Context

Thesis subject

- *Random-based testing of C programs.*

Test case generation

- white-box : we have a program,
- criterion guided : we want to cover some elements in the program (eg. : all instructions),
- random : we draw paths at random in the program wrt. some probability distribution.

Why random ?

- exhaustively
  - all paths are chosen,
  - what happens if there are infinitely many paths ?
- deterministically
  - the test-set is an arbitrary solution-set,
  - what happens if there exists another interesting solution-set ?
- randomly : any path has a "fair" chance to be chosen.

In this thesis : structural biased random testing

- basis : control flow graph (CFG),
- elements : *complete* paths in the CFG,
- **but all paths in the CFG are not feasible !**

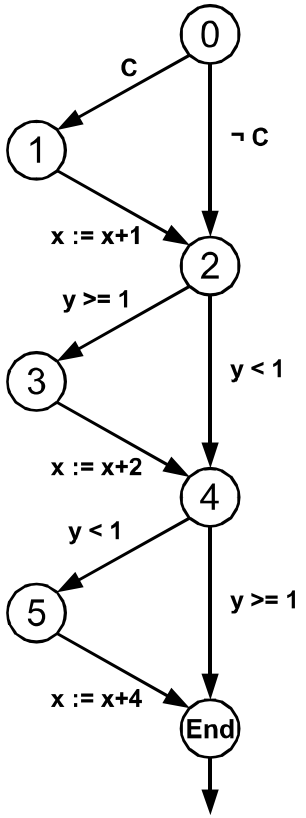
In this talk : how to eliminate infeasible paths from the CFG ?

Infeasible paths in white-box testing

"Path-based" white-box testing

- select a path,
- compute path predicate and check for feasibility using constraint solvers :
  - Ok : produce suitable input values,
  - Ko : select a new path.

```
0  if (C)
1      x := x+1;
2  if (y >= 1)
3      x := x+2;
4  if (y < 1)
5      x := x+4;
```

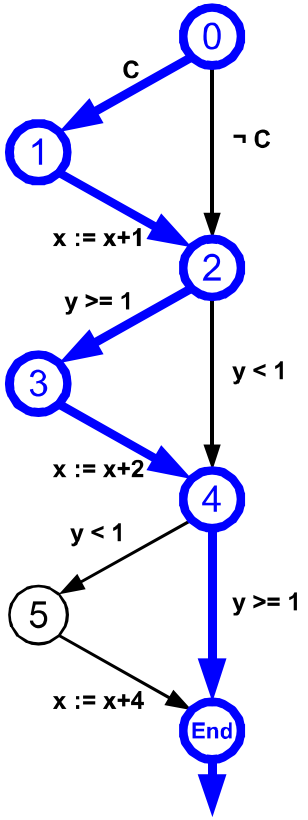


Infeasible paths in white-box testing

"Path-based" white-box testing

- select a path,
- compute path predicate and check for feasibility using constraint solvers :
  - Ok : produce suitable input values,
  - Ko : select a new path.

```
0  if (C)
1      x := x+1;
2  if (y >= 1)
3      x := x+2;
4  if (y < 1)
5      x := x+4;
```

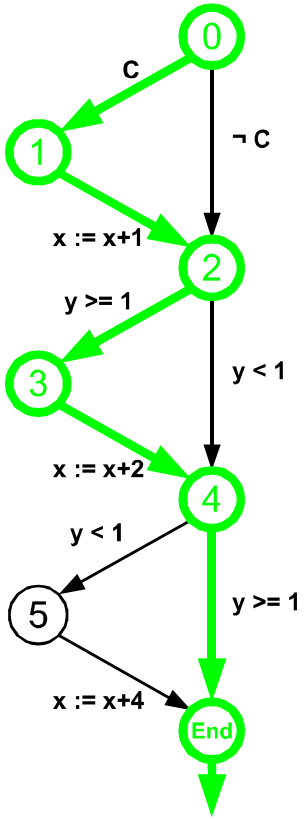


Infeasible paths in white-box testing

"Path-based" white-box testing

- select a path,
- compute path predicate and check for feasibility using constraint solvers :
  - Ok : produce suitable input values,
  - Ko : select a new path.

```
0  if (C)
1      x := x+1;
2  if (y >= 1)
3      x := x+2;
4  if (y < 1)
5      x := x+4;
```



