
cea

list

Test Generation Strategies to Measure Worst-Case Execution Time

Nicky Williams, Muriel Roger



Worst Case Execution Time

list

Needed to schedule real-time software

Difficult to find even for sequential uninterrupted module

Naive solution :

sum execution times instructions longest path(s)

but:

software more and more complex, number of paths \uparrow

anticipation mechanisms microprocessor architectures

(data cache, branch prediction,..) \Rightarrow execution time

depends on context, cache miss \gg simple instruction



WCET : two alternative approaches

1. Static analysis :

complex

specific to one microprocessor

manufacturers may not divulge algorithms

2. Measurement (on target platform or simulator) :

which input values ?

PathCrawler generates inputs for path coverage

Pathcrawler (ASE'04, EDCC'05) : concolic-type tool generating input data for path coverage of ANSI C code

No approximation (unlike CUTE, PeX,..) :

100% feasible path coverage if

- no constraint resolution timeout
- absence constructions not treated yet (pointer casts)

Uses Constraint Logic Programming: built-in backtrack



list

Measuring the execution time of each path

Measuring execution time of each feasible path in the source code will ensure we measure the longest effective execution time (\sim WCET) if

- Each feasible source code path corresponds to only one binary code path
- Path execution time is the same for all inputs activating the path
- We can put the machine in some initial worst state (cache, branch prediction registers)
- No influence from bus, DRAM refresh...



list

Measuring fewer paths : partial orders

Many real-life modules have too many feasible paths

Don't know the *longest* path but define partial orders depending on hypotheses on anticipation mechanisms which are weak enough to apply in many cases

ex. only difference between 2 paths is that one has more identical loop iterations than the other

ex. only difference between 2 paths is that one takes empty branch of if-then-else



If-then-else partial order

list

Empty ITE branch contains no data references

ITEE = ITE with an empty branch

Partial order : *path P_i longer than path P_j if the only difference between them is that*

P_i replaces at least one empty ITE branch in P_j by a non-empty branch

Restrictions :

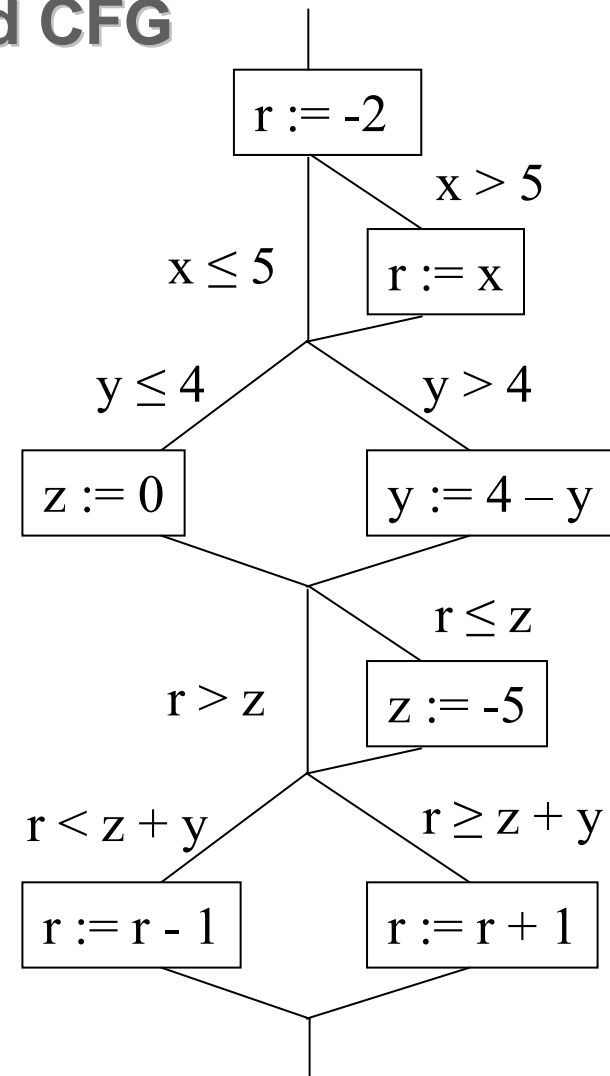
- 1) if data cache, don't apply if a pointer assigned in ITE is dereferenced afterwards
- 2) doesn't take branch prediction into account

Properties of modified strategy

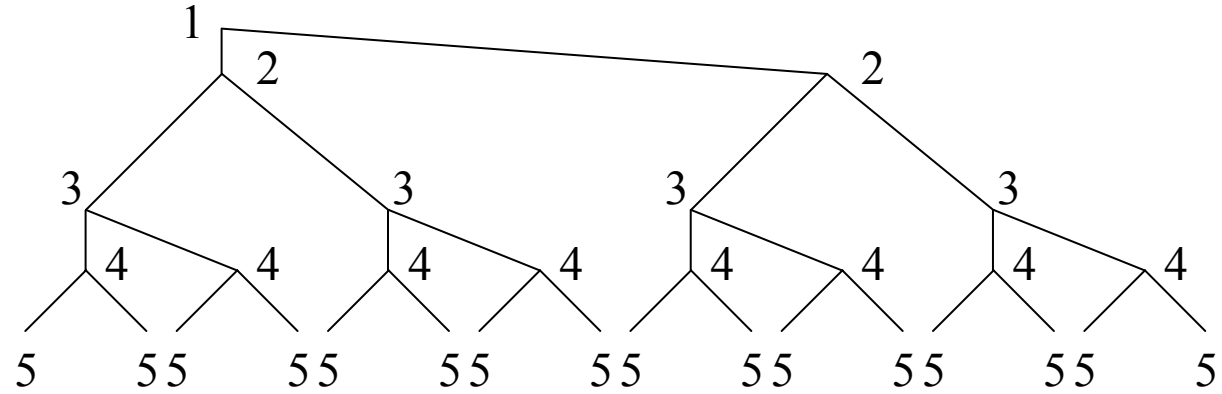
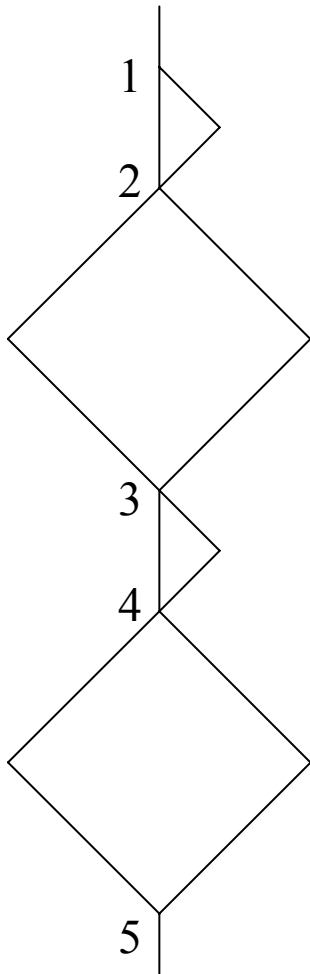
- **Generates inputs to cover all maximal paths**
- **Reduces (or, at worst, does not increase) generation of inputs to cover non-maximal paths**
- **Reduces (or, at worst, does not increase) unnecessary exploration of execution path tree**

Example: source code and CFG

```
f(int x, int y, int z){  
    int r = -2;  
    if (x > 5)  
        r = x;  
    if (y > 4)  
        y = 4 - y;  
    else  
        z = 0;  
    if (r <= z)  
        z = -5;  
    if (r >= (z + y))  
        r = r + 1;  
    else  
        r = r - 1;  
    return r;  
}
```

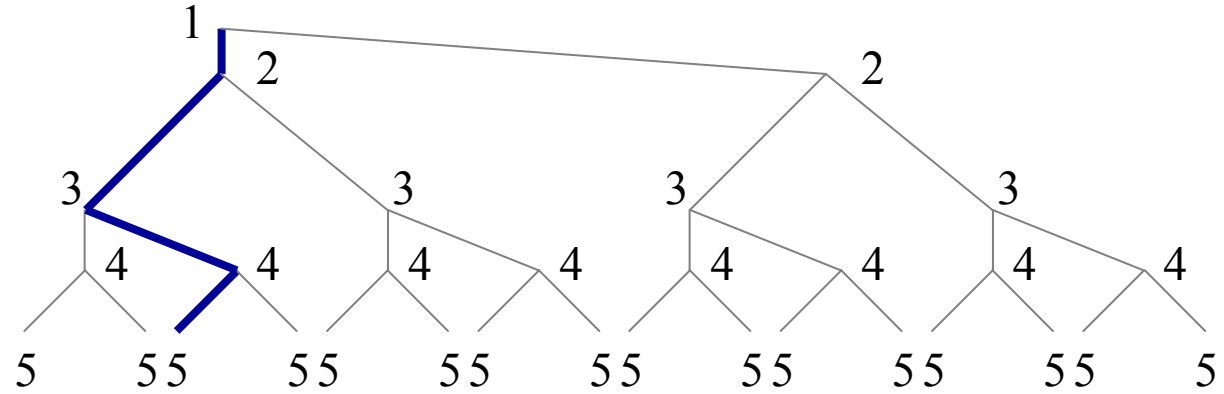
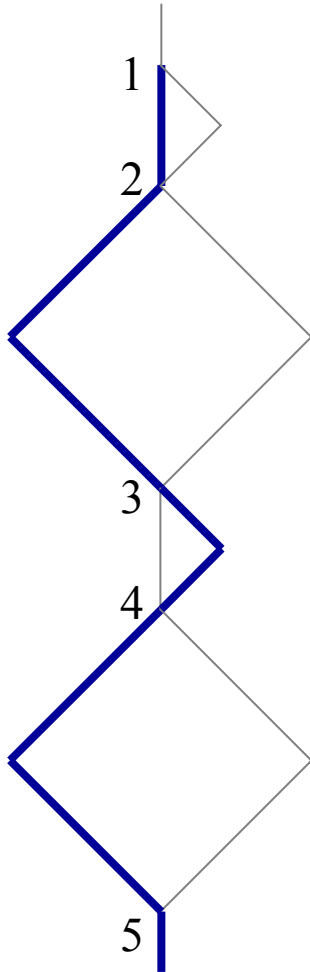


CFG and tree of execution paths

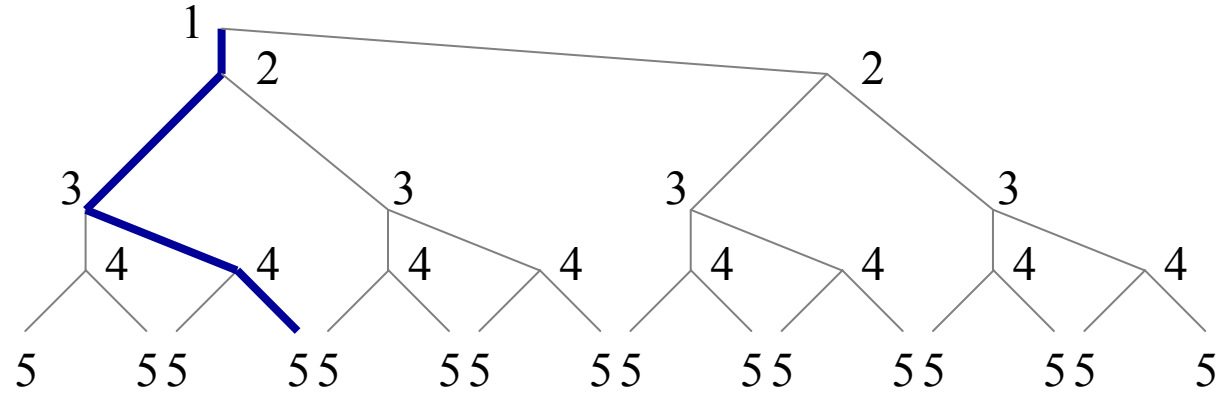
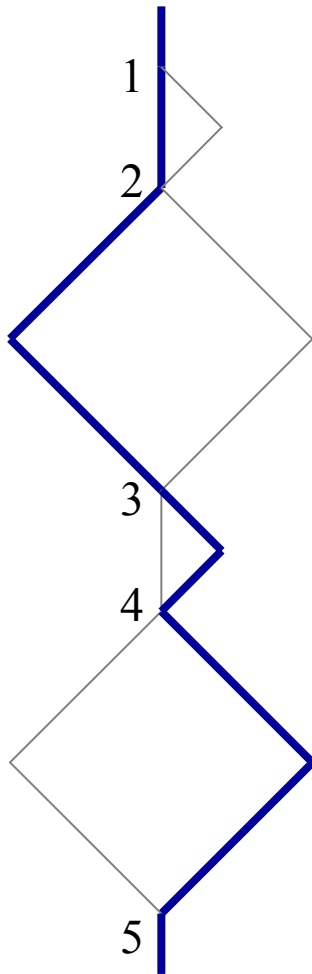


