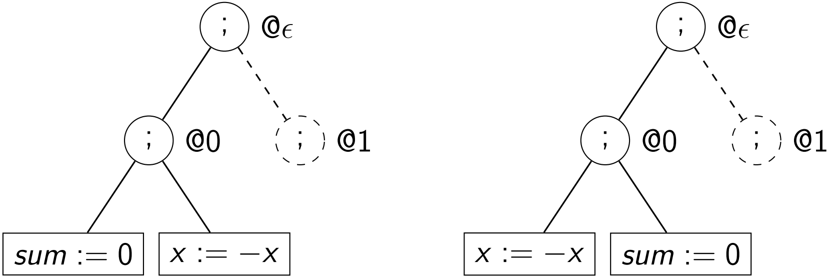
```
sum = 0;
x = -x;
count = 0;
while (sum < x) {
   count = count + 1;
   sum += 1 + 2 * count;
}
sum = 0;
```

P_2

```
x = -x;
sum = 0;
count = 0;
while (sum < x) {
   count = count + 1;
   sum += 1 + 2 * count;
}
sum = 0;
```

Assignment commutation

$SwapAssign@0$



Assignment commutation

Oracle features highlight

- Cannot always be called in both directions
- Need to maintain an internal state
- May predict 0, 1 or 2 steps
- Relate syntactic path to dynamic evaluation points

Assignment commutation: semantics

A peek into its internal state

This oracle maintains an internal state with:

- A list of *continuation modifiers* to relate the syntactic path to dynamic execution points
- A delta on stores during commutations
- The direction it was called on during commutations

Assignment commutation: semantics

$$
\begin{tikzcd}
c_1 \arrow[r, "\delta_1"] \arrow[d, dash, "\sim"'] \arrow[dr, phantom, "\downharpoonleft \upharpoonleft"] & c_2 \arrow[d, dash, "\sim"] \\
c_1' \arrow[r, dashed, two heads, "\delta_1"'] & c_2'
\end{tikzcd}
$$

$\dots$

$\delta_1 = ([\textbf{unfold-seq}], [])$

$\dots$

$\delta_1' = ([\textbf{unfold-seq}], [])$

Assignment commutation: semantics

$$\begin{tikzcd}
c_1 \arrow[r, "\delta_1"] \arrow[d, dash, "\sim"'] \arrow[dr, phantom, "\downharpoonleft\upharpoonleft"] & c_2 \arrow[r, "\delta_2"] \arrow[d, dash, "\sim"'] \arrow[dr, phantom, "\downharpoonleft"] & c_3 \arrow[d, dash, "\sim"'] \\
c_1' \arrow[r, dashed, twoheadrightarrow, "\delta_1"'] & c_2' \arrow[r, dashed, twoheadrightarrow, "="'] & c_2'
\end{tikzcd}$$

...

$\delta_1 = ($[**unfold-seq**]$, [])$

$\delta_2 = ($[**pop**]$, [sum := 0])$

...

$\delta_1' = ($[**unfold-seq**]$, [])$

Assignment commutation: semantics

$$
\begin{tikzcd}
c_1 \arrow[r, "\delta_1"] \arrow[d, dash, "\sim"'] \arrow[dr, phantom, "\downharpoonleft\upharpoonleft"] & c_2 \arrow[r, "\delta_2"] \arrow[d, dash, "\sim"'] \arrow[dr, phantom, "\downharpoonleft"] & c_3 \arrow[r, "\delta_3"] \arrow[d, dash, "\sim"'] \arrow[dr, phantom, "\downharpoonleft\upharpoonleft"] & c_4 \arrow[d, dash, "\sim"'] \\
c_1' \arrow[r, dashed, twoheadrightarrow, "\delta_1"'] & c_2' \arrow[r, dashed, twoheadrightarrow, "="'] & c_2' \arrow[r, dashed, twoheadrightarrow, "\delta_3\delta_2"'] & c_4'
\end{tikzcd}
$$

...

$\delta_1 = ([\textbf{unfold-seq}], [])$

$\delta_2 = ([\textbf{pop}], [sum := 0])$

$\delta_3 = ([\textbf{pop}], [x := 42])$

...

