Improving Performance of a Path-Based Equivalence Checker using Counter-Examples

Ramanuj Chouksey, Chandan Karfa, and Purandar Bhaduri

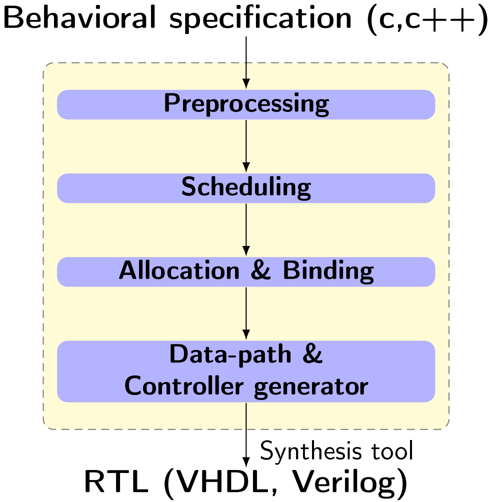
Indian Institute of Technology Guwahati

March 31, 2019

Outline

1. High-level Synthesis (HLS)
2. Path-based Equivalence Checker (PBEC)
3. Overall Approach
4. Experimental Results
5. Conclusion & Future Work

High-level Synthesis (HLS)



Behavioral specification (c,c++)

Preprocessing

Scheduling

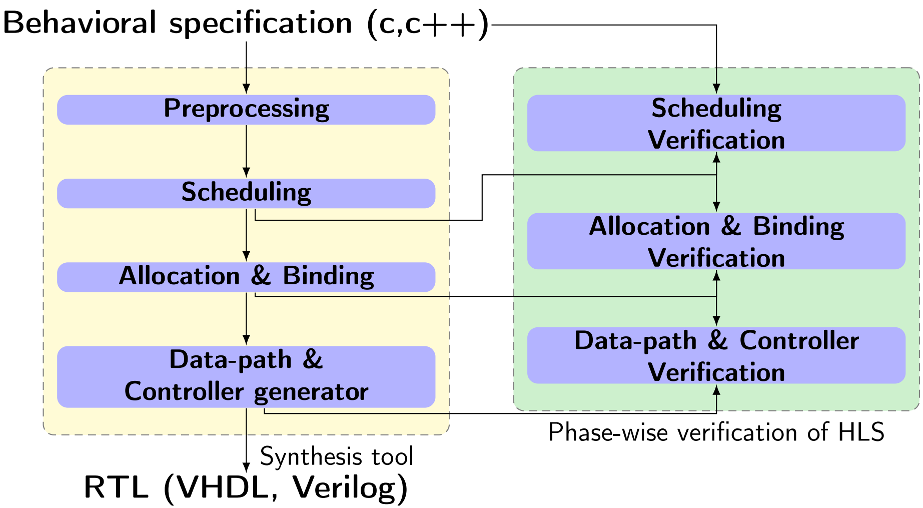
Allocation & Binding

Data-path & Controller generator

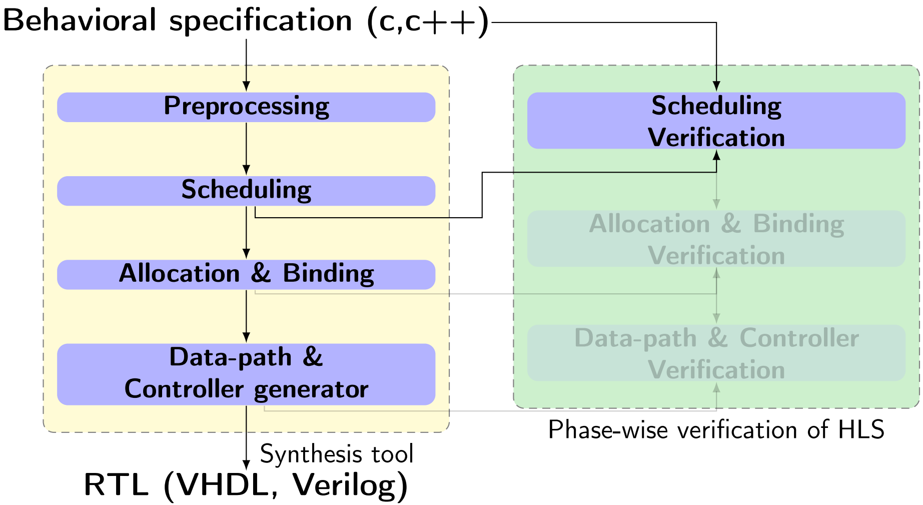
Synthesis tool

RTL (VHDL, Verilog)

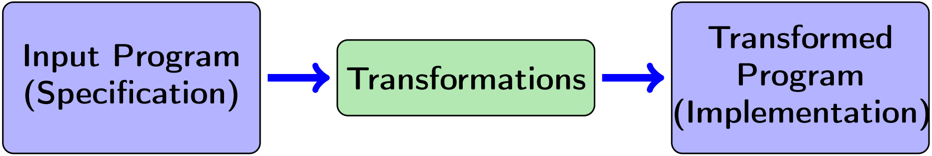
High-level Synthesis (HLS)



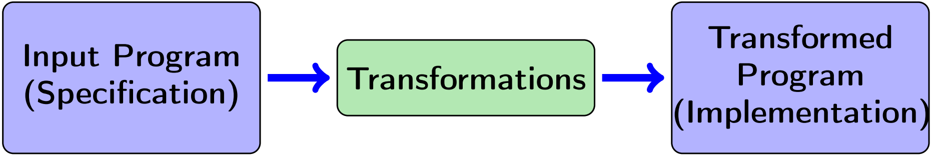
High-level Synthesis (HLS)



Verification of HLS

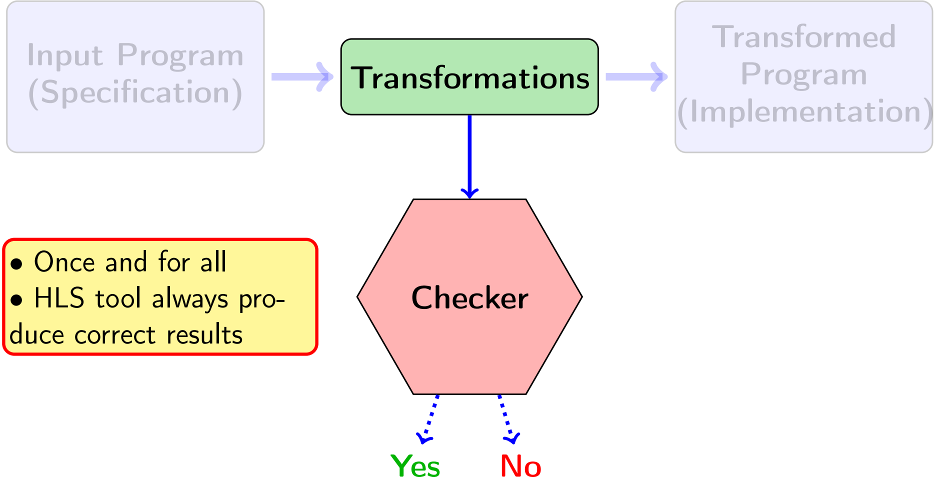


Verification of HLS



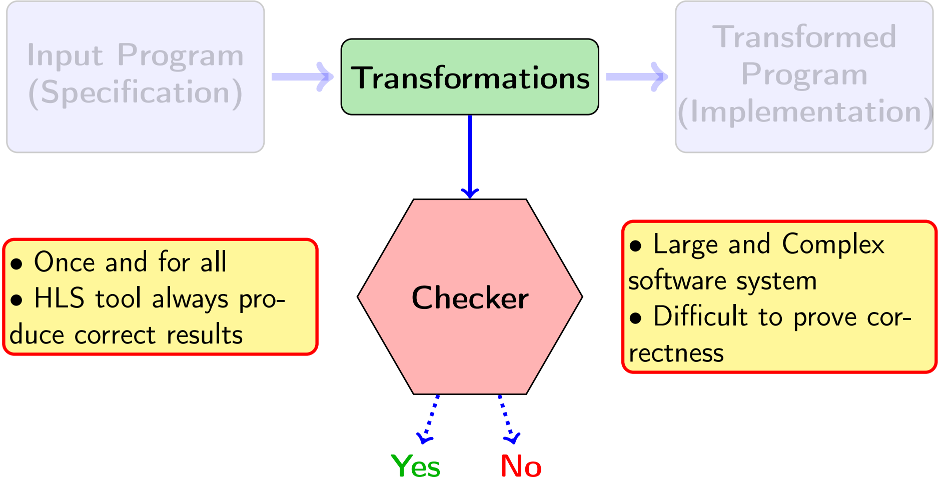
Is the Specification “functionally equivalent” to Implementation?