16 paths

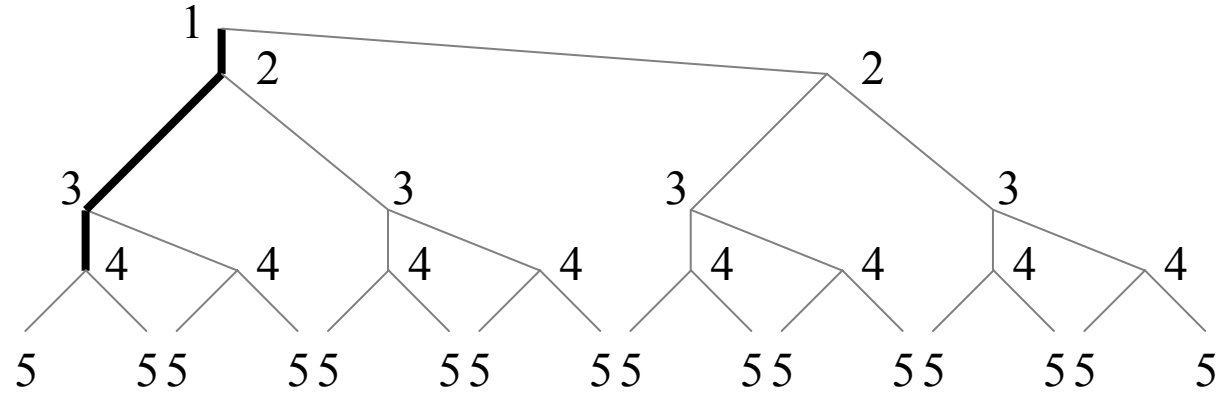
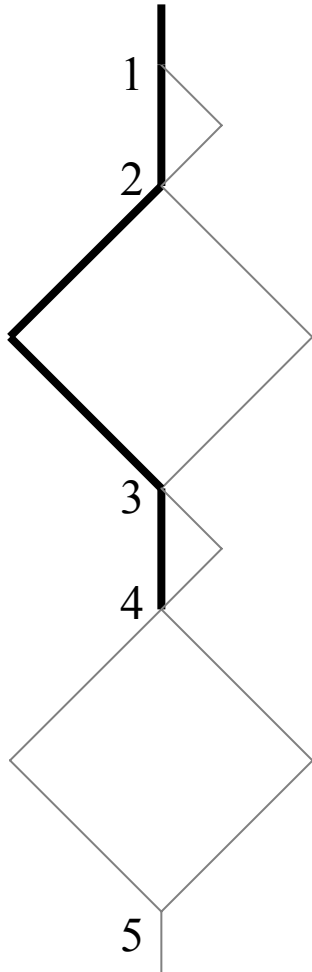
Default strategy : arbitrary 1st path covered



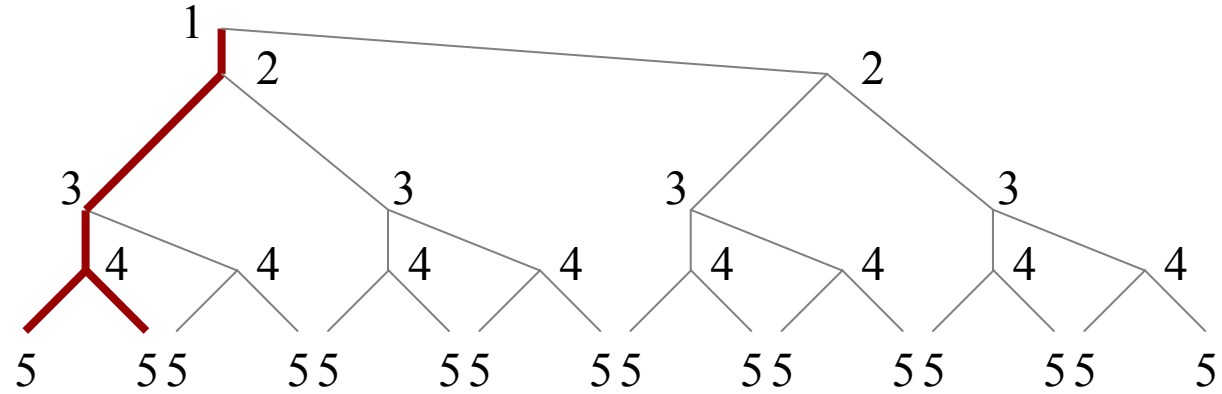
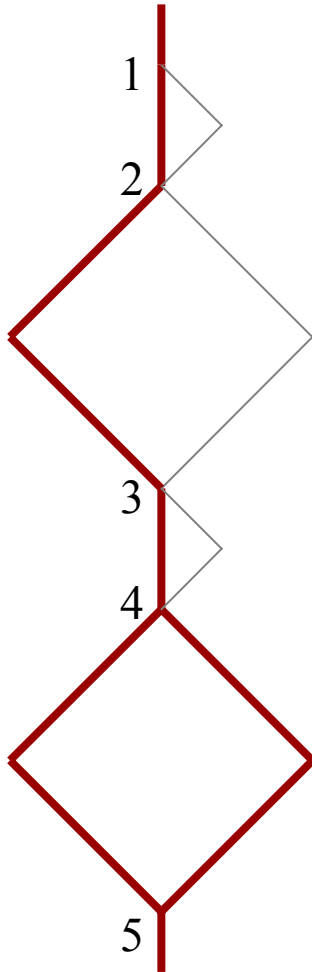
Depth-first search : new path covered



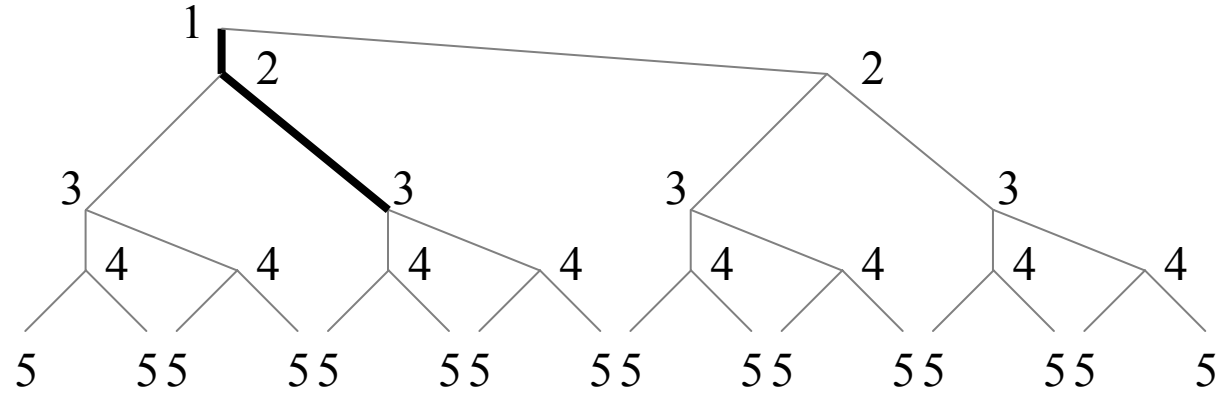
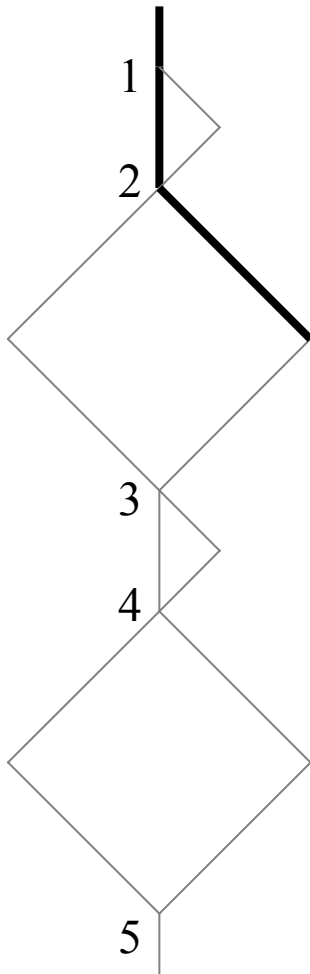
Depth-first search : try to cover partial path



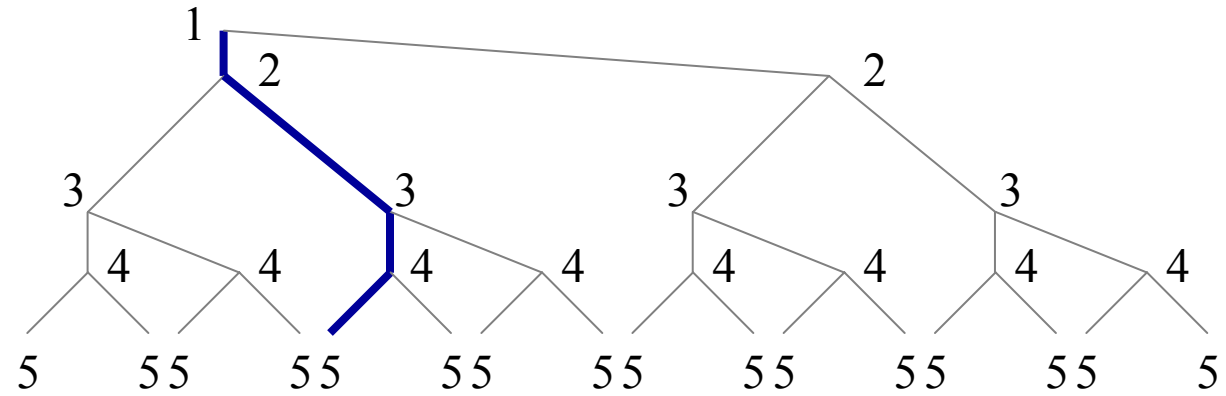
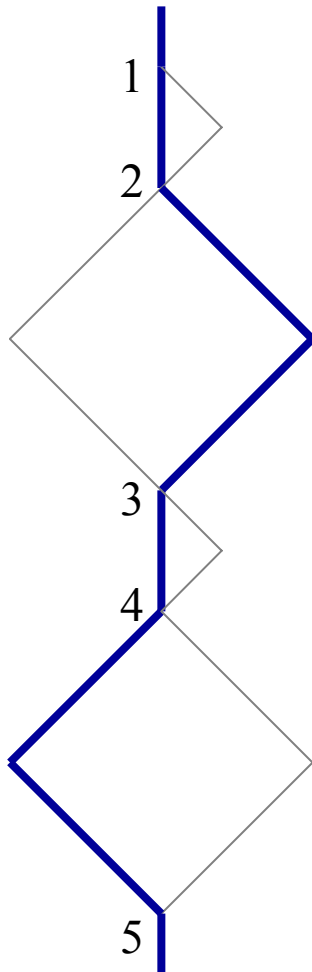
Infeasible partial path : tree pruned



Depth-first search : try to cover partial path



Left-right-non-determinist ...



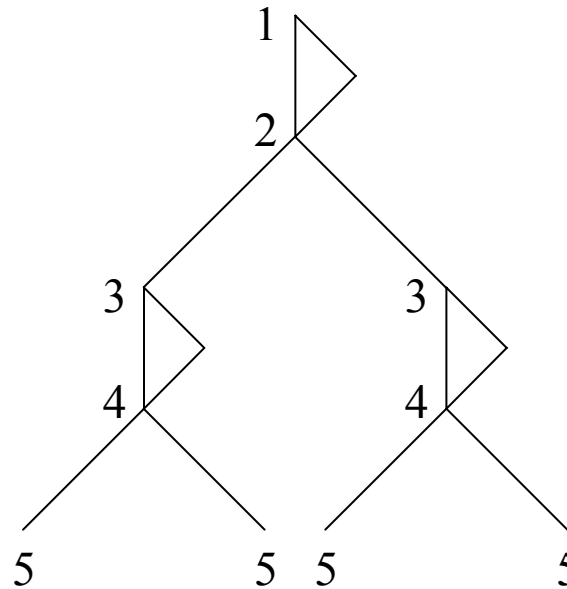
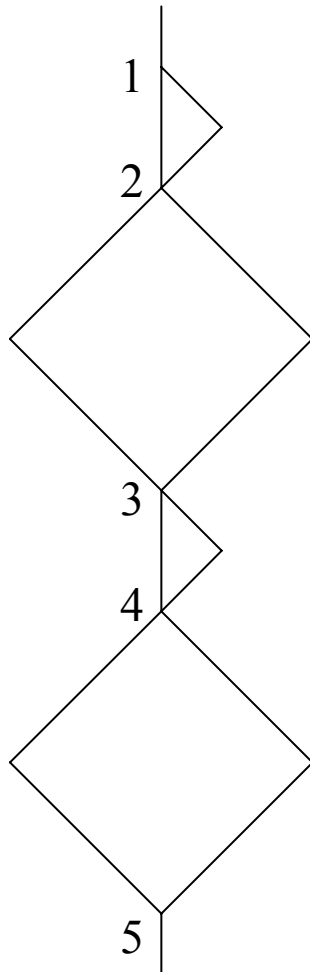
How to modify the strategy for ITE partial order

