A Mechanically Checked Generation of Correlating Programs directed by Structured Syntactic Differences

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, PPS, UMR 7126 CNRS, PiR2, INRIA Paris-Rocquencourt, F-75205 Paris, France

October 12, 2015

How to compute semantic differences between two versions of a program?

- One way is by performing static analysis on *Correlating Programs*
- *Abstract Semantic Differencing for Numerical Programs*, Nimrod Partush and Eran Yahav

What is a Correlating Program?

A **Correlating Program** is a program interleaving the instructions from two closely-related programs to analyze their behavior.

Two versions of the same program

Original version

```
void foo(int x) {
  int result;
  int i = 0;
  while (i <= 41) {
    i = i + 1;
    x = x + 1;
  };
  result = x;
}
```

Modified version

```
void foo(int x) {
  int result;
  int i = 0;
  while (i < 42) { //changed
    i = i + 1;
    x = x + 1;
  };
  result = x;
}
```

Overview of Partush and Yahav's approach

Step 1: Translation to guarded form

Guarded translation (computed by ccc) of the original program

Original program

```
void foo(int x) {
  int result;
  int i = 0;
  while (i <= 41) {

    i = i + 1;
    x = x + 1;

  };
  result = x;
}
```

Original program (guarded)

```
void foo(int x) {
  int result;
  int i = 0;
L1:;
    Guard G0 = (i <= 41);
    if (G0) i = i + 1;
    if (G0) x = x + 1;
    if (G0) goto L1;
L3:;
  result = x;
}
```

Overview of Partush and Yahav's approach

Step 2: Tagging

Guarded translations (computed by ccc) of both programs

Original version (guarded)

```
void foo(int x) {
  int result;
  int i = 0;
L1:;
    Guard G0 = (i <= 41);
    if (G0) i = i + 1;
    if (G0) x = x + 1;
    if (G0) goto L1;
L3:;
  result = x;
}
```

Modified version (guarded)

```
void foo(int T_x) {
  int T_result;
  int T_i = 0;
T_L1:;
    Guard T_G0 = (T_i < 42);
    if (T_G0) T_i = T_i + 1;
    if (T_G0) T_x = T_x + 1;
    if (T_G0) goto T_L1;
T_L3:;
  T_result = T_x;
}
```

Overview of Partush and Yahav's approach

Step 3: Instruction interleaving

Correlating program (computed by ccc)

```
void foo(int x) {
  int result;
  int i = 0;
L1:;
    Guard G0 = (i <= 41);
    if (G0) i = i + 1;
    if (G0) x = x + 1;
    if (G0) goto L1;
L3:;
  result = x;
}
```

```
  int T_x = x;
  int T_result;
  int T_i = 0;
T_L1:;
    Guard T_G0 = (T_i < 42);
    if (T_G0) T_i = T_i + 1;
    if (T_G0) T_x = T_x + 1;
    if (T_G0) goto T_L1;
T_L3:;
  T_result = T_x;
```

Such program transformations are hard to get right

Example of an incorrect correlating program produced by ccc

Two versions of the same program

Original version

```
void fail(int x) {
  int result;
  int i = 0;
  while (i <= 41) {


    i = i + 1;
    x = x + 1;



  };
  result = x;
}
```

Modified version

```
void fail(int x) {
  int result;
  int i = 0;
  while (i < 42) { //changed


    i = i + 1;
    x = x + 1;
    break; //added statement


  };
  result = x;
}
```

Such program transformations are hard to get right

Example of an incorrect correlating program produced by ccc

Two versions of the same program

Original version (guarded)

```
void fail(int x) {
  int result;
  int i = 0;
L1:;
    Guard G0 = (i <= 41);
    if (G0) i = i + 1;
    if (G0) x = x + 1;

    if (G0) goto L1;
L3:;
  result = x;
}
```