$\delta_1' = ([\textbf{unfold-seq}], [])$

$\delta_2' = ([\textbf{pop}], [x := 42])$

$\delta_3' = ([\textbf{pop}], [sum := 0])$

Assignment commutation: semantics

$$
\begin{tikzcd}
c_1 \arrow[r, "\delta_1"] \arrow[d, dash, "\sim"] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_2 \arrow[r, "\delta_2"] \arrow[d, dash, "\sim"] \arrow[dr, phantom, "\downharpoonleft"] & c_3 \arrow[r, "\delta_3"] \arrow[d, dash, "\sim"] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_4 \arrow[r, dashed, two heads, "\delta'_4"] \arrow[d, dash, "\sim"] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_5 \arrow[d, dash, "\sim"] \\
c'_1 \arrow[r, dashed, two heads, "\delta_1"'] & c'_2 \arrow[r, dashed, two heads, "="'] & c'_2 \arrow[r, dashed, two heads, "\delta_3\delta_2"'] & c'_4 \arrow[r, "\delta'_4"'] & c'_5
\end{tikzcd}
$$

$\ldots$

$\delta_1 = ([\textbf{unfold-seq}], [])$

$\delta_2 = ([\textbf{pop}], [\mathit{sum} := 0])$

$\delta_3 = ([\textbf{pop}], [x := 42])$

$\delta_4 = ([\textbf{unfold-seq}], [])$

$\ldots$

$\ldots$

$\delta'_1 = ([\textbf{unfold-seq}], [])$

$\delta'_2 = ([\textbf{pop}], [x := 42])$

$\delta'_3 = ([\textbf{pop}], [\mathit{sum} := 0])$

$\delta'_4 = ([\textbf{unfold-seq}], [])$

$\ldots$

Type of an oracle's prediction function

$$s \times d \times \delta c \nrightarrow$$
$$s \times \dot{d} \times ((\mathbb{N} \setminus \{0\} \times \delta c) + \mathbf{wait})$$

- $\dot{d} = d \uplus \{\downharpoonright \upharpoonright\}$
  (allowed directions for the next prediction)
- s: internal oracle state

Abstraction of equivalent sub-programs

P_2

```
x = -x;
sum = 0;
count = 0;
while (sum < x) {
  count = count + 1;
  sum += 1 + 2 * count;
}
sum = 0;
```

P_3

```
x = -x;
sum = 0;
if (x < 4)  { count = 1; }
else {
  count = x;
  sum = (x + 1) / 2;
  while (sum < count) {
    count = sum;
    sum = x / sum + sum;
    sum /= 2;
  }
};
sum = 0;
```

Abstraction of equivalent sub-programs

P_2

```
x = -x;
```

```
sum = 0;
count = 0;
while (sum < x) {
  count = count + 1;
  sum += 1 + 2 * count;
}
sum = 0;
```

P_3

```
x = -x;
```

```
sum = 0;
if (x < 4) { count = 1; }
else {
  count = x;
  sum = (x + 1) / 2;
  while (sum < count) {
    count = sum;
    sum = x / sum + sum;
    sum /= 2;
  }
};
sum = 0;
```

Abstraction of equivalent sub-programs

Oracle features highlight

- May predict 0, 1, or a dynamic number of steps
- Abstracts away from the small-step presentation
- Defer work to proof obligations

Abstraction of equivalent sub-programs

$$AbstractEquiv@1, \ldots$$

- Path of equivalent subprograms (@1)
- The sub-programs themselves
- Proof of big-step equivalence
- Proof termination for each sub-program

Abstraction of equivalent sub-programs: semantics

A peek into its internal state

- List of *continuation modifiers* the same length as the continuation

or

- The current *prediction direction* ($\downharpoonleft$ or $\upharpoonleft$)
- The number of remaining continuation elements from the sub-program being evaluated
- A snapshot of the store before executing the sub-program (used to compute the bound on execution steps)
- The accumulated delta on the store so far

Abstraction of equivalent sub-programs: semantics

$$
\begin{tikzcd}
c_2 \arrow[r, "\delta_2"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonright"] & c_3 \arrow[d, dash, "\sim" description] \\
c_2' \arrow[r, dashed, two heads, "=" below] & c_2'
\end{tikzcd}
$$

