Inapproximability of the Standard Pebble Game and Hard to Pebble Graphs

ERIK D. DEMAINE AND QUANQUAN C. LIU

Pebble Games

Games played on DAGs by placing, removing and sliding pebbles

Originally used to model memory used in computation (i.e. register allocation)

Care about bounded indegree DAGs (usually bounded indegree-2 graphs) in practical sense

Using *standard pebble game* $\boldsymbol{O(n/\log n)}$ **space can be used to compute DAG with** $\boldsymbol{n}$ **nodes** (i.e. $DTIME(n) \subseteq DSPACE\left(\frac{n}{\log n}\right)$) [Hopcroft, Paul, Valient 77]

Now: various applications in proof complexity and resolution, cryptography—proofs of work and space using time/space tradeoffs

Minimizing Space Used in Computation

A natural question: what is the minimum amount of space necessary to perform a computation?

Many different types of pebble games used to model space usage in many different settings

- Reversible Pebble Game: used to model reversible computation
- Black-White Pebble Game: used to model non-deterministic computation
- Red-Blue Pebble Game: used to study I/O Complexity

All such games are PSPACE-complete to find minimum number of pebbles to pebble DAG.
[Gilbert, Lengauer, Tarjan 79], [Chan, Lauria, Nordstrom, Vinyals 15], [Hertel, Pitassi 10], [Liu 17]

- Notably, result in [HP10] is for unbounded indegree (and also very large indegree) so less useful in the practical sense
- Open question whether bounded indegree black-white is PSPACE-complete

Approximating Minimum Number of Pebbles Needed to Pebble DAG

The only approximation result for minimizing number of pebbles necessary to pebble a DAG was given by [CLNV15]

- They proved it is PSPACE-hard to approximate the minimum number of pebbles necessary to pebble a DAG to within any *constant additive term* for both the standard and reversible pebble games

We strengthen this result for the standard pebble game to PSPACE-hard to approximate to an additive term of $n^{\frac{1}{3}-\epsilon}$ for all $\epsilon > 0$

Use different approach from that presented in [CLNV15] instead using an approach based on the construction in [GLT79]

Reduce from PSPACE-complete problem Quantified Boolean Formula (QBF)

Standard Pebble Game

Standard pebble game (often called black pebble game) played by placing and sliding black pebbles on DAG

Pebbles can be moved according to a specific set of moves:

- A pebble can be placed on any source node
- A pebble can be removed from any node
- A pebble can be placed on any non-source node if and only if all direct predecessors of the node are pebbled
- A pebble can be slid from a predecessor of a node to the node if and only if all direct predecessors of the node are pebbled

Terminology

Pebbling space cost is the **minimum number of pebbles necessary to pebble DAG**

Pebbling time cost given k pebbles is the **minimum number of moves necessary to pebble DAG using k pebbles**

Given QBF instance $B = Q_1 x_1 \cdots Q_u x_u F$:

- u is the number of number of variables and corresponding quantifiers
- c is the number of clauses in F

K is the width of each literal in each variable

PSPACE-hardness Reduction

Gap-reduction from Quantified Boolean Formula (QBF)

QBF: given formula $B = Q_1 x_1 \cdots Q_u x_u F$ can a setting of all existential variables make F true for all truth settings of universal variables

Given a QBF instance B construct DAG G such that:

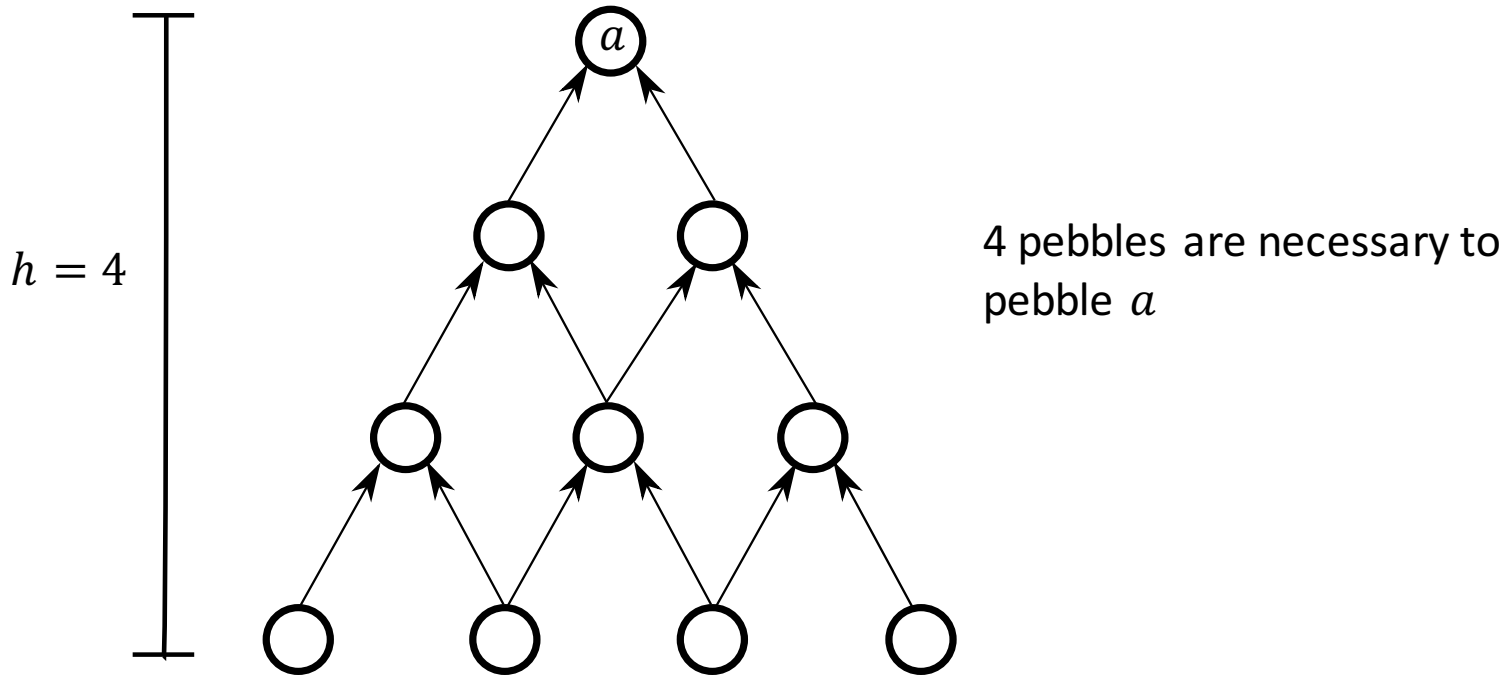
- G can be pebbled using $3Ku + 4K + 1$ pebbles if and only if B is a true instance
- Otherwise $3Ku + 5K$ pebbles are necessary to pebble G

We construct the following gadgets:

- Variable gadgets
- Clause gadgets
- Quantifier blocks

Important Subgraphs

Pyramid graphs Π_h were proven to require h pebbles to pebble the apex where h is the height of the pyramid [GLT79]

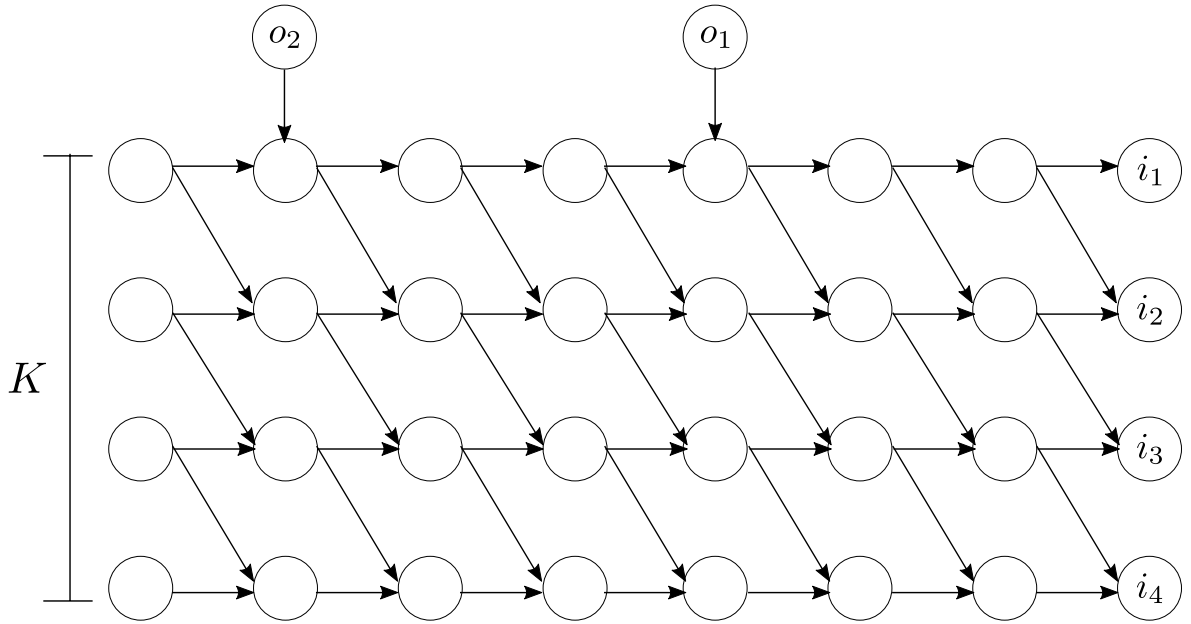


4 pebbles are necessary to pebble a

Important Subgraphs

Road graphs are graphs that require $w + |O| - 1$ pebbles to pebble a set of $O \subseteq \{o_1, \cdots, o_w\}$ outputs where w is the width of the road graph

In our constructions, $w = K$



5 pebbles are necessary to pebble o_1 and o_2 persistently with pebbles

Normal and Regular Strategies

A *normal strategy* [GLT79]:

- Once a pebble has been placed on a pyramid, no pebbles are placed on nodes outside of the pyramid until the apex pebbled
- All other pebbles are removed from the pyramid
- No unnecessary pebble placements anywhere (*frugal)*

A *regular strategy*:

- Once a pebble has been placed on a road graph, no pebbles are placed on nodes outside of the road graph until all desired outputs pebbled
- All other pebbles are removed from the road graph.

Any strategy can be transformed into a *normal* and *regular* strategy.

