A Linear Time Algorithm for Automatic Generation of Multiplicative Planar Proof Nets

Satoshi Matsuoka
AIST

Motivation

- Discovery of a new linear time correctness condition for MLL proof nets
  - Efficient implementation (in *Proof Net Calculator*)
  - Found a theory bug in the first implementation: may result in deadlock
  - The second implementation seems correct: adding a deadlock prevention mechanism
- Formal verification (not yet, but seems to be OK by using an easy *state machine encoding*)
- Testing using test data
- Effective test data generation: automatic generation of MLL proof nets

The system MLL

MLL formulas: $A ::= p \mid p^{\perp} \mid A \otimes B \mid A \parr B$

$\text{negation}: \quad (p^{\perp})^{\perp} := p, \qquad (A \otimes B)^{\perp} := B^{\perp} \parr A^{\perp}, \qquad (A \parr B)^{\perp} := B^{\perp} \otimes A^{\perp}$

Inference rules:

$$\text{ID} \qquad \frac{}{\vdash A, A^{\perp}}$$

$$\otimes \qquad \frac{\vdash \Sigma_1, A \qquad \vdash B, \Sigma_2}{\vdash \Sigma_1, \Sigma_2, A \otimes B}$$

$$\parr \qquad \frac{\vdash \Sigma, A, B}{\vdash \Sigma, A \parr B}$$

$$\text{Cut} \qquad \frac{\vdash \Sigma_1, A \qquad \vdash A^{\perp}, \Sigma_2}{\vdash \Sigma_1, \Sigma_2}$$

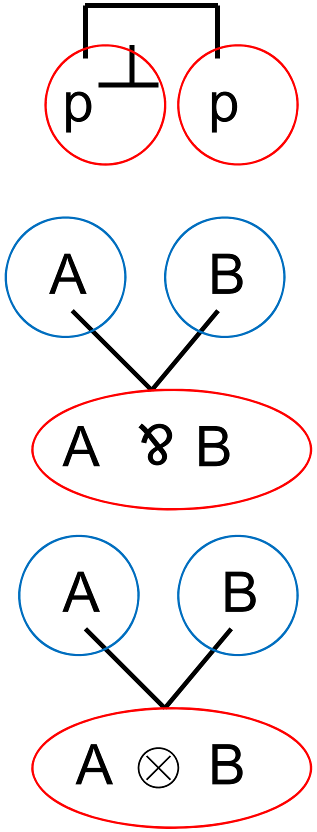
Cut is almost $\otimes$

$\Sigma, \Sigma_1, \Sigma_2$ are multisets of MLL formulas

MLL Proof Nets

- Abstract formal proofs of formulas of Multiplicative Linear Logic
- A subset of the set of MLL proof structures
- Sequentializable MLL proof structures