Abstraction of equivalent sub-programs: semantics

$$
\begin{tikzcd}
c_2 \arrow[r, "\delta_2"] \arrow[d, dash, "\sim"] \arrow[dr, phantom, "\downharpoonleft"] & c_3 \arrow[r, "\cdots"] \arrow[d, dash, "\sim"] \arrow[dr, phantom, "\downharpoonleft \ \ldots \ \downharpoonleft"] & c_n \arrow[r, "\delta_n"] \arrow[d, dash, "\sim"] \arrow[dr, phantom, "\downharpoonleft"] & c_{n+1} \arrow[d, dash, "\sim"] \\
c'_2 \arrow[r, dashed, two heads, "="'] & c'_2 \arrow[r, dashed, two heads, "\overset{\cdots}{=}"'] & c'_2 \arrow[r, dashed, two heads, "="'] & c'_2
\end{tikzcd}
$$

Abstraction of equivalent sub-programs: semantics

$$
\begin{tikzcd}
c_2 \arrow[r, "\delta_2"] \arrow[d, dash, "\sim"] \arrow[dr, phantom, "\downharpoonleft"] & c_3 \arrow[r, dashed, "\cdots"] \arrow[d, dash, "\sim"] \arrow[dr, phantom, "\downharpoonleft \; \ldots \; \downharpoonleft"] & c_n \arrow[r, "\delta_n"] \arrow[d, dash, "\sim"] \arrow[dr, phantom, "\downharpoonleft"] & c_{n+1} \arrow[r, "\delta_{n+1}"] \arrow[d, dash, "\sim"] \arrow[dr, phantom, "\downharpoonleft \upharpoonleft"] & c_{n+2} \arrow[d, dash, "\sim"] \\
c'_2 \arrow[r, dashed, two heads, "="'] & c'_2 \arrow[r, dashed, two heads, "\overset{\cdots}{=}"'] & c'_2 \arrow[r, dashed, two heads, "="'] & c'_2 \arrow[r, dashed, two heads, "({[\mathbf{pop}]}, \ldots)"'] & c'_m
\end{tikzcd}
$$

Control-flow-preserving value changes

$$\mathit{ValueChange}\,[\mathit{pow}, a]\,(42 \rightarrow 10)@0$$

F_0

```
a = 42;
n = 5; pow = 1;
while (0 < n) {
  n = n - 1;
  pow = pow * a;
};
```

F_1

```
a = 10;
n = 5; pow = 1;
while (0 < n) {
  n = n - 1;
  pow = pow * a;
};
```

Control-flow-preserving value changes

Oracle features highlight

- Always predict exactly one step
- Always bidirectional
- Relate programs that are **not equivalent**
- Soundness follows from (overly restrictive) syntactic criteria

Control-flow-preserving value changes

$$ValueChange\,[pow, a]\,(42 \to 10)@0$$

- Path of an assignment to change (@0)
- Expression to substitute in the assignment's right-hand side $(42 \to 10)$
- Variables possibly affected ($[pow, a]$)

Control-flow-preserving value changes

A peek into its internal state

This oracle maintains an internal state with:

- A list of *continuation modifiers*
  to relate the syntactic path to dynamic execution points
- A store for each program

Control-flow-preserving value changes: semantics (simpl.)

$$
\begin{tikzcd}
c_8 \arrow[r, "{(\delta\kappa_8, [n{:=}4])}"] \arrow[d, dash, dashed, "\sim"] \arrow[dr, phantom, "\downharpoonleft\,\upharpoonright"] & c_9 \arrow[r, dashed, two heads, "{(\delta\kappa_9, [pow{:=}42])}"] \arrow[d, dash, dashed, "\sim"] \arrow[dr, phantom, "\downharpoonleft\,\upharpoonright"] & c_{10} \arrow[d, dash, dashed, "\sim"] \\
c'_8 \arrow[r, dashed, two heads, "{(\delta\kappa_8, [n{:=}4])}"'] & c'_9 \arrow[r, "{(\delta\kappa_9, [pow{:=}10])}"'] & c'_{10}
\end{tikzcd}
$$

Oracle languages on `Imp`: recap

We presented:

- Renaming
- Assignment commutation
- Abstraction of equivalent terminating sub-programs
- Control-flow-preserving value changes

We have also implemented:

- Sequence associativity
- Condition negation / Branch commutation
- Abstraction of equivalent unbounded sub-programs
- Abstraction of inequivalent sub-programs
- Crash avoidance