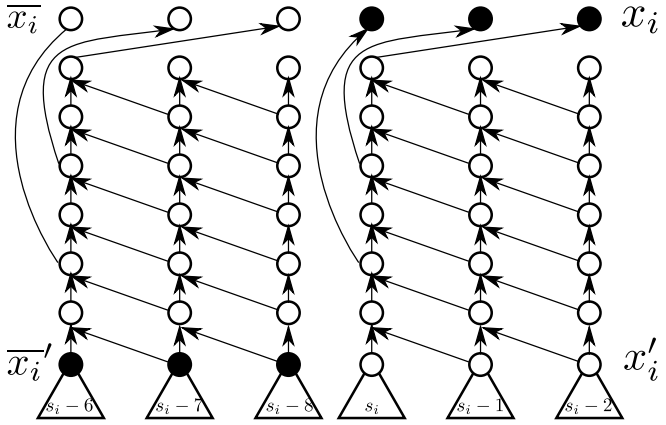
Variable Gadgets

Variable gadgets are used to represent the variables in B

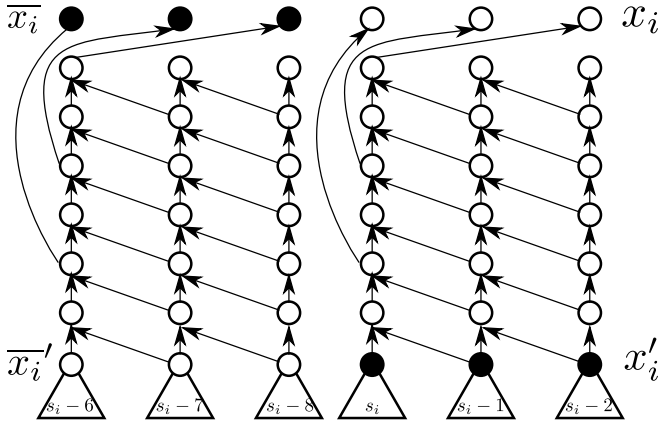
There exists a gadget for each **literal**

Each literal can be set in the true or false configuration

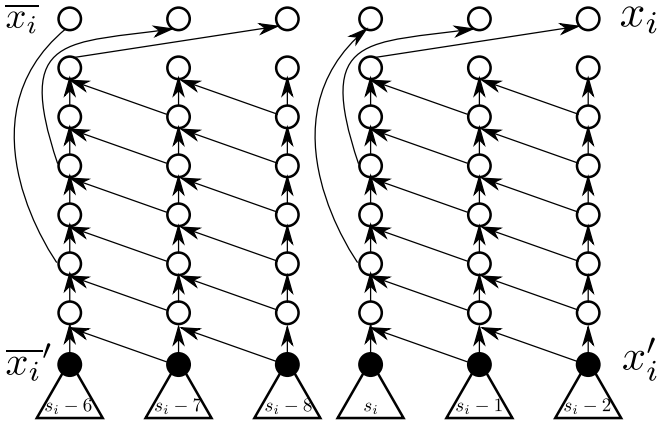
There are three possible valid configurations of the variable gadgets (made of road graphs)



$x_i = True;\ \overline{x_i} = False$ (***true***)



$x_i = False;\ \overline{x_i} = True$ (***false)***



$x_i = False;\ \overline{x_i} = False$ (***double false)***

Clause Gadgets

Checks whether setting of variables satisfies CNF formula F

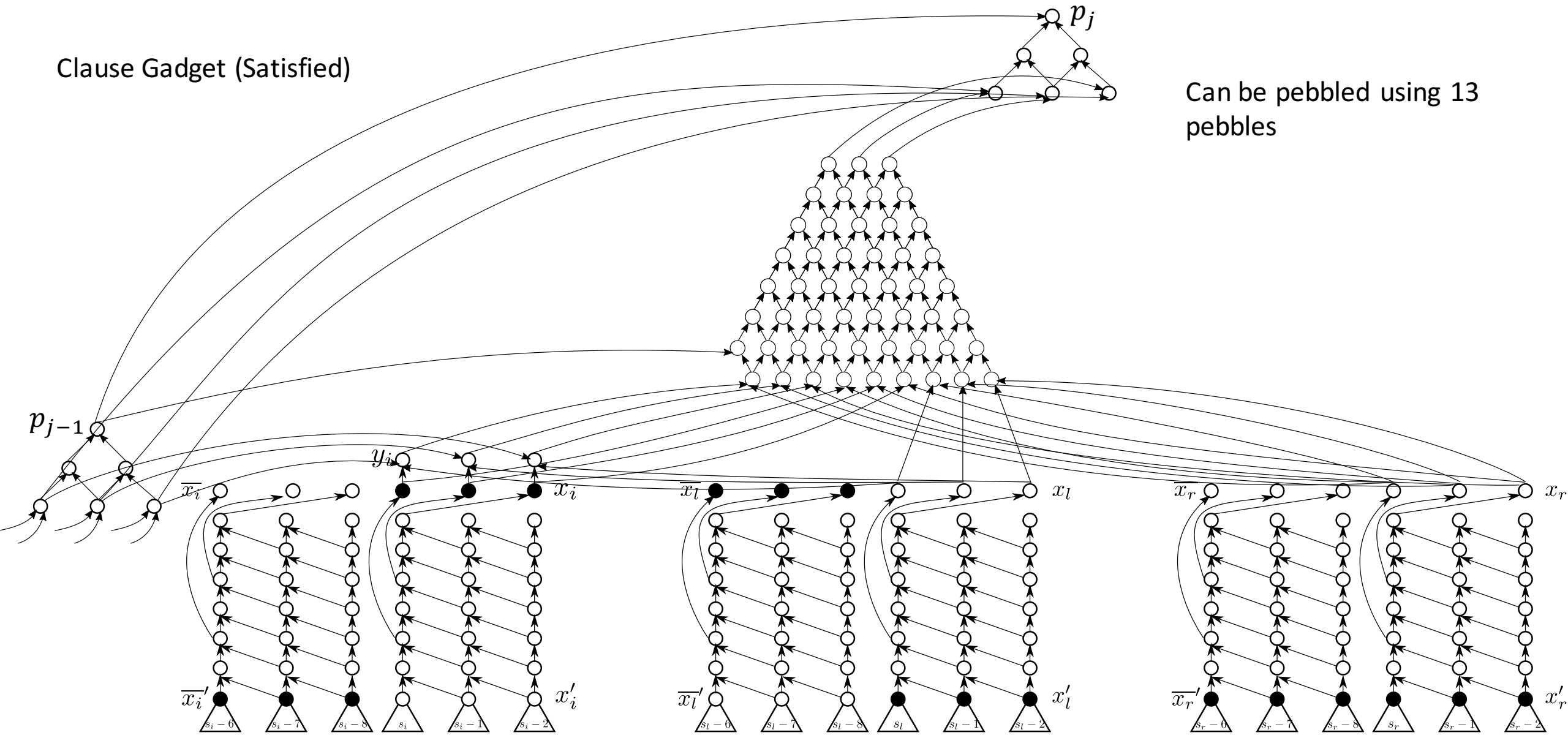
Constructed with pyramid whose bottom nodes connected to the three literals that are present in the clause

Requires $4K + 1$ pebbles if at least one literal true

Otherwise, requires $5K$ pebbles to pebble an unsatisfied clause

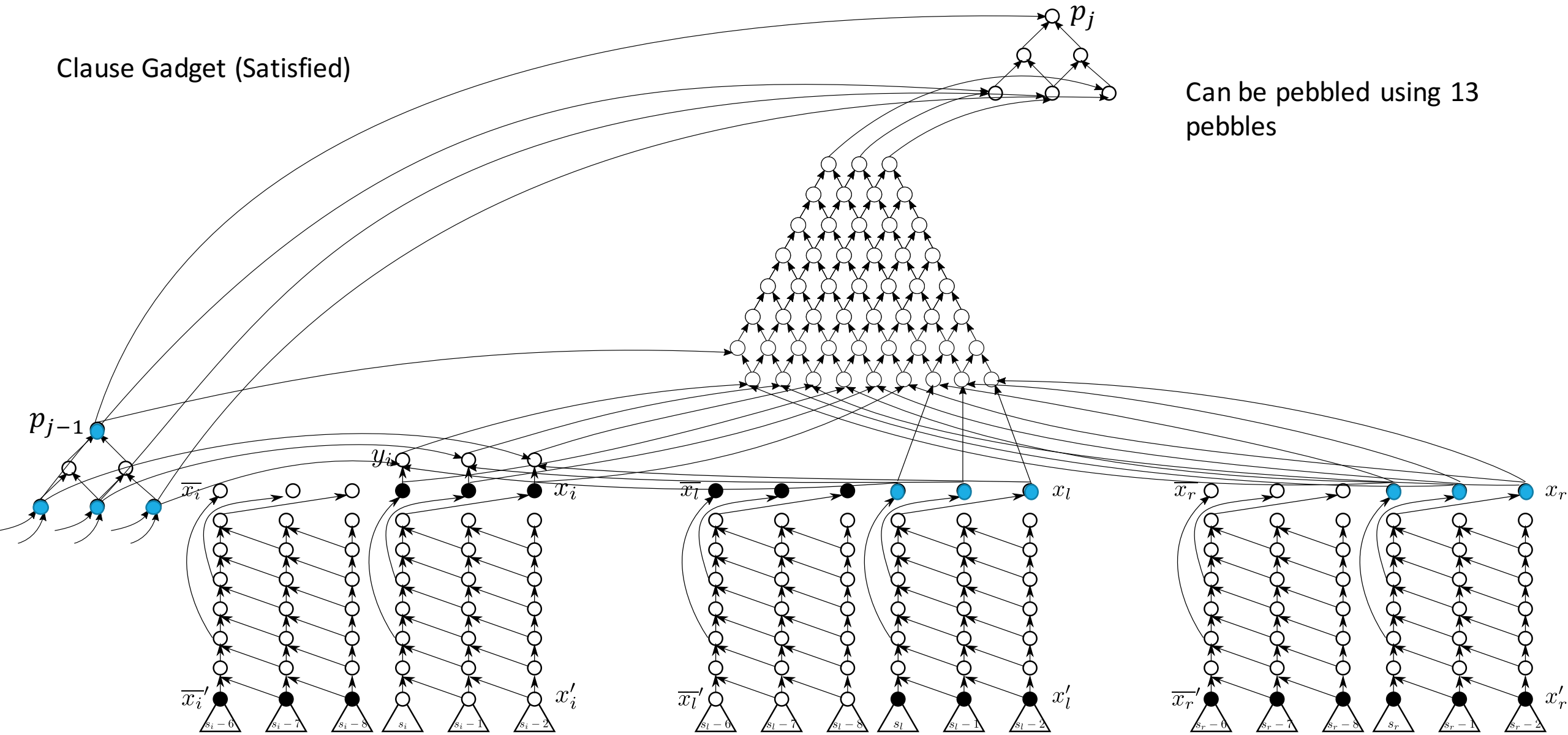
Clause Gadget (Satisfied)