Verification of HLS:Correct by Construction



- Once and for all
- HLS tool always produce correct results

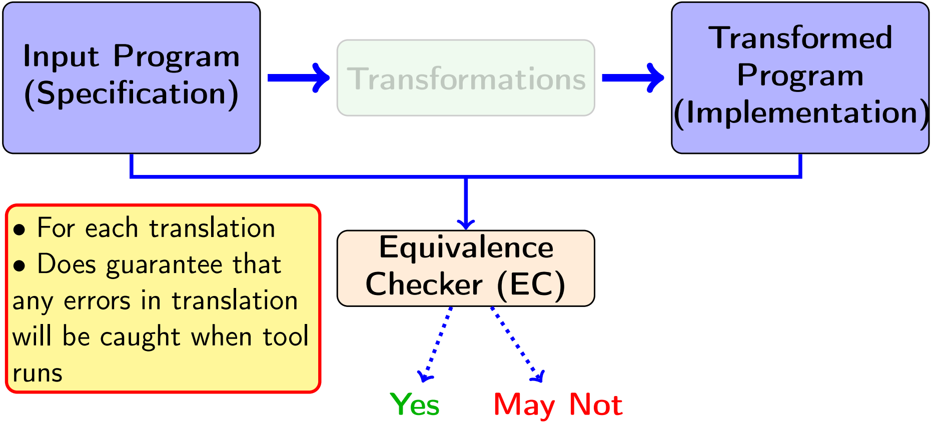
Verification of HLS:Correct by Construction



- Once and for all
- HLS tool always produce correct results

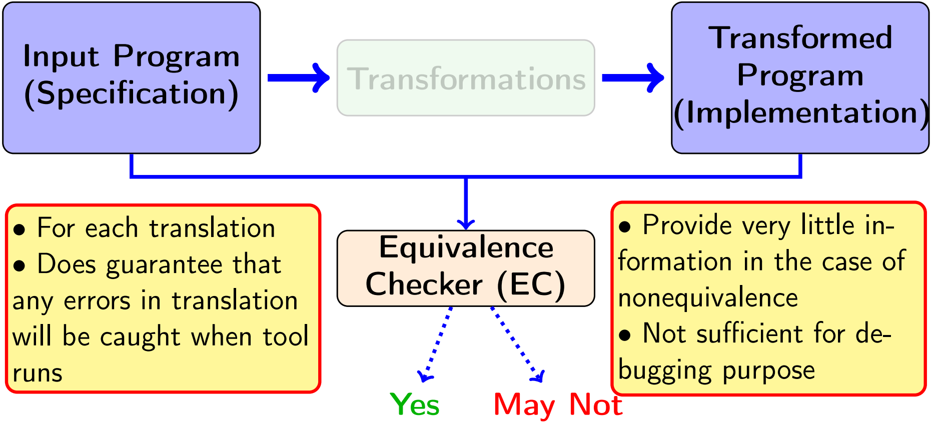
- Large and Complex software system
- Difficult to prove correctness

Verification of HLS:Translation Validation


Input Program (Specification)
Transformations
Transformed Program (Implementation)
Equivalence Checker (EC)
Yes
May Not

- For each translation
- Does guarantee that any errors in translation will be caught when tool runs

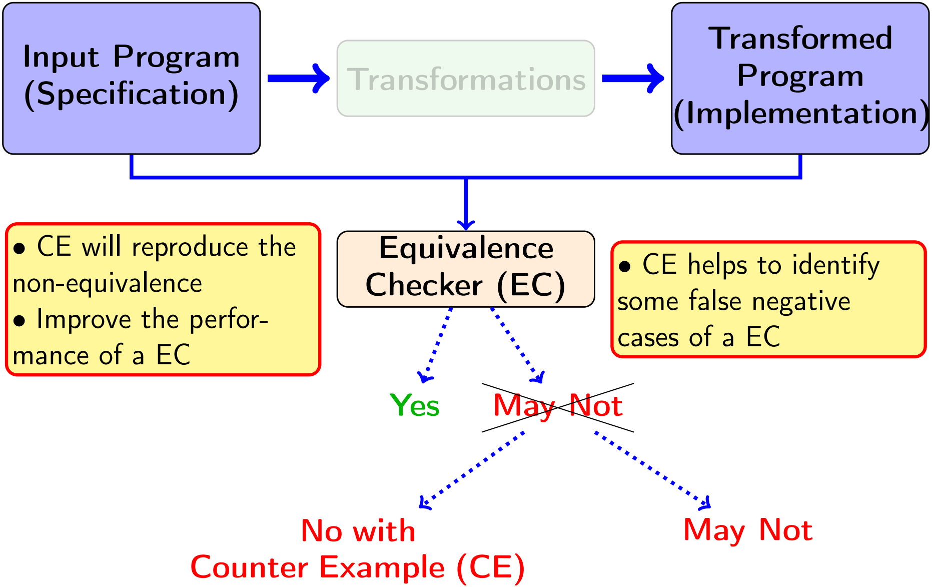
Verification of HLS:Translation Validation


Input Program (Specification)
Transformations
Transformed Program (Implementation)
Equivalence Checker (EC)
Yes
May Not

- For each translation
- Does guarantee that any errors in translation will be caught when tool runs

- Provide very little information in the case of nonequivalence
- Not sufficient for debugging purpose

Motivation



Input Program (Specification)

Transformations

Transformed Program (Implementation)

Equivalence Checker (EC)

Yes

~~May Not~~

No with Counter Example (CE)

May Not

- CE will reproduce the non-equivalence
- Improve the performance of a EC

- CE helps to identify some false negative cases of a EC

Path-based Equivalence Checkers (PBEC)–Related Works

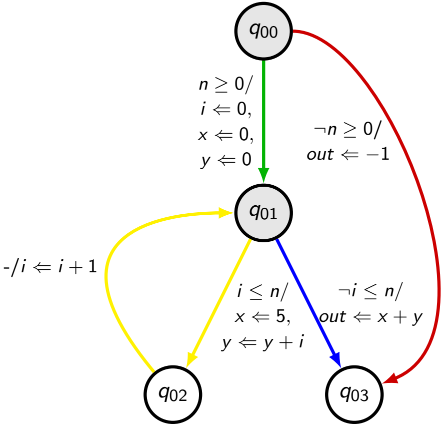
- C. Karfa et al, "*An equivalence-checking method for scheduling verification in high-level synthesis*,"IEEE TCAD, (2008).
- Lee et al, "*Equivalence checking of scheduling with speculative code transformations in high-level synthesis*", ASP-DAC (2011).
- K. Banerjee et al, "*Verification of code motion techniques using value propagation*", IEEE TCAD, (2014). [VP Method]
- J. Hu et al "Equivalence checking between SLM and RTL using machine learning techniques," ISQED (2016).
- R. Chouksey et al, "*Translation validation of code motion transformations involving loops*", IEEE TCAD, (2018). [EVP Method]

PBEC

- Behaviors are represented by Finite State Machine with Data (FSMD).
- Decompose each FSMD using cutpoints.
- Path: Finite sequence of states from a cutpoint to another cutpoint.
- Equivalence of FSMDs is established by showing path level equivalence between two FSMDs.

Representing a program using FSMD

```
  if(n≥0){
   x=0,y=0;
  for(i=0;i≤n;i++){
   x=5;
   y=y+i;}
 out=x+y;}
  else
out=−1;
```



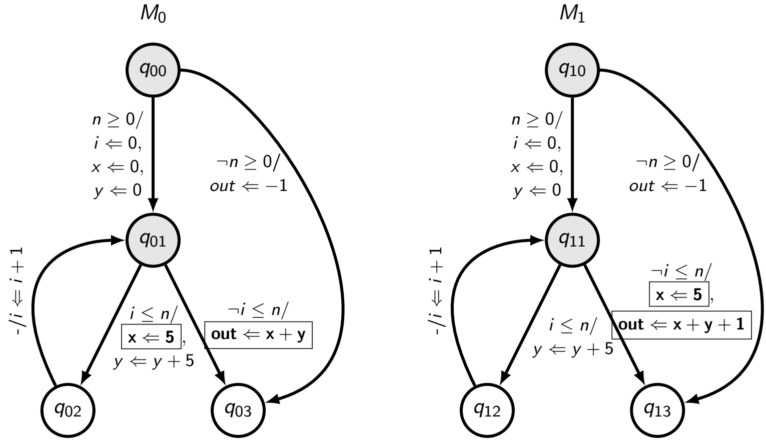
q_{00}
$n \geq 0/$ $i \Leftarrow 0,$ $x \Leftarrow 0,$ $y \Leftarrow 0$
$\neg n \geq 0/$ $out \Leftarrow -1$
q_{01}
$\text{-}/i \Leftarrow i + 1$
$i \leq n/$ $x \Leftarrow 5,$ $y \Leftarrow y + i$
$\neg i \leq n/$ $out \Leftarrow x + y$
q_{02}
q_{03}

Overall Approach

In the case of nonequivalence reported by PBEC

1. Counter Trace (*cTrace*) generation.
2. Modeling the *cTrace* to generate counter example using CBMC.
3. Incorporate the results with PBEC.