Modified version (guarded)

```
void fail(int x) {
  int result;
  int i = 0;
L1:;
    Guard G0 = (i < 42);
    if (G0) i = i + 1;
    if (G0) x = x + 1;
    if (G0) goto L3; //break
    if (G0) goto L1;
L3:;
  result = x;
}
```

Such program transformations are hard to get right

Example of an incorrect correlating program produced by ccc

Incorrect correlating program (computed by ccc)

```
void fail(int x) {
  int result;
  int i = 0;
L1:;
    Guard G0 = (i <= 41);
    if (G0) i = i + 1;
    if (G0) x = x + 1;

    if (G0) goto L1;
L3:;
  result = x;
}
```

```
  int T_x = x;
  int T_result;
  int T_i = 0;
T_L1:;
    Guard T_G0 = (T_i < 42);
    if (T_G0) T_i = T_i + 1;
    if (T_G0) T_x = T_x + 1;
    if (T_G0) goto T_L3; //!
    if (T_G0) goto T_L1;
T_L3:;
  T_result = T_x;
```

Such program transformations are hard to get right

- Lots of tricky corner cases, difficult to get right...
- ... Partush and Yahav's ccc is **unsound** for goto (and loops)

Towards structured syntactic differences

Root of issues

- Issues stem from a lack of **structure**
- and a lack of **formalization**

Towards structured syntactic differences

Root of issues

- Issues stem from a lack of **structure**
- and a lack of **formalization**

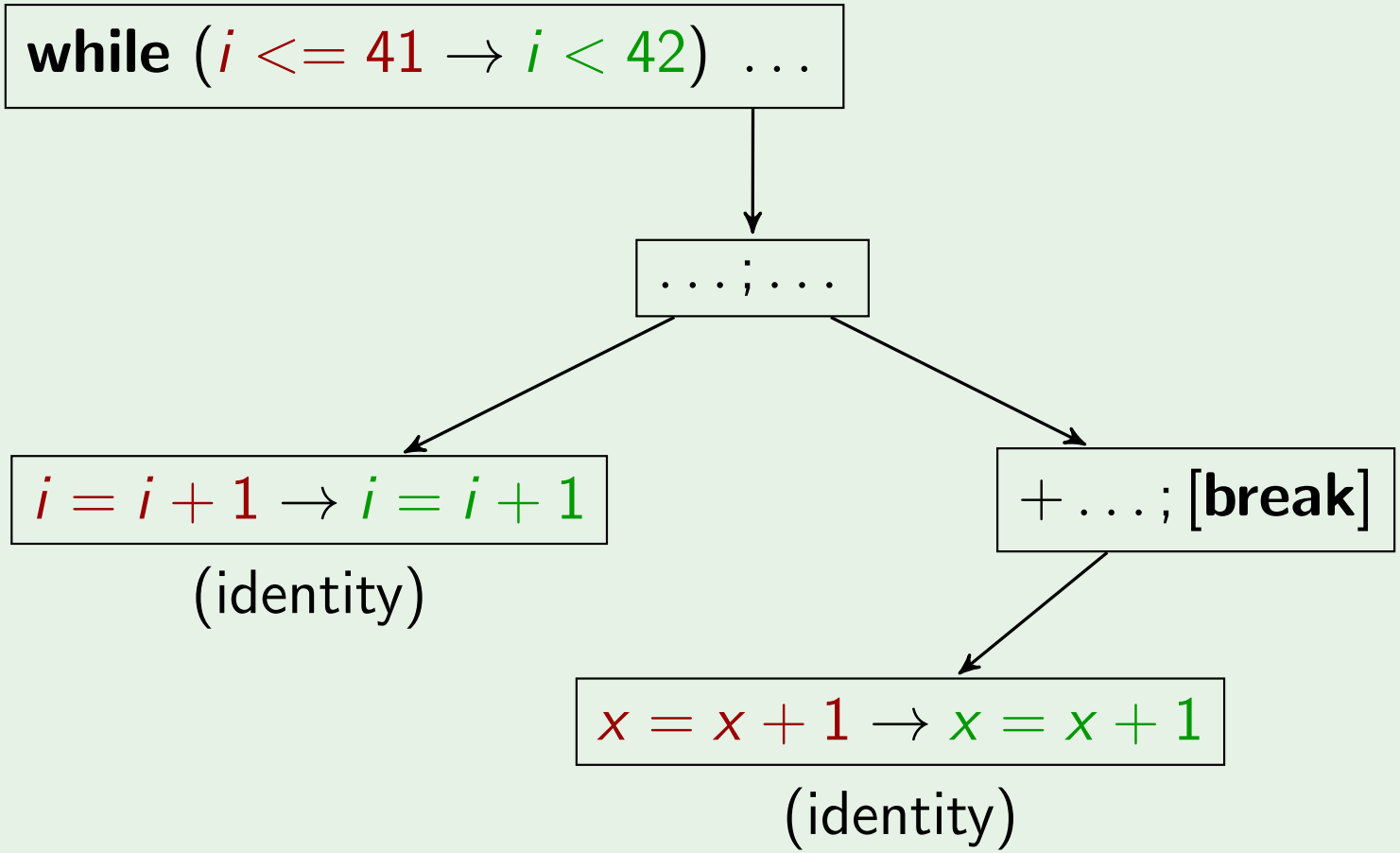
Our approach

This work presents a (new) mechanically-checked generation algorithm for Correlating Programs which makes use of:

- Structured syntactic (as opposed to textual) differences
- Interleaving following the structure of the difference

