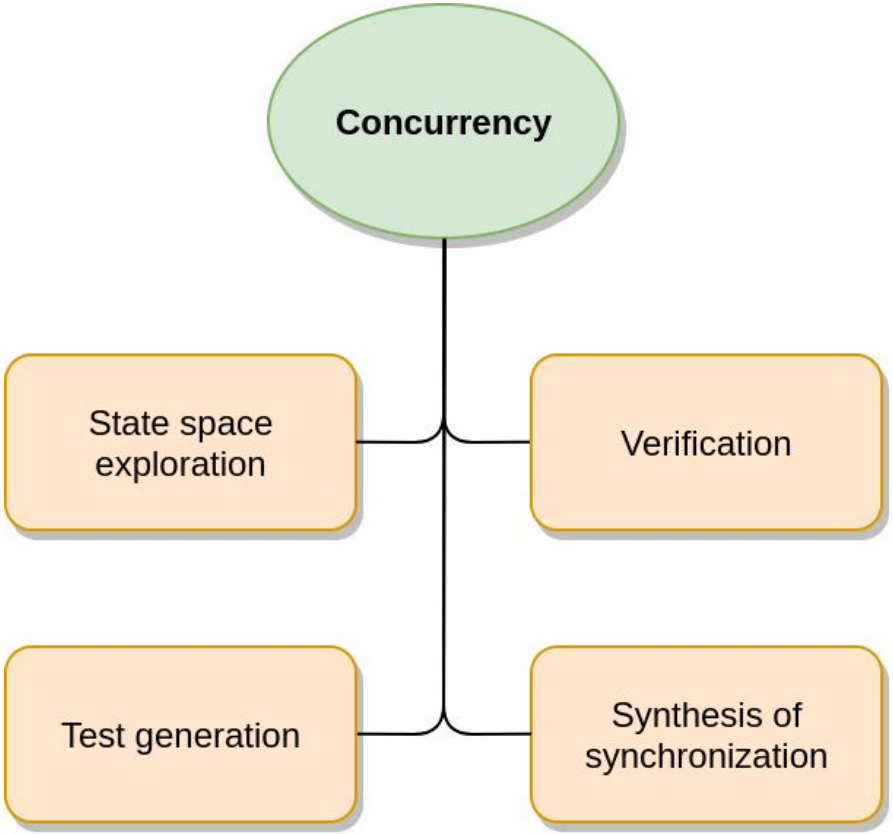
Relational Interpretation of Concurrency