Can be pebbled using 13 pebbles



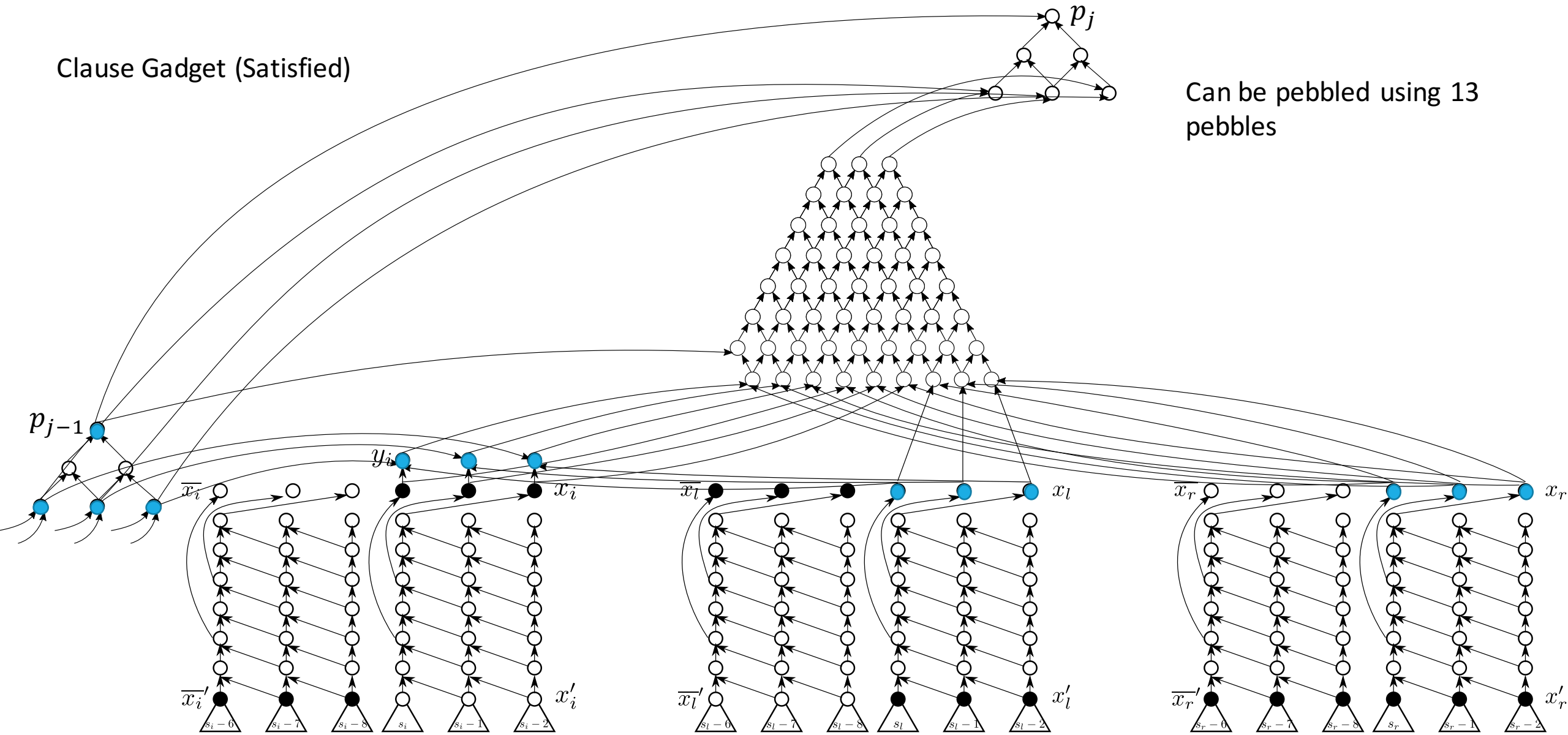
Clause Gadget (Satisfied)

Can be pebbled using 13 pebbles



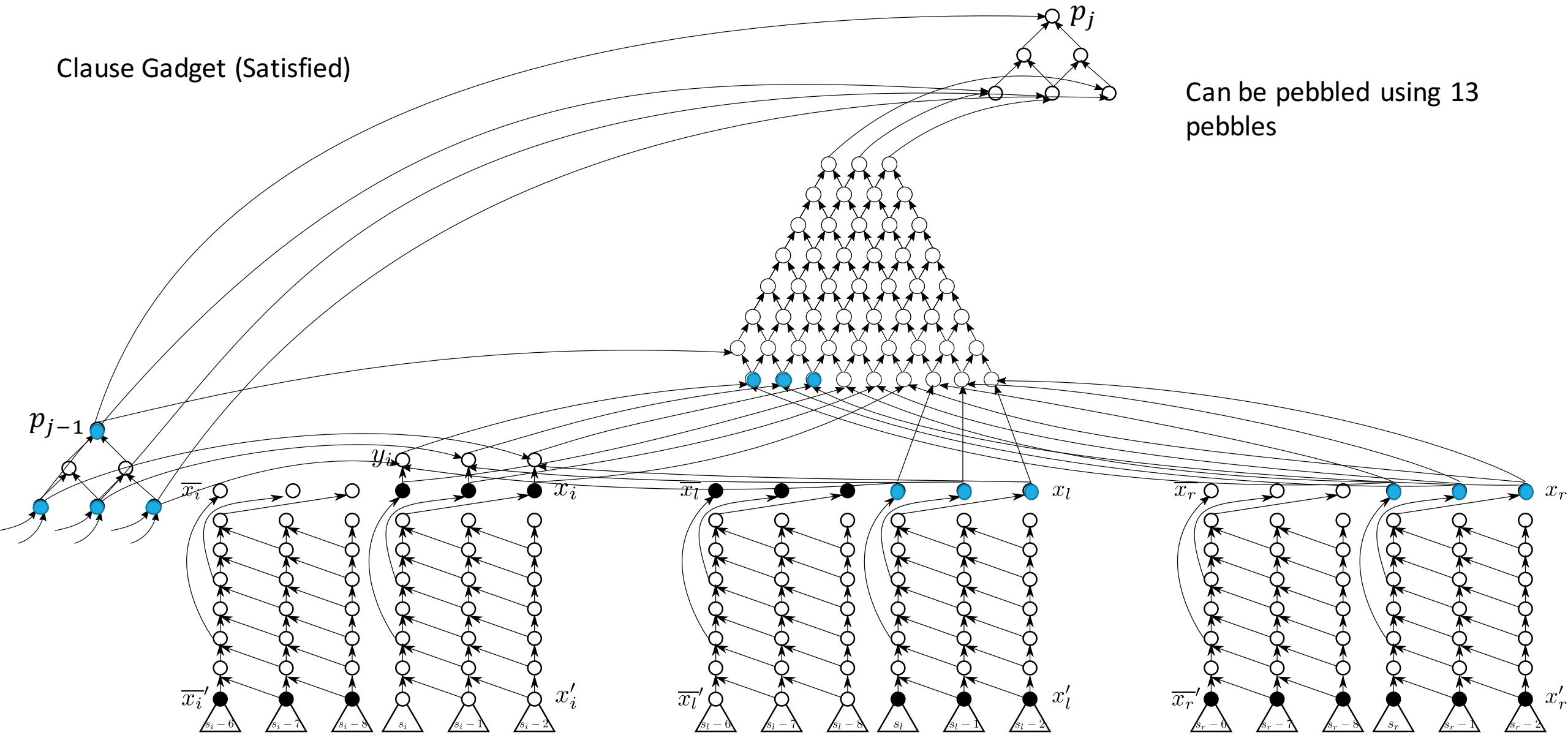
Clause Gadget (Satisfied)

Can be pebbled using 13 pebbles



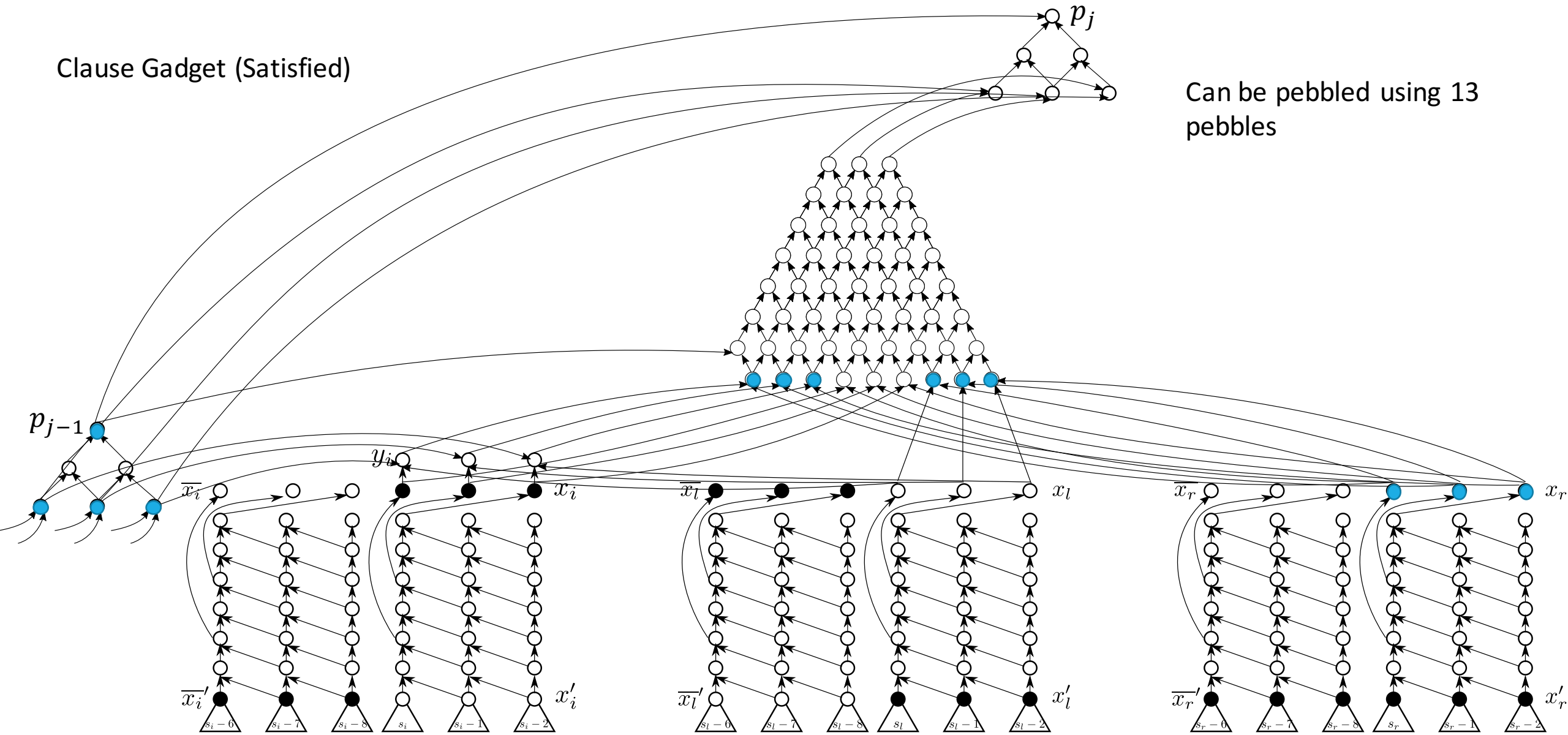
Clause Gadget (Satisfied)