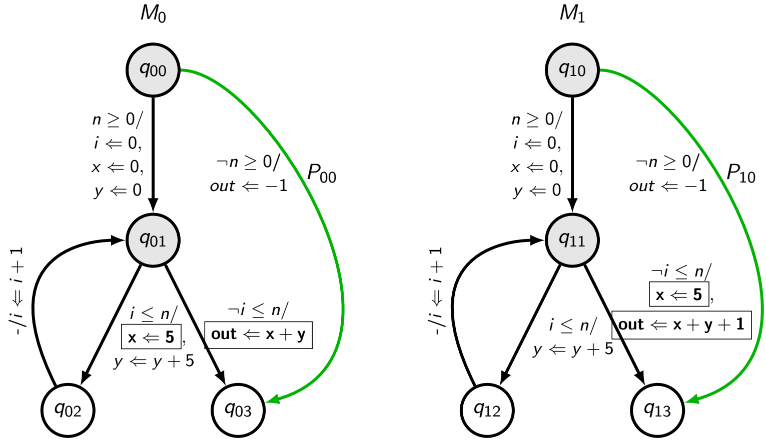
cTrace Generation in the EVP Method



EQ_LIST = ☒

C_LIST = ☒

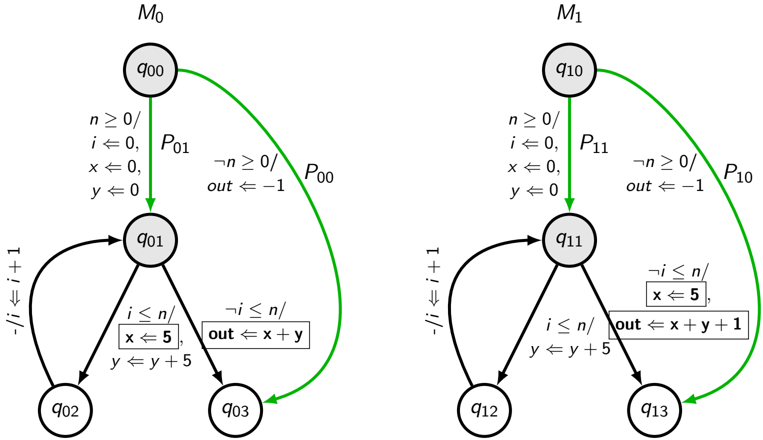
cTrace Generation in the EVP Method



EQ_LIST = (P_{00}, P_{10}) ● → ☒

C_LIST = ☒

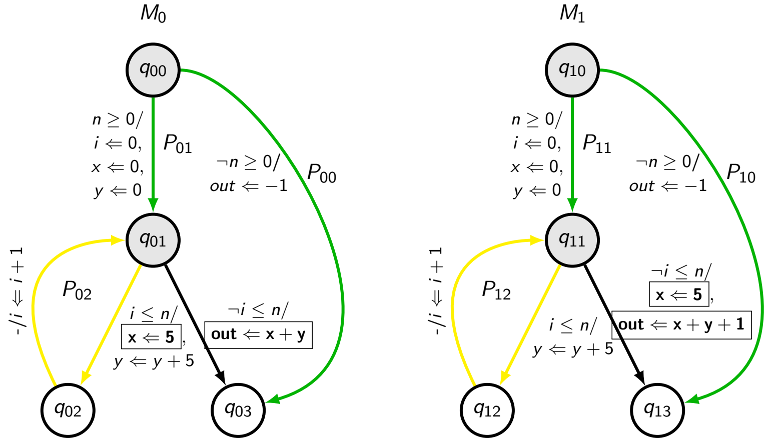
cTrace Generation in the EVP Method



EQ_LIST = (P_{00}, P_{10}) ● → (P_{01}, P_{11}) ● → ⊠

C_LIST = ⊠

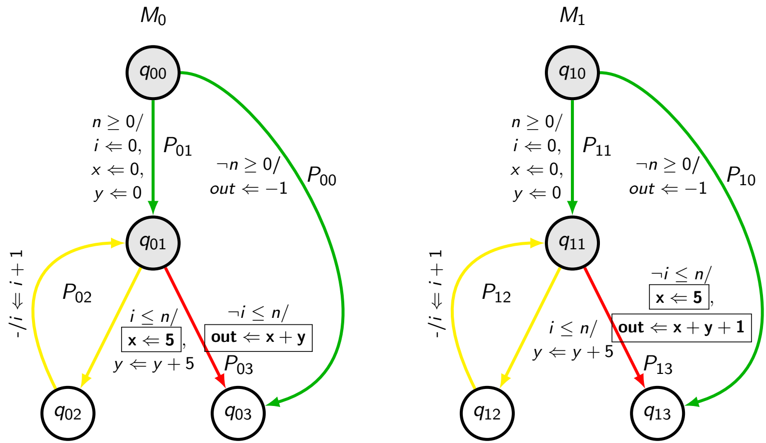
cTrace Generation in the EVP Method



EQ_LIST = (P_{00}, P_{10}) ● → (P_{01}, P_{11}) ● → ⊠

C_LIST = (P_{02}, P_{12}) ● → ⊠

cTrace Generation in the EVP Method



EQ_LIST = (P_{00}, P_{10}) ● → (P_{01}, P_{11}) ● → ☒

C_LIST = (P_{02}, P_{12}) ● → ☒

cTrace Generation in EVP Method

In case of nonequivalence

$$\textit{cTrace} \text{ of } M_0 = \langle \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \rangle$$

cTrace Generation in EVP Method

In case of nonequivalence

- Can not find equivalent path (say α)

$$\textit{cTrace} \text{ of } M_0 = \langle \qquad \boxed{\alpha} \; \rangle$$

cTrace Generation in EVP Method

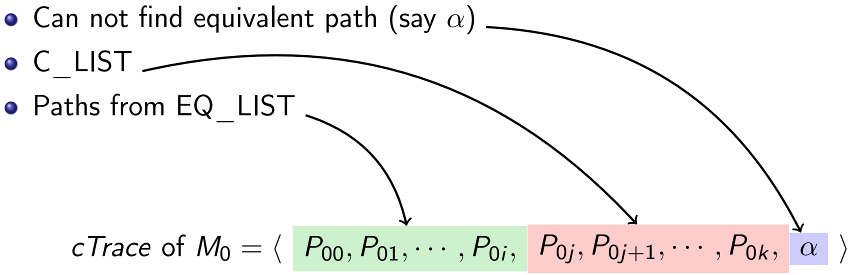
In case of nonequivalence

- Can not find equivalent path (say α)
- C_LIST

cTrace of $M_0 = \langle \quad P_{0j}, P_{0j+1}, \cdots, P_{0k}, \ \alpha \ \rangle$

cTrace Generation in EVP Method

In case of nonequivalence

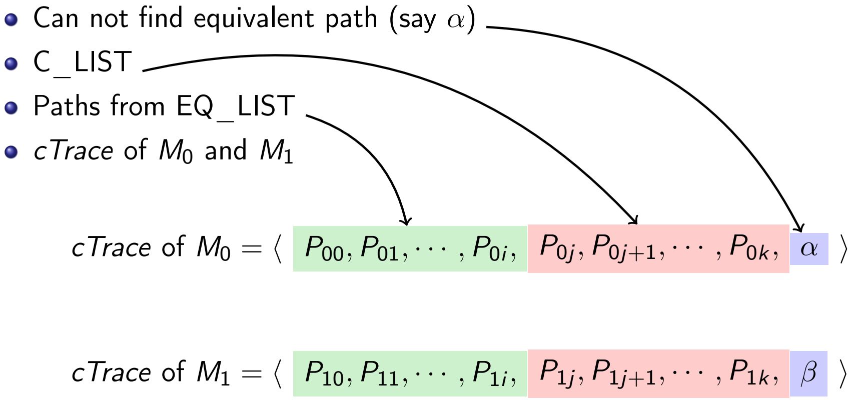


- Can not find equivalent path (say α)
- C_LIST
- Paths from EQ_LIST

cTrace of $M_0 = \langle P_{00}, P_{01}, \cdots, P_{0i}, P_{0j}, P_{0j+1}, \cdots, P_{0k}, \alpha \rangle$

cTrace Generation in EVP Method

In case of nonequivalence

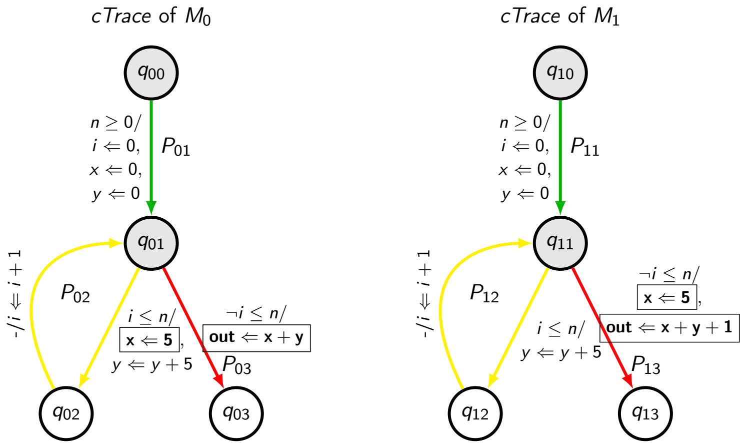


- Can not find equivalent path (say α)
- C_LIST
- Paths from EQ_LIST
- *cTrace* of M_0 and M_1

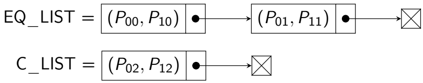
$$\textit{cTrace of } M_0 = \langle\ P_{00}, P_{01}, \cdots, P_{0i},\ P_{0j}, P_{0j+1}, \cdots, P_{0k},\ \alpha\ \rangle$$

$$\textit{cTrace of } M_1 = \langle\ P_{10}, P_{11}, \cdots, P_{1i},\ P_{1j}, P_{1j+1}, \cdots, P_{1k},\ \beta\ \rangle$$

cTrace Generation in EVP Method


cTrace of M_0

cTrace of M_1



```
#include<assert.h>
void main()
{
  int i_s,x_s,y_s,n,out_s;
  int i_t,x_t,y_t,out_t;
  __CPROVER_assume(n>=0);
  assert(!(n>=0));
```

```
// cTrace for M0
  if(n>=0)
  {
    i_s=0;x_s=0;y_s=0;
    __CPROVER_assume(i_s<=n);
    assert(!(i_s<=n));
    while(i_s<=n)
    {
      x_s=5;
      y_s=y_s+5;
      i_s=i_s+1;
    }
    out_s=x_s+y_s;
  }
```

```
//cTrace for M1
  if(n>=0)
  {
    i_t=0;x_t=0;y_t=0;
    __CPROVER_assume(i_t<=n);
    assert(!(i_t<=n));
    while(i_t<=n)
    {
      y_t=y_t+5;
      i_t=i_t+1;
    }
    x_t=5;
    out_t=x_t+y_t+1;
  }
```

```
//Live variables
  assert(x_s = x_t);
  assert(y_s = y_t);
//Output variable
 assert(out_s = out_t);
}
```

```
#include<assert.h>
void main()
{
  int i_s,x_s,y_s,n,out_s;
  int i_t,x_t,y_t,out_t;
  __CPROVER_assume(n>=0);
  assert(!(n>=0));
```

```
// cTrace for M0
  if(r
  {
    i
    _
    as
    wl
    {

      y_s=y_s+5;
      i_s=i_s+1;
    }
    out_s=x_s+y_s;
  }
```

```
//cTrace for M1
  if(n>=0)
  {
    i_t=0;x_t=0;y_t=0;
    __CPROVER_assume(i_t<=n);
    assert(!(i_t<=n));
    while(i_t<=n)
    {
      y_t=y_t+5;
```