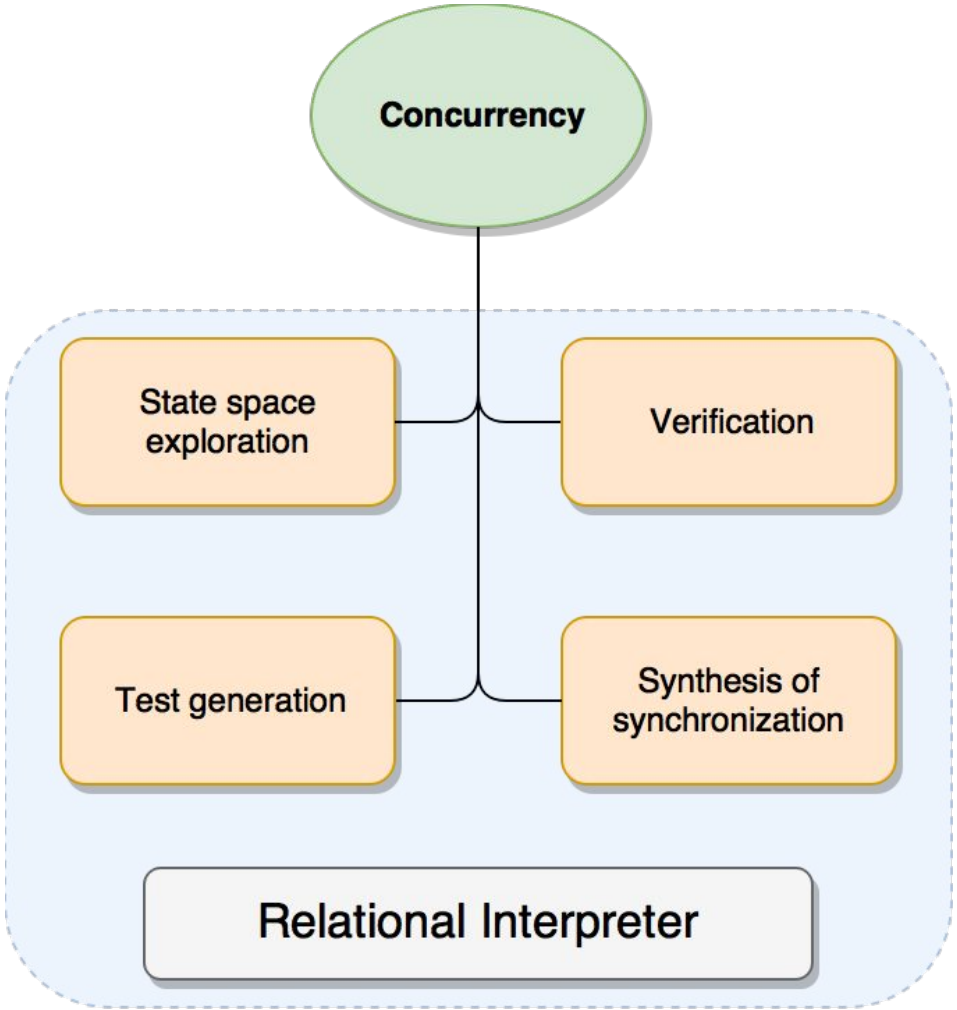
Evgenii Moiseenko

Anton Podkopaev

Plan

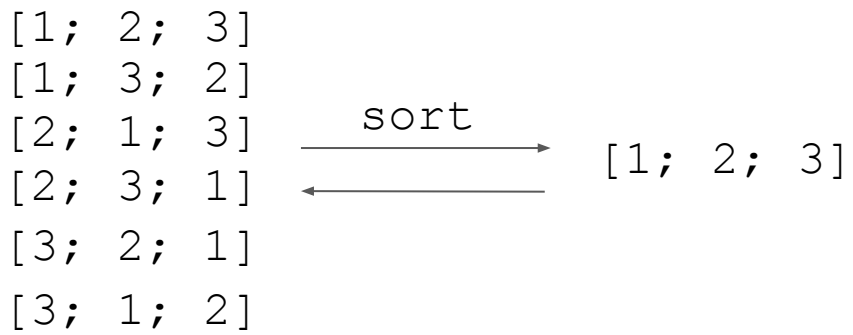
- Relational Programming
- Relational Interpreter
- Relational Interpreter for Concurrency


Concurrency
State space exploration
Verification
Test generation
Synthesis of synchronization


Concurrency
State space exploration
Verification
Test generation
Synthesis of synchronization
Relational Interpreter