Can be pebbled using 13 pebbles



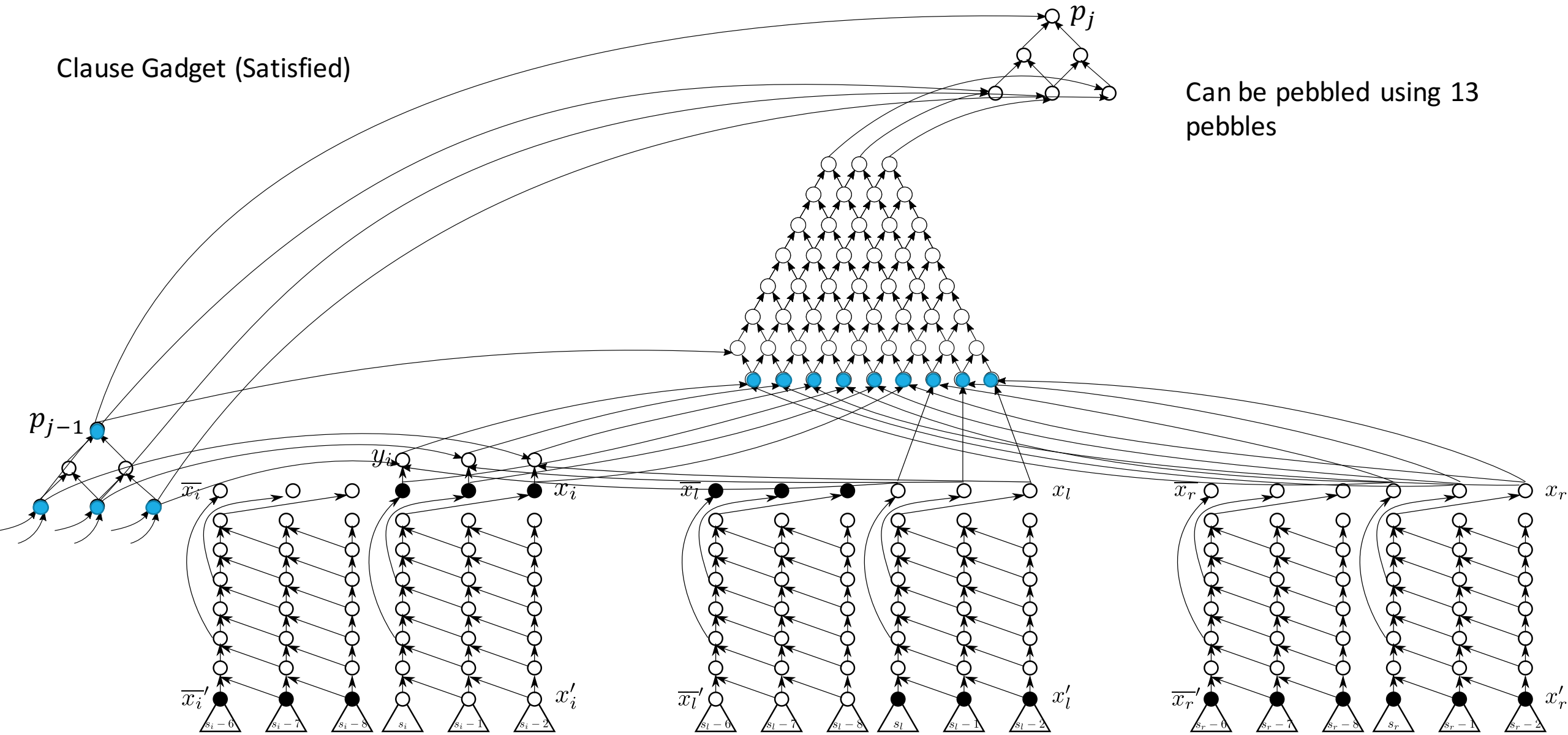
Clause Gadget (Satisfied)

Can be pebbled using 13 pebbles



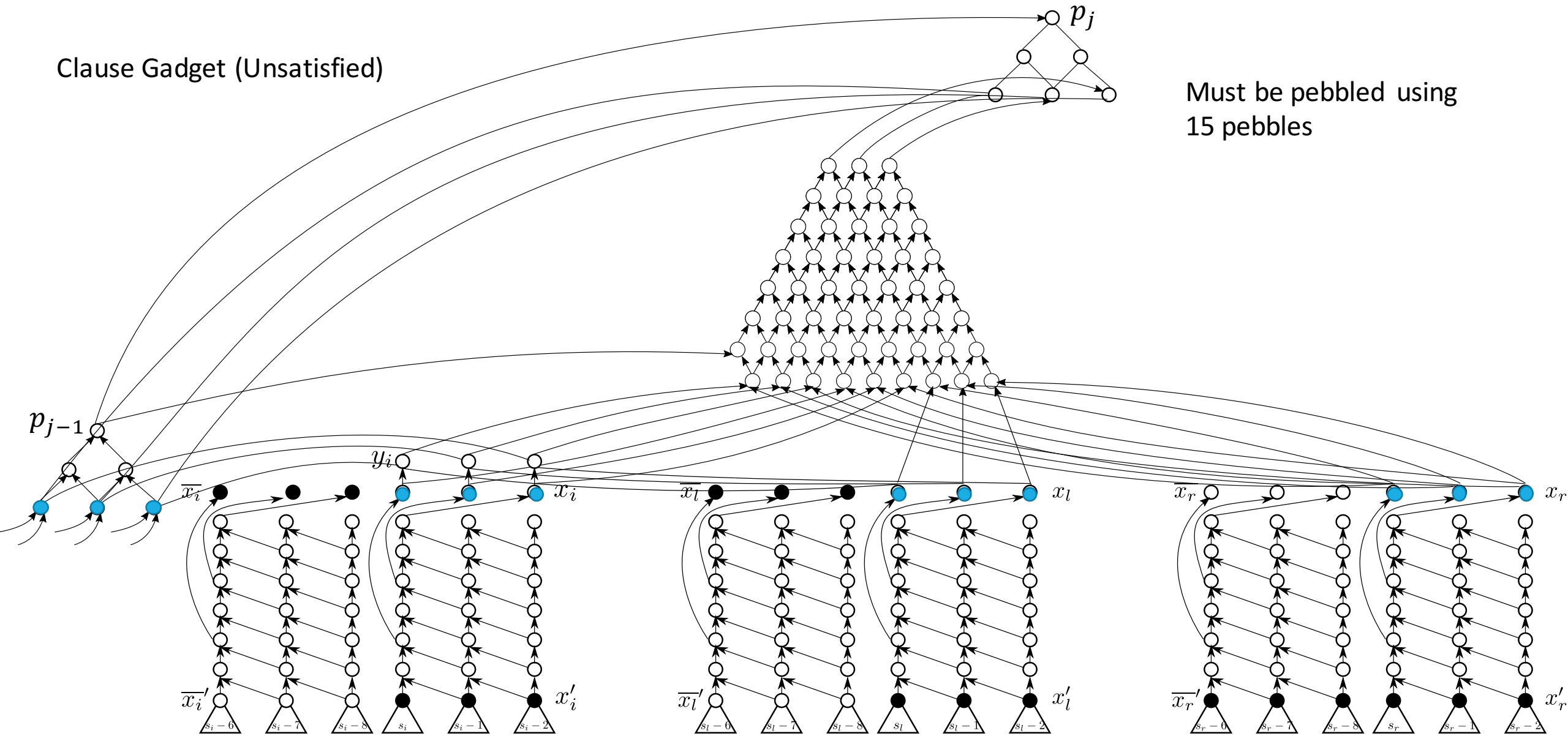
Clause Gadget (Satisfied)