Links

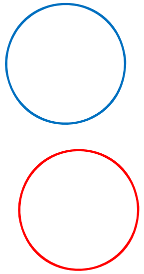


ID-Links

Par-Links

Tensor-Links

Multiplicative-Links



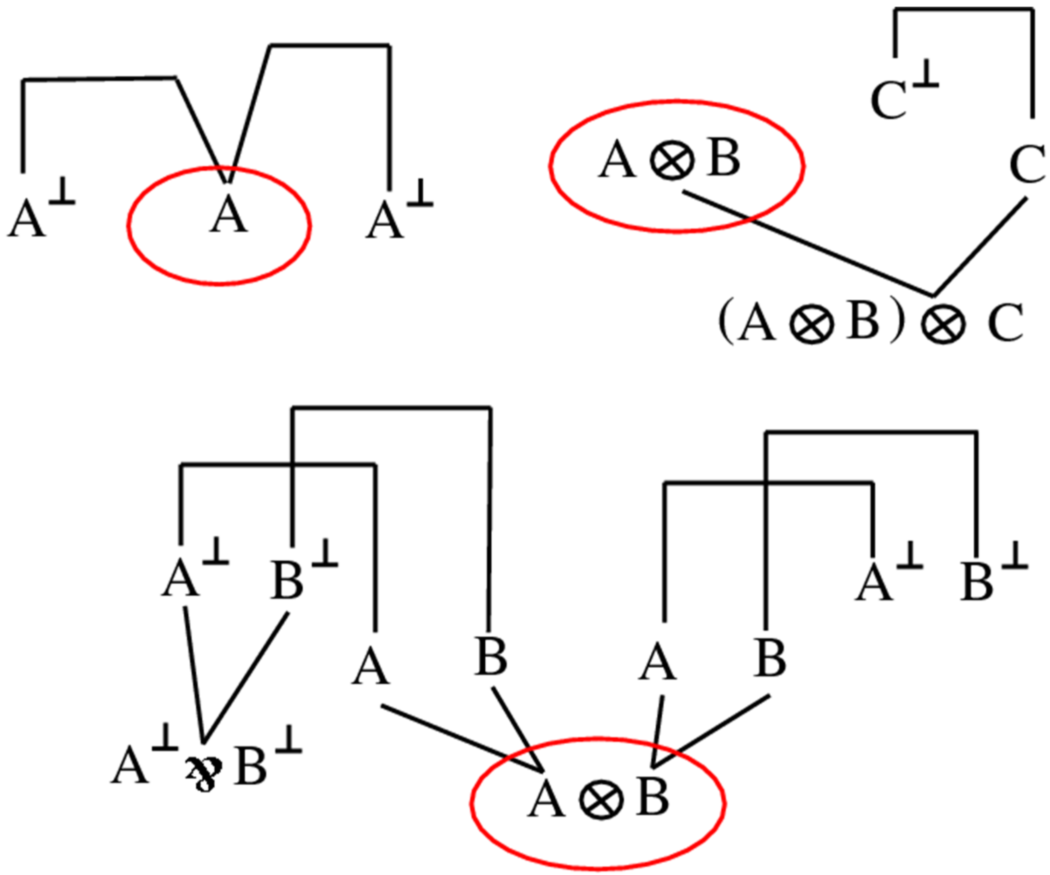
premises

conclusions

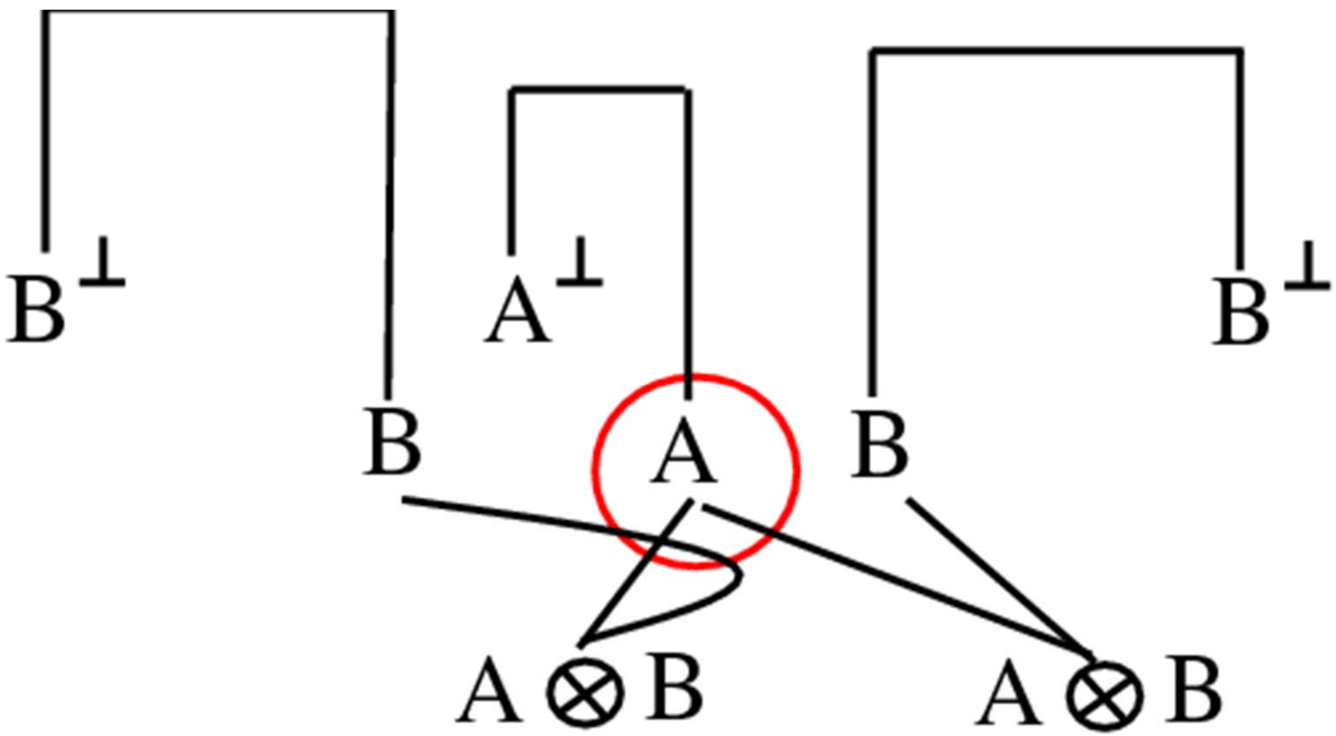
MLL proof structure

- A set of links
- Satisfying two conditions

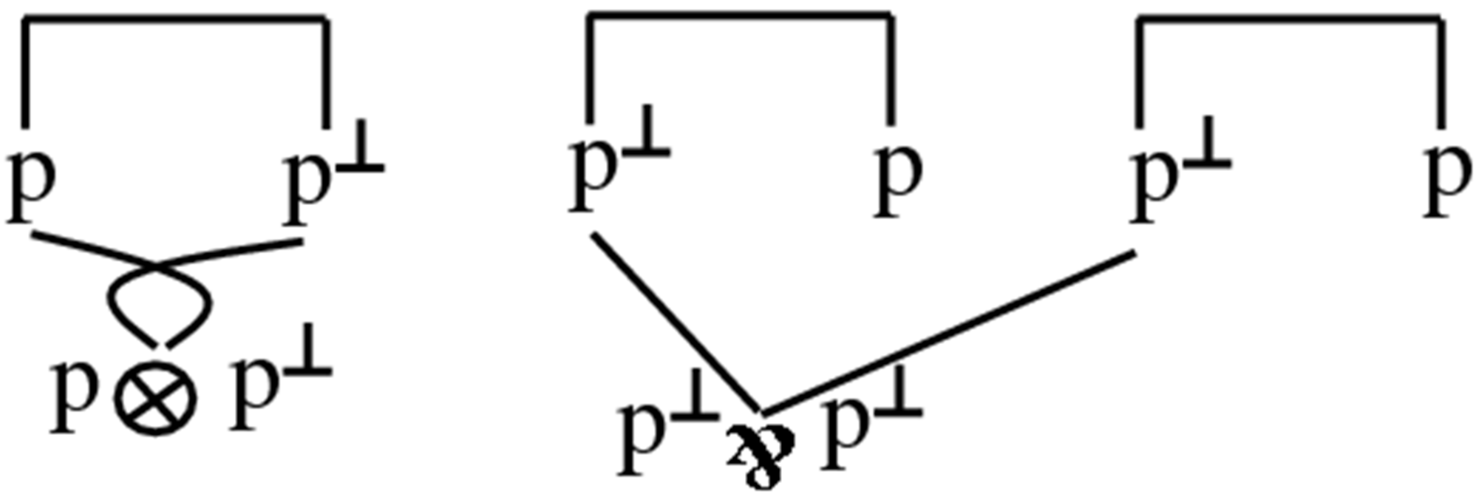
Cond. (1): any formula (occurrence) is a conclusion of exactly one link



Cond.(2): any formula (occurrence) does not become a premise of more than one link

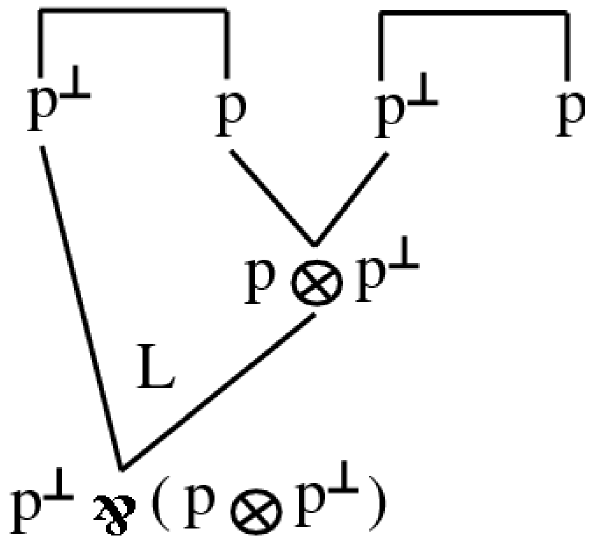


MLL proof structures (but not MLL proof nets)



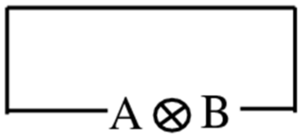
MLL Proof Structure(also MLL proof net)

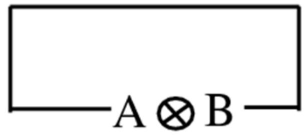
$\Theta_1 =$

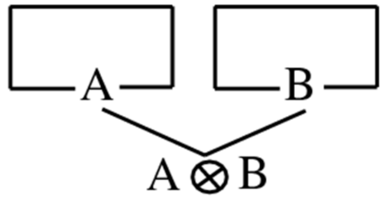


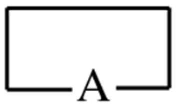
Definition of Proof Nets: Sequentializability

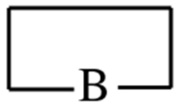
(1) ID−links are sequentializable

(2) 

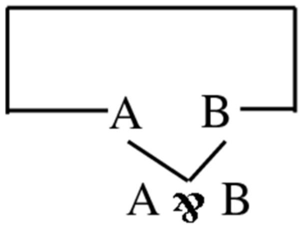
sequentializable if 

= 

and bofh 

and 

sequentializable

(3) 

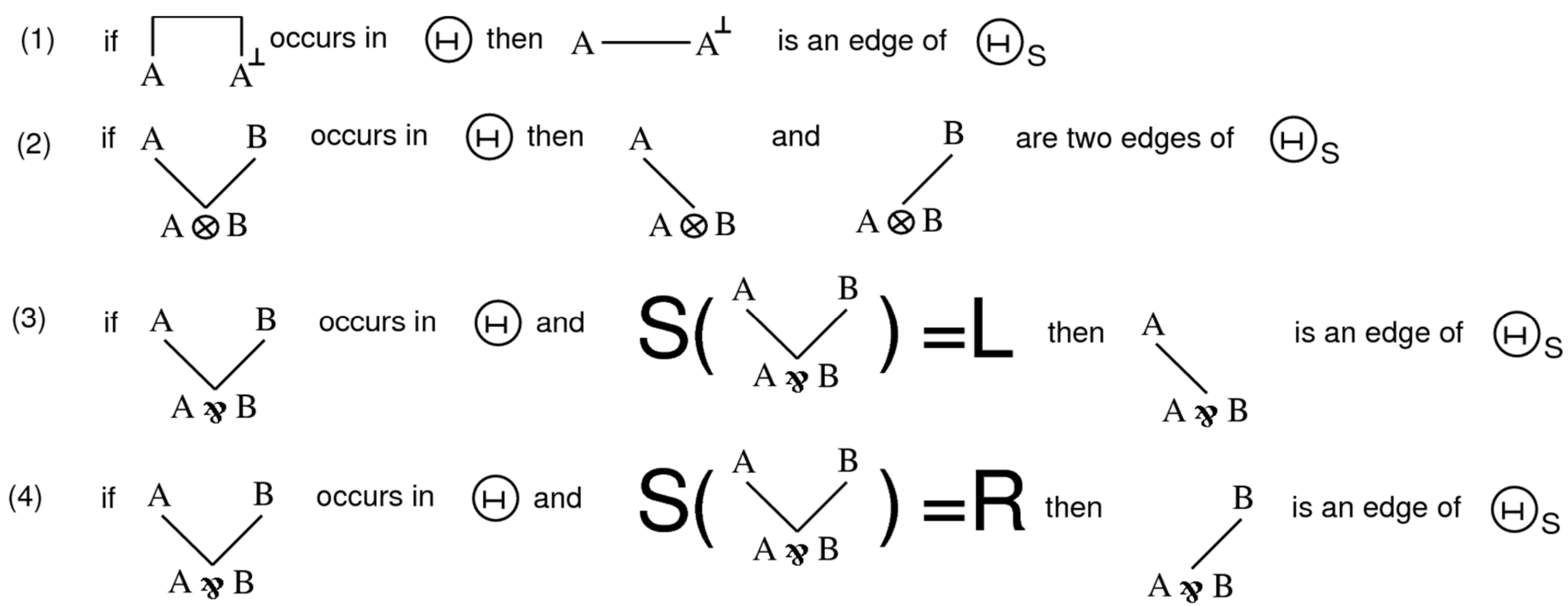
sequentializable if 

sequentializable

A DR-switching S for a proof structure Θ

- A function from the set of par-links in Θ to {L, R}

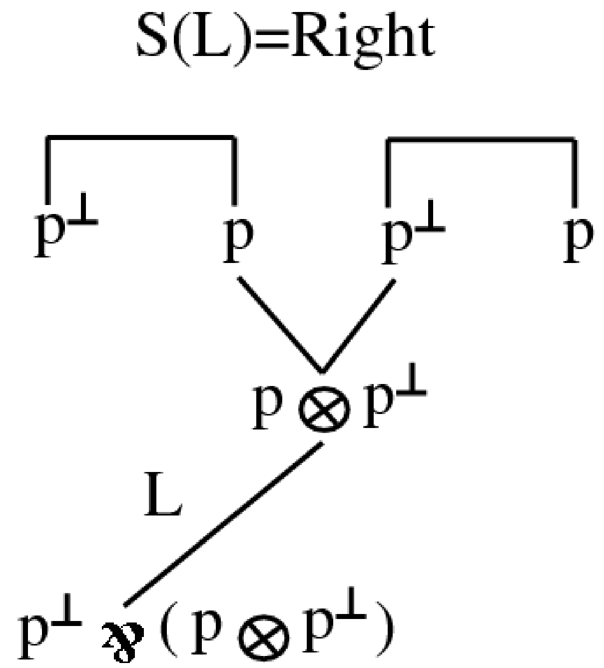
The DR graph ΘS induced by DR-switching S



A Graph-theretic characterization of proof nets

- Theorem (Girard, Danos-Regnier)
  A MLL proof structure Θ is a MLL proof net
  iff
  for any DR-switching S for Θ,
  the DR-graph ΘS is acyclic and connected

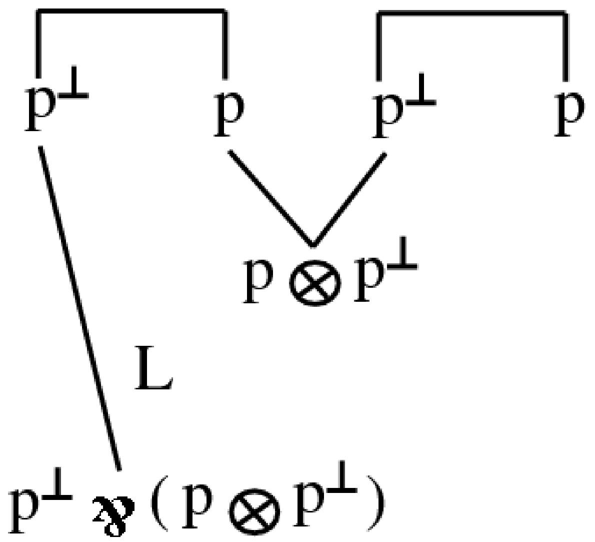
A DR-graph of Θ_1



acyclic and connected(tree)