Functional Programming

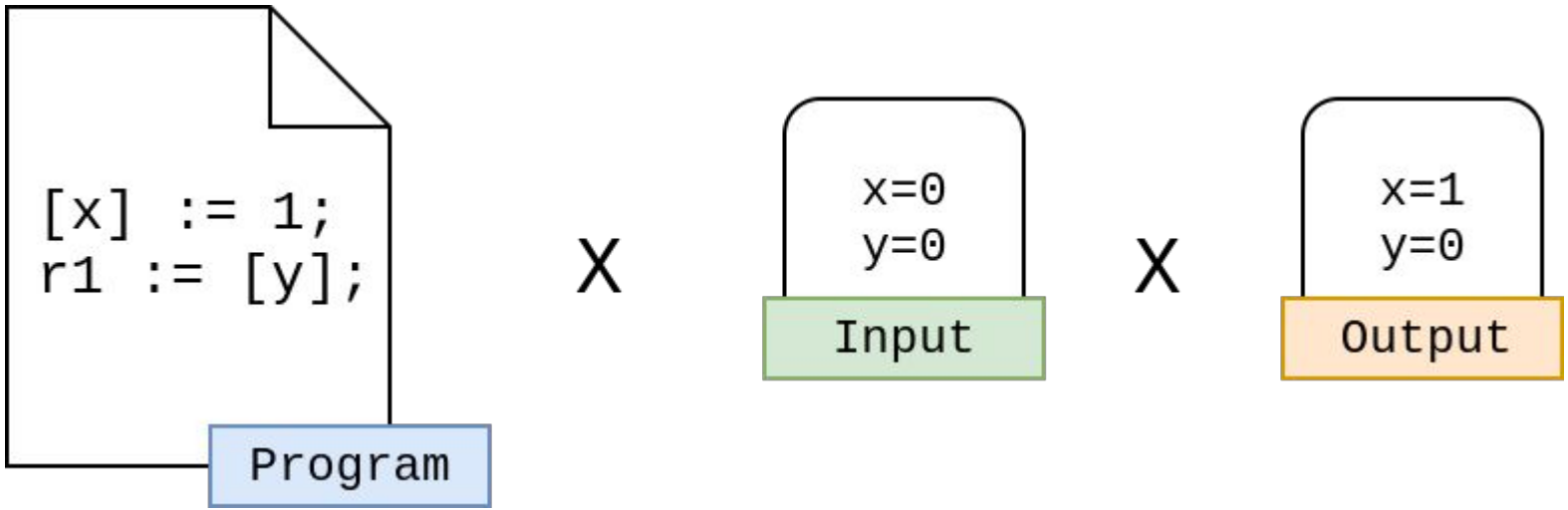
Relational Programming



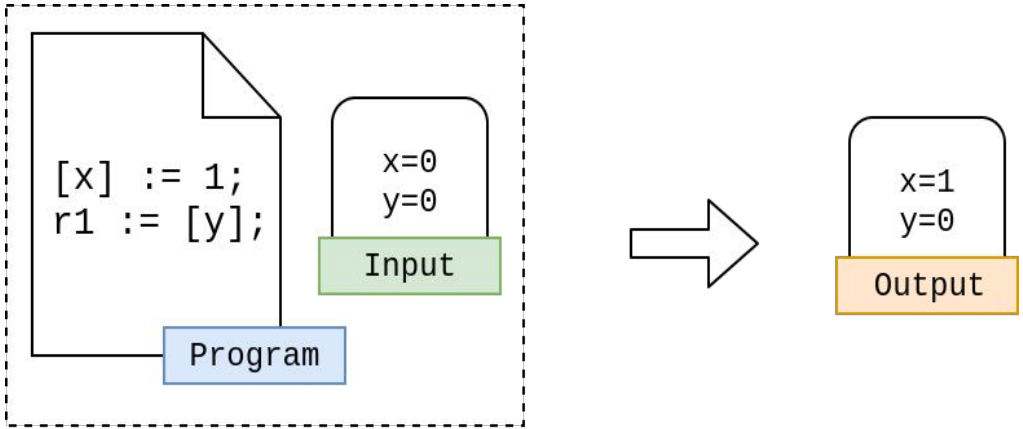
OCanren

- OCaml DSL for Relational Programming
  - Declarative specification of relations
  - Complete backtracking search