Can be pebbled using 13 pebbles



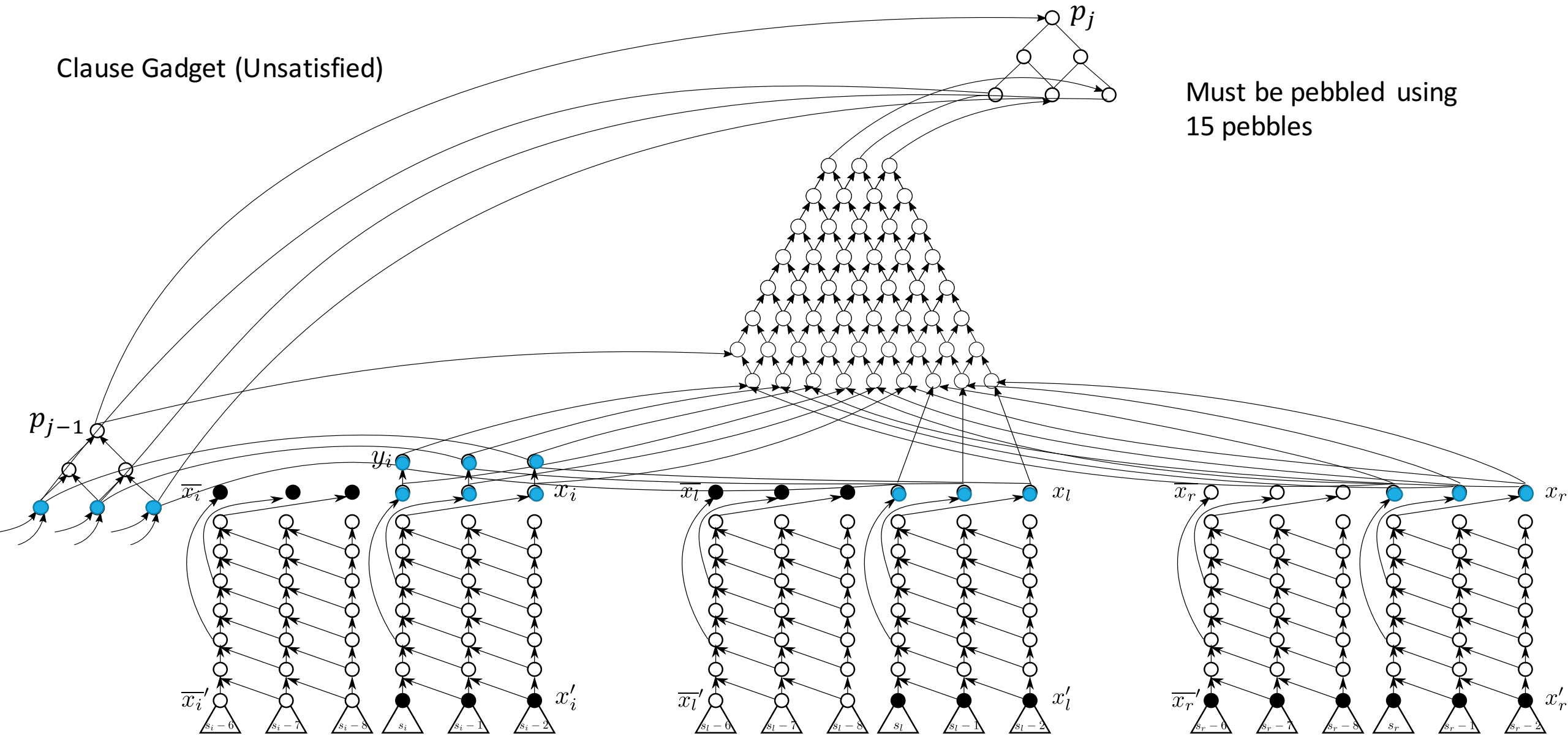
Clause Gadget (Unsatisfied)

Must be pebbled using 15 pebbles



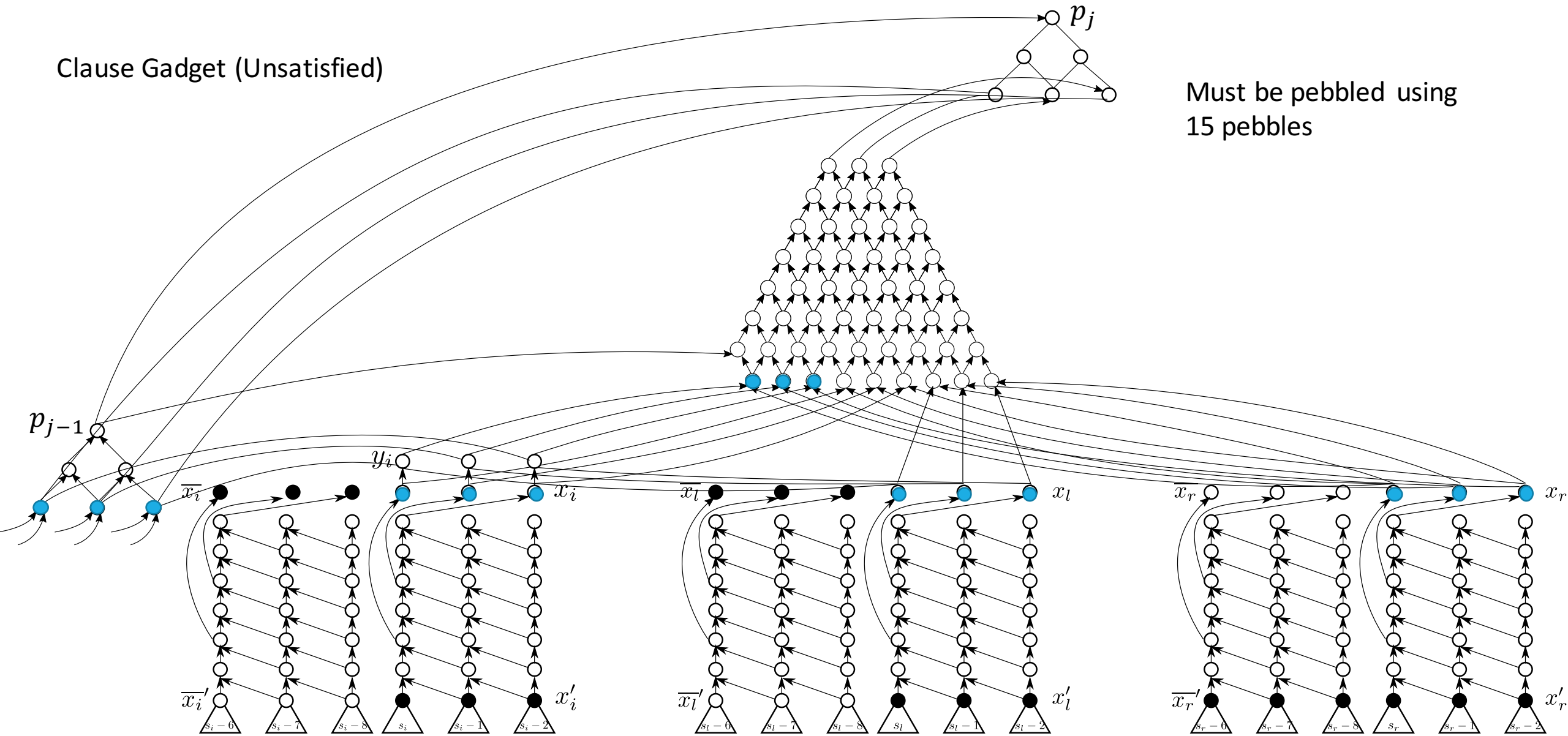
Clause Gadget (Unsatisfied)

Must be pebbled using 15 pebbles



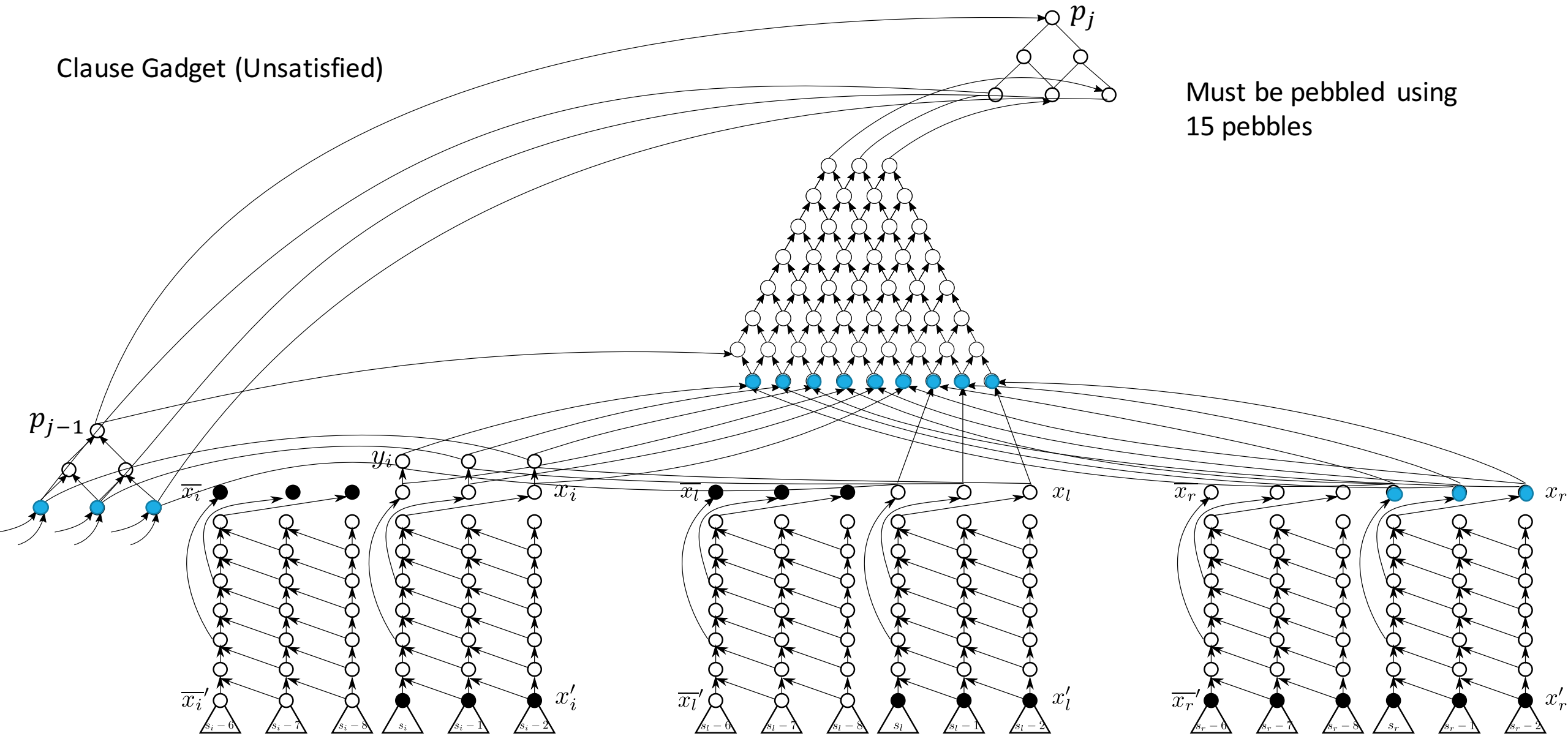
Clause Gadget (Unsatisfied)