Goal : optimise generation but cover maximal paths

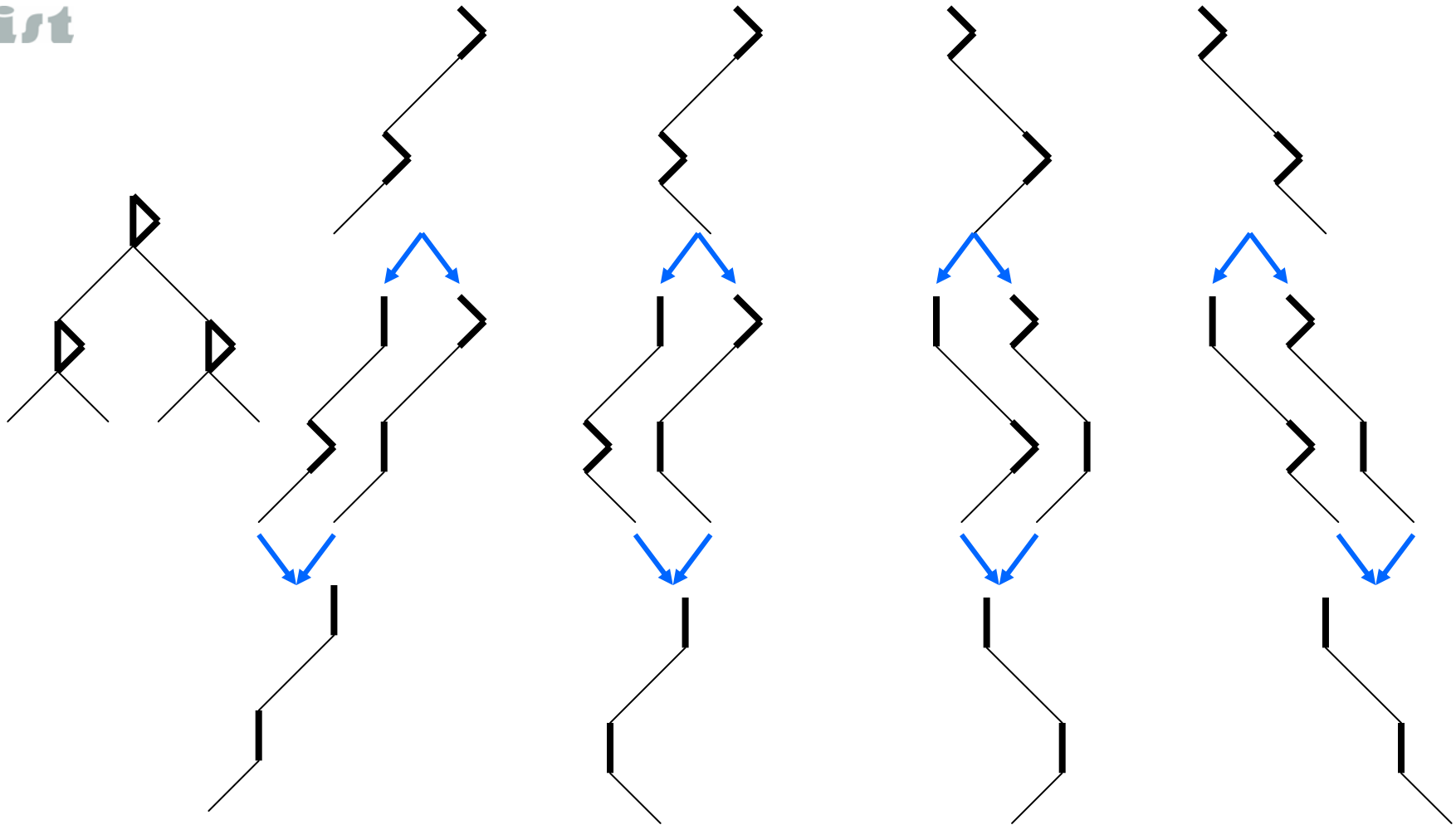
Mechanisms :

- ***Abstract partial path*** = each ITEE replaced by abstract ITEE containing empty and non-empty branches
- ***Memorise*** feasible paths & abstract infeasible partial paths
- DFS modified to explore non-empty ITEE branches first => discover all maximal infeasible partial paths first
- LR non-determinism modified to explore maximal paths: on backtrack, if partial path ends in empty ITE branch then only cover *continuations* whose abstraction was infeasible

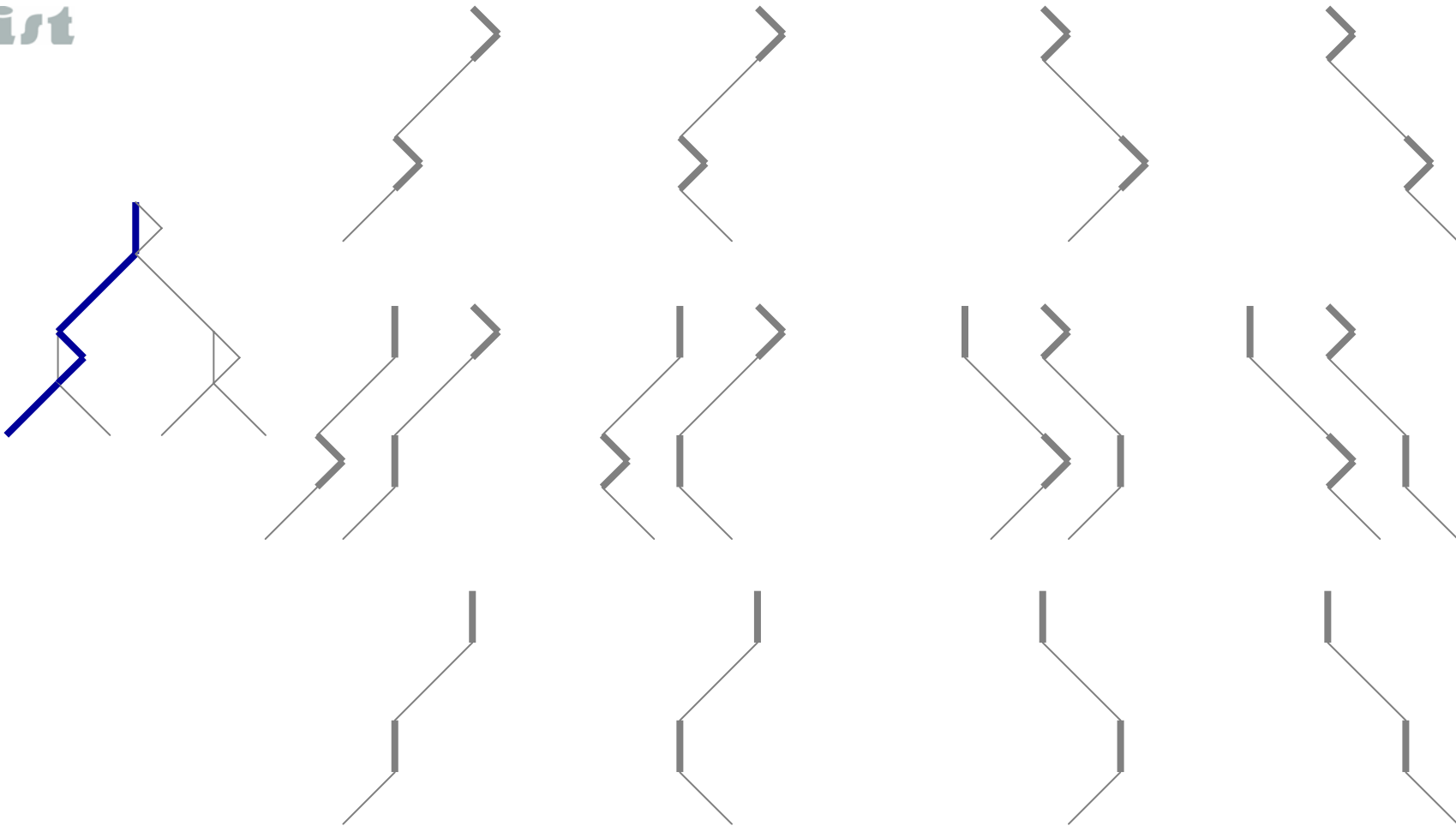
CFG and tree of abstract execution paths