Relational Interpreter

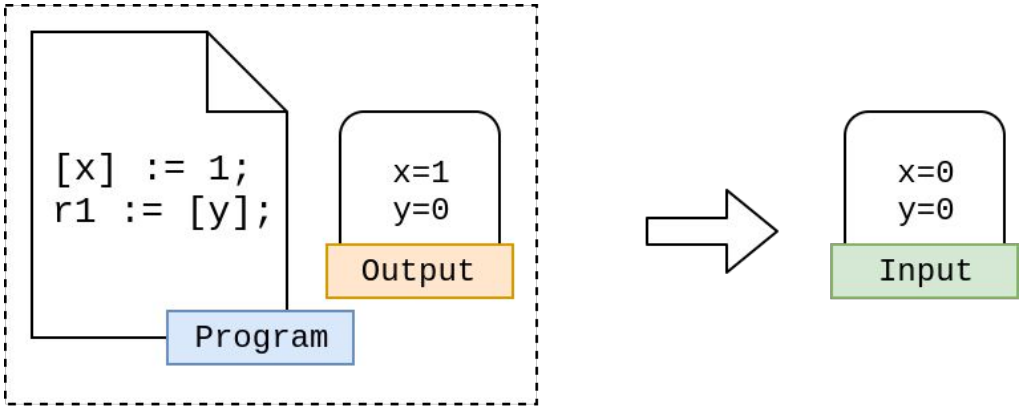


Relational Interpreter



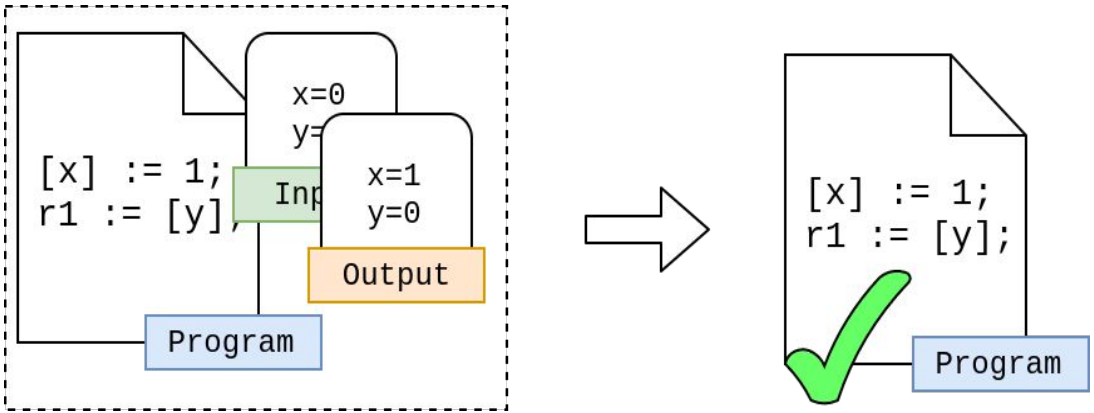
- Execution

Relational Interpreter



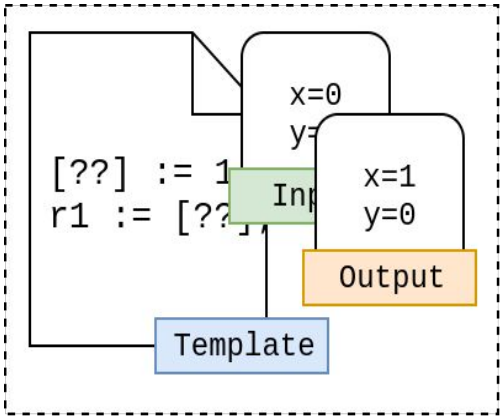
- Execution
- Test generation

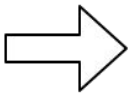
Relational Interpreter

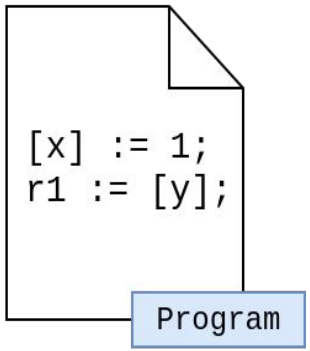


- Execution
- Test generation
- Verification

Relational Interpreter







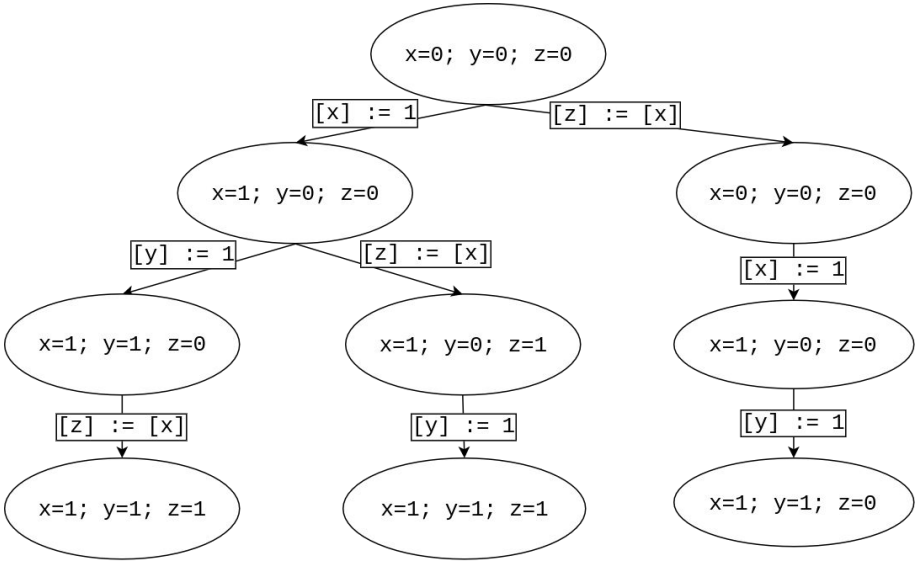
- Execution
- Test generation
- Verification
- Synthesis

Results

- Relational Interpreter for Concurrency
  - Compact and declarative (~1.5K lines of code)
- Extension of OCanren
  - Tabling – memoization for relational programs
    - Efficient state space exploration
  - Constructive negation
    - Allows to express negative examples