Composition: $P_0 \sim P_2$

$$
\begin{array}{ccc}
c_0^0 & \xrightarrow{\delta_0^0} & c_1^0 \\
| & & \\
\sim & & \\
| & & \\
c_0^1 & &
\end{array}
$$

$$
\begin{array}{c}
c_0^1 \\
| \\
\sim \\
| \\
c_0^2
\end{array}
$$

Composition: $P_0 \sim P_2$

$$
\begin{tikzcd}
c_0^0 \arrow[r, "\delta_0^0"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1^0 \arrow[d, dash, "\sim" description] \\
c_0^1 \arrow[r, dashed, twoheadrightarrow] & c_1^1
\end{tikzcd}
$$

$$
\begin{tikzcd}
c_0^1 \arrow[d, dash, "\sim" description] \\
c_0^2
\end{tikzcd}
$$

Composition: $P_0 \sim P_2$

$$
\begin{tikzcd}
c_0^0 \arrow[r, "\delta_0^0"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1^0 \arrow[d, dash, "\sim" description] \\
c_0^1 \arrow[r, dashed, two heads] & c_1^1
\end{tikzcd}
$$

$$
\begin{tikzcd}
c_0^1 \arrow[r, "\delta_0^1"] \arrow[d, dash, "\sim" description] & c_1^1 \\
c_0^2 &
\end{tikzcd}
$$

Composition: $P_0 \sim P_2$

$$
\begin{tikzcd}
c_0^0 \arrow[r, "\delta_0^0"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1^0 \arrow[d, dash, "\sim" description] \\
c_0^1 \arrow[r, dashed, twoheadrightarrow] & c_1^1
\end{tikzcd}
$$

$$
\begin{tikzcd}
c_0^1 \arrow[r, "\delta_0^1"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1^1 \arrow[d, dash, "\sim" description] \\
c_0^2 \arrow[r, dashed, twoheadrightarrow] & c_1^2
\end{tikzcd}
$$

Composition: $P_0 \sim P_2$

$$
\begin{tikzcd}
c_0^0 \arrow[r, "\delta_0^0"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1^0 \arrow[r, "\delta_1^0"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_2^0 \arrow[d, dash, "\sim" description] \\
c_0^1 \arrow[r, dashed, twoheadrightarrow] & c_1^1 \arrow[r, dashed, twoheadrightarrow] & c_2^1
\end{tikzcd}
$$

$$
\begin{tikzcd}
c_0^1 \arrow[r, "\delta_0^1"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1^1 \arrow[d, dash, "\sim" description] \\
c_0^2 \arrow[r, dashed, twoheadrightarrow] & c_1^2
\end{tikzcd}
$$

Composition: $P_0 \sim P_2$

$$
\begin{tikzcd}
c_0^0 \arrow[r, "\delta_0^0"] \arrow[d, dash, "\wr" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1^0 \arrow[r, "\delta_1^0"] \arrow[d, dash, "\wr" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_2^0 \arrow[d, dash, "\wr" description] \\
c_0^1 \arrow[r, dashed, twoheadrightarrow] & c_1^1 \arrow[r, dashed, twoheadrightarrow] & c_2^1
\end{tikzcd}
$$

$$
\begin{tikzcd}
c_0^1 \arrow[r, "\delta_0^1"] \arrow[d, dash, "\wr" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1^1 \arrow[r, "\delta_1^1"] \arrow[d, dash, "\wr" description] \arrow[dr, phantom, "\downharpoonleft"] & c_2^1 \arrow[d, dash, "\wr" description] \\
c_0^2 \arrow[r, dashed, twoheadrightarrow] & c_1^2 \arrow[r, dashed, twoheadrightarrow] & c_2^2
\end{tikzcd}
$$

Composition: $P_0 \sim P_2$

$$
\begin{tikzcd}
c_0^0 \arrow[r, "\delta_0^0"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1^0 \arrow[r, "\delta_1^0"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_2^0 \arrow[r, "\delta_2^0"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_3^0 \arrow[d, dash, "\sim" description] \\
c_0^1 \arrow[r, dashed, two heads] & c_1^1 \arrow[r, dashed, two heads] & c_2^1 \arrow[r, dashed, two heads] & c_3^1
\end{tikzcd}
$$

$$
\begin{tikzcd}
c_0^1 \arrow[r, "\delta_0^1"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1^1 \arrow[r, "\delta_1^1"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft"] & c_2^1 \arrow[d, dash, "\sim" description] \\
c_0^2 \arrow[r, dashed, two heads] & c_1^2 \arrow[r, dashed, two heads] & c_2^2
\end{tikzcd}
$$