Must be pebbled using 15 pebbles



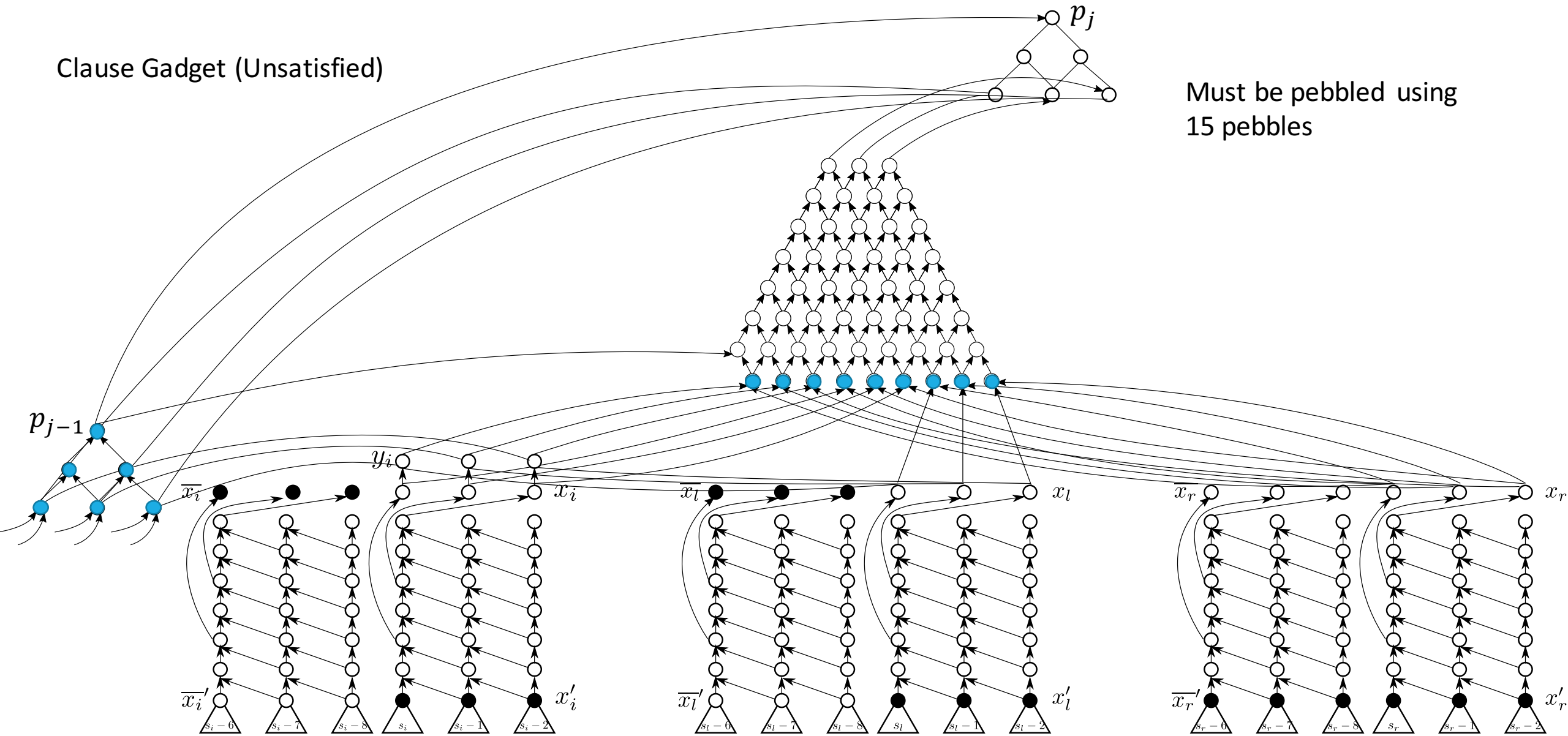
Clause Gadget (Unsatisfied)

Must be pebbled using 15 pebbles



Clause Gadget (Unsatisfied)

Must be pebbled using 15 pebbles



Quantifier Blocks

Quantifier blocks used to make sure clause gadgets are evaluated when all universal variables are set to either:

- Both the ***True*** and the ***False*** configurations
- ***Double False*** configuration

And when existential variables are set in a valid configuration

All quantifier blocks contain the variable gadget of the variable it quantifies

Universal quantifier blocks are pebbled twice if x_i is first set to true and then false or once if x_i is set to double false

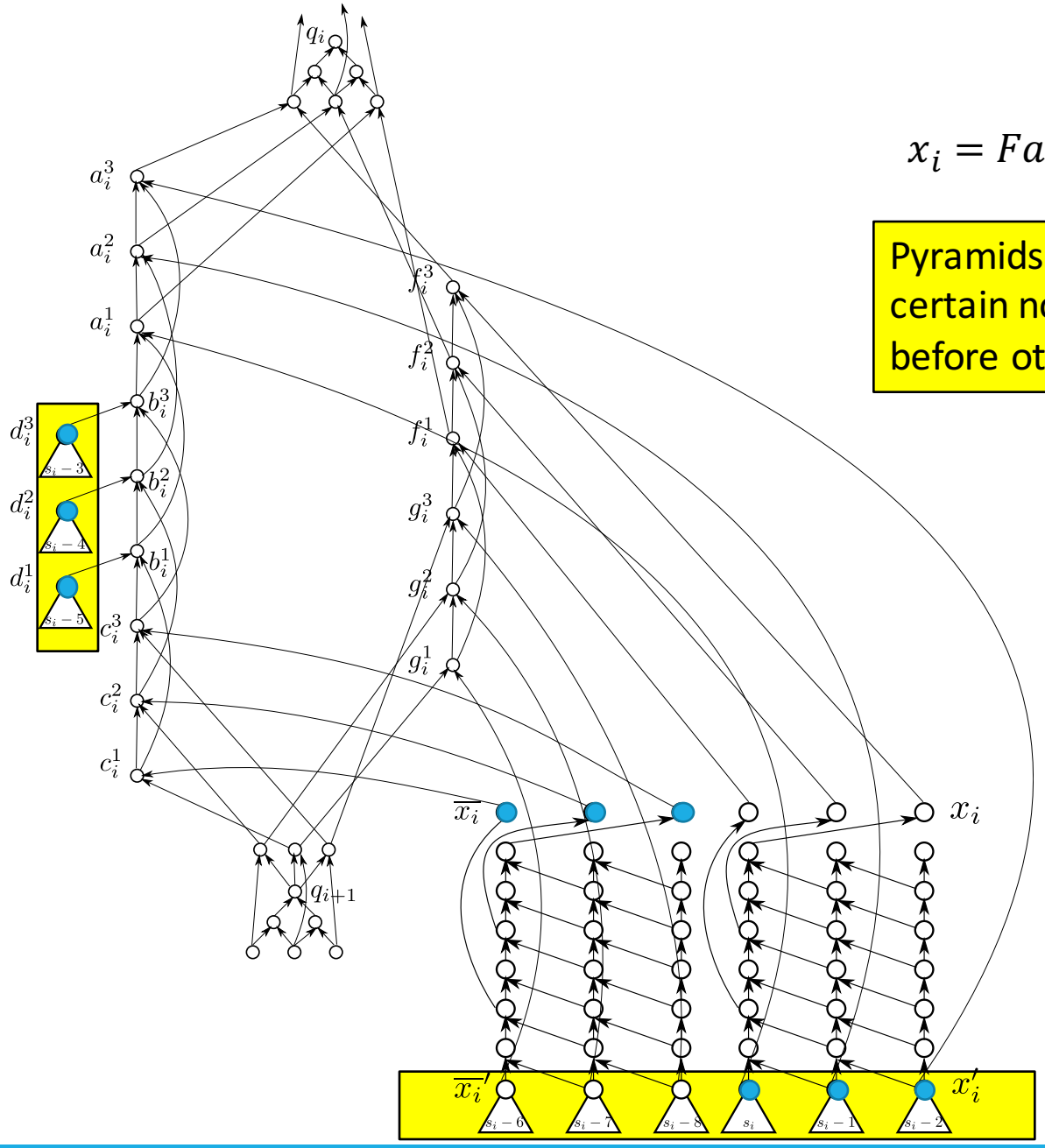
Existential quantifier blocks are pebbled only once regardless of the configuration of literals

Last clause connected to last quantifier block

Universal Quantifier Block

$x_i = False;\ \overline{x_i} = True$

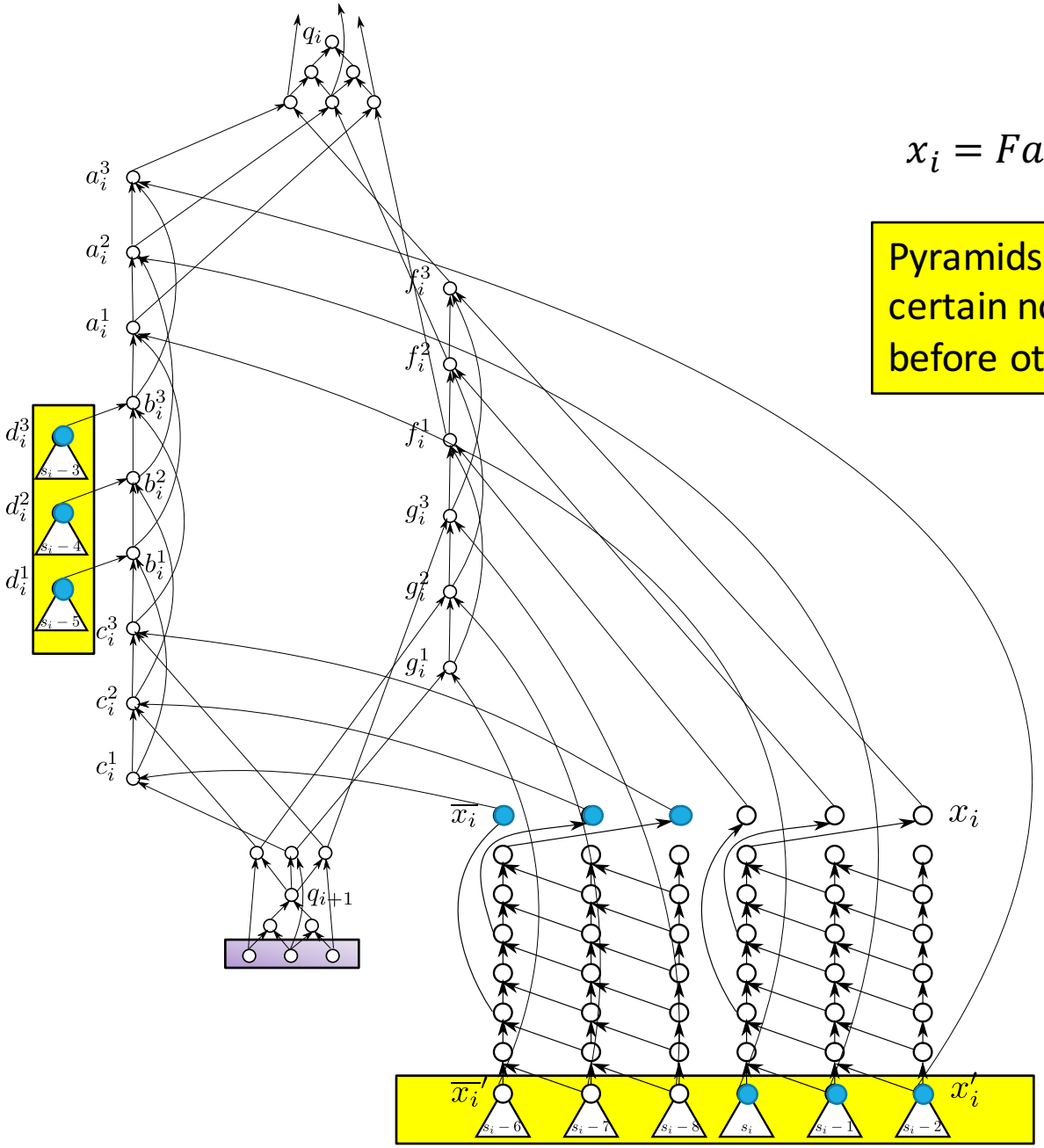
Pyramids are used to make sure certain nodes are pebbled before others