Towards structured syntactic differences

Excerpt of a syntactic difference on the two Abstract Syntax Trees

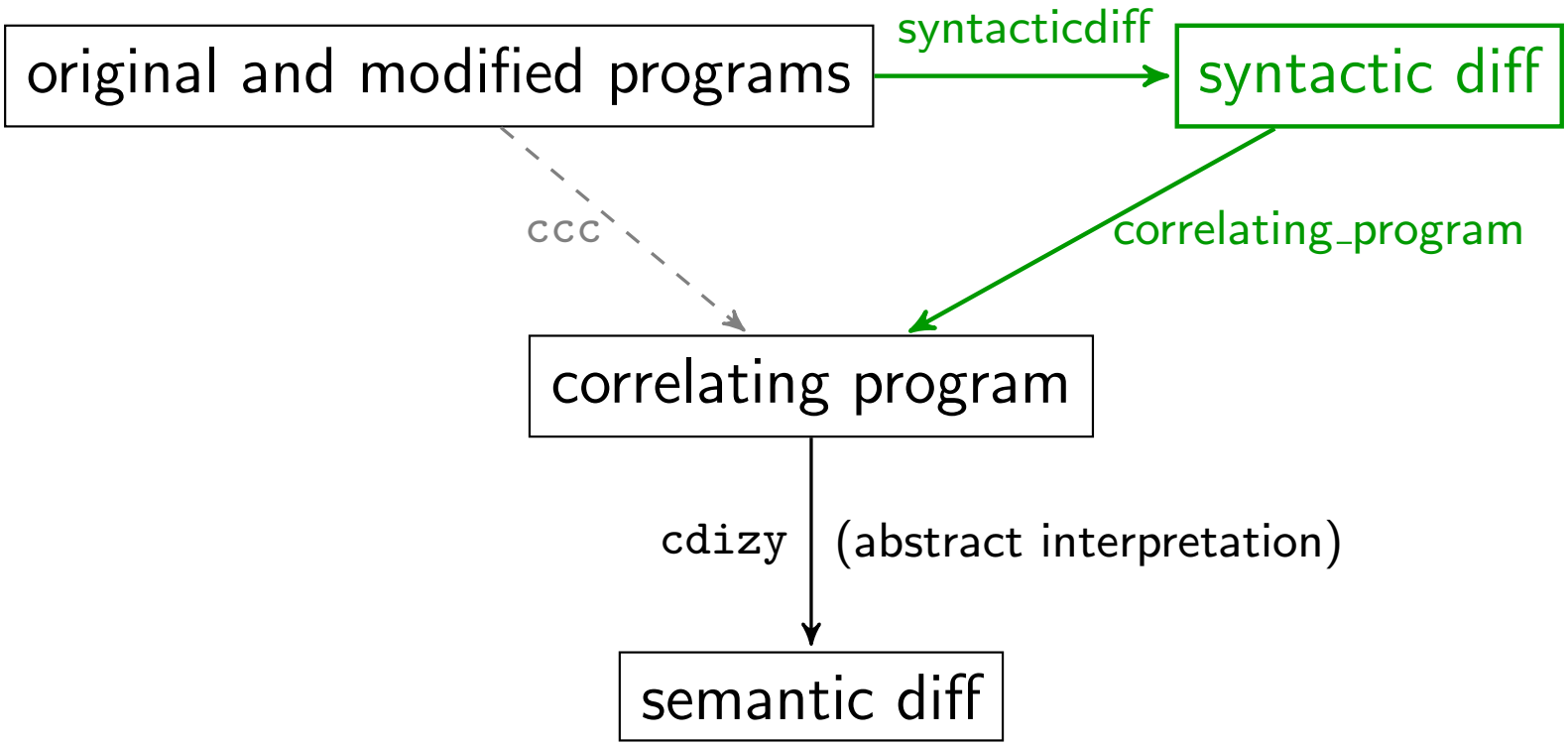


Towards structured syntactic differences

Textual representation of a syntactic difference on the two ASTs

```
 void fail(int x) {
   int i, result;
   i = 0;
   while (⟨i <= 41⟩ → ⟨i < 42⟩) {
     i = (i + 1);
     x = (x + 1);
+    break;
   };
   result = x;
 }
```

Replacing ccc



Replacing ccc

A new guarded form preserving the structure of loops

Comparison of the guarded translations

Guarded version (ccc)

```
void fail(int x) {
  int result;
  int i = 0;
L1:;
    Guard G1 = (i <= 41);
    if (G1) i = i + 1;
    if (G1) x = x + 1;

    if (G1) goto L1;
L3:;
  result = x;
}
```

Guarded version (used in proof)

```
void fail(int x) {
  int result;
  int i = 0;
  Guard G1 = (i <= 41);
  while (G1) {
    if (G1) i = i + 1;
    if (G1) x = x + 1;

    if (G1) G1 = (i <= 41);
  };
  result = x;
}
```

Replacing ccc

A new guarded form preserving the structure of loops