Path predicate

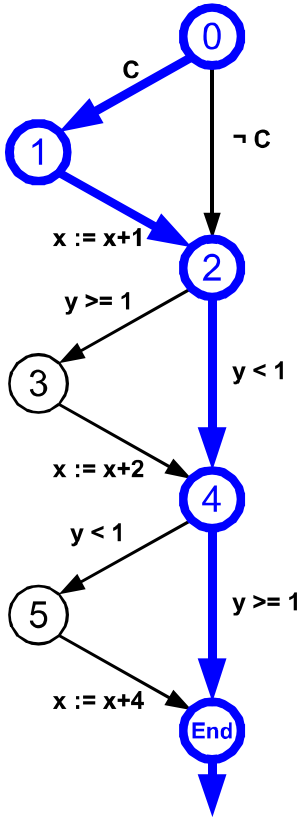
$C \wedge y \geq 1 \wedge y \geq 1 \rightarrow$ SAT.

Infeasible paths in white-box testing

"Path-based" white-box testing

- select a path,
- compute path predicate and check for feasibility using constraint solvers :
  - Ok : produce suitable input values,
  - Ko : select a new path.

```
0  if (C)
1      x := x+1;
2  if (y >= 1)
3      x := x+2;
4  if (y < 1)
5      x := x+4;
```

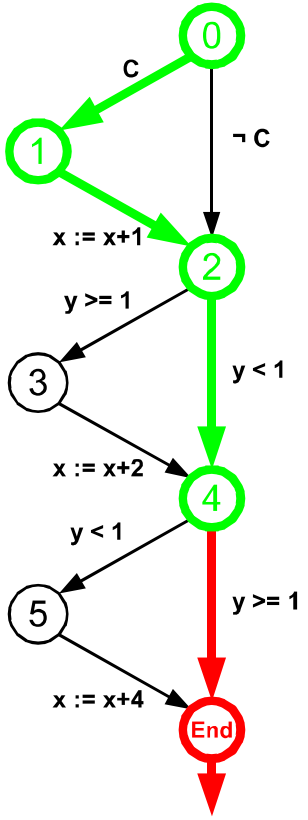


Infeasible paths in white-box testing

"Path-based" white-box testing

- select a path,
- compute path predicate and check for feasibility using constraint solvers :
  - Ok : produce suitable input values,
  - Ko : select a new path.

```
0  if (C)
1      x := x+1;
2  if (y >= 1)
3      x := x+2;
4  if (y < 1)
5      x := x+4;
```



Path predicate

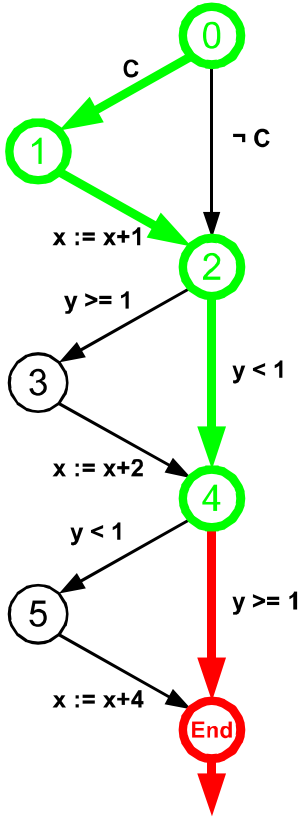
$C \wedge y \geq 1 \wedge y < 1 \rightarrow$ UNSAT.

Infeasible paths in white-box testing

"Path-based" white-box testing

- select a path,
- compute path predicate and check for feasibility using constraint solvers :
  - Ok : produce suitable input values,
  - Ko : select a new path.

```
0  if (C)
1      x := x+1;
2  if (y >= 1)
3      x := x+2;
4  if (y < 1)
5      x := x+4;
```



Path predicate

$C \wedge y \geq 1 \wedge y < 1 \rightarrow$ UNSAT.

Infeasible paths : don't correspond to an actual execution, ie. no suitable input values.

Infeasible paths detection

In practice : a big problem for testing real programs.

Idea

- detect sets of infeasible paths,
- build a new CFG of the program where infeasible paths have been pruned,
- draw paths in the new CFG.

Infeasible paths detection

- $\Longleftrightarrow$ path predicate satisfiability,
- undecidable when the logic is undecidable,
- solution : use of heuristics, ie. trading accuracy for effectiveness.

Common to many disciplines

- model-checking,
- WCET estimation,
- static analysis,
- etc.