Another DR-graph of Θ_1

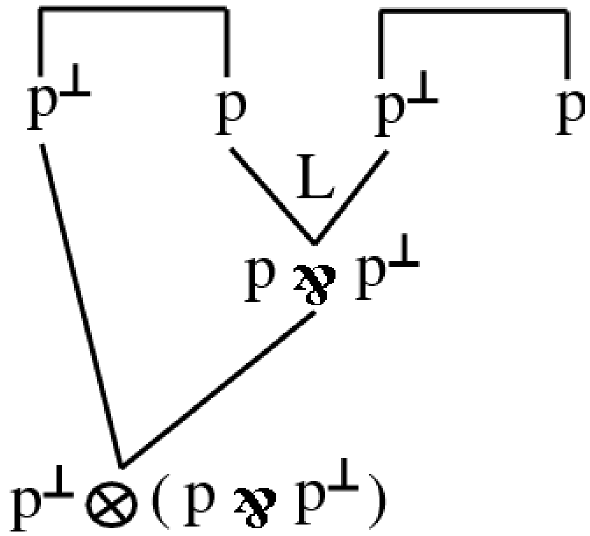
S(L)=Left



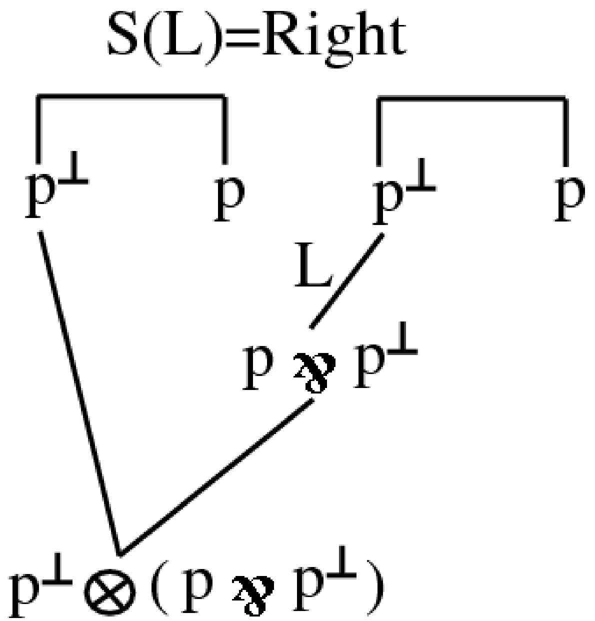
acyclic and connected(tree)

An MLL proof structure (But not an MLL proof net)

$\Theta_2 =$

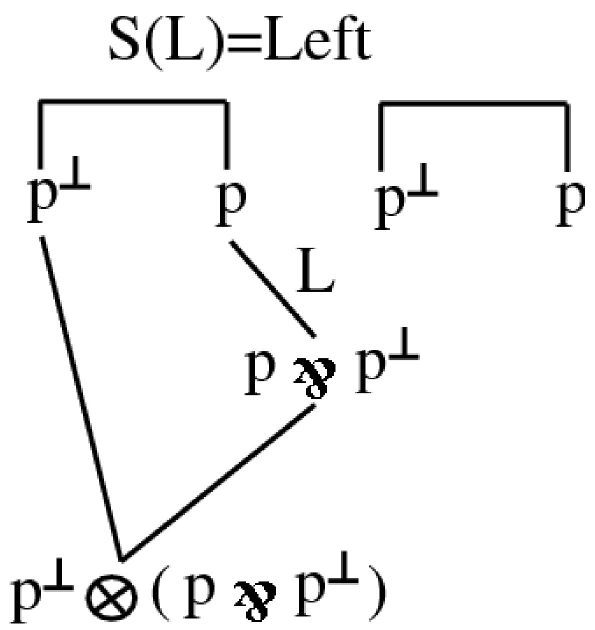


A DR graph for Θ_2



acyclic and connected(tree)

Another DR-graph for Θ_2



has a cycle

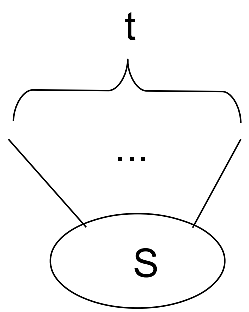
A new linear time correctness condition

- Given an MLL proof structure Θ
- Select a DR-switching S for Θ
- If the DR-graph S(Θ) is not acyclic and connected then Θ is not a proof net
- Otherwise, construct the *deNM*-tree T for Θ and S,
  1. check whether or not T can reduce to one node using three rewrite rules
  2. If it succeeds, then Θ is a proof net
  3. Otherwise, not a proof net
- To establish linear time termination, must use *union-find* data structure

deNM-trees

- Trees consisting of the following two types of nodes:

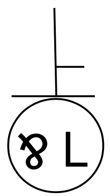
labeled-node



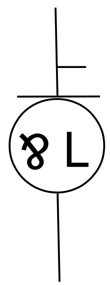
S is a set of labels l_L or r_L where L is a ⅋-link

$t \geq 0$

⅋ -node (degree 1)

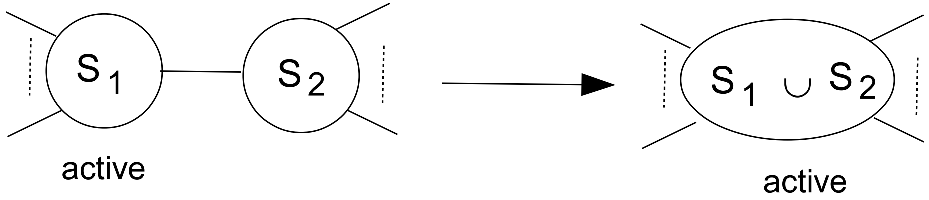


⅋ -node (degree 2)