Guarded translations of both programs

Original version (guarded)

```
void fail(int x) {
  int result;
  int i = 0;
  Guard G1 = (i <= 41);
  while (G1) {
    if (G1) i = i + 1;
    if (G1) x = x + 1;

    if (G1) G1 = (i <= 41);
  };
  result = x;
}
```

Modified version (guarded)

```
void fail(int x) {
  int result;
  int i = 0;
  Guard G1 = (i < 42);
  while (G1) {
    if (G1) i = i + 1;
    if (G1) x = x + 1;
    if (G1) G1 = 0;
    if (G1) G1 = (i < 42);
  };
  result = x;
}
```

Replacing ccc

A new Correlating Program

Correlating program (computed by correlating_program)

```
void fail(int O_x) {
  int O_i = 0;
  int O_result;
  Guard G1 = (O_i <= 41);
  while (    G1          ||
    if (G1) O_i = O_i + 1;
    if (G1) O_x = O_x + 1;

    if (G1) G1 = (O_i <= 41);
  };
  O_result = O_x;
}
```

```
int T_O_x = O_x;
int T_O_i = 0;
int T_O_result;
Guard T_G1 = (T_O_i < 42);
T_G1    ) {
if (T_G1) T_O_i = T_O_i + 1;
if (T_G1) T_O_x = T_O_x + 1;
if (T_G1) T_G1 = 0;
if (T_G1) T_G1 = (T_O_i < 42);

T_O_result = T_O_x;
```