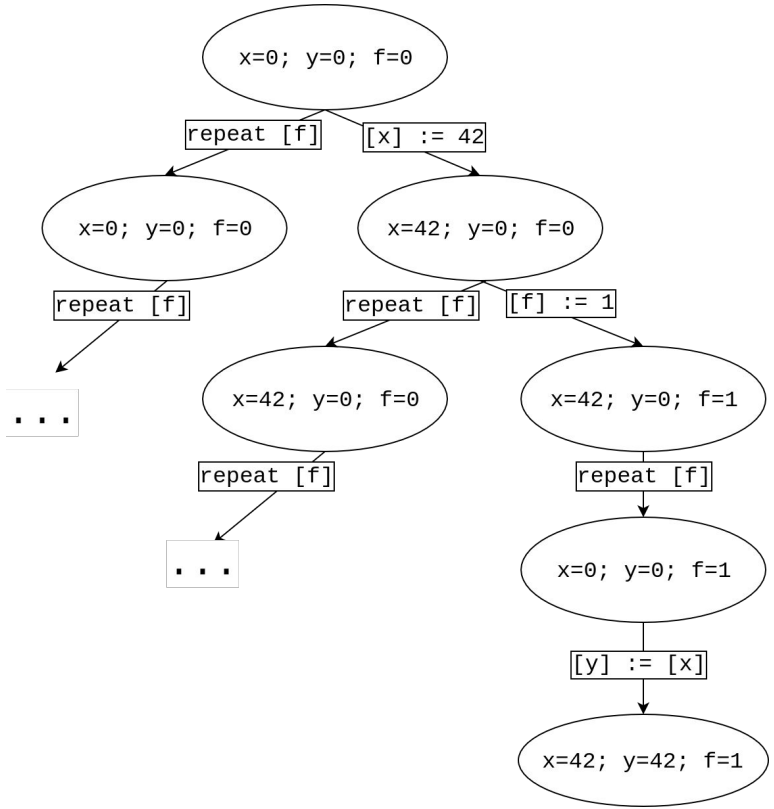
Transition Labeled System

```
[x] := 1;  ||  [z] := [x];
[y] := 1;  ||
```



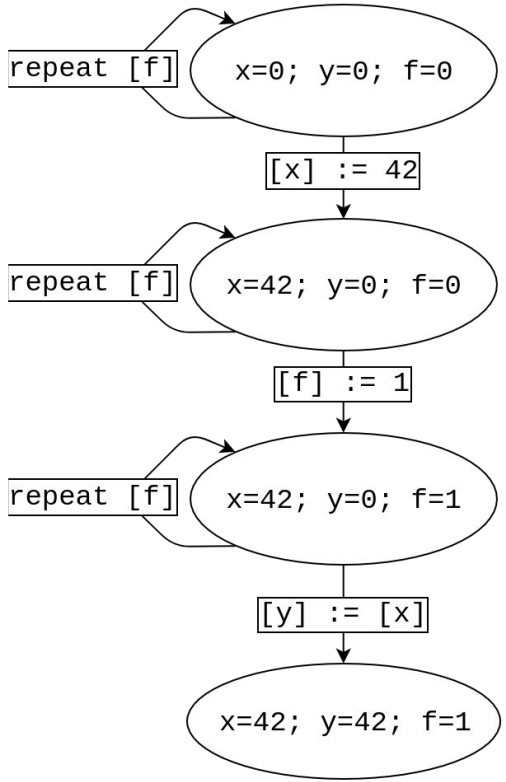
Memoization

```
[x] := 42;  ||  repeat [f];
[f] := 1;   ||  [y] := [x];
```