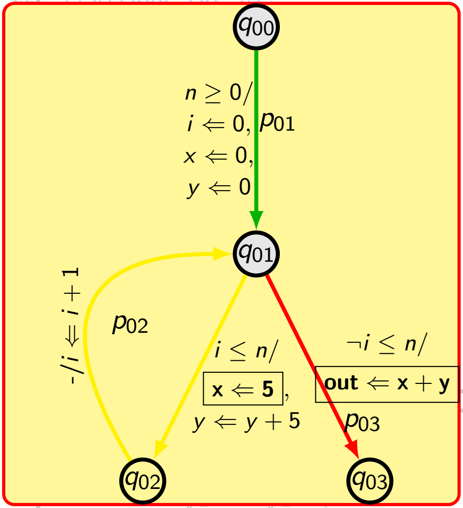
```
  assert(x_s = x_t);
  assert(y_s = y_t);
//Output variable
 assert(out_s = out_t);
}
```

- _s – Variables appearing in the cTrace of M_0.
- _t – Variables appearing in the cTrace of M_1.
- n – common Variable.
- __CPROVER_assume – allow only those computations that satisfy a given condition.

```
#include<assert.h>
void main()
{
  int i_s,x_s,y_s,n,out_s;
  int i_t,x_t,y_t,out_t;
  __CPROVER_assume(n>=0);
  assert(!(n>=0));
```

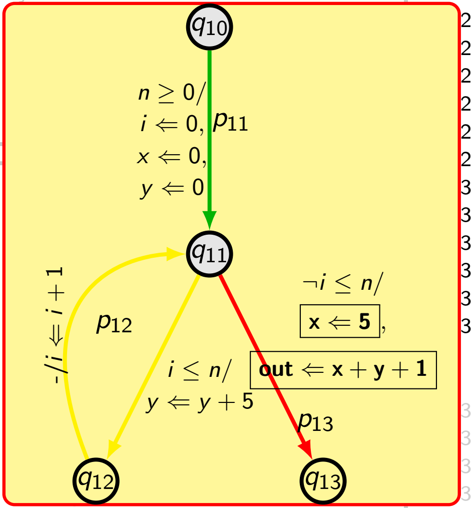
```
// cTrace for M0
  if(n>=0)
  {
    i_s=0;x_s=0;y_s=0;
    __CPROVER_assume(i_s<=n);
    assert(!(i_s<=n));
    while(i_s<=n)
    {
      x_s=5;
      y_s=y_s+5;
      i_s=i_s+1;
    }
    out_s=x_s+y_s;
  }
```

```
//cTrace for M1
```



```
//Output variable
 assert(out_s = out_t);
}
```

```
#include<assert.h>
void main()
{
```



```
    out_s=x_s+y_s;
  }
```

```
//cTrace for M1
  if(n>=0)
  {
    i_t=0;x_t=0;y_t=0;
    __CPROVER_assume(i_t<=n);
    assert(!(i_t<=n));
    while(i_t<=n)
    {
      y_t=y_t+5;
      i_t=i_t+1;
    }
    x_t=5;
    out_t=x_t+y_t+1;
  }
```

```
//Live variables
  assert(x_s = x_t);
  assert(y_s = y_t);
//Output variable
 assert(out_s = out_t);
}
```

```
#include<assert.h>
void main()
{
  int i_s,x_s,y_s,n,out_s;
  int
  __C
  asse
```

```
// cT
  if(n
  {
    i_s=0;x_s=0;y_s=0;
    __CPROVER_assume(i_s<=n);
    assert(!(i_s<=n));
    while(i_s<=n)
    {
      x_s=5;
      y_s=y_s+5;
      i_s=i_s+1;
    }
    out_s=x_s+y_s;
  }
```

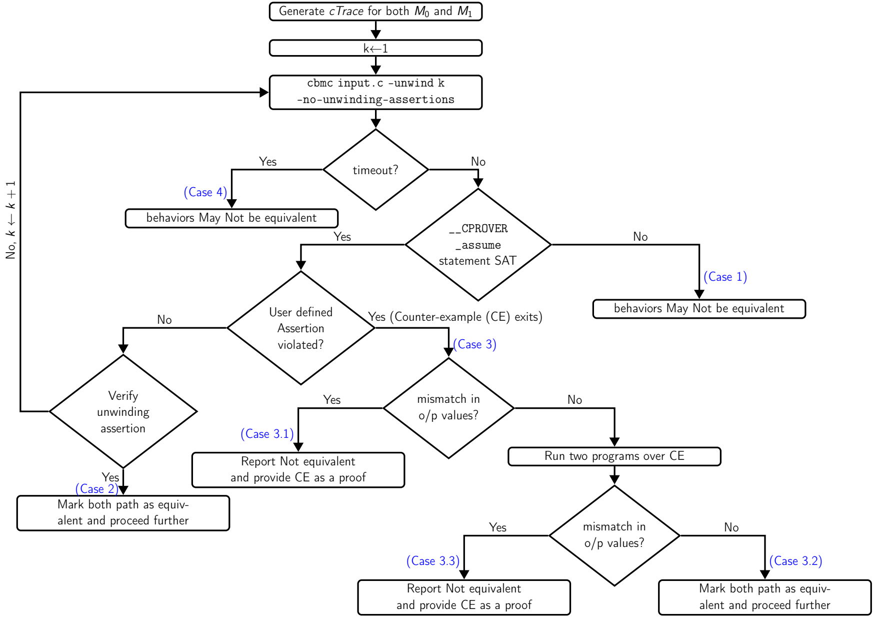
```
//cTrace for M1
  if(n>=0)
  {
    i_t=0;x_t=0;y_t=0;
                      n);
```

```
    }
    x_t=5;
    out_t=x_t+y_t+1;
  }
```

- CBMC verifies the specified assertions.
- If any violation of an assertion is detected, a counter-example is generated.
- $n = 0$ the values of the variable ‘out’ differs.

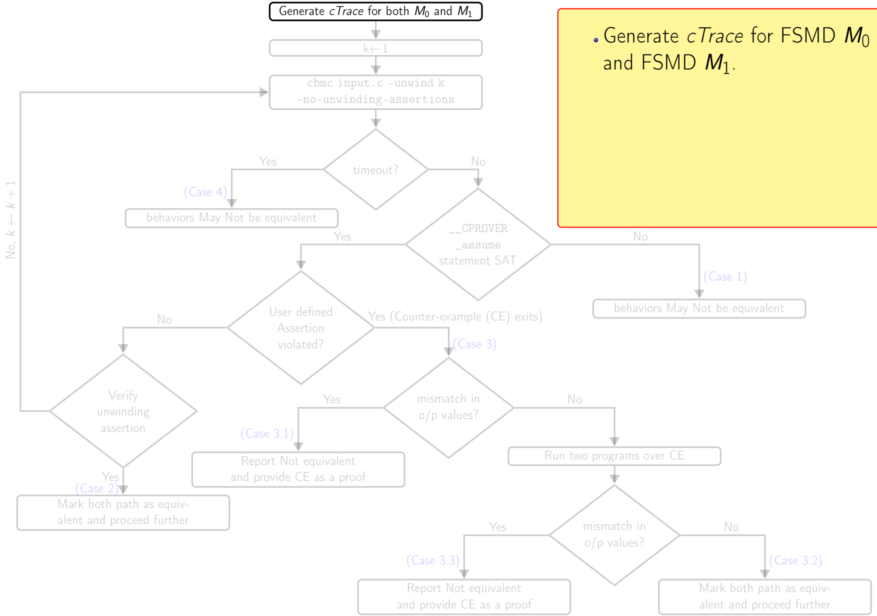
```
//Live variables
  assert(x_s = x_t);
  assert(y_s = y_t);
//Output variable
 assert(out_s = out_t);
}
```

PBEC+Counter Exmaple Generator (CEG)



Generate *cTrace* for both M_0 and M_1
k←1
`cbmc input.c -unwind k -no-unwinding-assertions`
timeout?
Yes
(Case 4)
behaviors May Not be equivalent
No
__CPROVER _assume statement SAT
Yes
No
(Case 1)
behaviors May Not be equivalent
User defined Assertion violated?
No
Yes (Counter-example (CE) exits)
(Case 3)
Verify unwinding assertion
No, $k \leftarrow k+1$
Yes
(Case 2)
Mark both path as equivalent and proceed further
mismatch in o/p values?
Yes
(Case 3.1)
Report Not equivalent and provide CE as a proof
No
Run two programs over CE
mismatch in o/p values?
Yes
(Case 3.3)
Report Not equivalent and provide CE as a proof
No
(Case 3.2)
Mark both path as equivalent and proceed further

PBEC+Counter Exmaple Generator (CEG)



Generate *cTrace* for both M_0 and M_1

k←1

`cbmc input.c -unwind k -no-unwinding-assertions`

timeout?

Yes

(Case 4)

behaviors May Not be equivalent

No

`__CPROVER_assume` statement SAT

No

(Case 1)

behaviors May Not be equivalent

Yes

User defined Assertion violated?

No

Verify unwinding assertion

No, $k \leftarrow k+1$

Yes

(Case 2)

Mark both path as equivalent and proceed further

Yes (Counter-example (CE) exits)

(Case 3)

mismatch in o/p values?

Yes

(Case 3.1)

Report Not equivalent and provide CE as a proof

No

Run two programs over CE

mismatch in o/p values?