Replacing ccc

A new Correlating Program

Warning: about guarded forms

- Whole-program individual guarded forms not actually computed
- Correlating Program directly computed from syntactic difference
- Still an interleaving of the guarded forms (used in proof)

Replacing ccc

A new Correlating Program: an animated example

Correlating program (computed by correlating_program)

```
 void fail(int x) {
  int i = 0;
  int result;

  while (⟨i <= 41⟩ → ⟨i < 42⟩) {
    i = (i + 1);
    x = (x + 1);
+   break;

  };
  result = x;
 }
```

Replacing ccc

A new Correlating Program: an animated example

Correlating program (computed by correlating_program)

```
 void fail(int x) {
  ⟨int i = 0⟩              →      ⟨int i = 0⟩;
  ⟨int result⟩             →      ⟨int result⟩;

  while (⟨i <= 41⟩          →      ⟨i < 42⟩) {
    ⟨i = (i + 1)⟩          →      ⟨i = (i + 1)⟩;
    ⟨x = (x + 1)⟩          →      ⟨x = (x + 1)⟩;
+   break;

  };
  ⟨result = x⟩             →      ⟨result = x⟩;
 }
```

Replacing ccc

A new Correlating Program: an animated example

Correlating program (computed by correlating_program)

```
 void fail(int O_x) {
  ⟨int O_i = 0⟩          →      ⟨int T_O_i = 0⟩;
  ⟨int O_result⟩         →      ⟨int T_O_result⟩;

  while (⟨O_i <= 41⟩     →      ⟨T_O_i < 42⟩) {
    ⟨O_i = (O_i + 1)⟩    →      ⟨T_O_i = (T_O_i + 1)⟩;
    ⟨O_x = (O_x + 1)⟩    →      ⟨T_O_x = (T_O_x + 1)⟩;
+   break;

  };
  ⟨O_result = O_x⟩       →      ⟨T_O_result = T_O_x⟩;
 }
```

Replacing ccc

A new Correlating Program: an animated example

Correlating program (computed by correlating_program)

```
 void fail(int O_x) {                    int T_O_x = O_x;
  ⟨int O_i = 0⟩              →          ⟨int T_O_i = 0⟩;
  ⟨int O_result⟩             →          ⟨int T_O_result⟩;

  while (⟨O_i <= 41⟩         →          ⟨T_O_i < 42⟩) {
    ⟨O_i = (O_i + 1)⟩        →          ⟨T_O_i = (T_O_i + 1)⟩;
    ⟨O_x = (O_x + 1)⟩        →          ⟨T_O_x = (T_O_x + 1)⟩;
+   break;

  };
  ⟨O_result = O_x⟩           →          ⟨T_O_result = T_O_x⟩;
 }
```

Replacing ccc

A new Correlating Program: an animated example

Correlating program (computed by correlating_program)

```
 void fail(int O_x) {                    int T_O_x = O_x;
  int O_i = 0;                           int T_O_i = 0;
  int O_result;                          int T_O_result;

  while (⟨O_i <= 41⟩      →       ⟨T_O_i < 42⟩) {
    ⟨O_i = (O_i + 1)⟩     →       ⟨T_O_i = (T_O_i + 1)⟩;
    ⟨O_x = (O_x + 1)⟩     →       ⟨T_O_x = (T_O_x + 1)⟩;
+   break;

  };
  ⟨O_result = O_x⟩        →       ⟨T_O_result = T_O_x⟩;
 }
```