Memoization

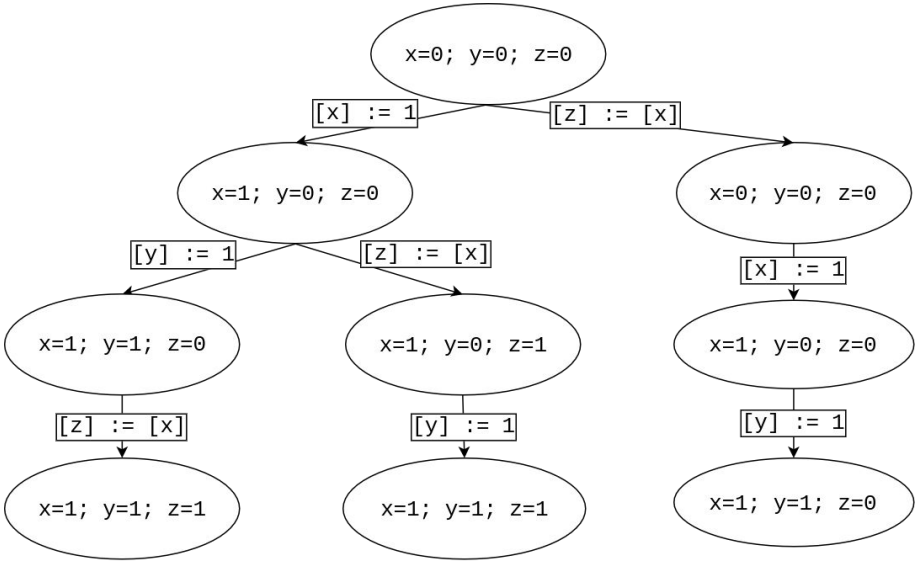
```
[x] := 42;  ||  repeat [f];
[f] := 1;   ||  [y] := [x];
```



Angelic Execution

```
Input: x=0; y=0; z=0
```

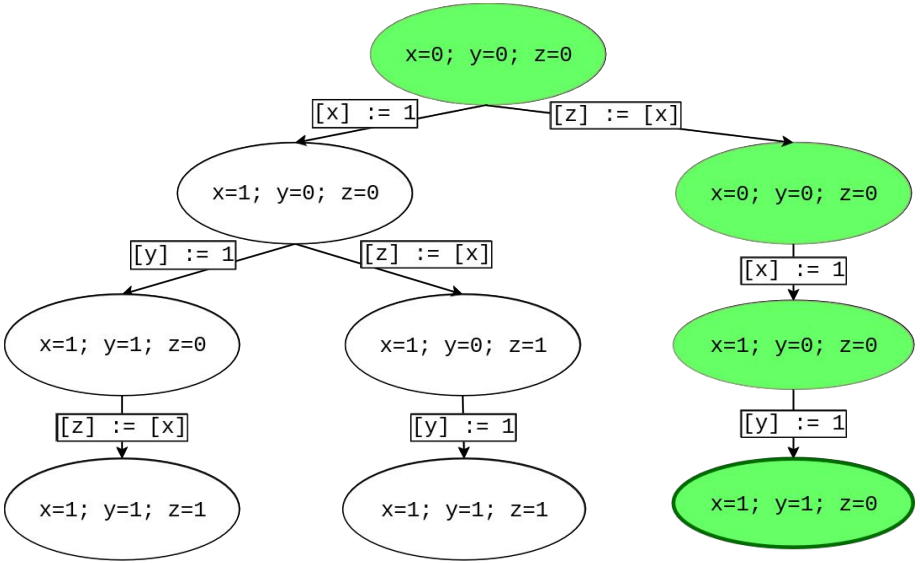
```
Exists output s.t. z=0 ?
```



Angelic Execution

Input: x=0; y=0; z=0

Exists output s.t. z=0 ?

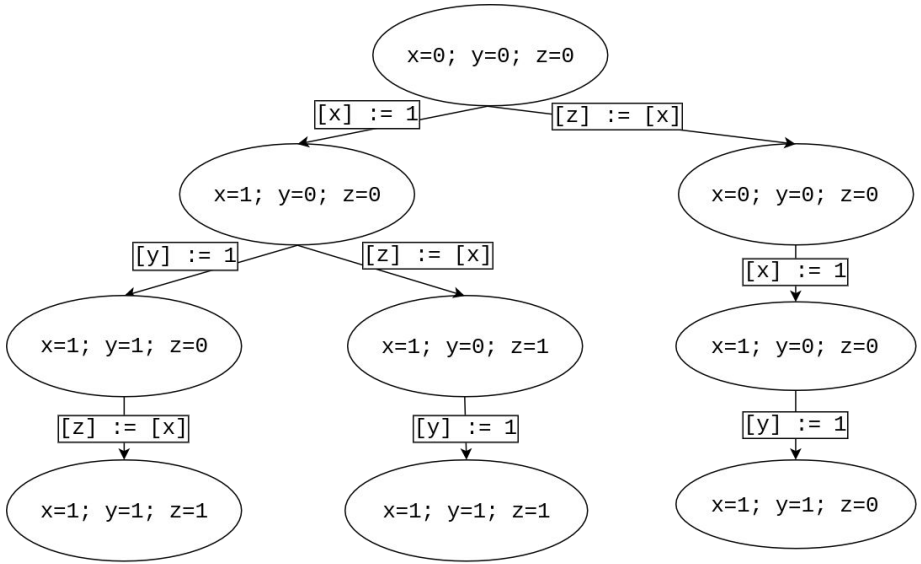


Verification

Input: x=0; y=0; z=0

Forall outputs z=0 ?