Composition: $P_0 \sim P_2$

$$
\begin{tikzcd}
c_0^0 \arrow[r, "\delta_0^0"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1^0 \arrow[r, "\delta_1^0"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_2^0 \arrow[r, "\delta_2^0"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_3^0 \arrow[d, dash, "\sim" description] \\
c_0^1 \arrow[r, dashed, twoheadrightarrow] & c_1^1 \arrow[r, dashed, twoheadrightarrow] & c_2^1 \arrow[r, dashed, twoheadrightarrow] & c_3^1
\end{tikzcd}
$$

$$
\begin{tikzcd}
c_0^1 \arrow[r, "\delta_0^1"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1^1 \arrow[r, "\delta_1^1"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft"] & c_2^1 \arrow[r, "\delta_2^1"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_3^1 \arrow[d, dash, "\sim" description] \\
c_0^2 \arrow[r, dashed, twoheadrightarrow] & c_1^2 \arrow[r, dashed, twoheadrightarrow] & c_2^2 \arrow[r, dashed, twoheadrightarrow, "=" '] & c_2^2
\end{tikzcd}
$$

Composition: $P_0 \sim P_2$

$$
\begin{tikzcd}
c_0^0 \arrow[r, "\delta_0^0"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1^0 \arrow[r, "\delta_1^0"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft"] & c_2^0 \arrow[r, "\delta_2^0"] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_3^0 \arrow[d, dash, "\sim" description] \\
c_0^2 \arrow[r, dashed, twoheadrightarrow] & c_1^2 \arrow[r, dashed, twoheadrightarrow] & c_2^2 \arrow[r, dashed, twoheadrightarrow, "=" '] & c_2^2
\end{tikzcd}
$$

Another composition example

$$
\begin{tikzcd}
c_0 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1 \arrow[d, dash, "\sim" description] \\
c_0' \arrow[r, dashed, two heads] & c_1'
\end{tikzcd}
$$

$$
\begin{tikzcd}
c_0' \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1' \arrow[d, dash, "\sim" description] \\
c_0'' \arrow[r, dashed, two heads] & c_1''
\end{tikzcd}
$$

Another composition example

$$
\begin{tikzcd}
c_0 \arrow[r] \arrow[d, no head, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1 \arrow[rr] \arrow[d, no head, "\sim" description] \arrow[drr, phantom, "\downharpoonleft\upharpoonright"] & & c_2 \arrow[d, no head, "\sim" description] \\
c_0' \arrow[r, dashed, two heads] & c_1' \arrow[rr, dashed, two heads] & & c_3'
\end{tikzcd}
$$

$$
\begin{tikzcd}
c_0' \arrow[r] \arrow[d, no head, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1' \arrow[d, no head, "\sim" description] \\
c_0'' \arrow[r, dashed, two heads] & c_1''
\end{tikzcd}
$$

Another composition example

$$
\begin{tikzcd}
c_0 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_2 \arrow[d, dash, "\sim" description] \\
c_0' \arrow[r, dashed, twoheadrightarrow] & c_1' \arrow[r, dashed, twoheadrightarrow] & c_3'
\end{tikzcd}
$$

$$
\begin{tikzcd}
c_0' \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1' \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft"] & c_2' \arrow[d, dash, "\sim" description] \\
c_0'' \arrow[r, dashed, twoheadrightarrow] & c_1'' \arrow[r, dashed, twoheadrightarrow] & c_2''
\end{tikzcd}
$$

Another composition example

$$
\begin{tikzcd}
c_0 \arrow[r] \arrow[d, no head, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\,\upharpoonright"] & c_1 \arrow[rr] \arrow[d, no head, "\sim" description] \arrow[drr, phantom, "\downharpoonleft\,\upharpoonright"] & & c_2 \arrow[d, no head, "\sim" description] \\
c'_0 \arrow[r, dashed, ->>] & c'_1 \arrow[rr, dashed, ->>] & & c'_3
\end{tikzcd}
$$

$$
\begin{tikzcd}
c'_0 \arrow[r] \arrow[d, no head, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\,\upharpoonright"] & c'_1 \arrow[r] \arrow[d, no head, "\sim" description] \arrow[dr, phantom, "\downharpoonleft"] & c'_2 \arrow[r] \arrow[d, no head, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\,\upharpoonright"] & c'_3 \arrow[d, no head, "\sim" description] \\
c''_0 \arrow[r, dashed, ->>] & c''_1 \arrow[r, dashed, ->>] & c''_2 \arrow[r, dashed, ->>] & c''_3
\end{tikzcd}
$$