Replacing ccc

A new Correlating Program: an animated example

Correlating program (computed by correlating_program)

```
 void fail(int O_x) {                    int T_O_x = O_x;
  int O_i = 0;                           int T_O_i = 0;
  int O_result;                          int T_O_result;
  Guard G1 = (O_i <= 41);                Guard T_G1 = (T_O_i < 42);
  while (    G1          ||              T_G1     ) {
    ⟨O_i = (O_i + 1)⟩    →              ⟨T_O_i = (T_O_i + 1)⟩;
    ⟨O_x = (O_x + 1)⟩    →              ⟨T_O_x = (T_O_x + 1)⟩;
+   break;
    if (G1) G1 = (O_i <= 41);            if (T_G1) T_G1 = (T_O_i < 42);
  };
  ⟨O_result = O_x⟩       →              ⟨T_O_result = T_O_x⟩;
 }
```

Replacing ccc

A new Correlating Program: an animated example

Correlating program (computed by correlating_program)

```
 void fail(int O_x) {                    int T_O_x = O_x;
  int O_i = 0;                           int T_O_i = 0;
  int O_result;                          int T_O_result;
  Guard G1 = (O_i <= 41);                Guard T_G1 = (T_O_i < 42);
  while (    G1          ||              T_G1    ) {
    if (G1) O_i = O_i + 1;               if (T_G1) T_O_i = T_O_i + 1;
    if (G1) O_x = O_x + 1;               if (T_G1) T_O_x = T_O_x + 1;
+   break;
    if (G1) G1 = (O_i <= 41);            if (T_G1) T_G1 = (T_O_i < 42);
  };
  ⟨O_result = O_x⟩          →          ⟨T_O_result = T_O_x⟩;
 }
```

Replacing ccc

A new Correlating Program: an animated example

Correlating program (computed by correlating_program)

```
void fail(int O_x) {                    int T_O_x = O_x;
 int O_i = 0;                           int T_O_i = 0;
 int O_result;                          int T_O_result;
 Guard G1 = (O_i <= 41);                Guard T_G1 = (T_O_i < 42);
 while (    G1          ||              T_G1     ) {
   if (G1) O_i = O_i + 1;               if (T_G1) T_O_i = T_O_i + 1;
   if (G1) O_x = O_x + 1;               if (T_G1) T_O_x = T_O_x + 1;
                                        if (T_G1) T_G1 = 0;
   if (G1) G1 = (O_i <= 41);            if (T_G1) T_G1 = (T_O_i < 42);
 };
 ⟨O_result = O_x⟩          →          ⟨T_O_result = T_O_x⟩;
}
```

Replacing ccc

A new Correlating Program: an animated example

Correlating program (computed by correlating_program)

```
void fail(int O_x) {
  int O_i = 0;
  int O_result;
  Guard G1 = (O_i <= 41);
  while (    G1          ||
    if (G1) O_i = O_i + 1;
    if (G1) O_x = O_x + 1;

    if (G1) G1 = (O_i <= 41);
  };
  O_result = O_x;
}
```

```
int T_O_x = O_x;
int T_O_i = 0;
int T_O_result;
Guard T_G1 = (T_O_i < 42);
T_G1    ) {
if (T_G1) T_O_i = T_O_i + 1;
if (T_G1) T_O_x = T_O_x + 1;
if (T_G1) T_G1 = 0;
if (T_G1) T_G1 = (T_O_i < 42);

T_O_result = T_O_x;
```

Soundness theorem

The correlating program simulates the execution of the two input programs

Theorem: The correlating program is sound

$$S_1 \vdash P_1 \Downarrow S_1' \ \wedge \ S_2 \vdash P_2 \Downarrow S_2' \implies$$
$$T(S_1) \uplus T'(S_2) \vdash \texttt{correlating\_program}\ P_1\ P_2 \Downarrow T(S_1') \uplus T'(S_2')$$

Soundness theorem

The correlating program simulates the execution of the two input programs

Theorem: The correlating program is sound

$$S_1 \vdash P_1 \Downarrow S_1' \ \wedge \ S_2 \vdash P_2 \Downarrow S_2' \implies$$
$$T(S_1) \uplus T'(S_2) \vdash \texttt{correlating\_program}\ P_1\ P_2 \Downarrow T(S_1') \uplus T'(S_2')$$

For any two programs P_1 and P_2 for which the execution in initial memory S_1 and S_2 results in memories S_1' and S_2', the execution of the correlating program computed from P_1 and P_2 in the tagged union of S_1 and S_2 results in the tagged union of S_1' and S_2'.