Not Exists output s.t. z!=0 ?

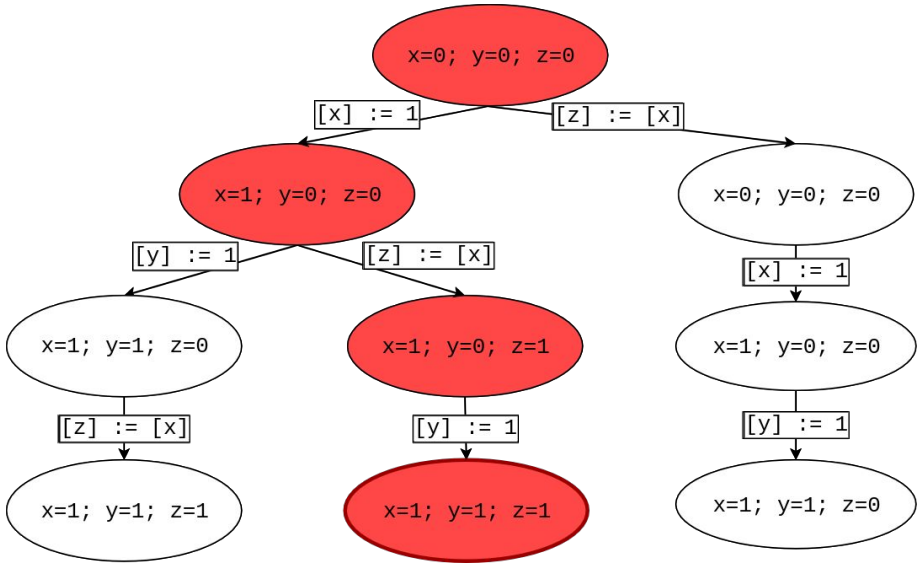


Verification

Input: x=0; y=0; z=0

Forall outputs z=0 ?

Not Exists output s.t. z!=0 ?

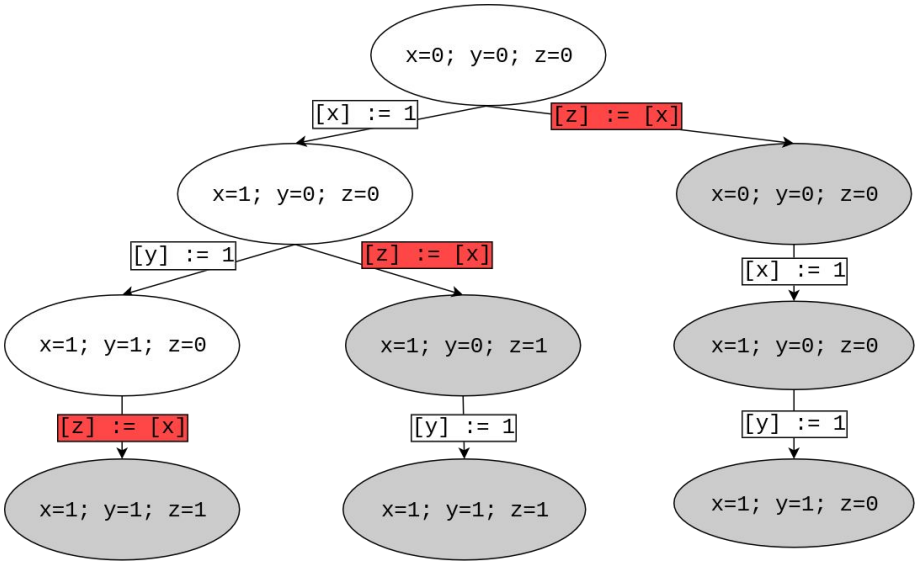


Verification

Input: x=0; y=0; z=0

Forall outputs z=0 ?