Another composition example

$$\begin{tikzcd}
c_0 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_2 \arrow[d, dash, "\sim" description] \\
c_0'' \arrow[r, dashed, two heads] & c_1'' \arrow[r, dashed, two heads] & c_3''
\end{tikzcd}$$

Composition: incompatible directions

$$
\begin{tikzcd}
c_0 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft"] & c_1 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft"] & c_2 \arrow[d, dash, "\sim" description] \\
c'_0 \arrow[r, dashed, twoheadrightarrow] & c'_1 \arrow[r, dashed, twoheadrightarrow] & c'_2
\end{tikzcd}
$$

$$
\begin{tikzcd}
c'_0 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c'_1 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\upharpoonright"] & c'_2 \arrow[d, dash, "\sim" description] \\
c''_0 \arrow[r, dashed, twoheadrightarrow] & c''_1 \arrow[r, dashed, twoheadrightarrow] & c''_2
\end{tikzcd}
$$

Composition: incompatible directions

$$
\begin{tikzcd}
c_0 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft"] & c_1 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft"] & c_2 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_3 \arrow[d, dash, "\sim" description] \\
c_0' \arrow[r, dashed, two heads] & c_1' \arrow[r, dashed, two heads] & c_2' \arrow[r, dashed, two heads] & c_3'
\end{tikzcd}
$$

$$
\begin{tikzcd}
c_0' \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1' \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\upharpoonright"] & c_2' \arrow[d, dash, "\sim" description] \\
c_0'' \arrow[r, dashed, two heads] & c_1'' \arrow[r, dashed, two heads] & c_2''
\end{tikzcd}
$$

Composition: incompatible directions

$$
\begin{tikzcd}
c_0 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft"] & c_1 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft"] & c_2 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\,\upharpoonright"] & c_3 \arrow[d, dash, "\sim" description] \\
c_0' \arrow[r, dashed, two heads] & c_1' \arrow[r, dashed, two heads] & c_2' \arrow[r, dashed, two heads] & c_3'
\end{tikzcd}
$$

$$
\begin{tikzcd}
c_0' \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\,\upharpoonright"] & c_1' \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\upharpoonright"] & c_2' \arrow[r, dashed, two heads] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\,\upharpoonright"] & c_3' \arrow[d, dash, "\sim" description] \\
c_0'' \arrow[r, dashed, two heads] & c_1'' \arrow[r, dashed, two heads] & c_2'' \arrow[r] & c_3''
\end{tikzcd}
$$

Composition: incompatible directions

$$
\begin{tikzcd}
c_0 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\lrcorner"] & c_1 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "?"] & c_2 \arrow[r] \arrow[d, dash, "\sim" description] & c_3 \\
c_0'' \arrow[r, dashed, two heads] & c_1'' \arrow[r, dashed, two heads] & c_2'' \arrow[r] & c_3''
\end{tikzcd}
$$

Composition: incompatible directions (2)

$$\begin{tikzcd}
c_0 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1 \arrow[rr] \arrow[d, dash, "\sim" description] \arrow[drr, phantom, "\downharpoonleft\upharpoonright"] & & c_2 \arrow[d, dash, "\sim" description] \\
c'_0 \arrow[r, dashed, two heads] & c'_1 \arrow[rr, dashed, two heads] & & c'_3
\end{tikzcd}$$

$$\begin{tikzcd}
c'_0 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c'_1 \arrow[d, dash, "\sim" description] \\
c''_0 \arrow[r, dashed, two heads] & c''_1
\end{tikzcd}$$

Composition: incompatible directions (2)

$$
\begin{tikzcd}
c_0 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1 \arrow[rr] \arrow[d, dash, "\sim" description] \arrow[drr, phantom, "\downharpoonleft\upharpoonright"] & & c_2 \arrow[d, dash, "\sim" description] \\
c_0' \arrow[r, dashed, two heads] & c_1' \arrow[rr, dashed, two heads] & & c_3'
\end{tikzcd}
$$

$$
\begin{tikzcd}
c_0' \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1' \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\upharpoonright"] & c_2' \arrow[d, dash, "\sim" description] \\
c_0'' \arrow[r, dashed, two heads] & c_1'' \arrow[r, dashed, two heads] & c_2''
\end{tikzcd}
$$