Universal Quantifier Block

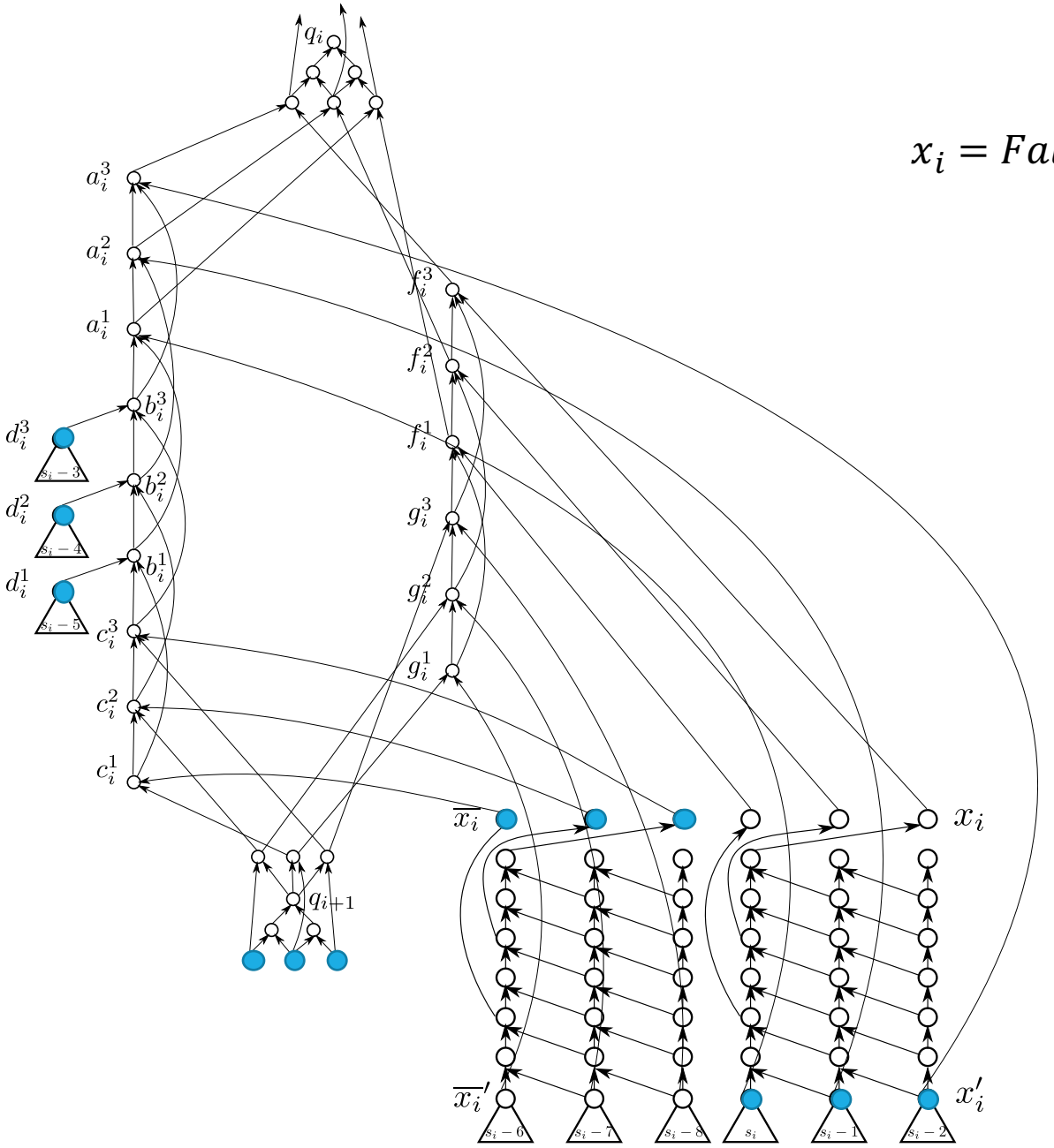
$x_i = False;\ \overline{x_i} = True$

Pyramids are used to make sure certain nodes are pebbled before others



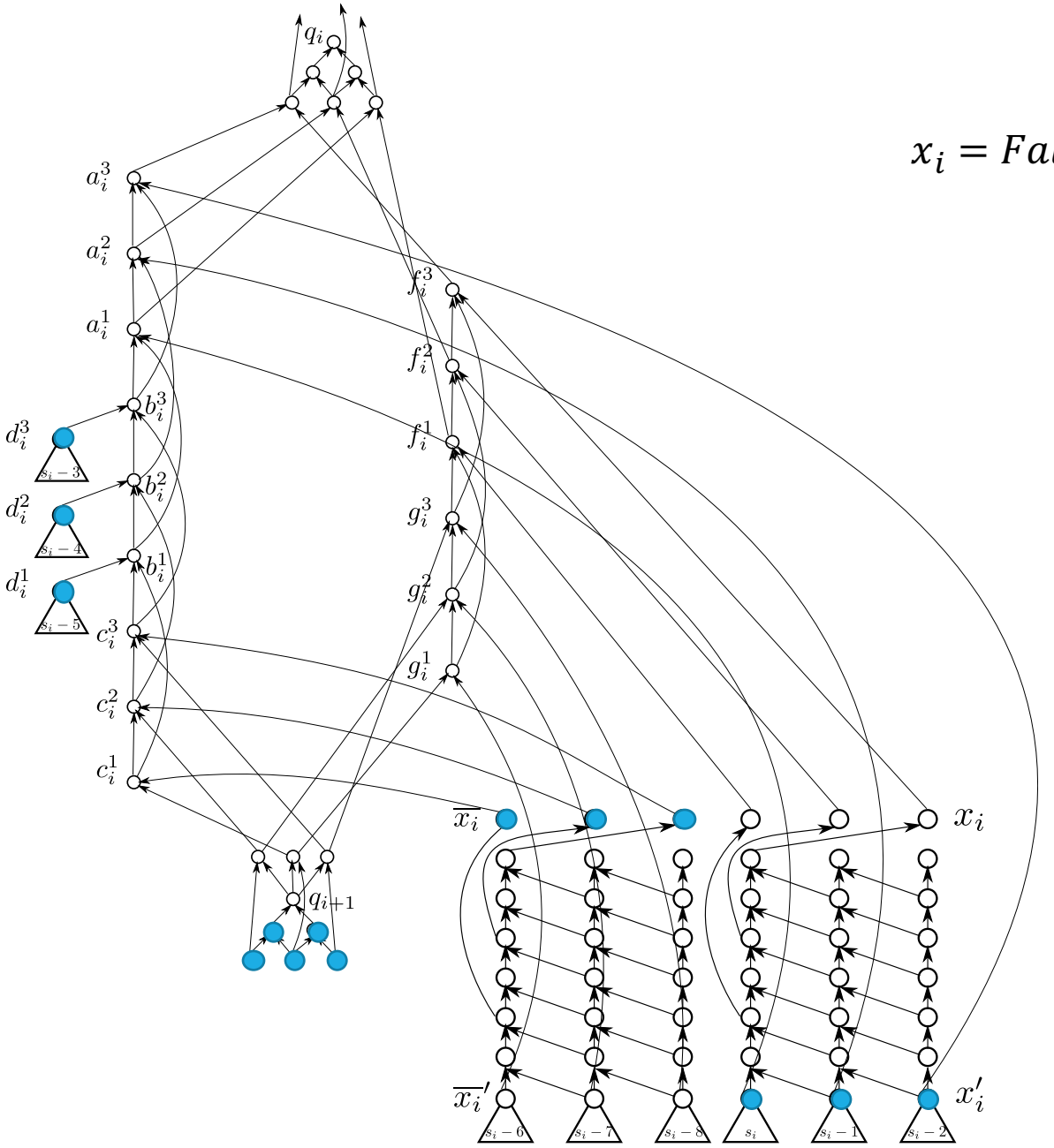
Universal Quantifier Block

$x_i = False;\ \overline{x_i} = True$