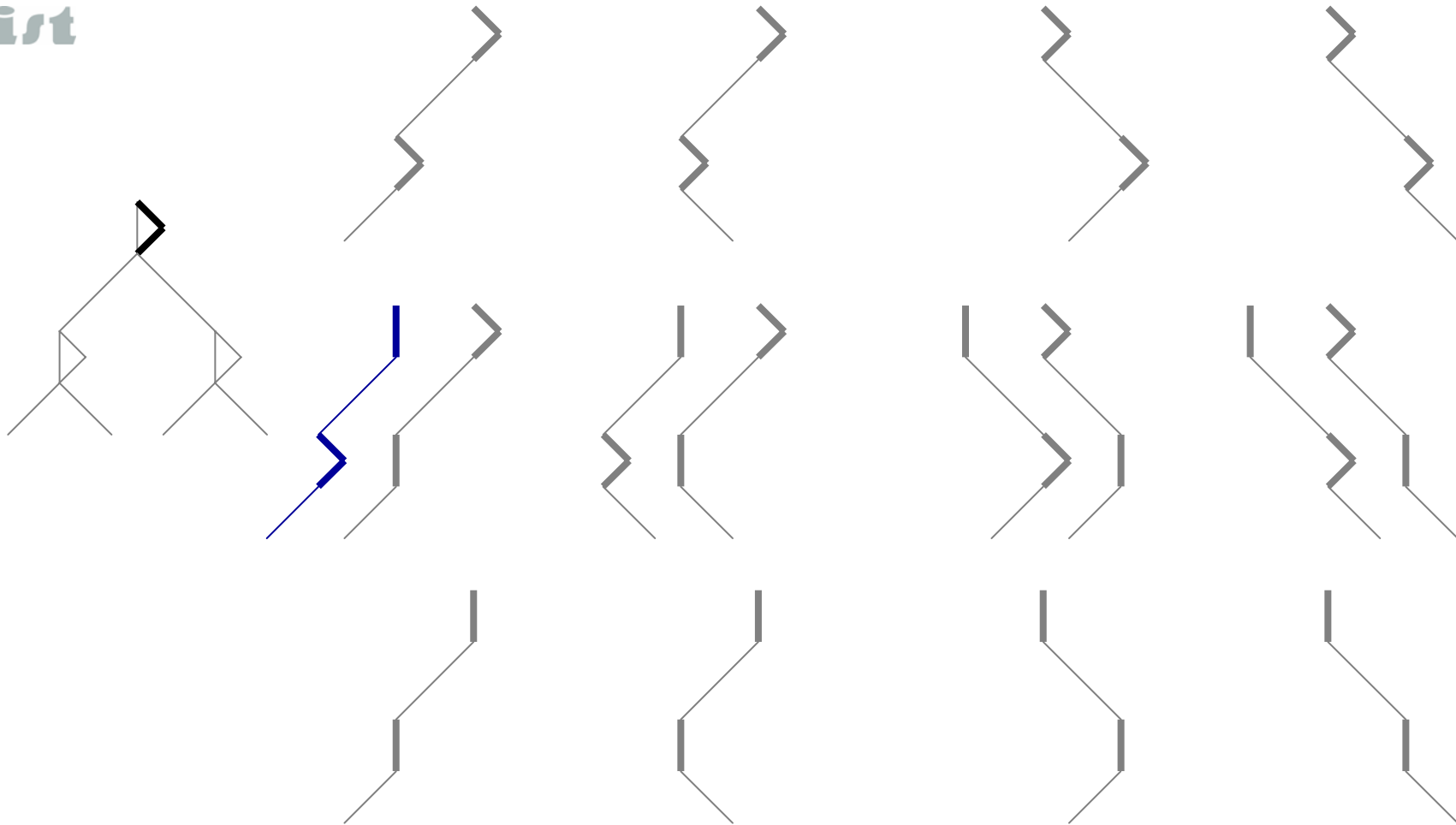
Partial order on execution paths



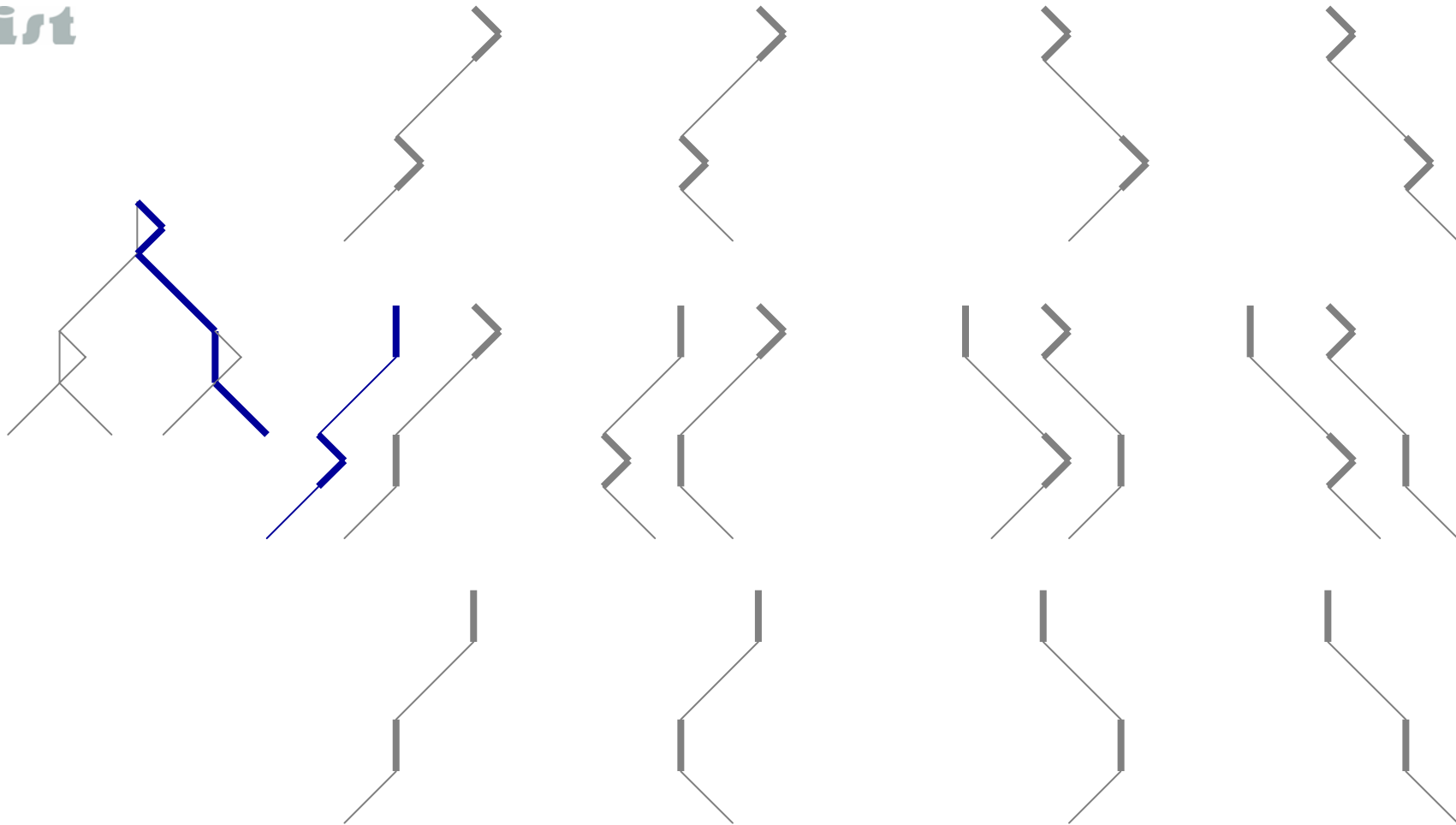
Modified strategy : arbitrary 1st path covered