Yes

(Case 3.3)

Report Not equivalent and provide CE as a proof

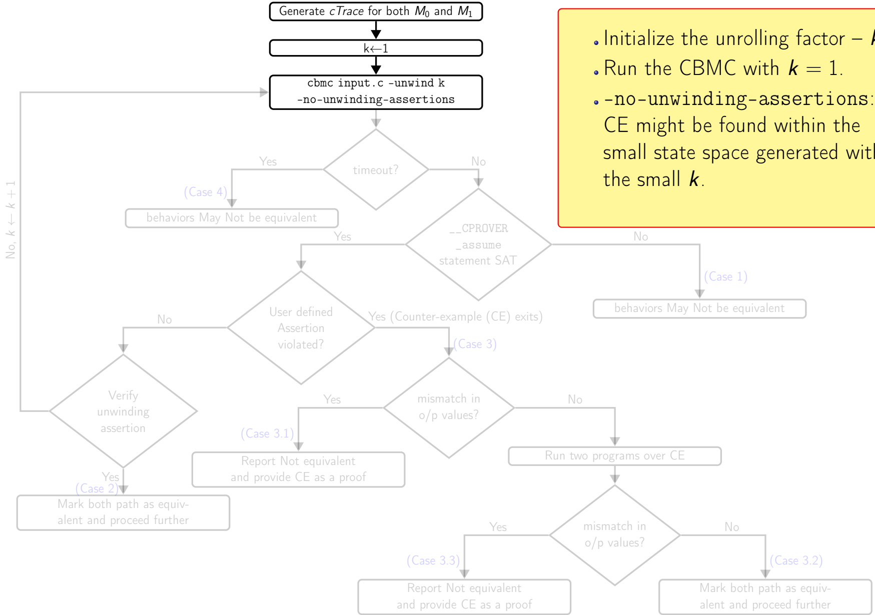
No

(Case 3.2)

Mark both path as equivalent and proceed further

- Generate *cTrace* for FSMD M_0 and FSMD M_1.

PBEC+Counter Exmaple Generator (CEG)



Generate *cTrace* for both M_0 and M_1

k←1

`cbmc input.c -unwind k -no-unwinding-assertions`

timeout?

Yes

(Case 4)

behaviors May Not be equivalent

No

`__CPROVER _assume` statement SAT

Yes

No

(Case 1)

behaviors May Not be equivalent

User defined Assertion violated?

No

Yes (Counter-example (CE) exits)

(Case 3)

Verify unwinding assertion

No, $k \leftarrow k + 1$

Yes

(Case 2)

Mark both path as equivalent and proceed further

mismatch in o/p values?

Yes

(Case 3.1)

Report Not equivalent and provide CE as a proof

No

Run two programs over CE

mismatch in o/p values?

Yes

(Case 3.3)

Report Not equivalent and provide CE as a proof

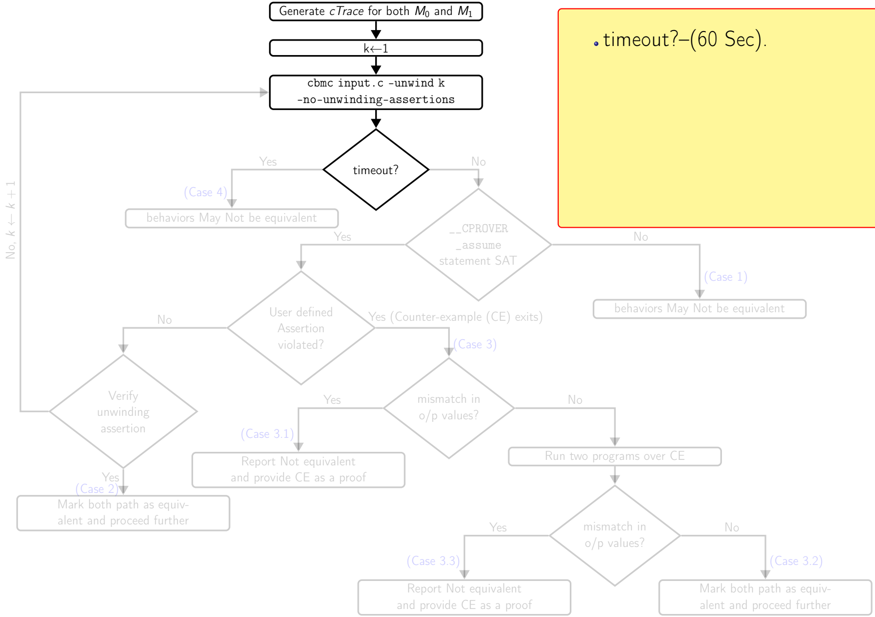
No

(Case 3.2)

Mark both path as equivalent and proceed further

- Initialize the unrolling factor – $\boldsymbol{k}$.
- Run the CBMC with $\boldsymbol{k} = 1$.
- `-no-unwinding-assertions`: CE might be found within the small state space generated with the small $\boldsymbol{k}$.

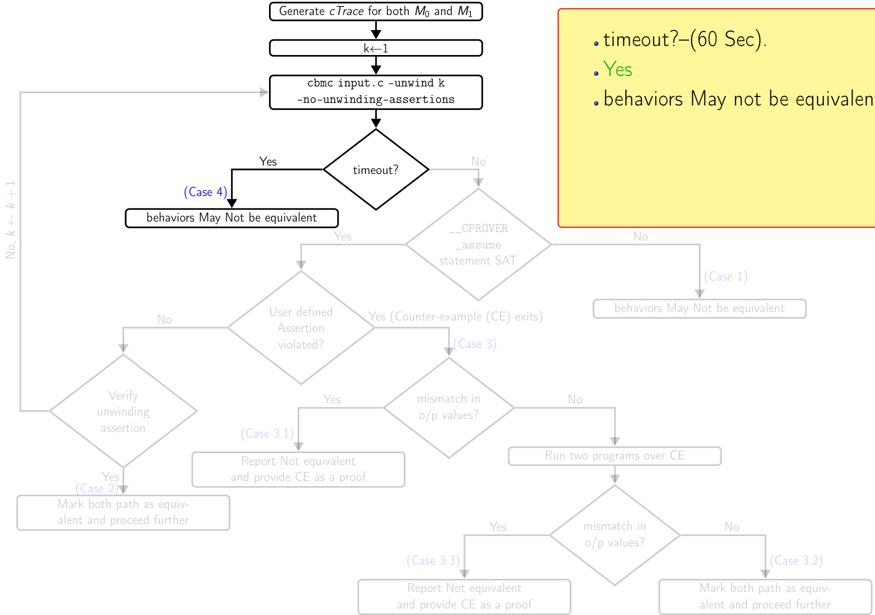
PBEC+Counter Exmaple Generator (CEG)



Generate *cTrace* for both M_0 and M_1
k←1
cbmc input.c -unwind k -no-unwinding-assertions
timeout?
Yes
No
(Case 4)
behaviors May Not be equivalent
__CPROVER _assume statement SAT
Yes
No
(Case 1)
behaviors May Not be equivalent
User defined Assertion violated?
No
Yes (Counter-example (CE) exits)
(Case 3)
Verify unwinding assertion
No, $k \leftarrow k+1$
Yes
(Case 2)
Mark both path as equiv- alent and proceed further
mismatch in o/p values?
Yes
No
(Case 3.1)
Report Not equivalent and provide CE as a proof
Run two programs over CE
mismatch in o/p values?
Yes
No
(Case 3.3)
Report Not equivalent and provide CE as a proof
(Case 3.2)
Mark both path as equiv- alent and proceed further

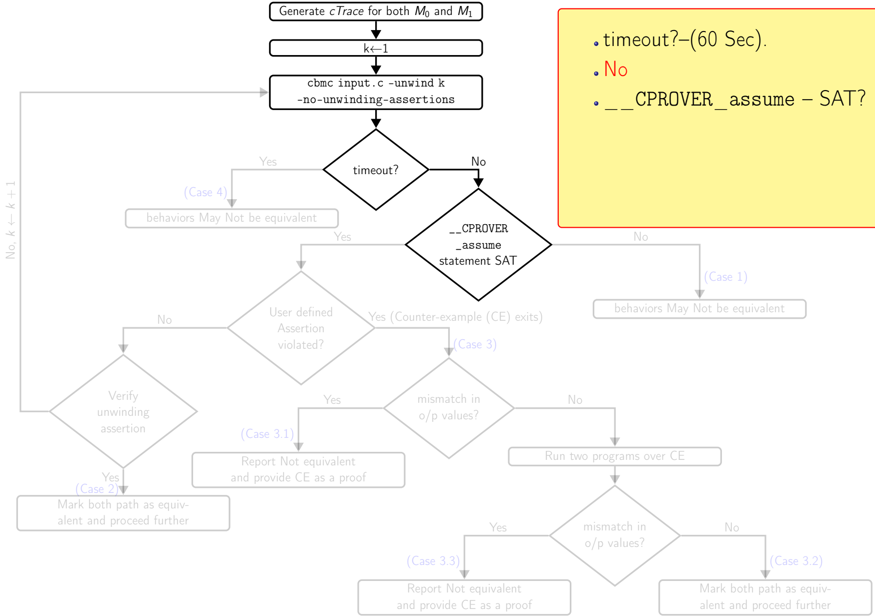
- timeout?–(60 Sec).

PBEC+Counter Exmaple Generator (CEG)



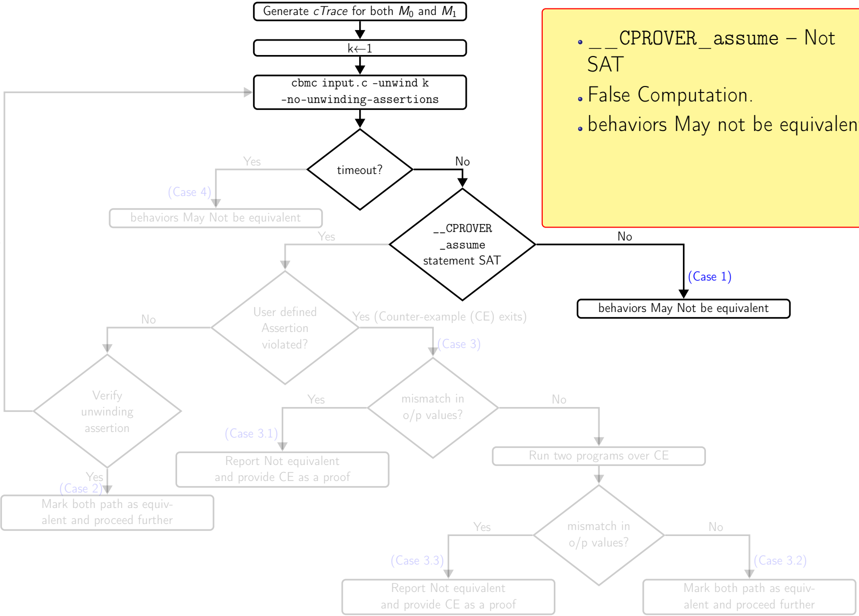
- timeout?–(60 Sec).
- Yes
- behaviors May not be equivalent.