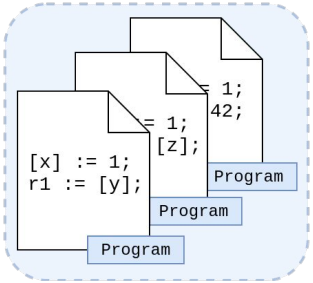
Not Exists output s.t. z!=0 ?



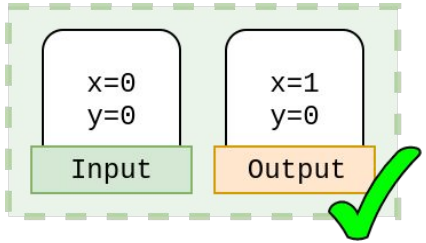
Constructive Negation

- Checks that some state is unreachable
- Otherwise provides counterexample or constraints on program
- Allows to express negative examples

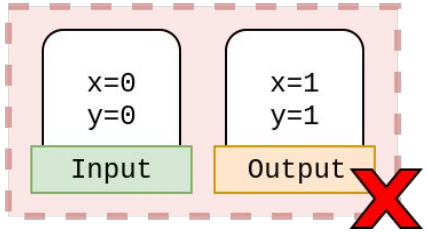
Programming By Examples



- Space of candidate programs
  - usually defined by the grammar



- Set of positive examples



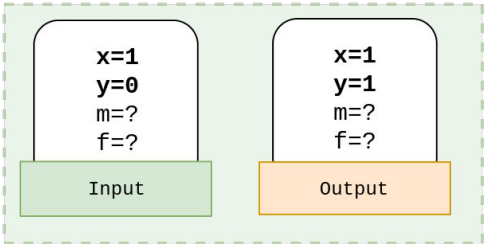
- Set of negative examples

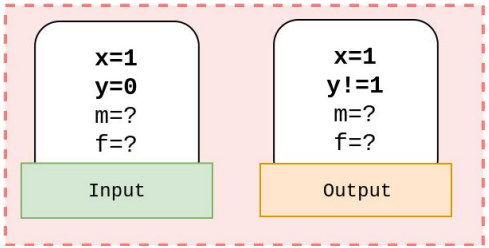
Programming By Examples: Message Passing

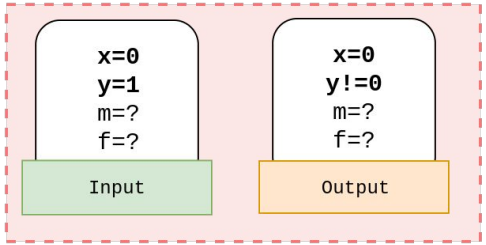
```
[m] := [x];  ||  {??};
{??};        ||  [y] := [m];
```



```
[m] := [x];  ||  repeat [f];
[f] := 1     ||  [y] := [m];
```







Synthesis Time

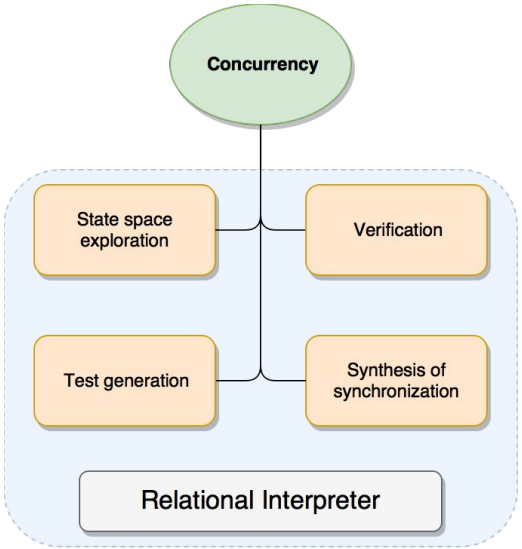
Name	#instructions	Time (s)
message passing 2 threads	4	0.39
consensus 2 threads	10	1.10
Dekker’s like 2 threads	10	15.88

Limitations and Future Work

- State space explosion
  - Abstract interpretation and Abstraction refinement
- Synthesis does not scale to big programs
  - More sophisticated algorithms
    - Counterexample guided inductive synthesis

Relational Interpretation of Concurrency

- Relational Programming
  - Declarative non-deterministic computations
- Relational Interpreter
  - Framework for verification and synthesis prototyping



https://github.com/eucpp/relcppmem

https://github.com/dboulytchev/OCanren