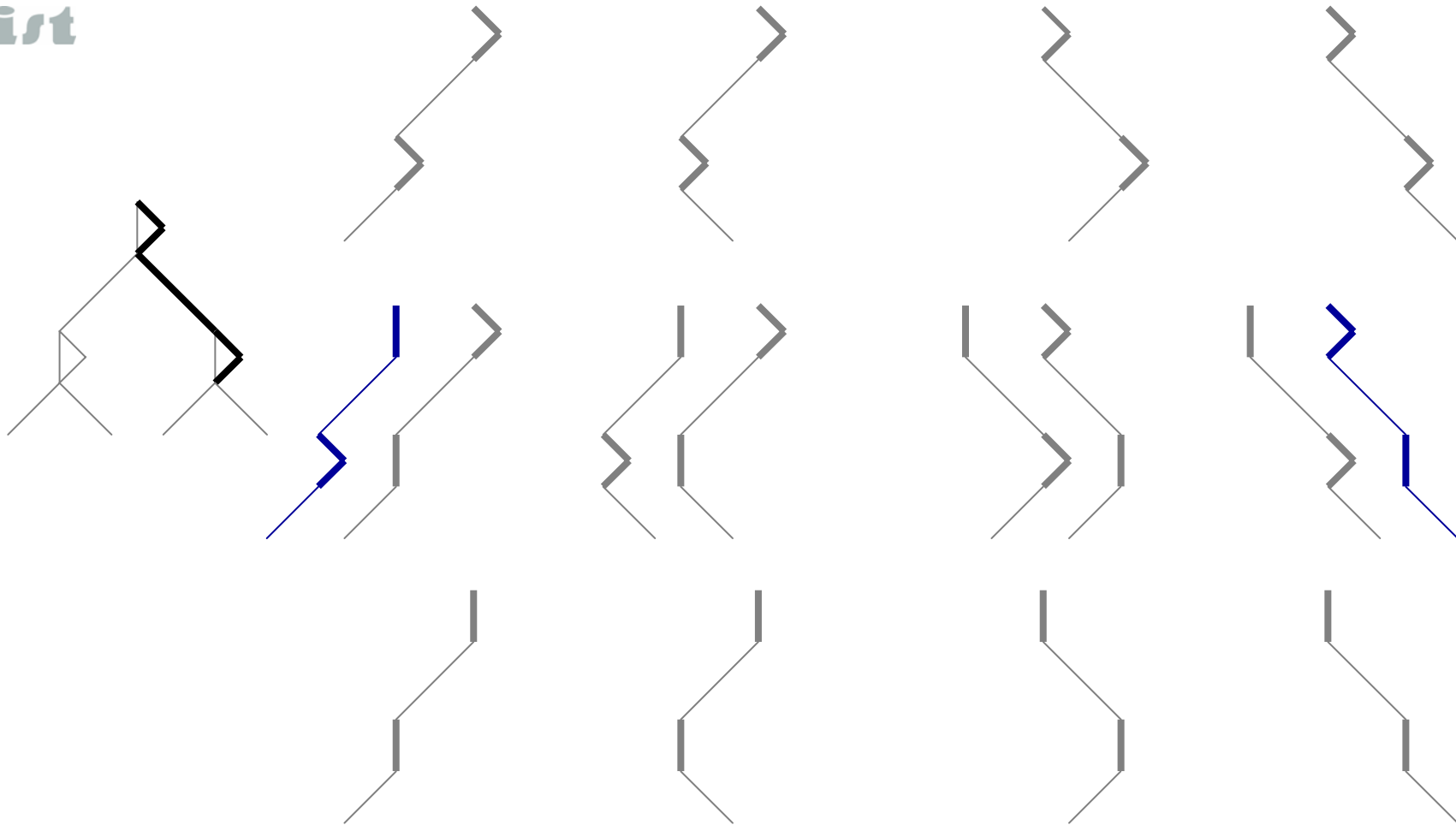
Try to cover 1st non-empty ITEE branch



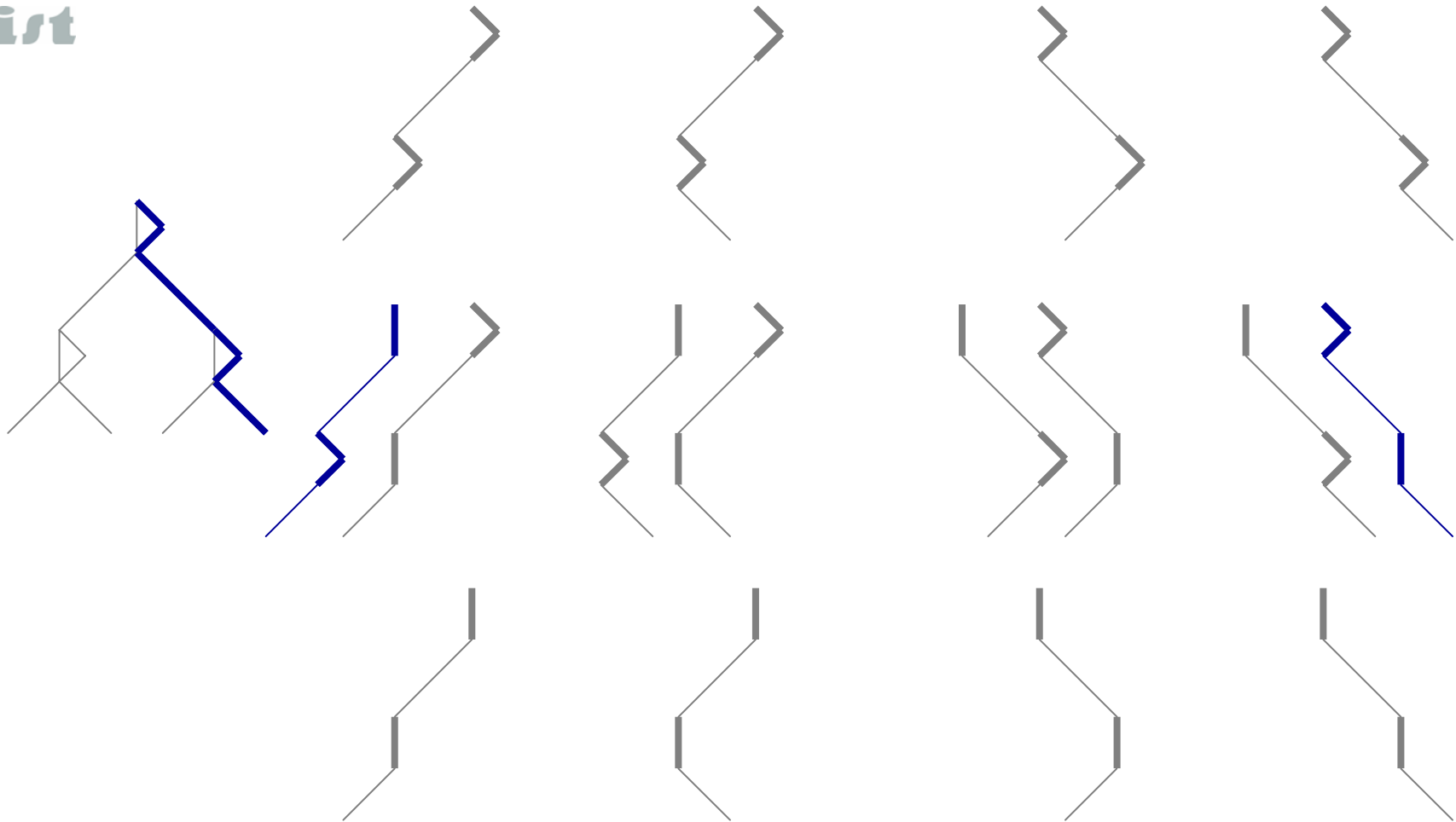
Path covered has 2nd empty ITEE branch



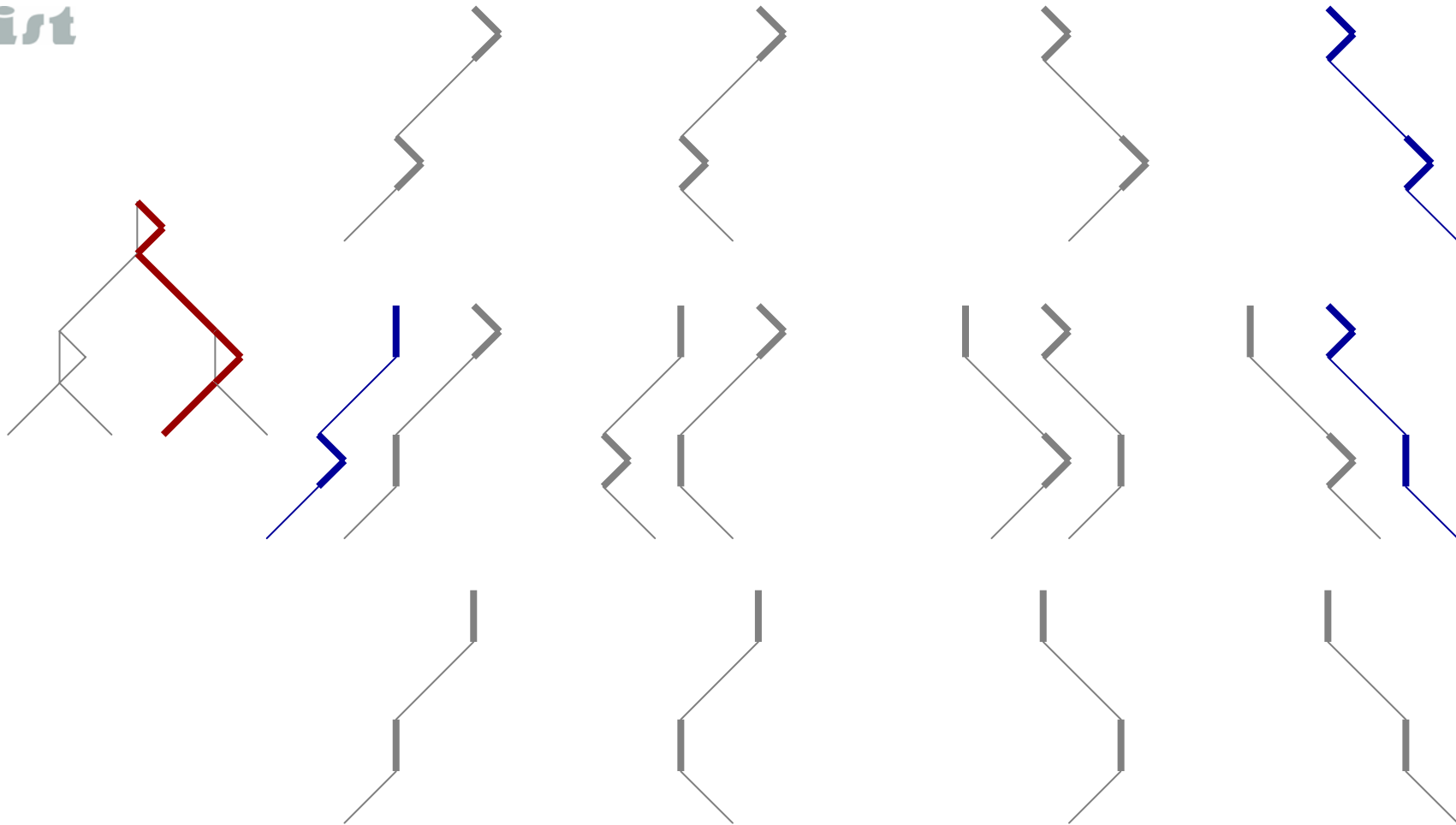
Try to cover 2nd non-empty ITEE branch



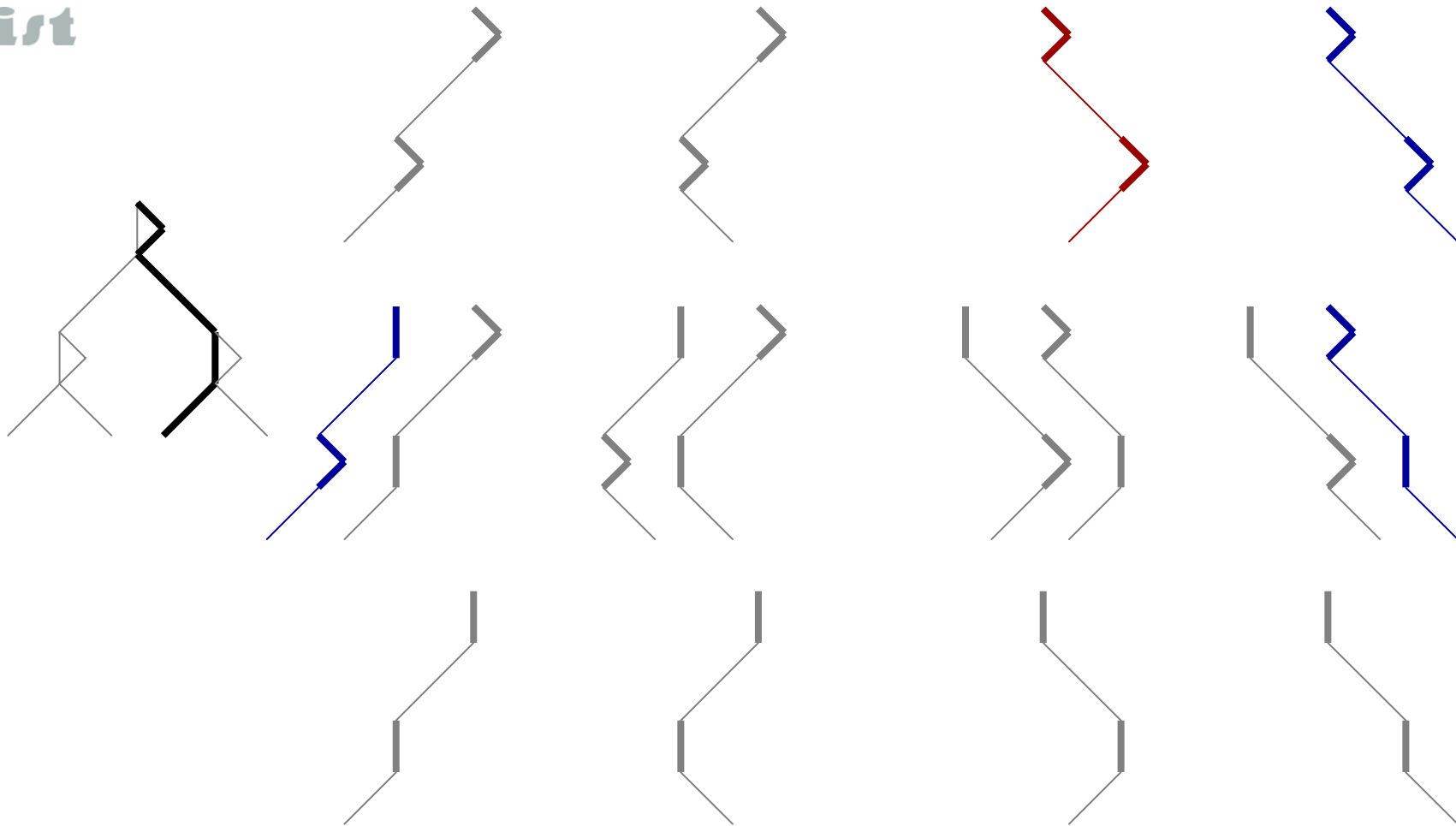
Maximal path covered



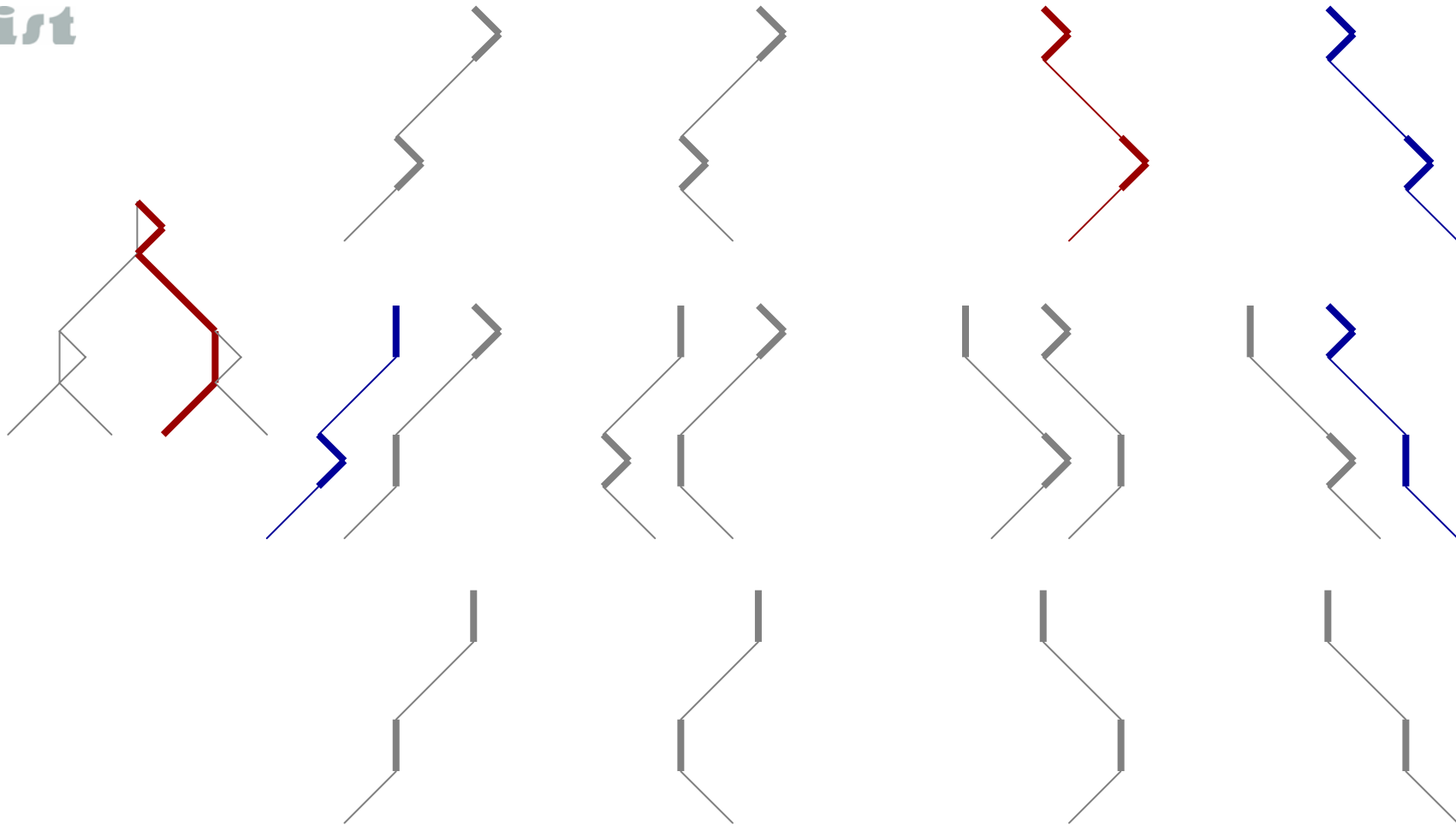
Depth-first search : next maximal path infeasible



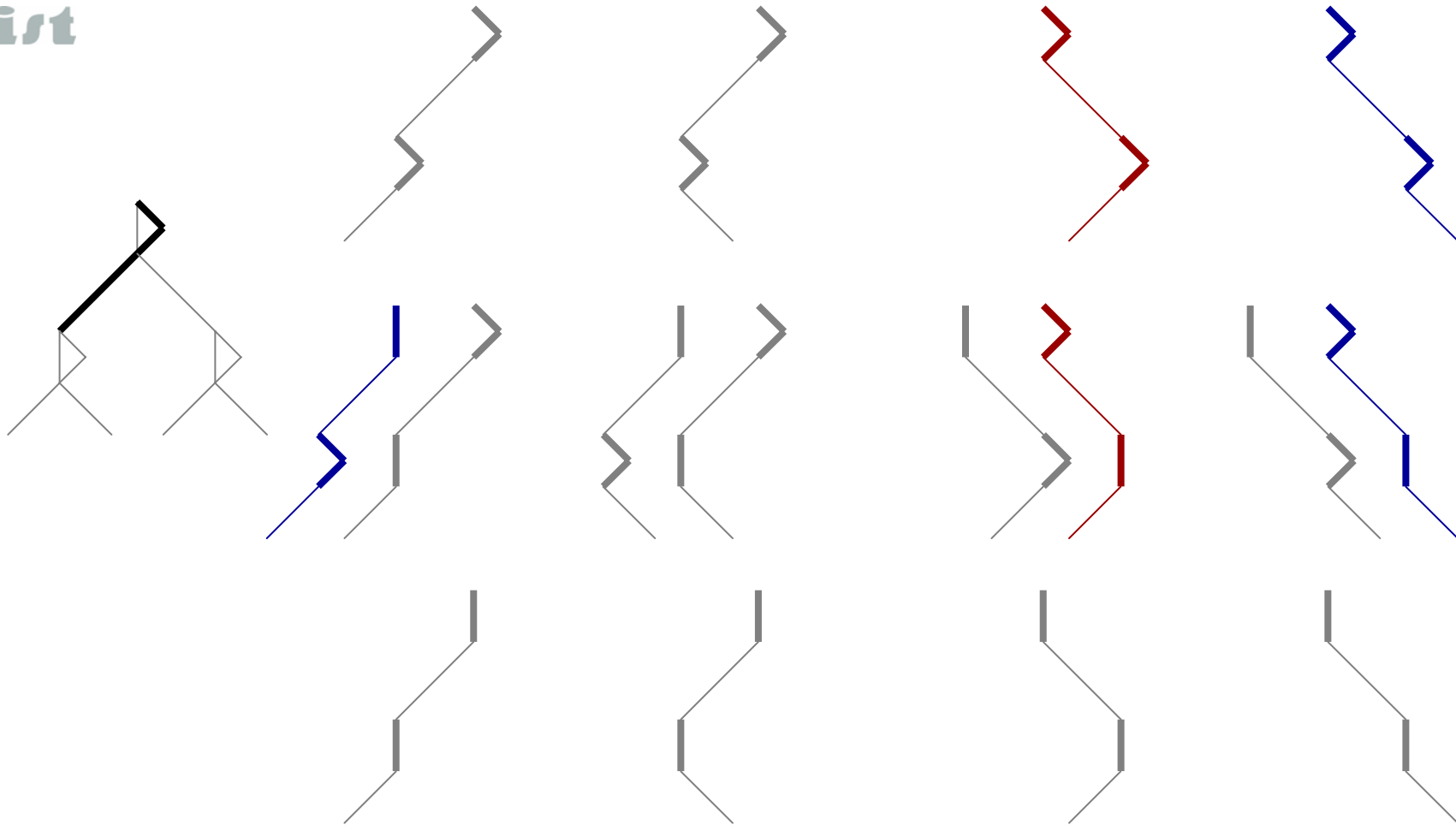
Try uncovered path with 2nd empty ITE branch