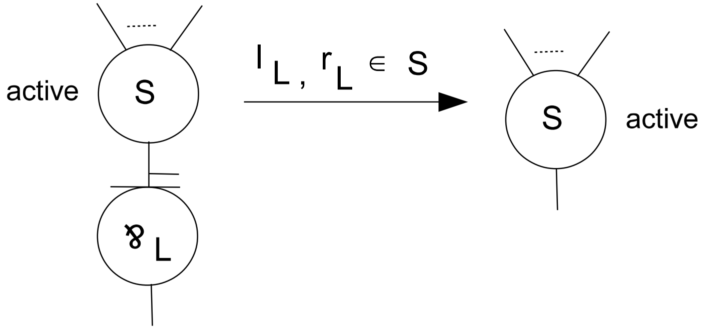


Three rewrite rules

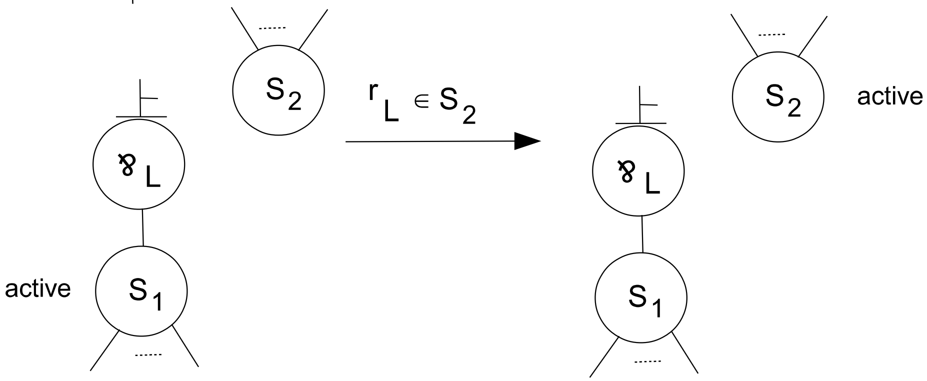
union



⅋-elimination

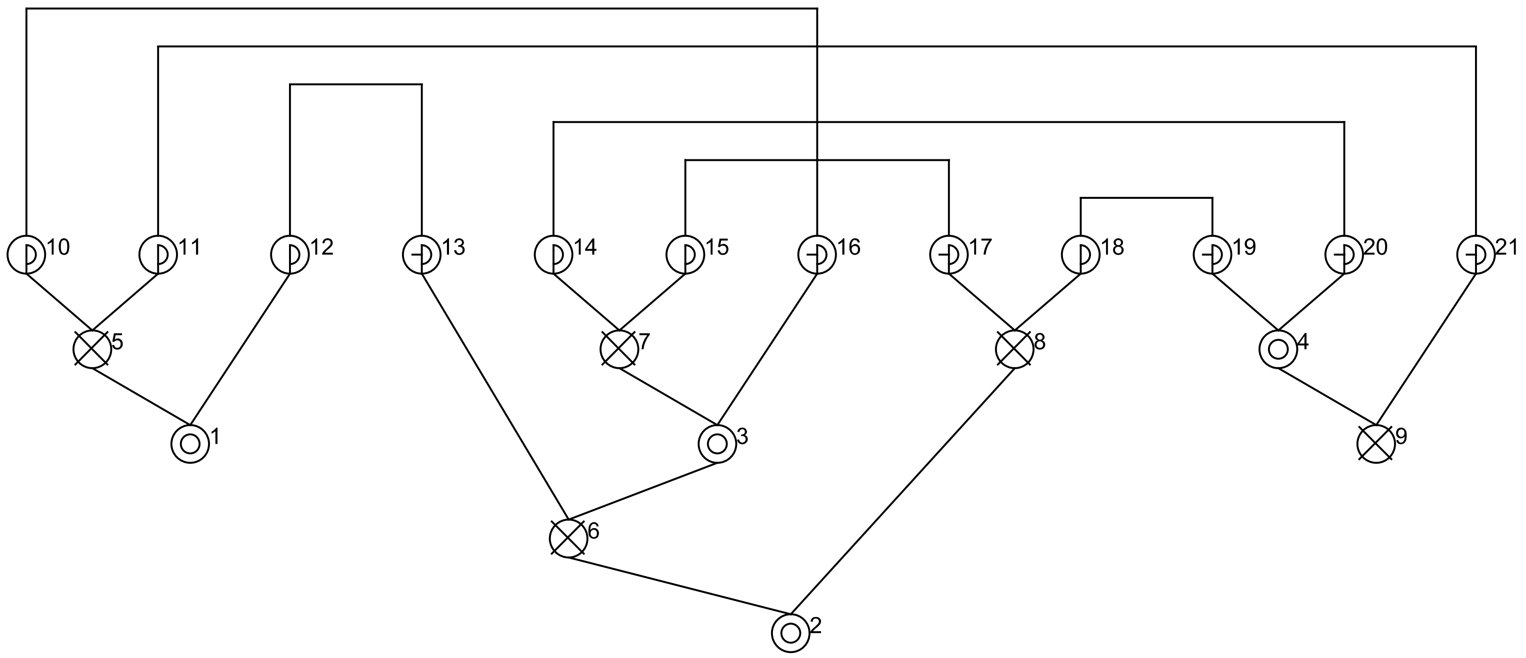


local jump

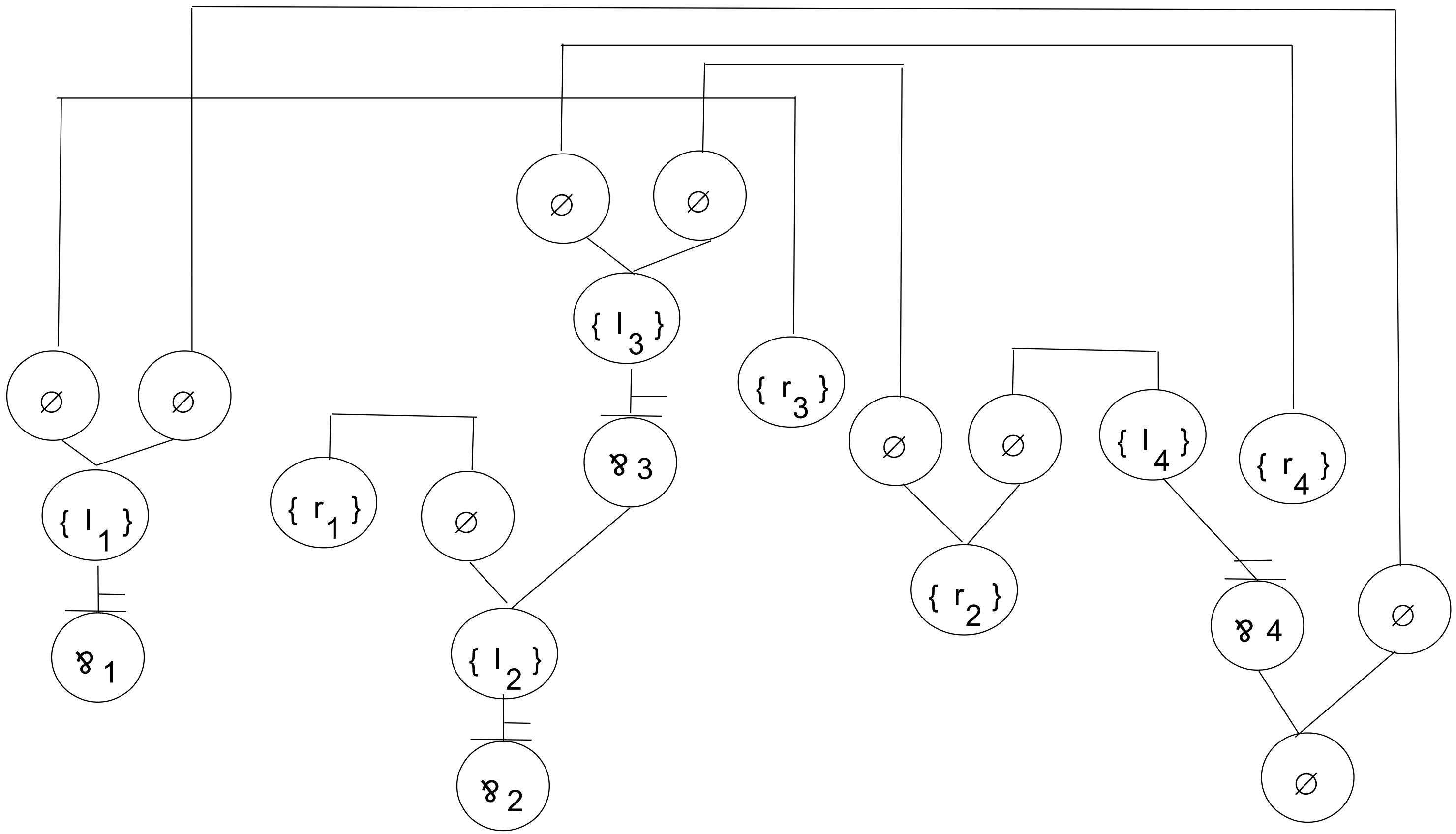


Example 1: An MLL proof net

PN-1.txt

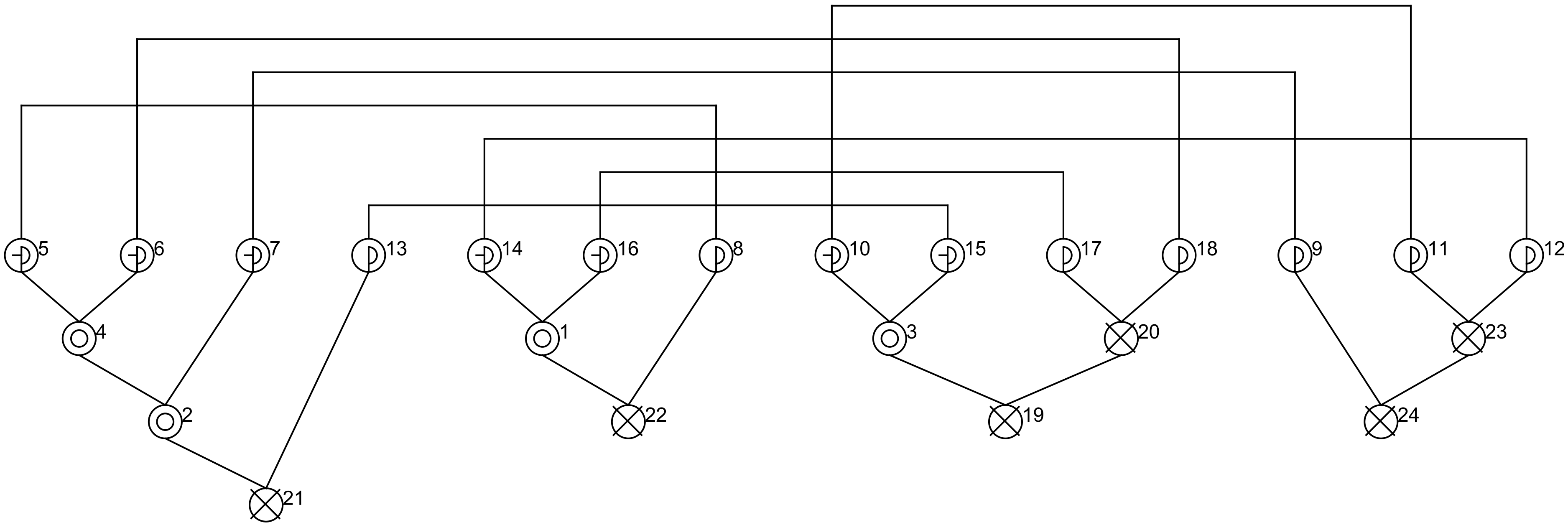


Example 1: its deNM-tree (by extremely left switching)