PBEC+Counter Exmaple Generator (CEG)


Generate *cTrace* for both M_0 and M_1
k←1
cbmc input.c -unwind k -no-unwinding-assertions
timeout?
Yes
No
(Case 4)
behaviors May Not be equivalent
No, $k \leftarrow k+1$
__CPROVER _assume statement SAT
Yes
No
(Case 1)
behaviors May Not be equivalent
User defined Assertion violated?
No
Yes (Counter-example (CE) exits)
(Case 3)
Verify unwinding assertion
Yes
(Case 2)
Mark both path as equiv-alent and proceed further
mismatch in o/p values?
Yes
(Case 3.1)
Report Not equivalent and provide CE as a proof
No
Run two programs over CE
mismatch in o/p values?
Yes
(Case 3.3)
Report Not equivalent and provide CE as a proof
No
(Case 3.2)
Mark both path as equiv-alent and proceed further

- timeout?–(60 Sec).
- No
- __CPROVER_assume – SAT?

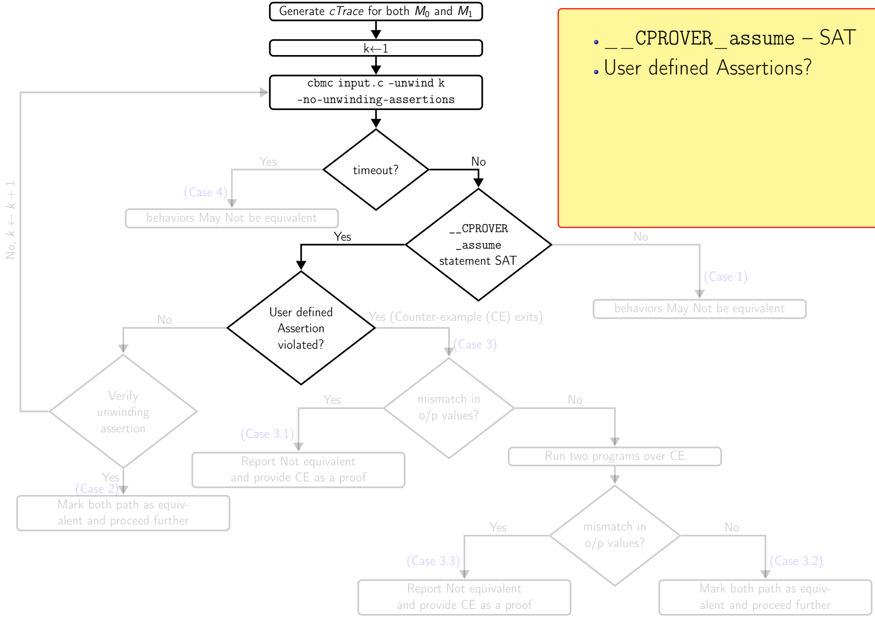
PBEC+Counter Exmaple Generator (CEG)



Generate *cTrace* for both M_0 and M_1
k←1
cbmc input.c -unwind k -no-unwinding-assertions
timeout?
Yes
No
(Case 4)
behaviors May Not be equivalent
__CPROVER _assume statement SAT
Yes
No
(Case 1)
behaviors May Not be equivalent
No, $k \leftarrow k+1$
User defined Assertion violated?
No
Yes (Counter-example (CE) exits)
(Case 3)
Verify unwinding assertion
Yes
(Case 2)
Mark both path as equivalent and proceed further
mismatch in o/p values?
Yes
No
(Case 3.1)
Report Not equivalent and provide CE as a proof
Run two programs over CE
mismatch in o/p values?
Yes
No
(Case 3.3)
Report Not equivalent and provide CE as a proof
(Case 3.2)
Mark both path as equivalent and proceed further

- `__CPROVER_assume` – Not SAT
- False Computation.
- behaviors May not be equivalent.

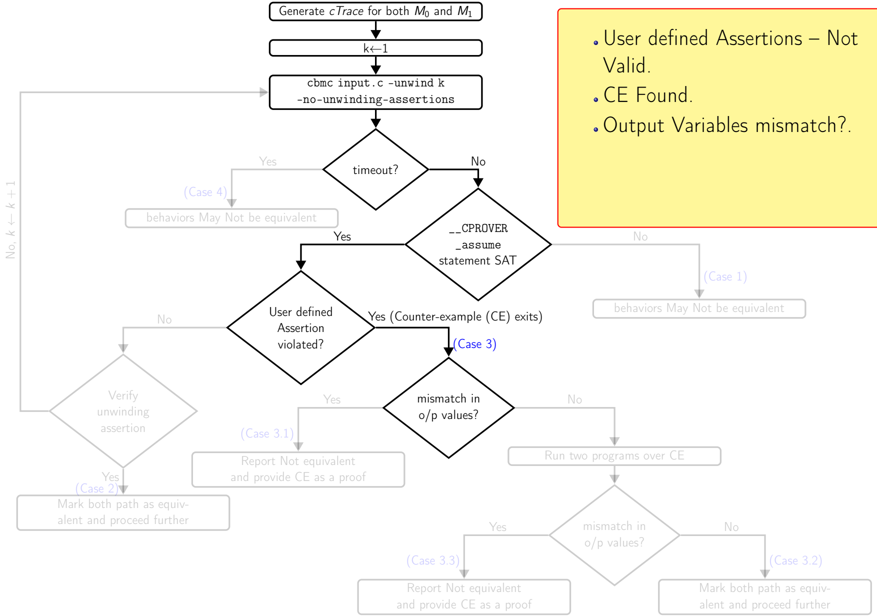
PBEC+Counter Exmaple Generator (CEG)



Generate $cTrace$ for both M_0 and M_1
k←1
cbmc input.c -unwind k -no-unwinding-assertions
timeout?
Yes
(Case 4)
behaviors May Not be equivalent
No
__CPROVER _assume statement SAT
Yes
No
(Case 1)
behaviors May Not be equivalent
User defined Assertion violated?
No
Yes (Counter-example (CE) exits)
(Case 3)
Verify unwinding assertion
No, $k \leftarrow k+1$
Yes
(Case 2)
Mark both path as equiv-alent and proceed further
mismatch in o/p values?
Yes
(Case 3.1)
Report Not equivalent and provide CE as a proof
No
Run two programs over CE
mismatch in o/p values?
Yes
(Case 3.3)
Report Not equivalent and provide CE as a proof
No
(Case 3.2)
Mark both path as equiv-alent and proceed further

- __CPROVER_assume – SAT
- User defined Assertions?

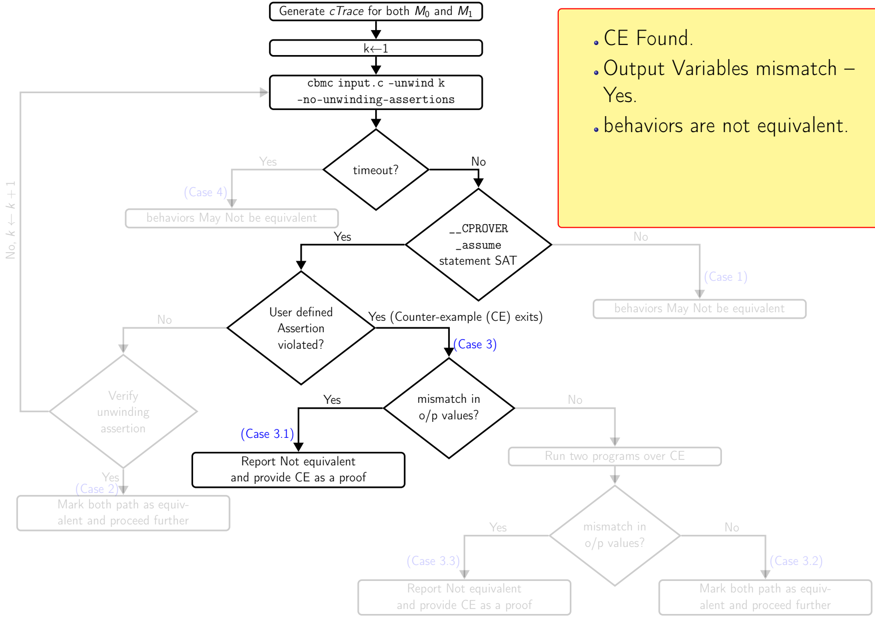
PBEC+Counter Exmaple Generator (CEG)



Generate *cTrace* for both M_0 and M_1
k←1
`cbmc input.c -unwind k -no-unwinding-assertions`
timeout?
Yes
No
(Case 4)
behaviors May Not be equivalent
No, $k \leftarrow k+1$
Yes
__CPROVER _assume statement SAT
No
(Case 1)
behaviors May Not be equivalent
No
User defined Assertion violated?
Yes (Counter-example (CE) exits)
(Case 3)
Verify unwinding assertion
Yes
mismatch in o/p values?
No
(Case 3.1)
Report Not equivalent and provide CE as a proof
Run two programs over CE
Yes
(Case 2)
Mark both path as equiv-alent and proceed further
Yes
mismatch in o/p values?
No
(Case 3.3)
(Case 3.2)
Report Not equivalent and provide CE as a proof
Mark both path as equiv-alent and proceed further

- User defined Assertions – Not Valid.
- CE Found.
- Output Variables mismatch?.