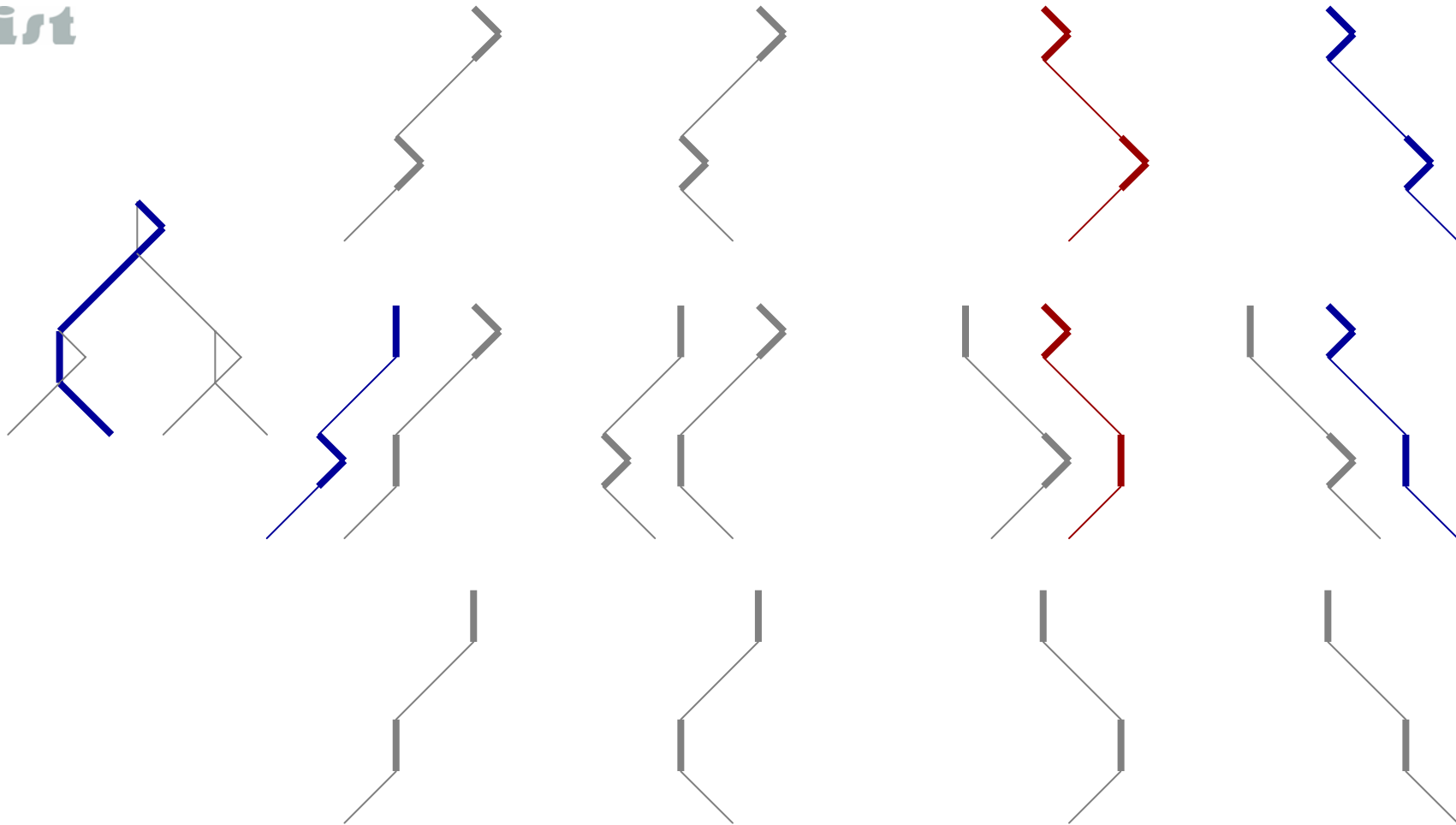
Infeasible with 2nd empty ITE branch too