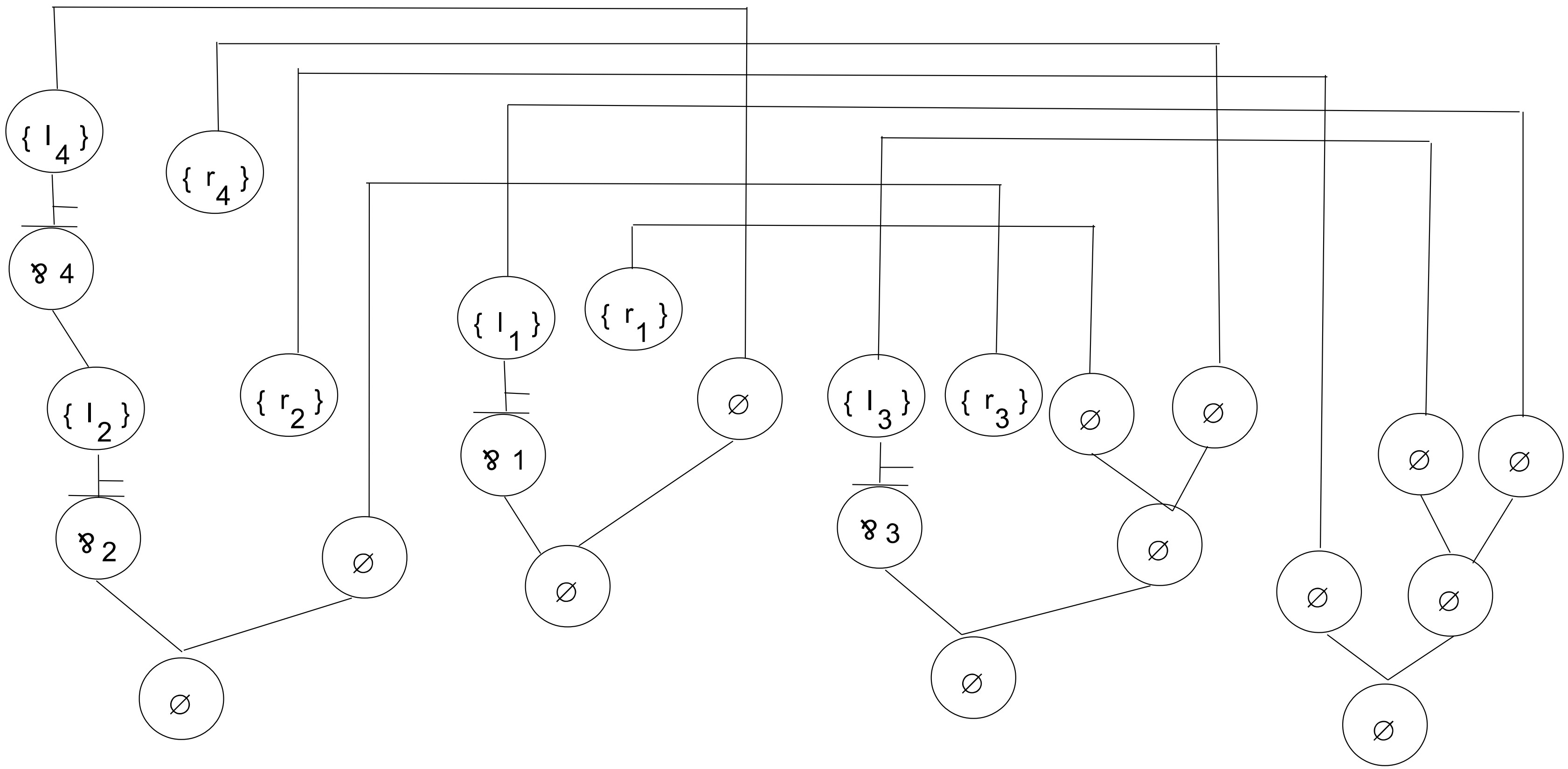


Example 2: An MLL proof structure, but not PN

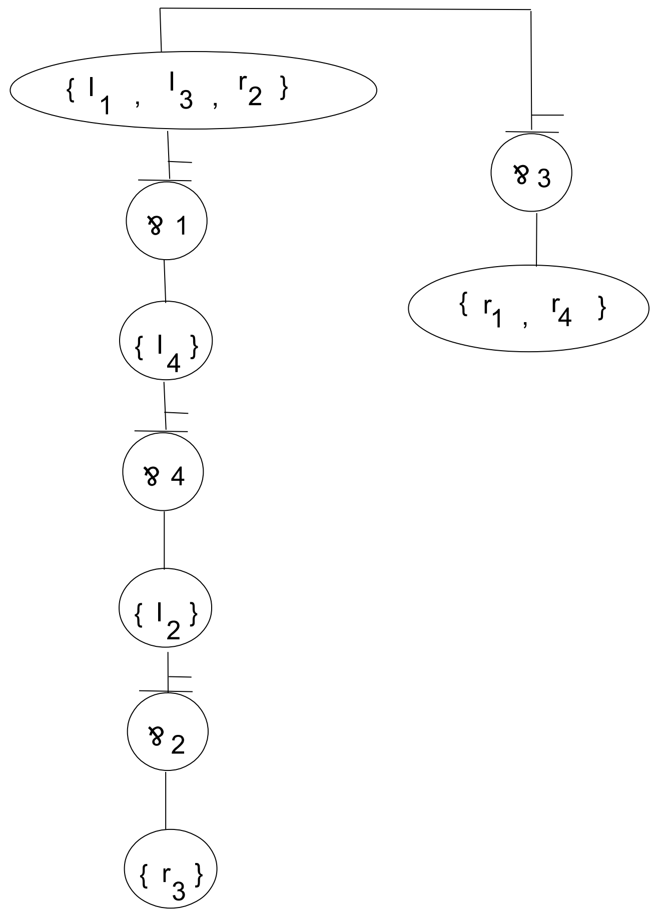
nonPN-1.txt



Example 2: its deNM-tree (by extremely left switching)



Example 2: the reduced deNM-tree



Union-Find Data Structure

- Dynamic operations on a finite set $S = \{a_1, a_2, \cdots, a_n\}$:
  - Make-Set(x): creates the singleton set {x}
  - Union(x, y): makes the union set $S_x \cup S_y$, where
    S_x (resp. S_y) is the current subset including x (resp. y)
  - Find-Set(x): return the representative element of S_x
- Can check whether or not two elements x and y belong to the same subset of S currently
- Runs in almost linear time (in practical sense)
- Runs in linear time over special cases, especially over *planar graphs*

A new linear time correctness condition

- Found bugs twice (at this moment)
- The first bug was a trivial mistake
- Then the algorithm was implemented
- The second bug was a subtle one:

  found it through a comparison test with an existing quadratic correctness condition
- Hope that the current one is the last one
- Would like to confirm its correctness through *tests*

The goal

- Generation of a number of MLL proof structures but not proof nets
  - The implementation must answer *"no"* for these
  - This part is relatively easy
- Generation of a number of MLL proof nets
  - The implementation must answer *"yes"* for these
  - This part is *not* so easy, especially for generating big proof nets