Soundness theorem

The correlating program simulates the execution of the two input programs

Theorem: The correlating program is sound

$$S_1 \vdash P_1 \Downarrow S_1' \ \wedge \ S_2 \vdash P_2 \Downarrow S_2' \implies$$
$$T(S_1) \uplus T'(S_2) \vdash \texttt{correlating\_program}\ P_1\ P_2 \Downarrow T(S_1') \uplus T'(S_2')$$

For any two programs P_1 and P_2 for which the execution in initial memory S_1 and S_2 results in memories S_1' and S_2', the execution of the correlating program computed from P_1 and P_2 in the tagged union of S_1 and S_2 results in the tagged union of S_1' and S_2'.

Soundness theorem

The correlating program simulates the execution of the two input programs

Theorem: The correlating program is sound

$$S_1 \vdash P_1 \Downarrow S_1' \ \wedge \ S_2 \vdash P_2 \Downarrow S_2' \implies$$
$$T(S_1) \uplus T'(S_2) \vdash \texttt{correlating\_program}\ P_1\ P_2 \Downarrow T(S_1') \uplus T'(S_2')$$

For any two programs P_1 and P_2 for which the execution in initial memory S_1 and S_2 results in memories S_1' and S_2', the execution of the correlating program computed from P_1 and P_2 in the tagged union of S_1 and S_2 results in the tagged union of S_1' and S_2'.

Soundness theorem

The correlating program simulates the execution of the two input programs

Theorem: The correlating program is sound

$$S_1 \vdash P_1 \Downarrow S_1' \ \wedge \ S_2 \vdash P_2 \Downarrow S_2' \implies$$
$$T(S_1) \uplus T'(S_2) \vdash \texttt{correlating\_program}\ P_1\ P_2 \Downarrow T(S_1') \uplus T'(S_2')$$

For any two programs P_1 and P_2 for which the execution in initial memory S_1 and S_2 results in memories S_1' and S_2', the execution of the correlating program computed from P_1 and P_2 in the tagged union of S_1 and S_2 results in the tagged union of S_1' and S_2'.