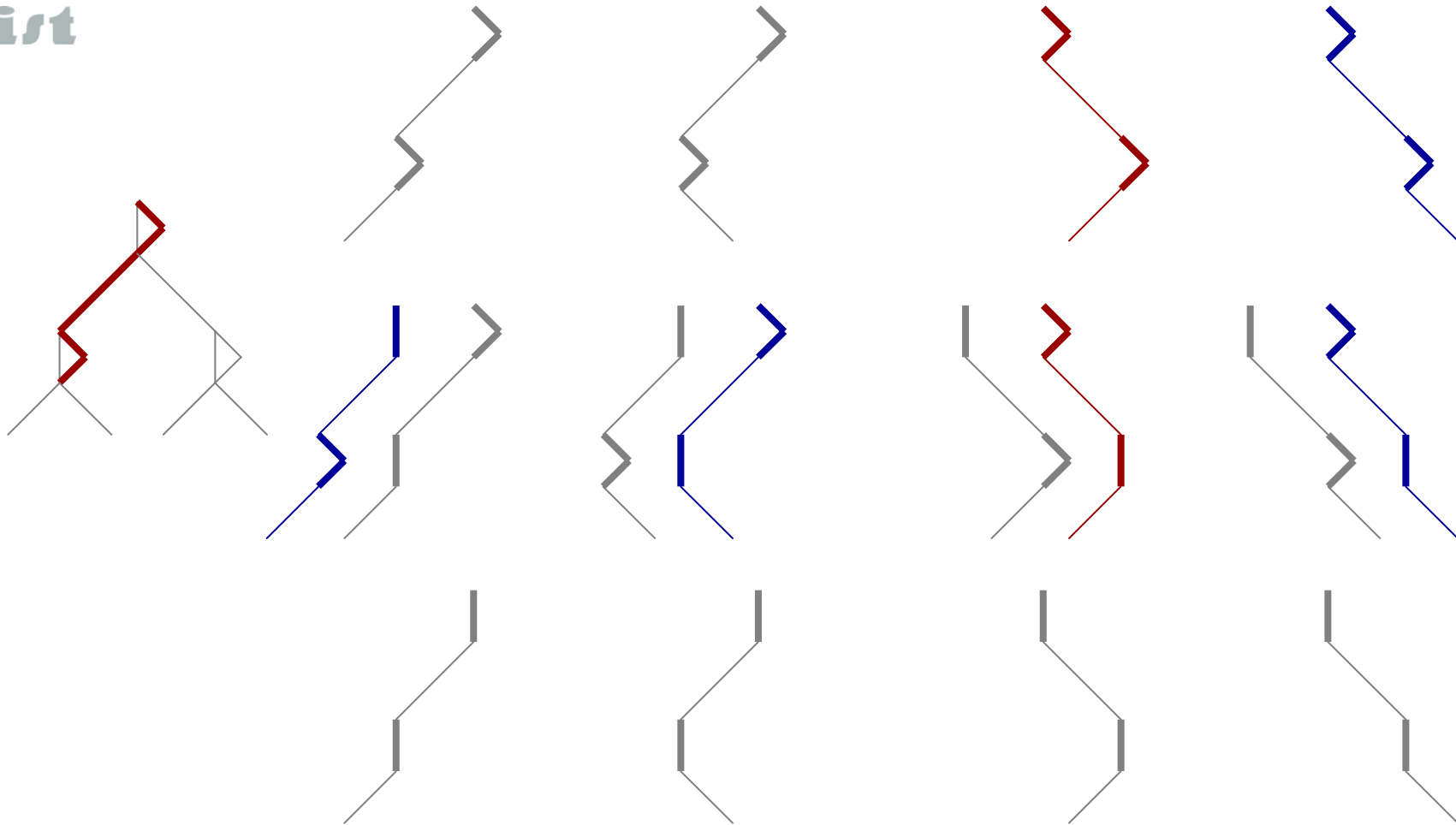
Depth-first search



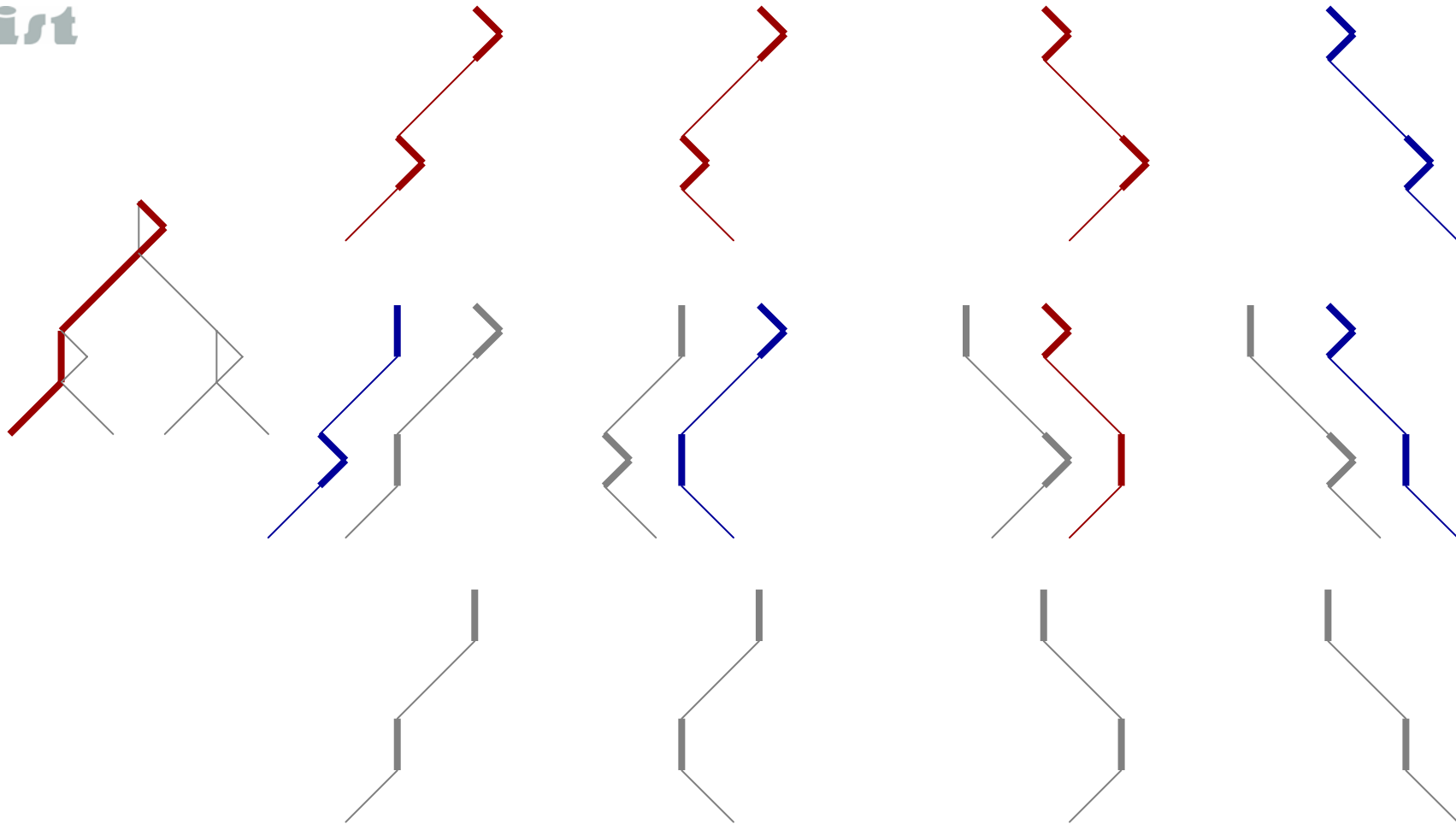
Path covered has 2nd empty ITE branch



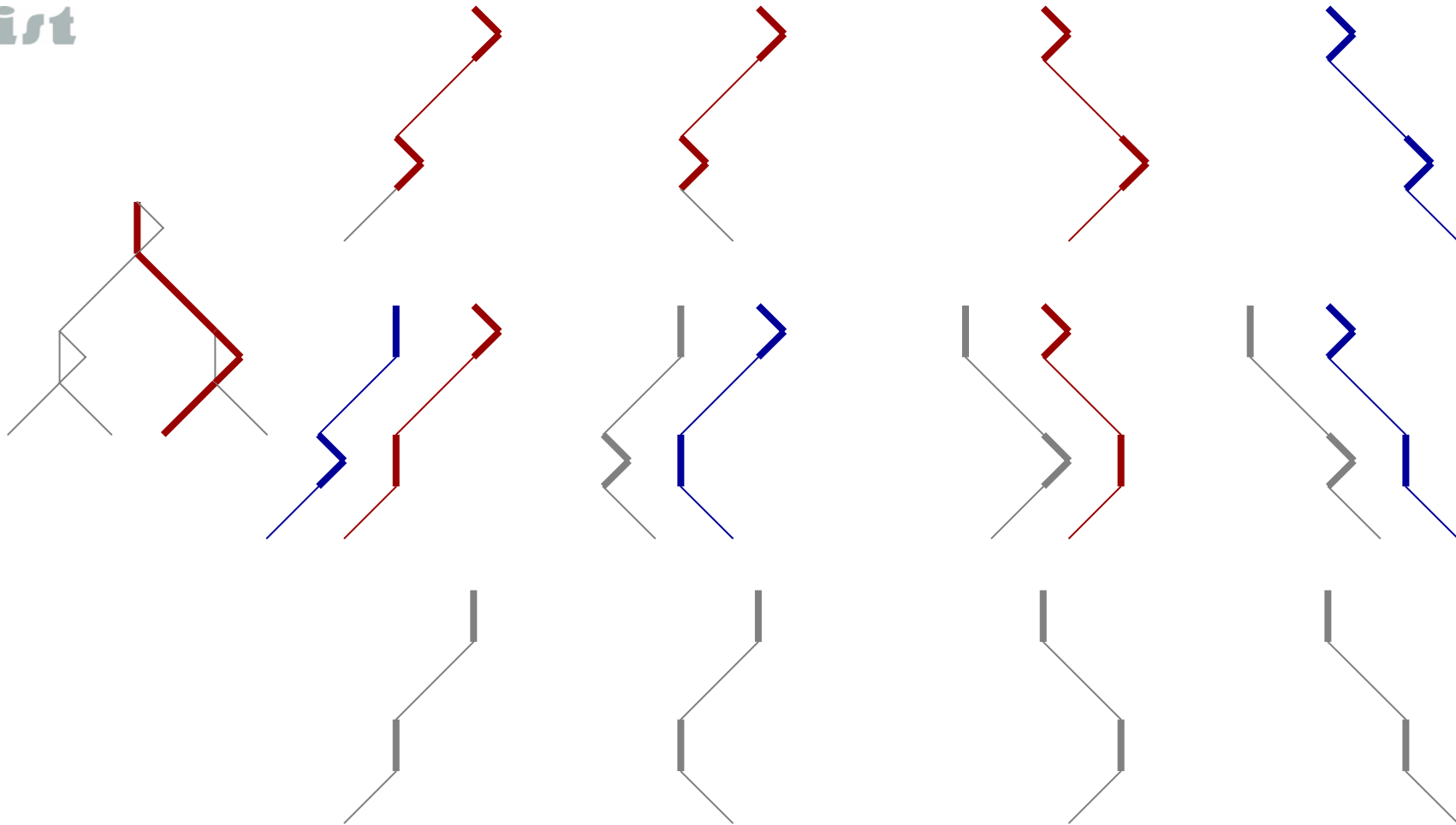
Paths with 2nd non-empty ITEE branch infeasible



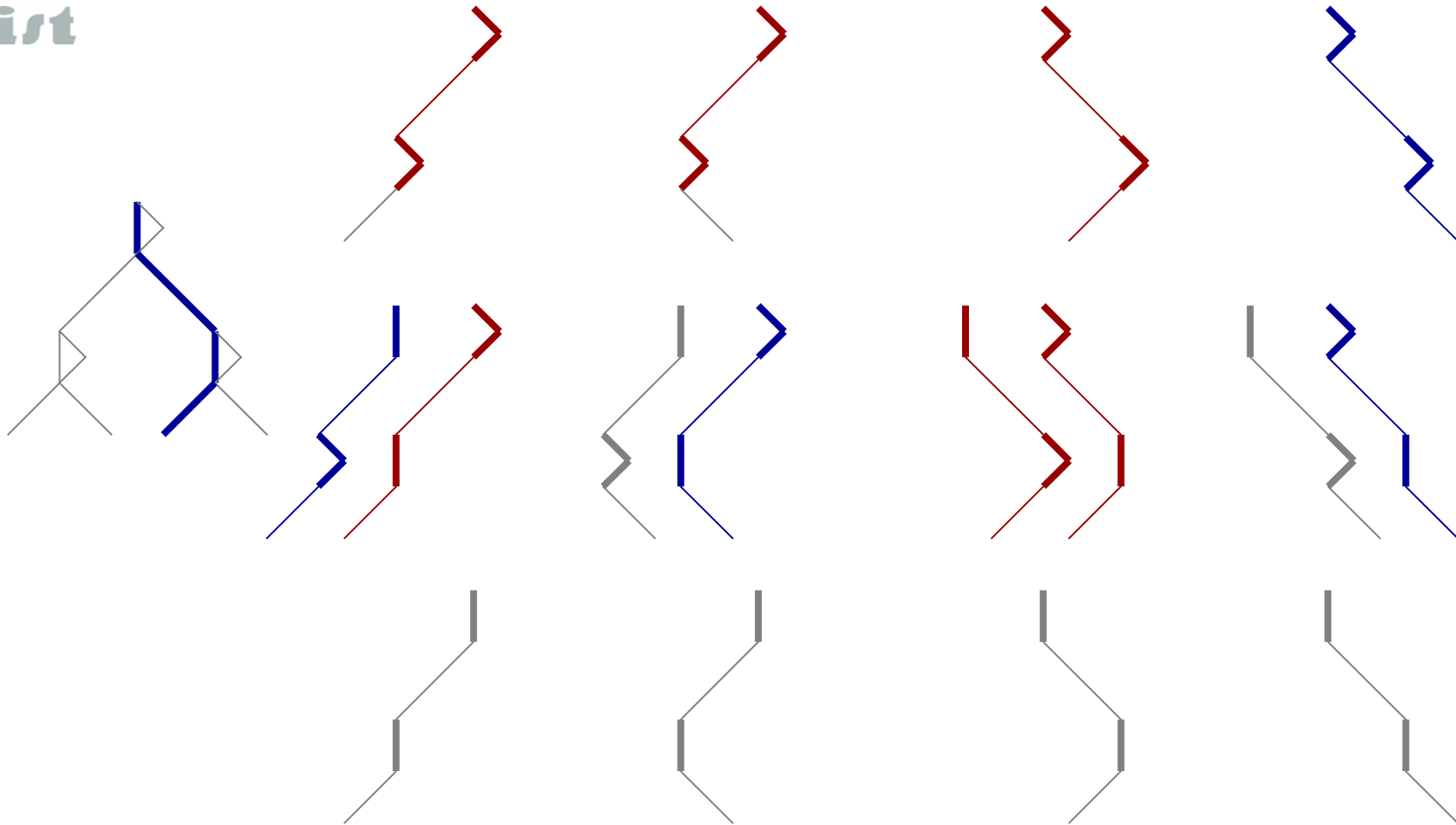
Depth-first search : other path infeasible



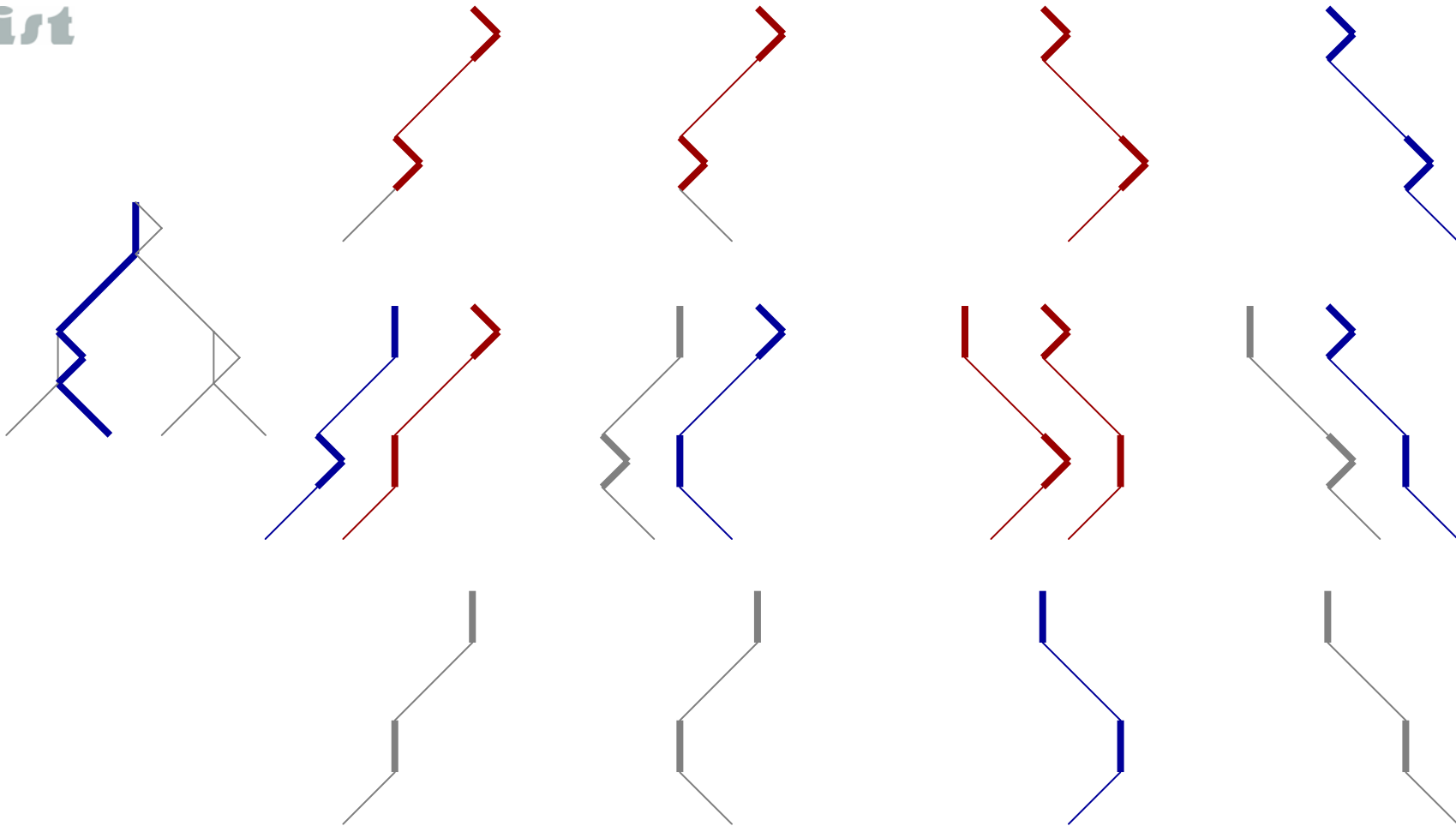
Infeasible with other empty ITE branch too