Generation of general MLL proof structures

1. Generation of *one-conclusion pre proof structure* P
2. Assignment of multiplicative links to P

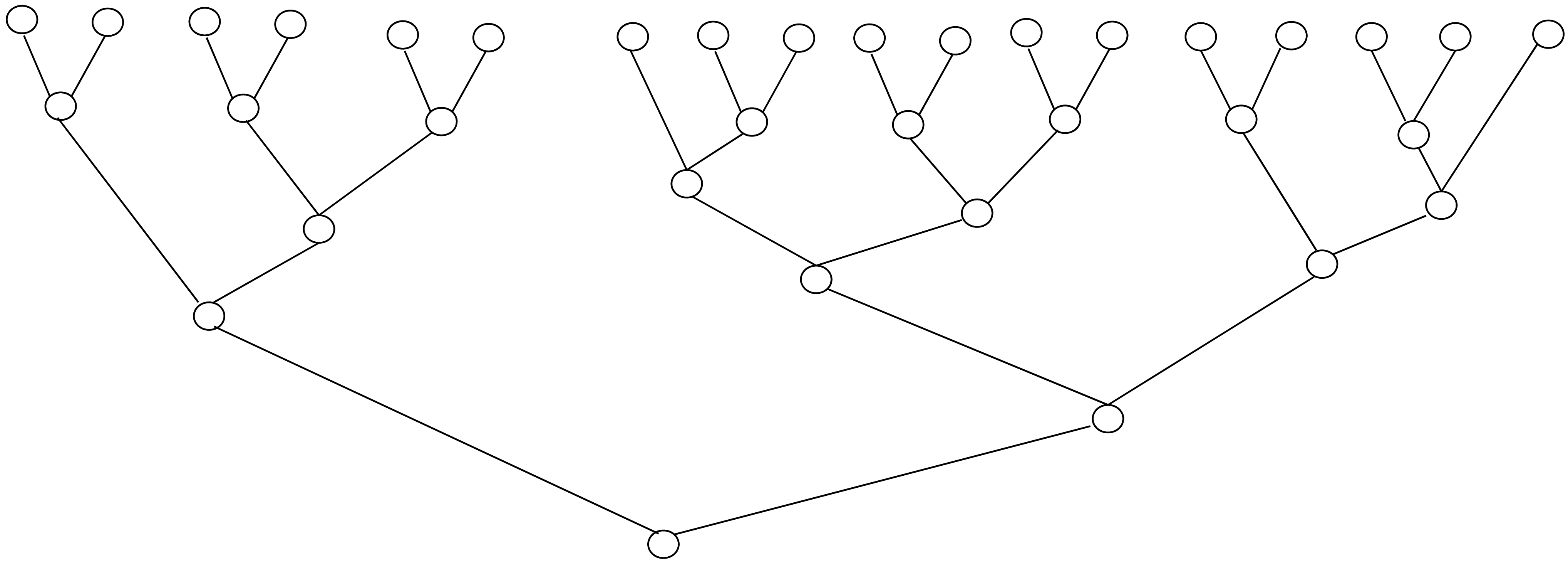
One-conclusion pre proof structure

- binary tree with *even-number* leaves

  plus
- *fixpoint free involution* over the leaves

- Can be easily extended to *connected multi-conclusion* pre proof structures

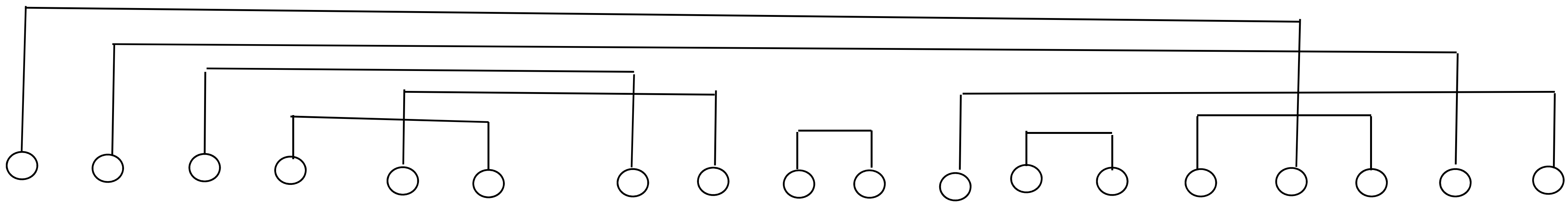
Binary tree with even-number leaves



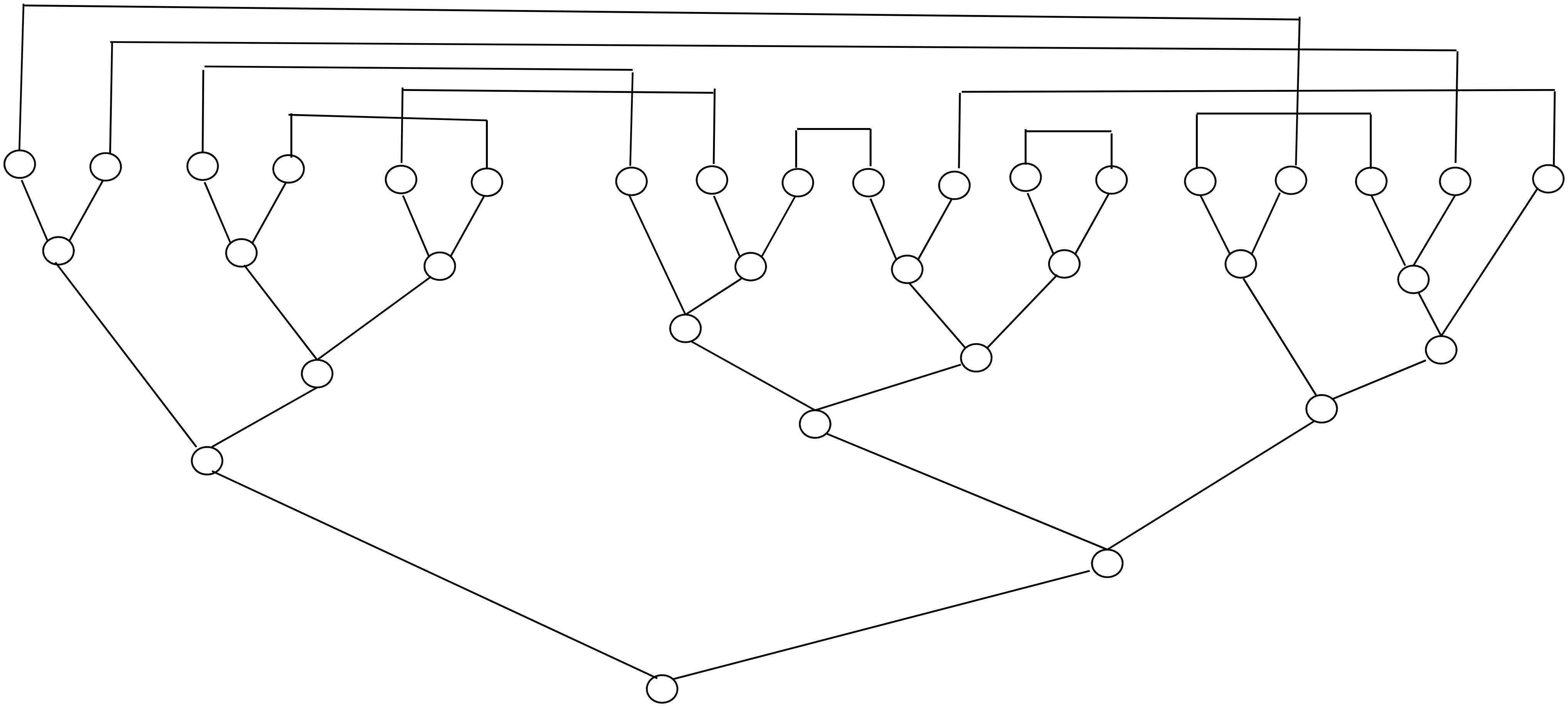
A simple observation about binary tree with even-number leaves

- When a binary tree has 2n leaves, the tree has 2n-1 internal nodes

fixpoint free involution over an even cardinal set



One conclusion pre proof structure



One conclusion proof structure

- One conclusion pre proof structure

  plus
- An assignment of $\{\otimes, \parr\}$ to the internal nodes
- An assignment of {+, -} to the leaves
- An assignment of an appropriate fixpoint involution

A necessary condition for a one conclusion MLL proof net

- $\mathbf{num}_{\mathrm{ID}} = \mathbf{num}_{⅋}$

- $\mathbf{num}_{\mathrm{ID}} = \mathbf{num}_{\otimes} + \mathbf{1}$

Random generation of MLL proof structures

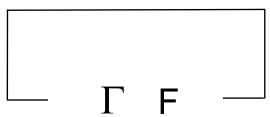
- Generate a random binary tree with even-number leaves
- Generate an assignment of multiplicative links to internal nodes satisfying the necessary condition
- Generate an appropriate fixpoint free involution with polarity

- Can generate MLL proof structures, but very few MLL proof nets

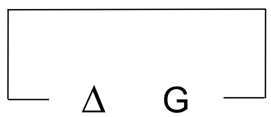
Inductive definition of MLL proof nets

is an MLL proof net.

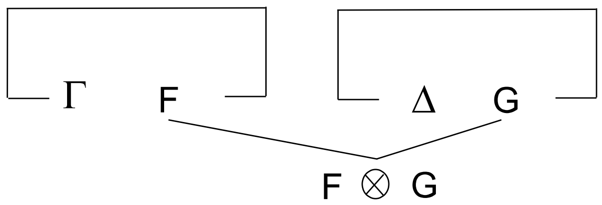
If



and

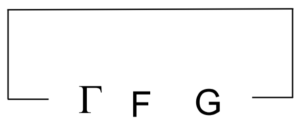


are MLL proof nets, then

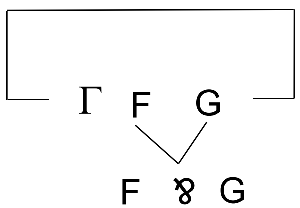


is an MLL proof net.

If



is an MLL proof net, then

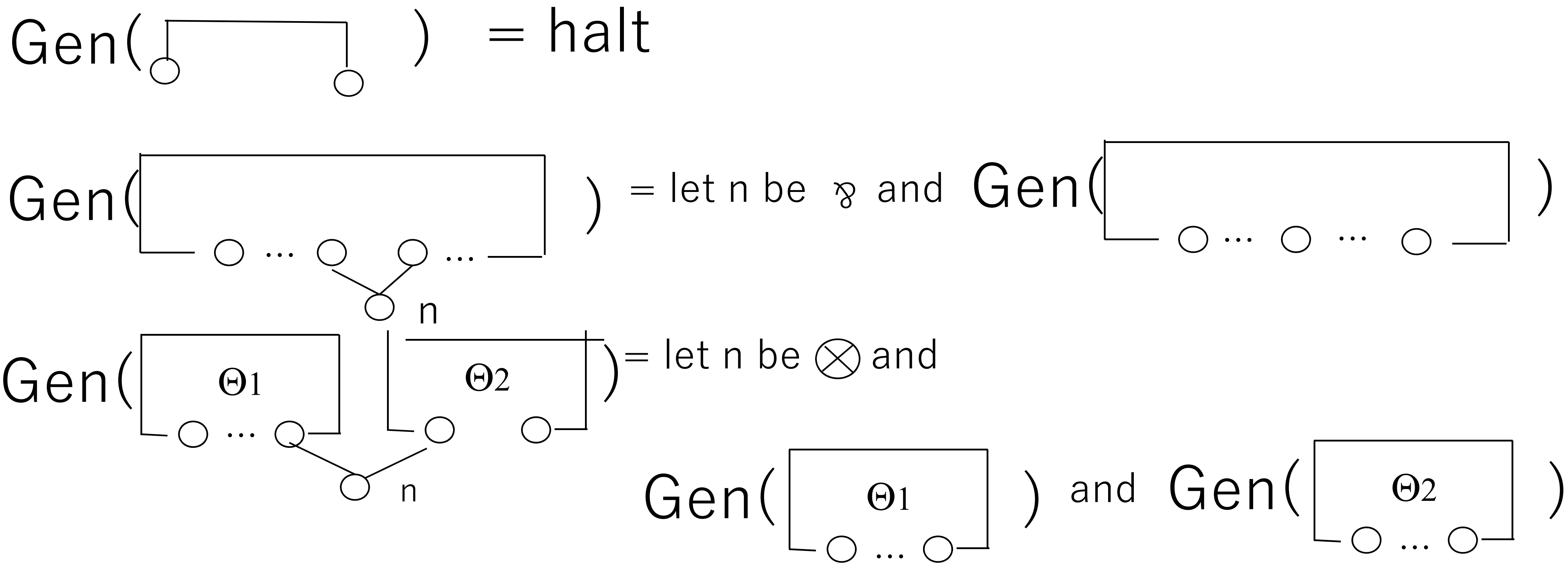


is an MLL proof net.

Use of ind. def. of MLL PNs for test data gen.

- Closely related to Galton-Watson branching processes

Proof Net Generation from *connected multi-conclusion* pre proof structures (connected pre PS's)