Composition: incompatible directions (2)

$$
\begin{tikzcd}
c_0 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_2 \arrow[d, dash, "\sim" description] \\
c_0' \arrow[r, dashed, two heads] & c_1' \arrow[r, dashed, two heads] & c_3'
\end{tikzcd}
$$

$$
\begin{tikzcd}
c_0' \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1' \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\upharpoonright"] & c_2' \arrow[r, dashed, two heads] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_3' \arrow[d, dash, "\sim" description] \\
c_0'' \arrow[r, dashed, two heads] & c_1'' \arrow[r, dashed, two heads] & c_2'' \arrow[r] & c_3''
\end{tikzcd}
$$

Composition: incompatible directions (2)

$$\begin{tikzcd}
c_0 \arrow[r] \arrow[d, dash, "\sim"] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1 \arrow[r] \arrow[d, dash, "\sim"] \arrow[dr, phantom, "?"] & c_2 \\
c_0'' \arrow[r, dashed, twoheadrightarrow] & c_1'' \arrow[r] & c_2''
\end{tikzcd}$$

Composition: stuck intermediate program

$$
\begin{tikzcd}
c_0 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft"] & c_2 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft"] & c_3 \\
c_0' \arrow[r, dashed, two heads] & c_1' \arrow[r, dashed, two heads] & c_2' \arrow[r, dashed, two heads, "/" marking] & {}
\end{tikzcd}
$$

$$
\begin{tikzcd}
c_0' \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1' \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_2' \arrow[r, "/" marking] \arrow[d, dash, "\sim" description] & {} \\
c_0'' \arrow[r, dashed, two heads] & c_1'' \arrow[r, dashed, two heads] & c_2''
\end{tikzcd}
$$

Composition: stuck intermediate program

$$
\begin{tikzcd}
c_0 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft\upharpoonright"] & c_1 \arrow[r] \arrow[d, dash, "\sim" description] \arrow[dr, phantom, "\downharpoonleft"] & c_2 \arrow[r] \arrow[d, dash, "\sim" description] & c_3 \\
c_0'' \arrow[r, dashed, twoheadrightarrow] & c_1'' \arrow[r, dashed, twoheadrightarrow] & c_2''
\end{tikzcd}
$$

Issues with composition

Composition is ill-defined if. . .

- The two underlying oracles require incompatible directions
- One oracle predicts a crash of the intermediate program

Sufficient criteria for well-defined composition

- Neither oracle predicts a crash of the intermediate program

and

- Either one of the oracles is always lock-step and bidirectional
- Or neither oracle forces a change of direction

Coq Library

In numbers

- Oracle languages meta-definitions and properties, including identity and composition (about 1400 lines of spec and 3200 lines of proofs)
- Imp syntax and semantics (about 500 lines of spec)
- Oracle languages on Imp (about 3900 lines of spec and 14500 lines of proofs)

Coq Library

Language-agnostic aspects

Generic definitions

- Oracle languages meta-definitions and properties
- Identity and universal oracles
- Oracle composition

Proof-Of-Concept oracle inference tool

```
difftool --search-depth 3 P0.imp P2.imp
```

```
Renaming [y → count]
SeqAssocOracle.Mod  Fold
AssignSwapOracle.Mod 0
SeqAssocOracle.Mod  Unfold
```

- Written in OCaml, with lots of heuristics
- Not complete, neither proved correct in Coq.
- **But**, it generates a difference.
- An **extracted checker** verifies its validity.

Proof-Of-Concept oracle inference tool

Implemented oracles

- Renaming
- Assignment commutation
- Condition negation / Branch commutation
- Sequence associativity
- Control-flow preserving value changes

Conclusion

- Differences between close programs can be formally defined
- Changes can usually be composed, but there are some limits

Future work

- Explore weaker conditions implying composability
- Need for a formal criteria characterizing “useful” oracles
- Instantiate the framework on more programming languages

Properties of oracle languages

	Renaming	SeqAssoc	SwapAssign	SwapBranch	AbstractEquiv	AbsEqNoBound	CrashFix	ValueChange	AbstractInequiv
Decidable Checking	✓	✓	✓	✓	✓[1]	✓[1]	✓[1]	✓	✓[1]
Applicative	✓	✓	✓	✓	✓	✓	✓	✓	✓
Cooperative	✓	✓	✓	✓	✓	×	✓	✓	✓
One-step & Bidirectional	✓	×	×	✓	×	×	×	✓	×

[1] Only decidable given underlying proofs