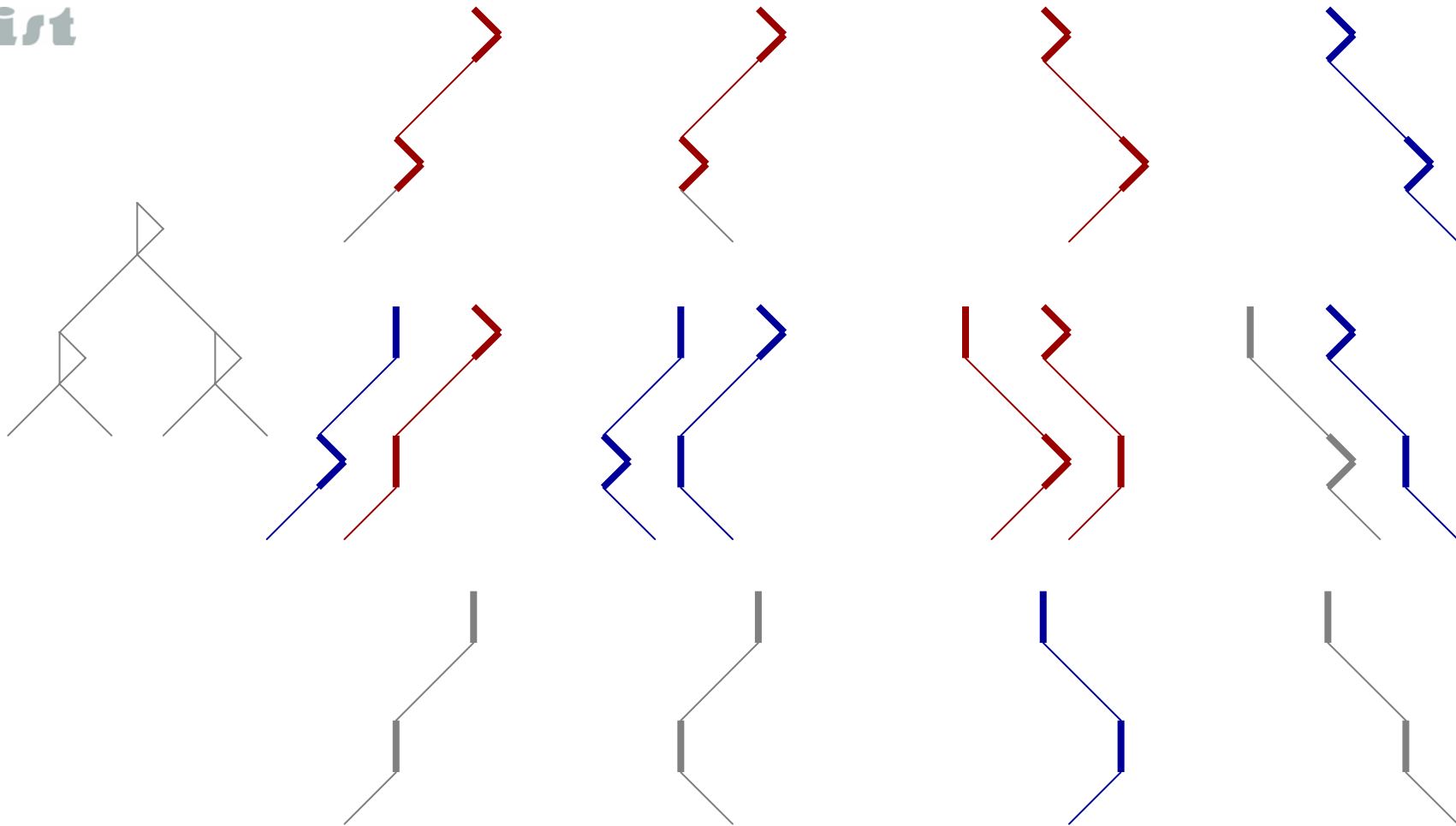
Maximal feasible path has 2 empty ITE branches



Path with other empty ITE branch is feasible too



All maximal feasible paths covered : stop here





Results of modified strategy

list

Example of industrial embedded real-time code :

1512 lines commented code

89 ITE, of which 45 with an empty branch, but no loops

default strategy: 846975 cases in several days

modified strategy: 6554 cases in one day

MaSCoTE project partner Geensys set up test rig to execute tests on a HCS12X simulator and measure execution time : the path with the longest execution time in the default set was also covered in the modified set





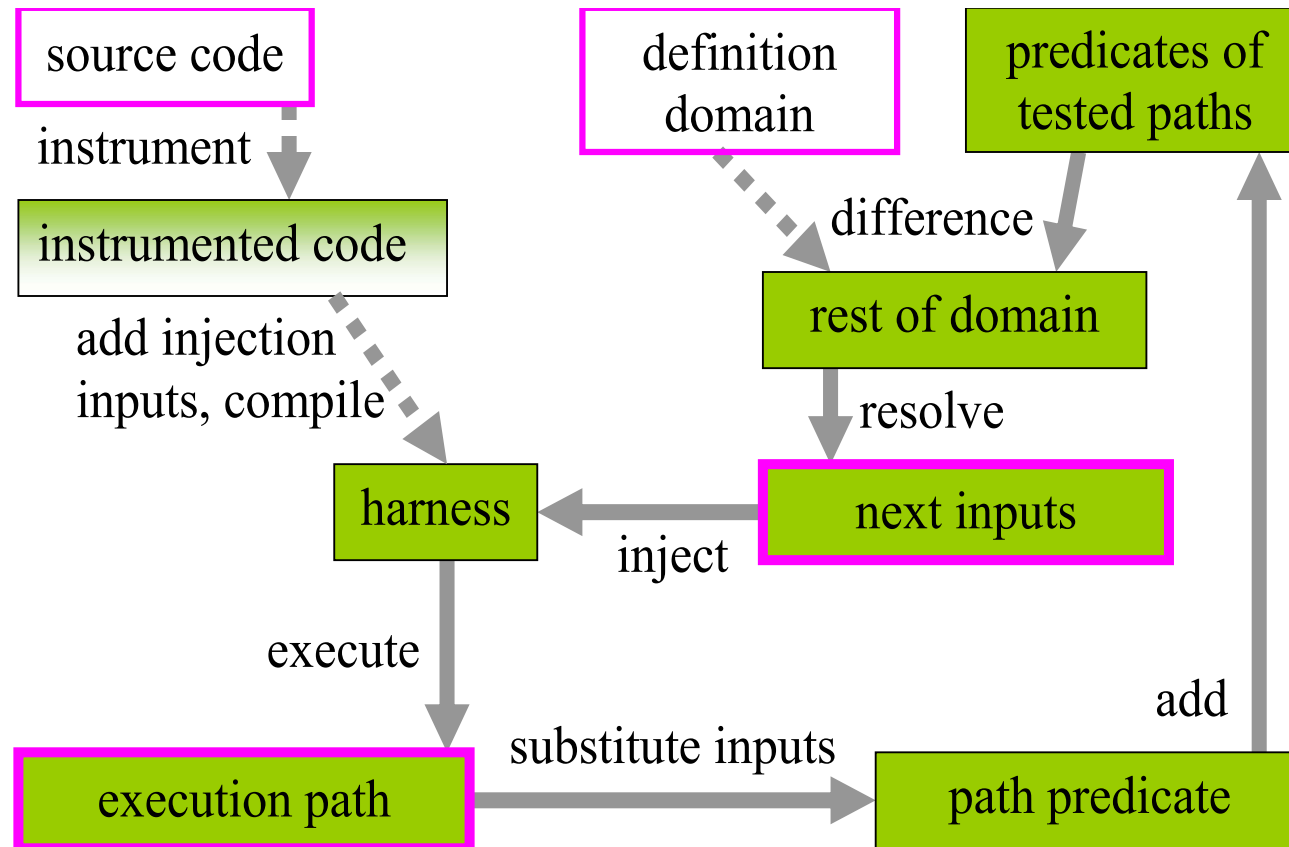
Future work

list

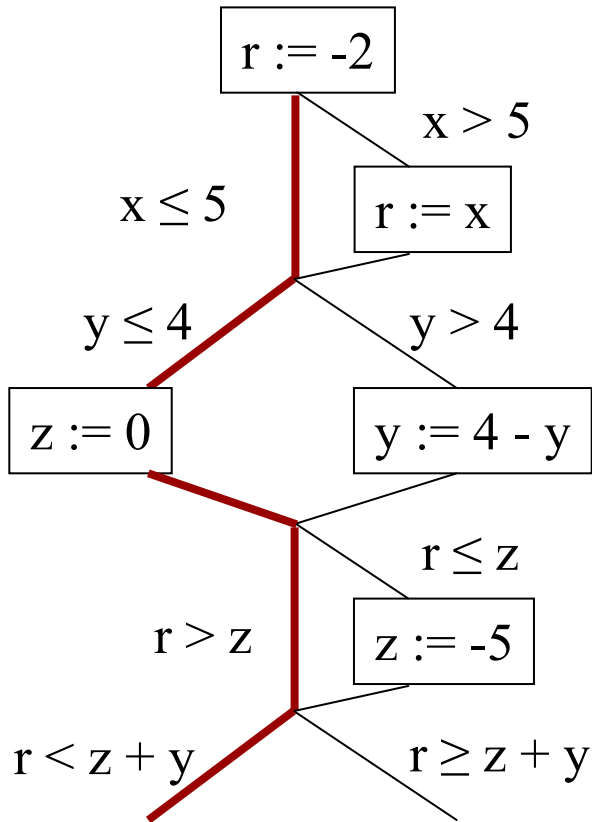
- Optimisation in progress : data dependences to avoid trying continuations of partial paths whose infeasibility cannot depend on ITEEs
- Restrictions : use PathCrawler to check no pointers are assigned in ITEE/loop and de-referenced after it
- Other partial orders: multiple conditions, function calls
- Uses of the same mechanisms in other test-generation strategies to treat ex. branch coverage, function calls,...



PathCrawler: process



CFG, infeasible path and unsatisfiable predicate



path predicate on input values:

$$x \leq 5 \wedge$$

$$y \leq 4 \wedge$$

$$-2 > 0 \wedge$$

$$-2 < -5 + y$$



Second partial order

list

P_i slower than P_j if their only difference is that for at least one loop all iterations are identical in P_i and P_j and P_i contains at least one iteration and P_i contains more

Hypothesis : no data cache or else don't apply if a pointer assigned in the loop is de-referenced afterwards

More complicated to implement because of multiple sub-conditions in loop-head

Occurs in many examples of embedded code using interpolation in discretised graphs coded as arrays



list

Default PathCrawler strategy (LR non-determinist DFS)

```
cover_all (P, i, PP) =  
  if BiP is_last_branch_in P then cover_subtree (PP:BiP, i)  
  else 1st pass : cover_all (P, i+1, PP:BiP)  
      on backtrack : cover_subtree (PP:BiP, i)      init: PP=Φ, i=1
```

```
cover_subtree (PP, i) =  
  if gen_test (PP) = P'  
  then if BiP' is_last_branch_in P'  
      then backtrack  
      else cover_all (P', i+1, PP)  
  else backtrack
```

P = execution path
i = branch number
PP = partial path
B_{iP} = branch i path P
B_{iP'} = opposite branch
:= append branch





list

Modified strategy for ITEE partial order: intro

- **cover_max** replaces **cover_all**
- **cover_max_subtree** and **cover_rest_subtree** replace **cover_subtree**
- *feas* = set of feasible paths
- *infeas* = set of abstract infeasible partial paths
- special treatment for
 - ♦ branches starting non-empty ITEEpaths
 - ♦ branches which are empty ITEEpaths



list

Modified strategy for ITEE partial order (1)

APP = abstract partial path, \cdot = append to APP

cover_max (P, i, PP, APP) =

if B_{iP} **is_last_branch_in** P *then* *if* B_{iP} **st_nonempty ITEEpath** *then* *bt*

else **cover_max_subtree** (PP: \underline{B}_{iP} , APP: \underline{B}_{iP} , i)

else if B_{iP} **st_nonempty ITEEpath** *then* *1st* : **cover_max** (P, i+1, PP: B_{iP} , APP: B_{iP})

bt : **cover_rest_subtree** (PP: \underline{B}_{iP} , APP: \underline{B}_{iP} , i)

else if B_{iP} **is_empty ITEEpath** *then*

if **gen_test** (PP: \underline{B}_{iP}) = P' *then*

1st : $feas := feas \cup P'$; *if* $B_{iP'}$ **is_last_branch_in** P' *then* *bt*

else **cover_max** (P', i+1, PP: \underline{B}_{iP} , APP: \underline{B}_{iP})

bt : **cover_rest_subtree** (i, PP: B_{iP} , APP: B_{iP})

else $infeas := infeas \cup APP \cdot \underline{B}_{iP}$; **cover_max** (P, i+1, PP: B_{iP} , APP: B_{iP})

else *1st* : **cover_max** (P, i+1, PP: B_{iP} , APP: B_{iP})

bt : **cover_max_subtree** (PP: \underline{B}_{iP} , APP: \underline{B}_{iP} , i)





Modified strategy for ITEE partial order (2)

list

cover_max_subtree (PP, APP, i) =

if **gen_test** (PP) = P' *then* **feas** := **feas** U P' ; *if* B_{iP'} **is_last_branch_in** P'
then *backtrack*

else **cover_max** (P', i+1, PP, APP)

else **infeas** := **infeas** U APP ; *backtrack*

cover_rest_subtree (i, PP, APP) =

foreach APPext \in **infeas** { *foreach_in_order* PPext **concretises** APPext {

if PP **is_a_prefix_of** PPext *then*

if not **slower_feas_paths** (Pext, APPext) *then*

if **gen_test** (PPext) = P' *then* **feas** := **feas** U P' ; *if* B_{iP'} **is_last_branch_in** P'
then *backtrack*

else **cover_max** (P', i+1, PP, APP) }

else *backtrack* } }



list

Modified strategy for ITEE partial order (3)

slower_feas_paths (Pext, APPext) *if*
exists Pext', PPext' *such_that* (
 Pext' \in feas
 and PPext' **is_a_prefix_of** Pext'
 and PPext' **concretises** APPext
 and Pext' **slower_than** Pext)