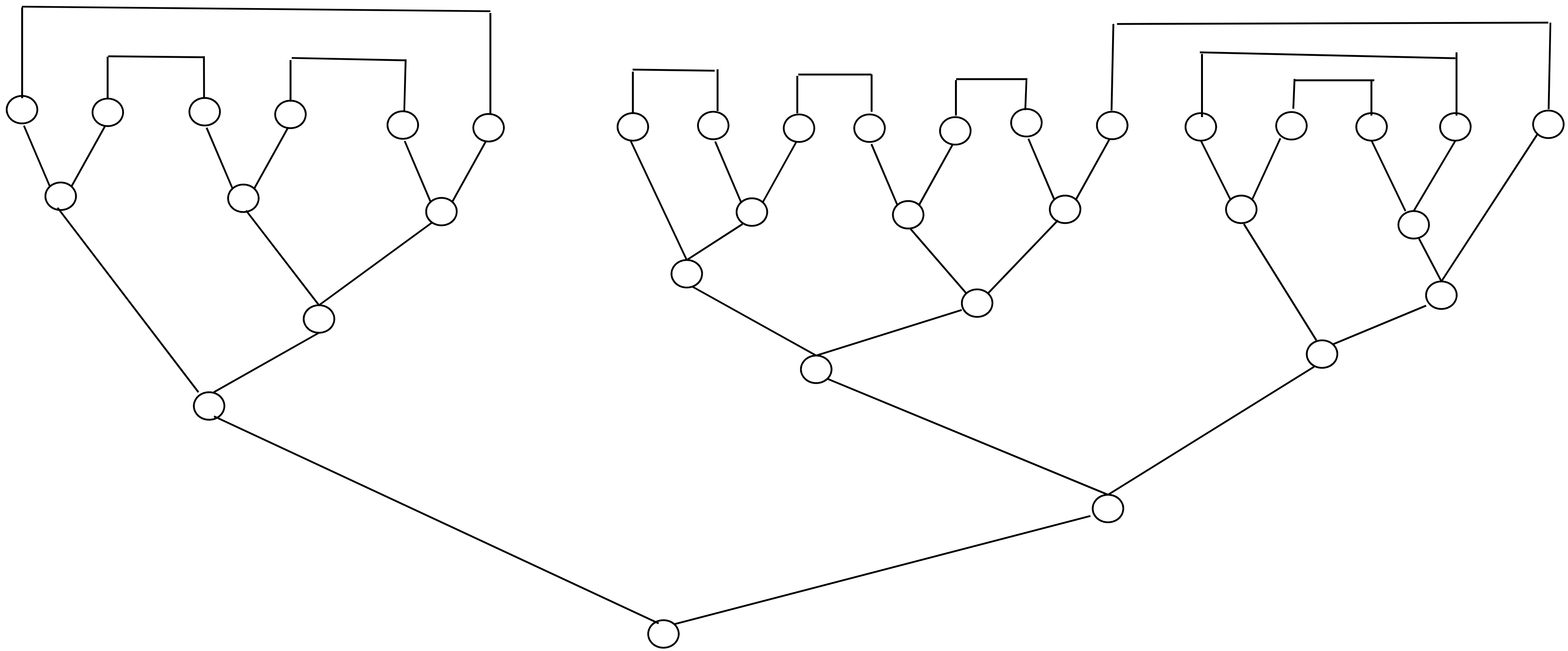
Remarks

- A forest can be linear-ordered in a bottom up manner by *topological sort* in linear time
- Naïve implementation of the Gen procedure is quadratic:

    connected check at each recursive step
- We have found a linear time Gen procedure:

    but in the special case, i.e., in the *planar* case

Planar pre proof structure



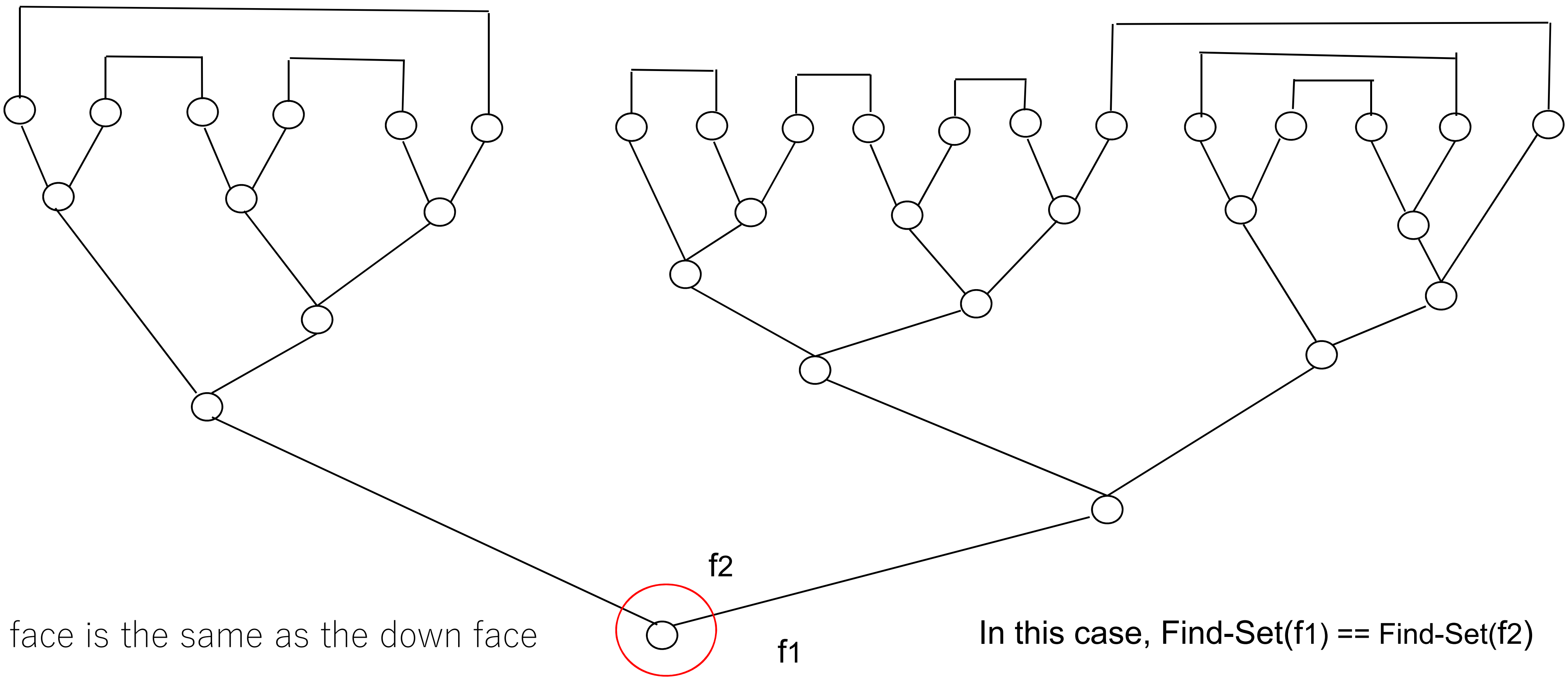
Quiz

- Can you find a solution?
- Hint: use of union-find data structure

Our solution

- Get attention to *faces*
- Observations:
    1. Each internal node has at most three faces
    2. Each conclusive internal node has (necessarily not different) two (up and down) faces

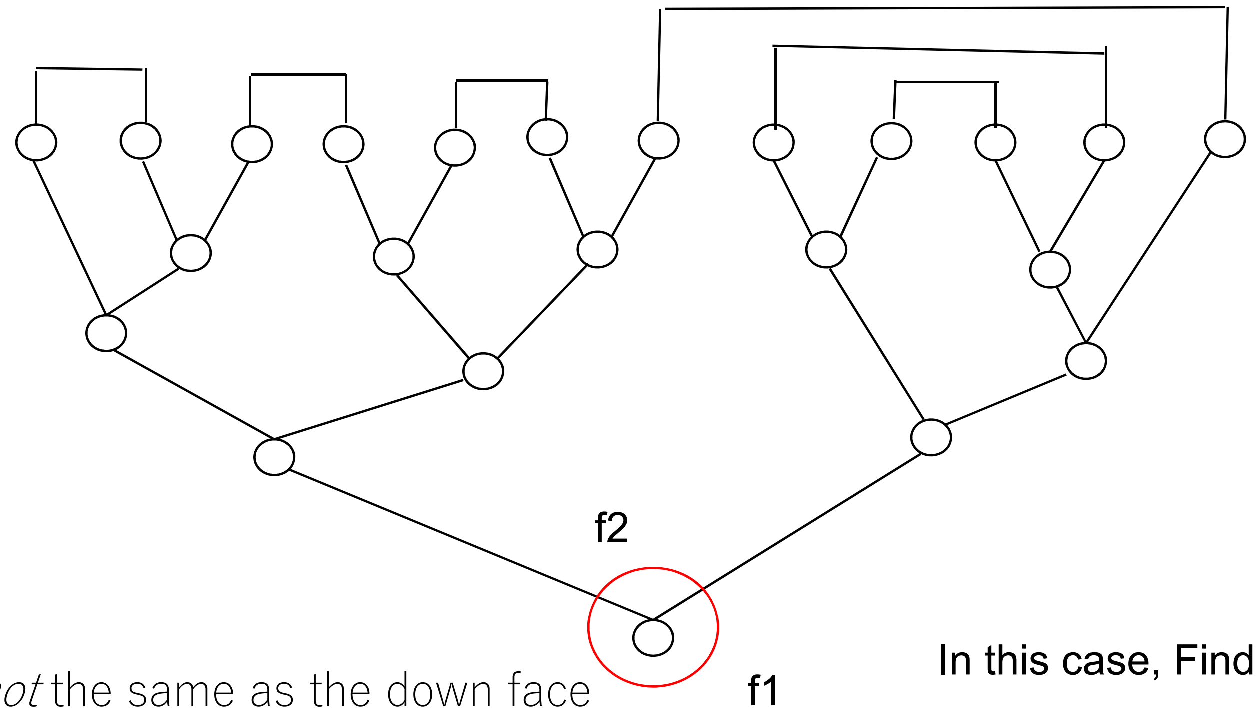
Case 1: removal of conclusion node results in two connected components



The up face is the same as the down face

In this case, Find-Set(f1) == Find-Set(f2)

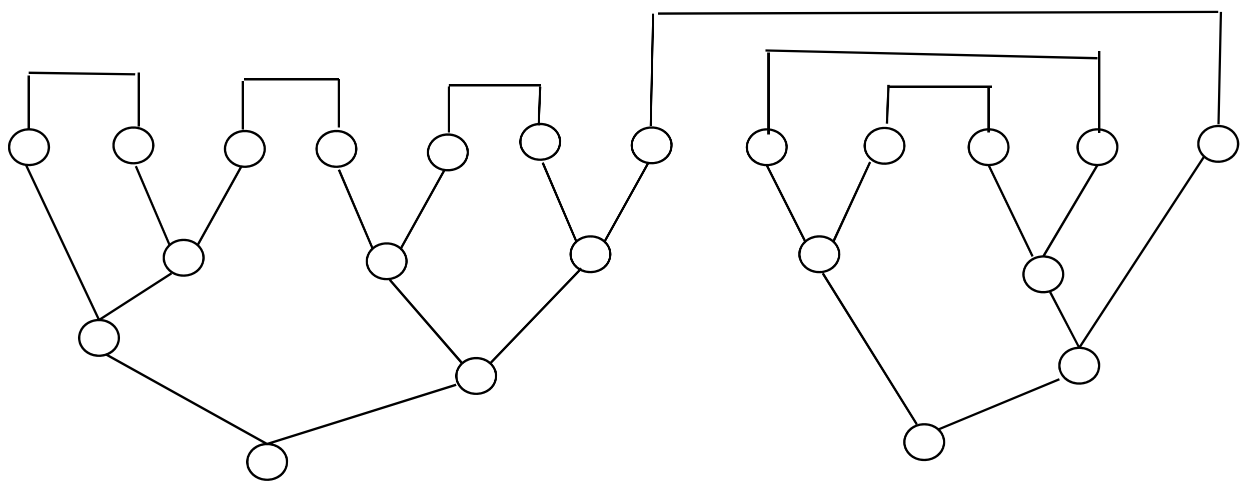
Case 2: Otherwise



The up face is *not* the same as the down face

In this case, Find-Set(f1) != Find-Set(f2)

Case 2: Otherwise (Cont.)