Soundness theorem

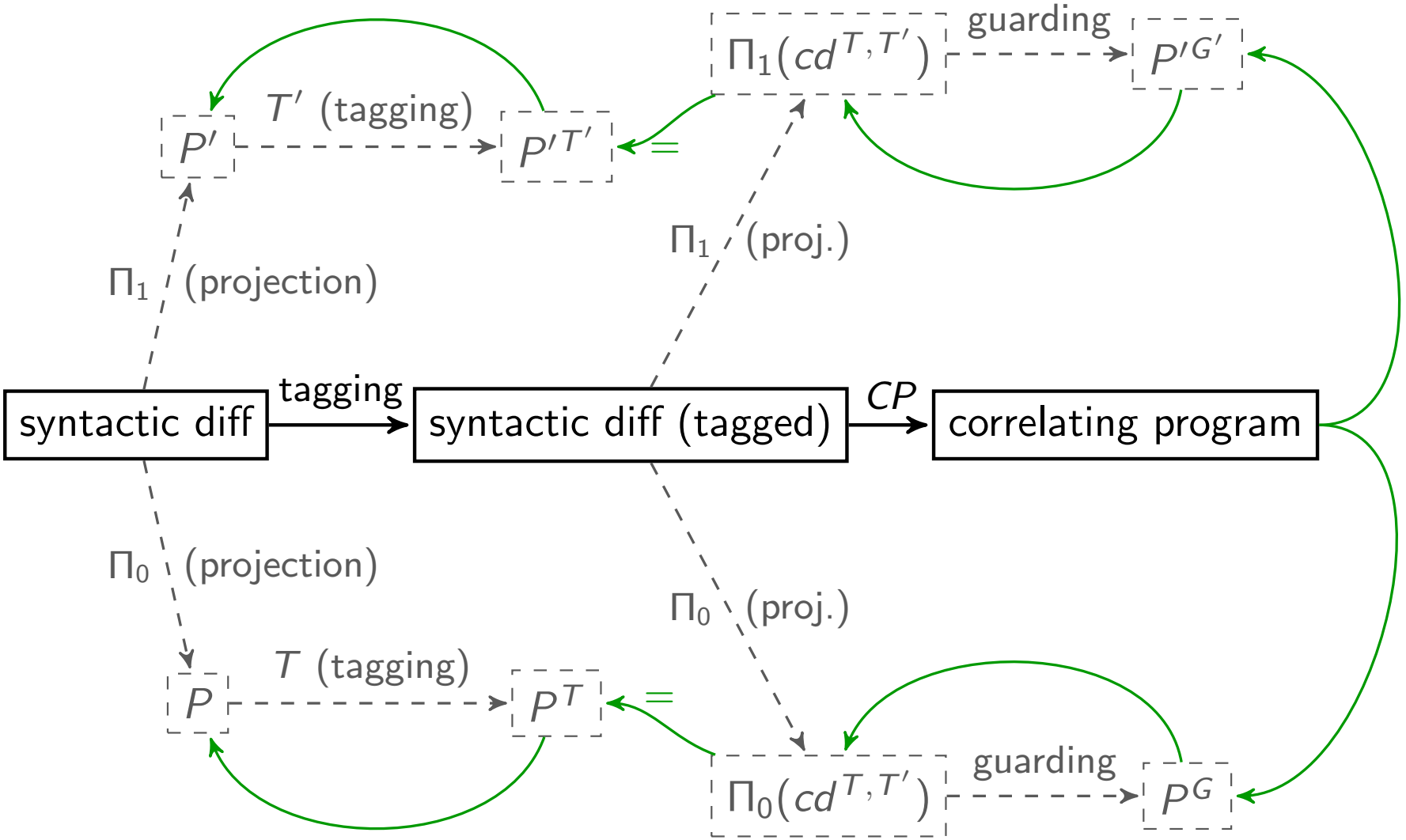
The correlating program simulates the execution of the two input programs

Theorem: The correlating program is sound

$$S_1 \vdash P_1 \Downarrow S_1' \ \wedge \ S_2 \vdash P_2 \Downarrow S_2' \implies$$
$$T(S_1) \uplus T'(S_2) \vdash \texttt{correlating\_program}\ P_1\ P_2 \Downarrow T(S_1') \uplus T'(S_2')$$

For any two programs P_1 and P_2 for which the execution in initial memory S_1 and S_2 results in memories S_1' and S_2', the execution of the correlating program computed from P_1 and P_2 in the tagged union of S_1 and S_2 results in the tagged union of S_1' and S_2'.

Soundness theorem in Coq (proof hints)



Curved green lines represent simulation proofs

Implementation

- `syntacticdiff`: 354 lines of OCaml, dynamically checked
- `correlating_program`:
  - $\sim$ 3800 lines of Coq ($\sim$ 10% definitions, 90% algorithm and proofs)
  - $\sim$ 700 lines of OCaml ($\sim$ 50% for parsing)

Implementation

Implementation and experiments available at:

http://www.pps.univ-paris-diderot.fr/~thib/atva15/

Conclusion

Our contributions:

- Use of syntactic differences to preserve structure
- Formalized languages and transformations
- Correlating Program generation implemented and proved in Coq

Conclusion

Our contributions:

- Use of syntactic differences to preserve structure
- Formalized languages and transformations
- Correlating Program generation implemented and proved in Coq

Thanks for listening!