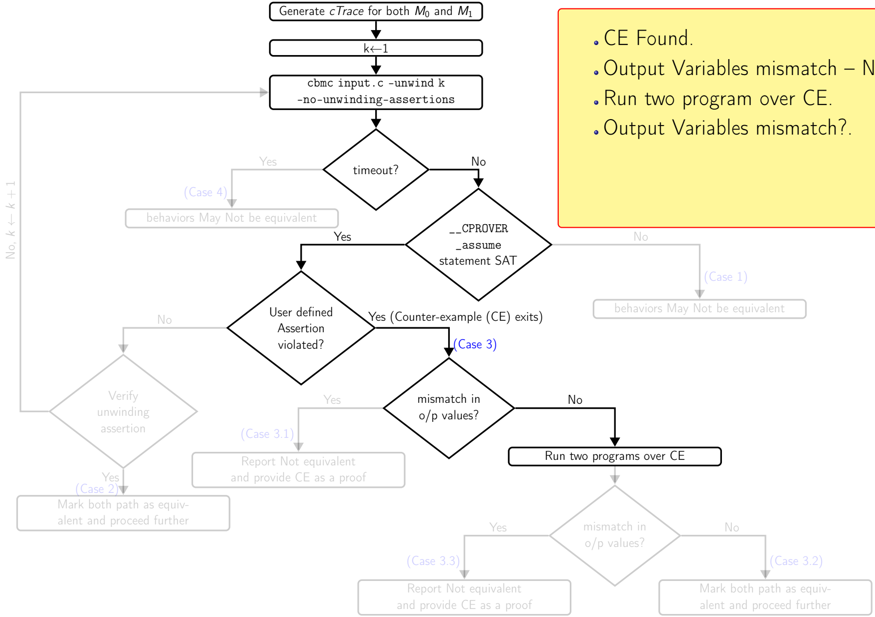
PBEC+Counter Exmaple Generator (CEG)



Generate $cTrace$ for both M_0 and M_1
k←1
cbmc input.c -unwind k -no-unwinding-assertions
timeout?
Yes
(Case 4)
behaviors May Not be equivalent
No
__CPROVER _assume statement SAT
No
(Case 1)
behaviors May Not be equivalent
Yes
User defined Assertion violated?
No
Verify unwinding assertion
No, $k \leftarrow k+1$
Yes
(Case 2)
Mark both path as equiv- alent and proceed further
Yes (Counter-example (CE) exits)
(Case 3)
mismatch in o/p values?
Yes
(Case 3.1)
Report Not equivalent and provide CE as a proof
No
Run two programs over CE
mismatch in o/p values?
Yes
(Case 3.3)
Report Not equivalent and provide CE as a proof
No
(Case 3.2)
Mark both path as equiv- alent and proceed further

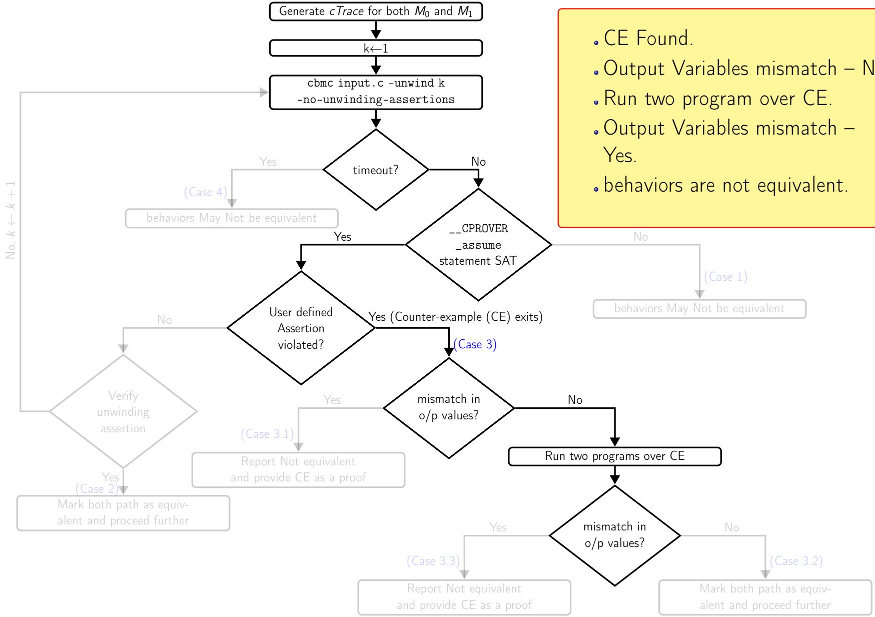
- CE Found.
- Output Variables mismatch – Yes.
- behaviors are not equivalent.

PBEC+Counter Exmaple Generator (CEG)



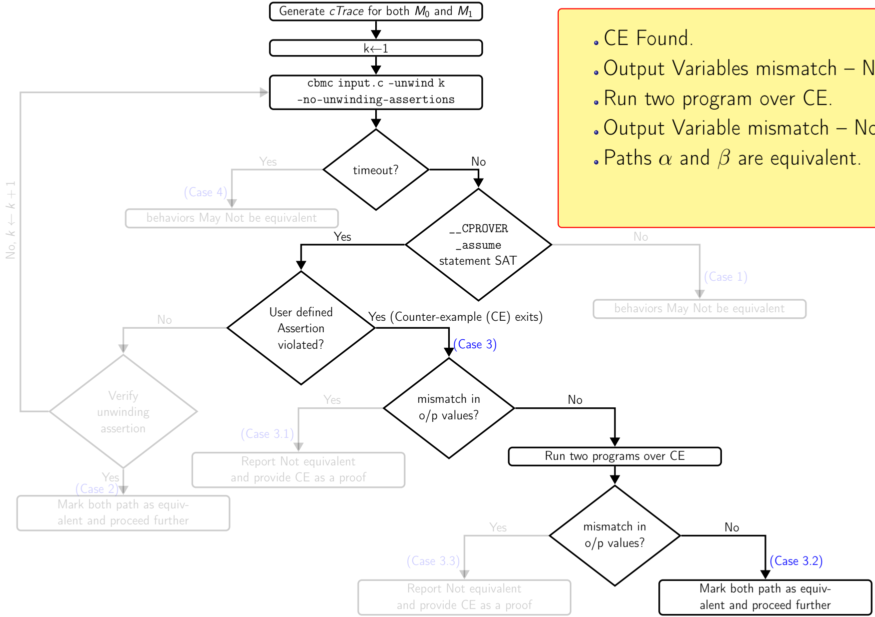
- CE Found.
- Output Variables mismatch – No.
- Run two program over CE.
- Output Variables mismatch?.

PBEC+Counter Exmaple Generator (CEG)



- CE Found.
- Output Variables mismatch – No.
- Run two program over CE.
- Output Variables mismatch – Yes.
- behaviors are not equivalent.

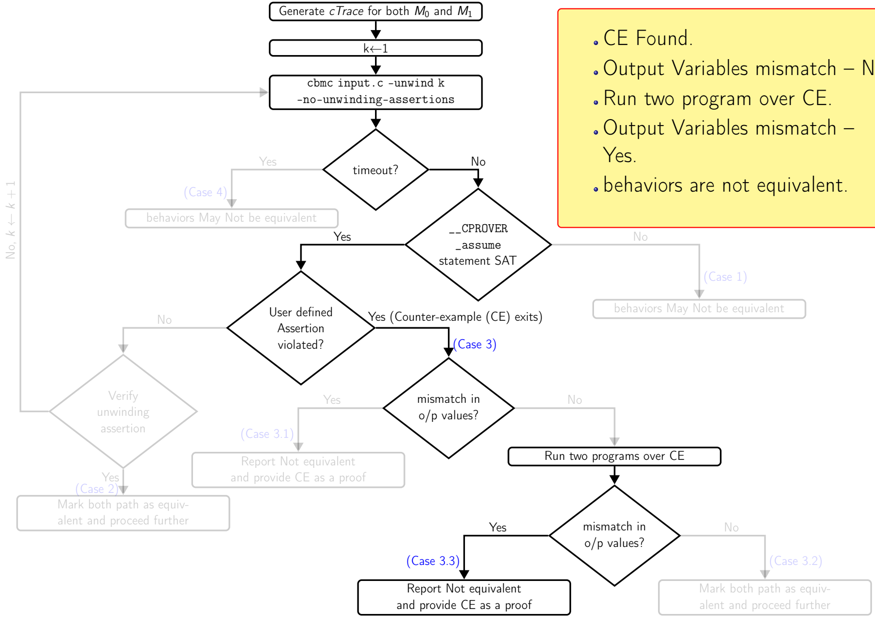
PBEC+Counter Exmaple Generator (CEG)



Generate *cTrace* for both M_0 and M_1
k←1
cbmc input.c -unwind k -no-unwinding-assertions
timeout?
Yes
(Case 4)
behaviors May Not be equivalent
No
__CPROVER _assume statement SAT
No
(Case 1)
behaviors May Not be equivalent
Yes
User defined Assertion violated?
No
Verify unwinding assertion
No, $k \leftarrow k+1$
Yes
(Case 2)
Mark both path as equiv-alent and proceed further
Yes (Counter-example (CE) exits)
(Case 3)
mismatch in o/p values?
Yes
(Case 3.1)
Report Not equivalent and provide CE as a proof
No
Run two programs over CE
mismatch in o/p values?
Yes
(Case 3.3)
Report Not equivalent and provide CE as a proof
No
(Case 3.2)
Mark both path as equiv-alent and proceed further

- CE Found.
- Output Variables mismatch – No.
- Run two program over CE.
- Output Variable mismatch – No.
- Paths α and β are equivalent.