Execute Union(f1, f2)

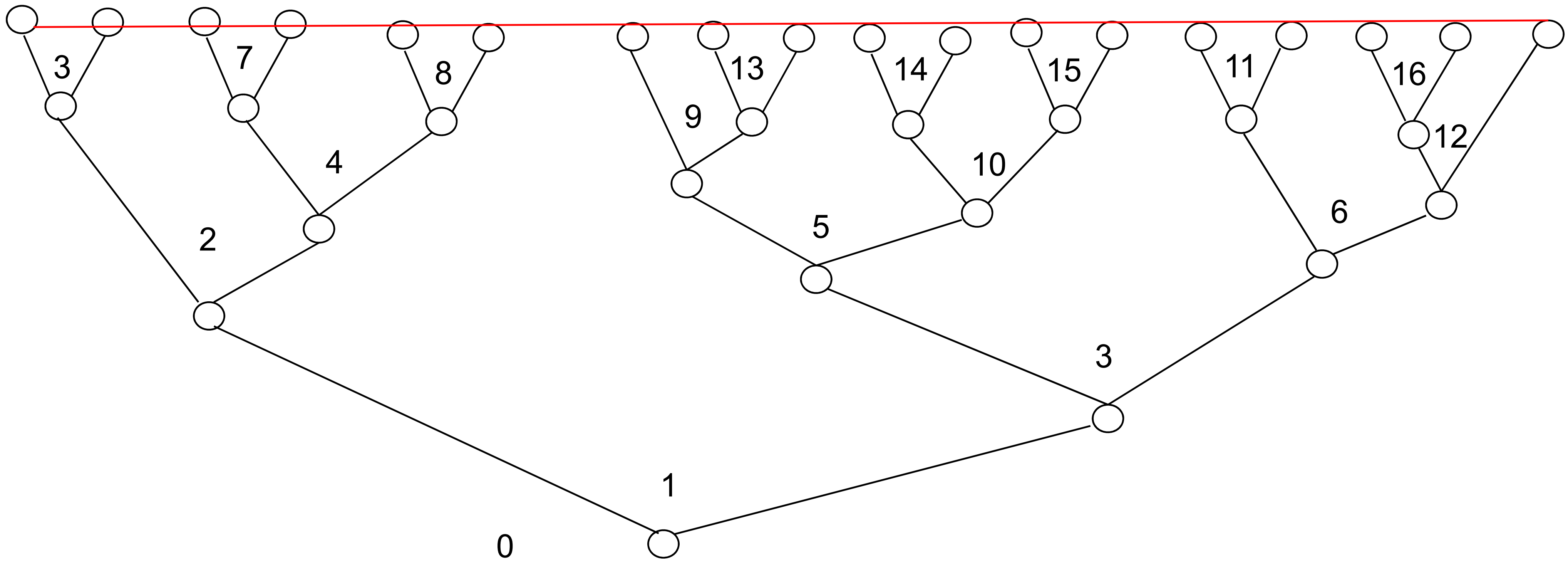
Question

- How can we assign faces to a pre proof structure?

Our answer

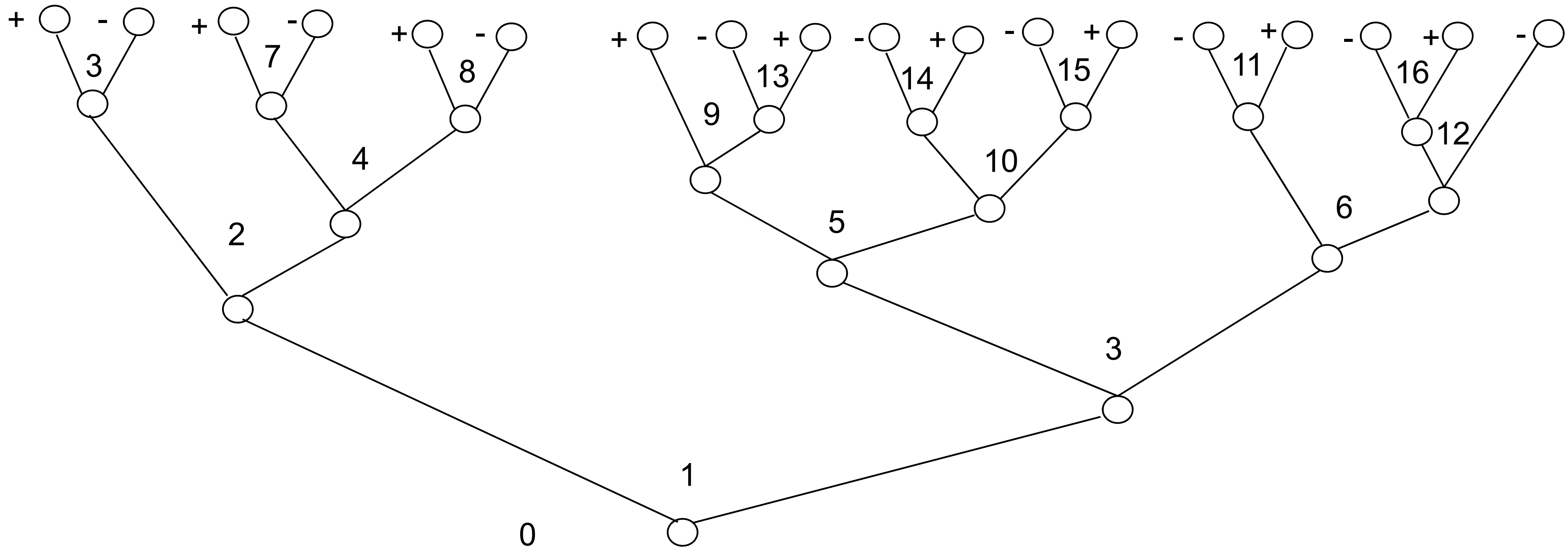
- Use of union-find data structure

Step 1: Assign numbers to the regions in an input binary tree



Step 2: Assign polarities to the leaves

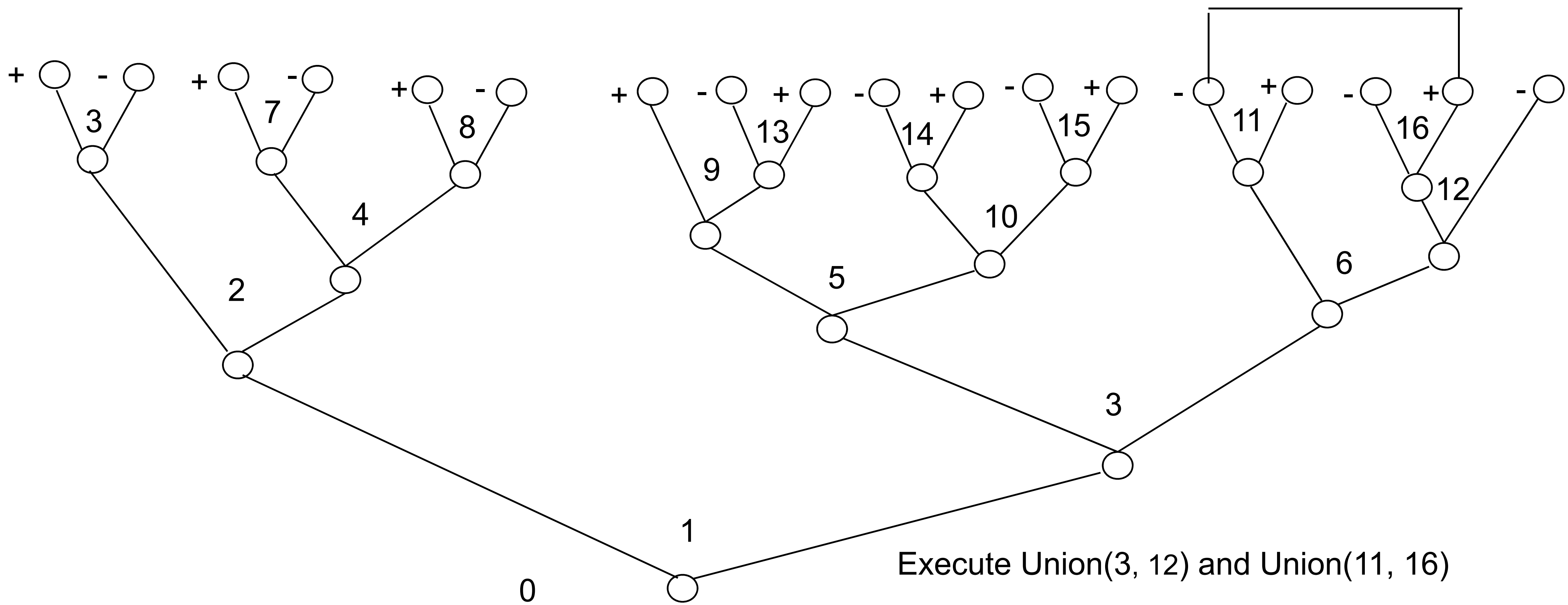
Positive leaf must connect to negative leaf in order to be planar (necessary condition)



Step 3: applying union operation to regions

Left region of left conclusion is unified to right region of right conclusion

Right region of left conclusion is unified to left region of right conclusion



Execute Union(3, 12) and Union(11, 16)

Open Question

- Generalization to the non-planar case
  - A special case to a general open question (*decremental graph connectivity problem*)
  - Already in quadratic time
  - Embedding into general surfaces and applying similar methods in linear time?