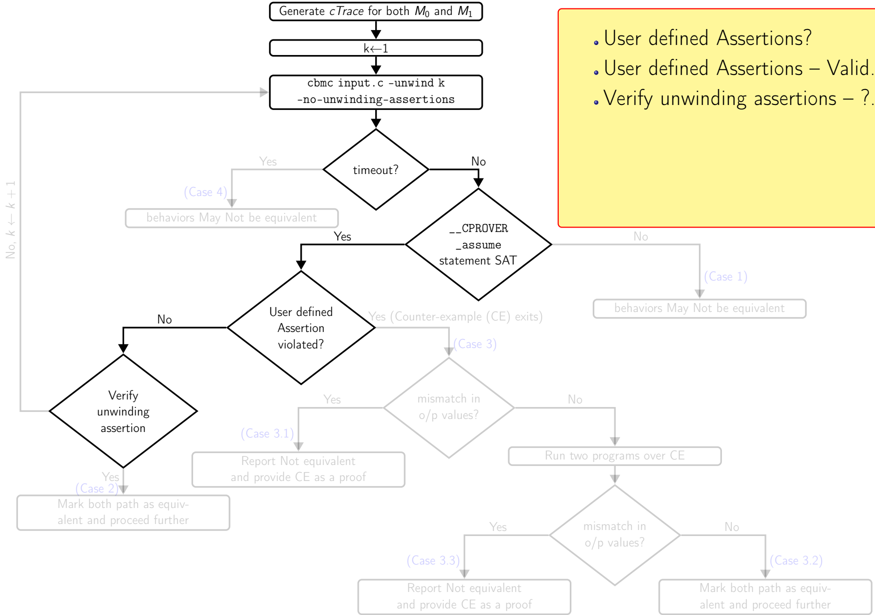
PBEC+Counter Exmaple Generator (CEG)



Generate *cTrace* for both M_0 and M_1
k←1
cbmc input.c -unwind k -no-unwinding-assertions
timeout?
Yes
(Case 4)
behaviors May Not be equivalent
No
__CPROVER _assume statement SAT
Yes
No
(Case 1)
behaviors May Not be equivalent
User defined Assertion violated?
Yes (Counter-example (CE) exits)
(Case 3)
No
Verify unwinding assertion
No, $k \leftarrow k+1$
Yes
(Case 2)
Mark both path as equivalent and proceed further
mismatch in o/p values?
Yes
(Case 3.1)
Report Not equivalent and provide CE as a proof
No
Run two programs over CE
mismatch in o/p values?
Yes
(Case 3.3)
Report Not equivalent and provide CE as a proof
No
(Case 3.2)
Mark both path as equivalent and proceed further

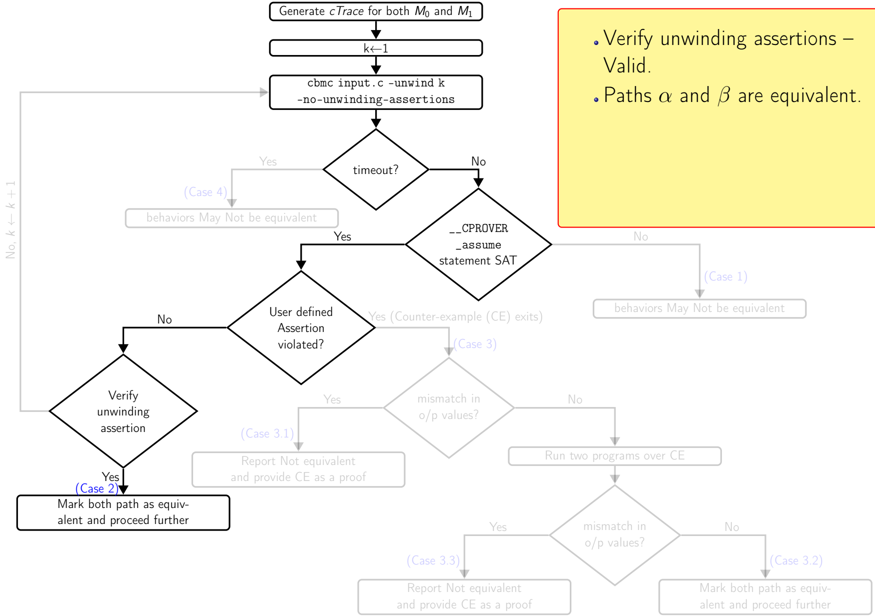
- CE Found.
- Output Variables mismatch – No.
- Run two program over CE.
- Output Variables mismatch – Yes.
- behaviors are not equivalent.

PBEC+Counter Exmaple Generator (CEG)



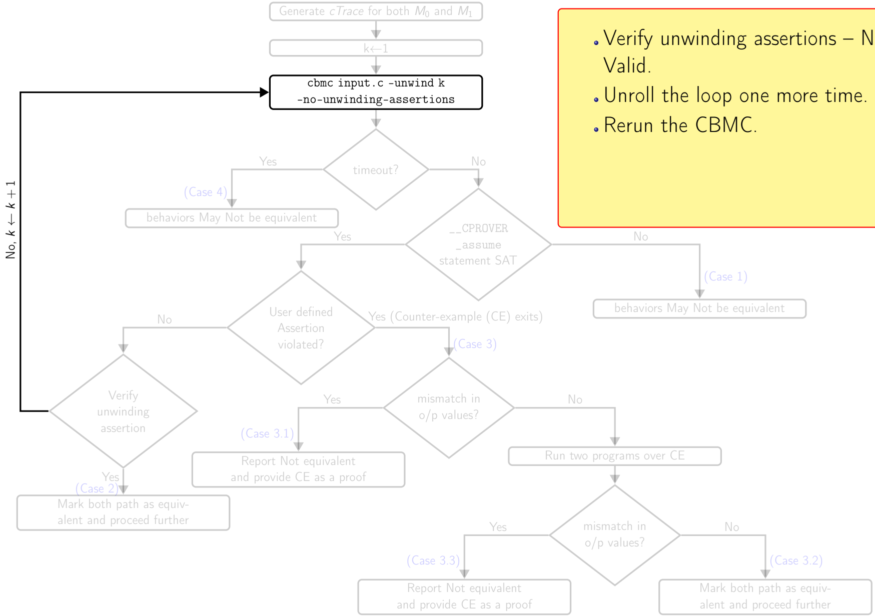
- User defined Assertions?
- User defined Assertions – Valid.
- Verify unwinding assertions – ?.

PBEC+Counter Exmaple Generator (CEG)



- Verify unwinding assertions – Valid.
- Paths α and β are equivalent.

PBEC+Counter Exmaple Generator (CEG)


Generate *cTrace* for both M_0 and M_1
k←1
cbmc input.c -unwind k -no-unwinding-assertions
timeout?
Yes
No
(Case 4)
behaviors May Not be equivalent
__CPROVER _assume statement SAT
Yes
No
(Case 1)
behaviors May Not be equivalent
User defined Assertion violated?
No
Yes (Counter-example (CE) exits)
(Case 3)
Verify unwinding assertion
No, $k \leftarrow k+1$
Yes
(Case 2)
Mark both path as equiv-alent and proceed further
mismatch in o/p values?
Yes
(Case 3.1)
Report Not equivalent and provide CE as a proof
No
Run two programs over CE
mismatch in o/p values?
Yes
(Case 3.3)
Report Not equivalent and provide CE as a proof
No
(Case 3.2)
Mark both path as equiv-alent and proceed further

- Verify unwinding assertions – Not Valid.
- Unroll the loop one more time.
- Rerun the CBMC.

Experimental Results

Benchmarks	#Path	#State		Decision		Time (ms)		Lines
		M_0	M_1	EVP	Our	EVP	Our	
DIFFEQ	3	15	9	E	E	25	25	-
LRU	39	33	32	E	E	1038	1038	-
DCT	1	8	16	MNE	NE	85	766	185
PERFECT	7	6	4	MNE	NE	56	227	74
MODN	9	8	9	MNE	NE	66	890	137
GCD	11	8	4	MNE	NE	31	100	97
Test Case	6	5	5	MNE	MNE	20	26	32

E – Equivalent, MNE – May Not be Equivalent, NE – Not Equivalent

- Implemented CEG on top of the EVP.

Experimental Results

Benchmarks	#Path	#State		Decision		Time (ms)		Lines
		M_0	M_1	EVP	Our	EVP	Our	
DIFFEQ	3	15	9	E	E	25	25	-
LRU	39	33	32	E	E	1038	1038	-
DCT	1	8	16	MNE	NE	85	766	185
PERFECT	7	6	4	MNE	NE	56	227	74
MODN	9	8	9	MNE	NE	66	890	137
GCD	11	8	4	MNE	NE	31	100	97
Test Case	6	5	5	MNE	MNE	20	26	32

E – Equivalent, MNE – May Not be Equivalent, NE – Not Equivalent

- No side effect on the existing method.

Experimental Results

Benchmarks	#Path	#State		Decision		Time (ms)		Lines
		M_0	M_1	EVP	Our	EVP	Our	
DIFFEQ	3	15	9	E	E	25	25	-
LRU	39	33	32	E	E	1038	1038	-
DCT	1	8	16	MNE	NE	85	766	185
PERFECT	7	6	4	MNE	NE	56	227	74
MODN	9	8	9	MNE	NE	66	890	137
GCD	11	8	4	MNE	NE	31	100	97
Test Case	6	5	5	MNE	MNE	20	26	32

E – Equivalent, MNE – May Not be Equivalent, NE – Not Equivalent

- The EVP takes strong decisions about the non-equivalence of behaviors.

Experimental Results

Benchmarks	#Path	#State		Decision		Time (ms)		Lines
		M_0	M_1	EVP	Our	EVP	Our	
DIFFEQ	3	15	9	E	E	25	25	-
LRU	39	33	32	E	E	1038	1038	-
DCT	1	8	16	MNE	NE	85	766	185
PERFECT	7	6	4	MNE	NE	56	227	74
MODN	9	8	9	MNE	NE	66	890	137
GCD	11	8	4	MNE	NE	31	100	97
Test Case	6	5	5	MNE	MNE	20	26	32

E – Equivalent, MNE – May Not be Equivalent, NE – Not Equivalent

- Finds a scenario where the EVP gives false negative result.

Conclusion & Future Works

- Proposed a CEG mechanism for the PBEC.
- PBEC is further strengthened with the CEG mechanism.
- For some scenarios PBEC reports not equivalent and provide CE as a proof.
- Identified a false negative result of the EVP method.
- Similar CEG mechanism can also be developed for other reported equivalence checking methods as well.
- Enhance the EVP to handle false negative cases.

Thank you!