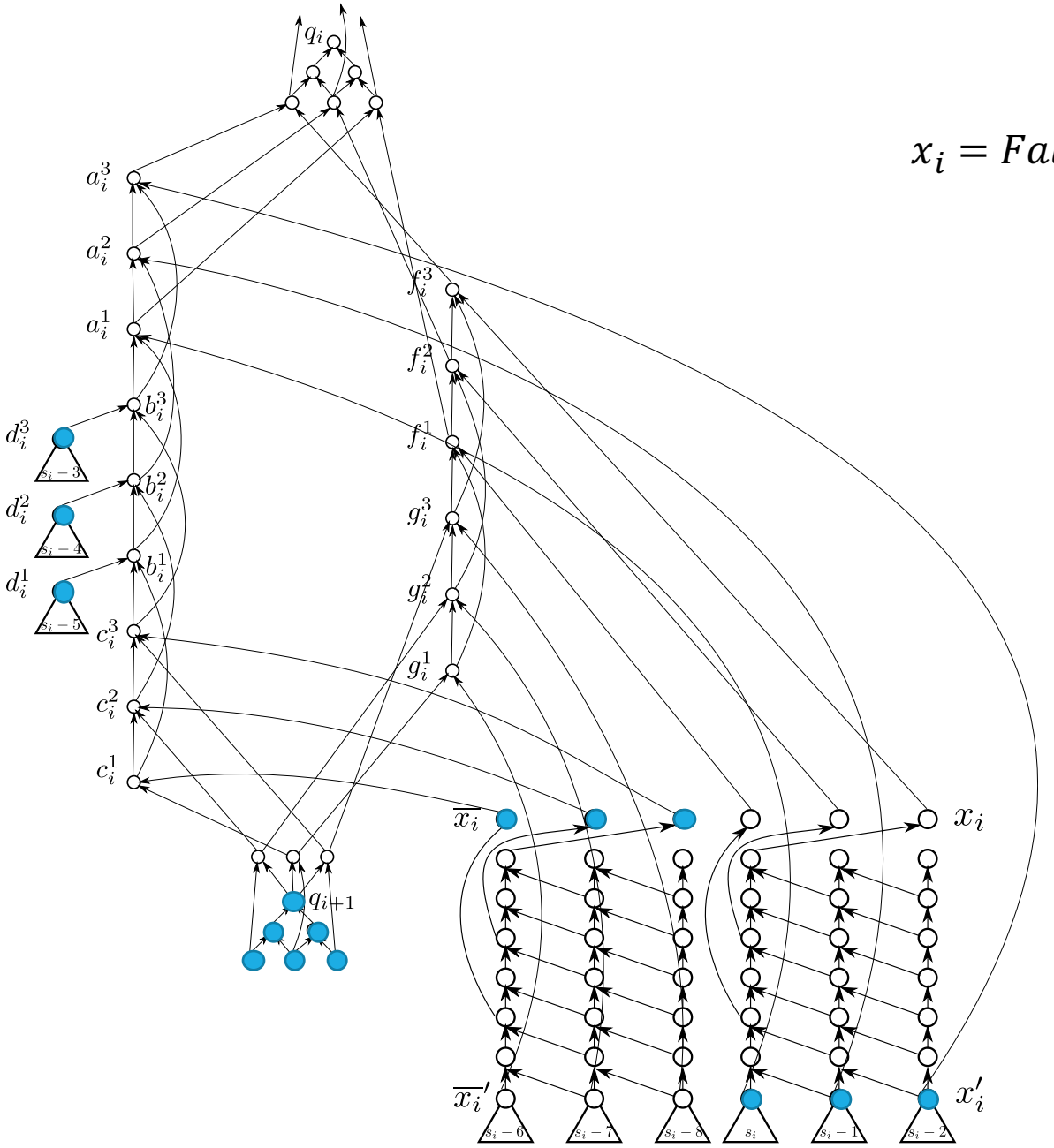
Universal Quantifier Block

$x_i = False;\ \overline{x_i} = True$



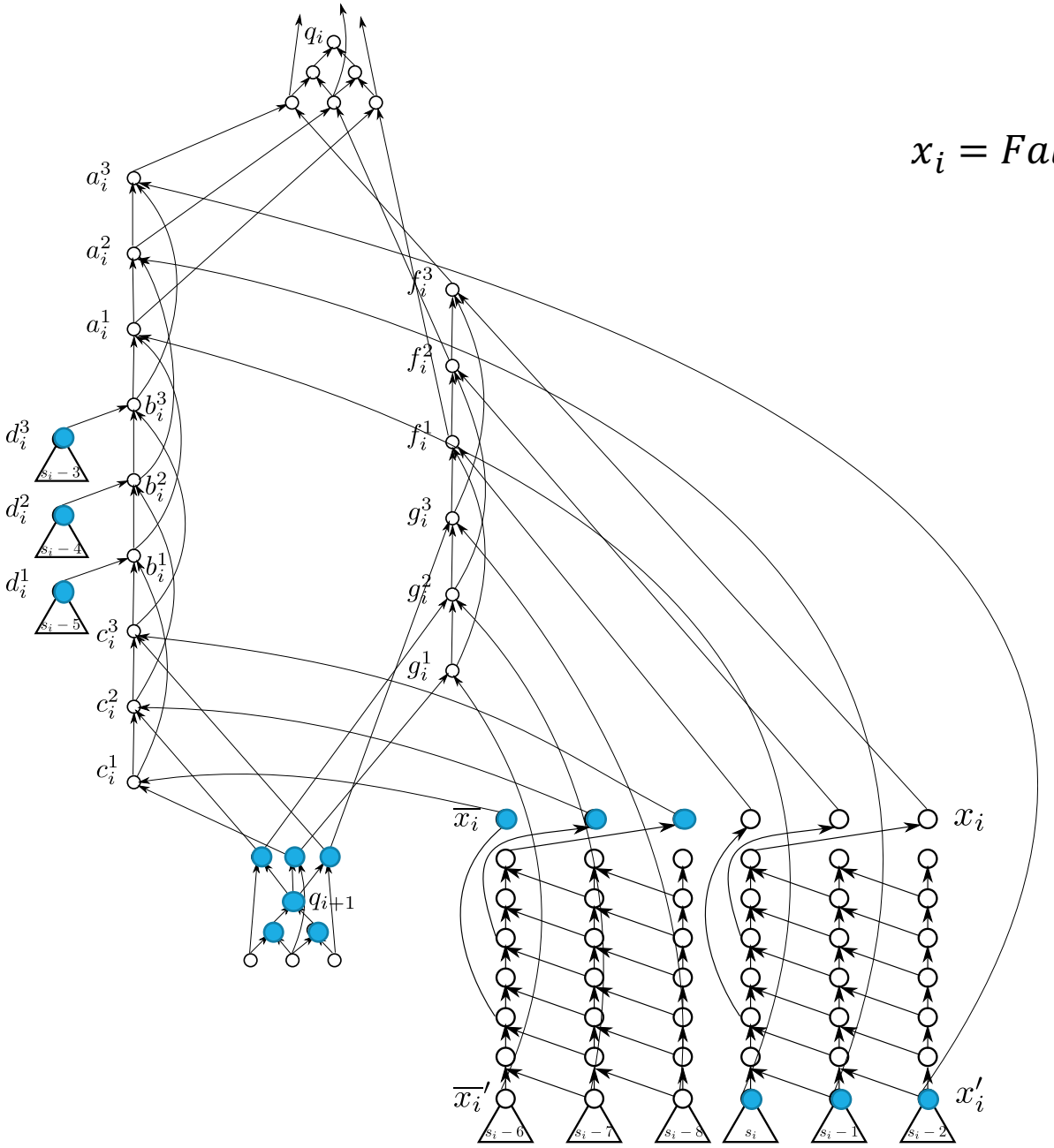
Universal Quantifier Block

$x_i = False;\ \overline{x_i} = True$



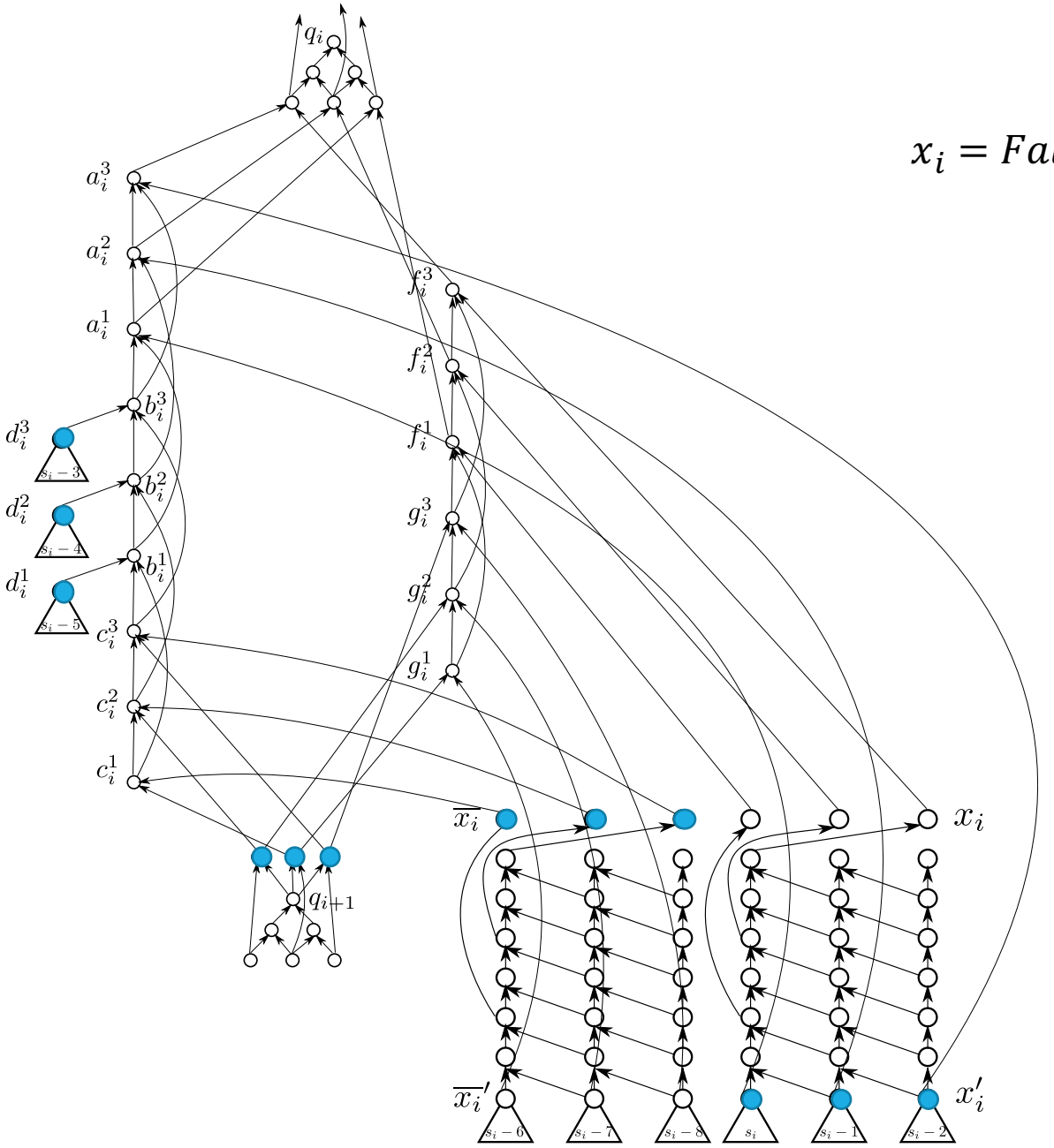
Universal Quantifier Block

$x_i = False;\ \overline{x_i} = True$



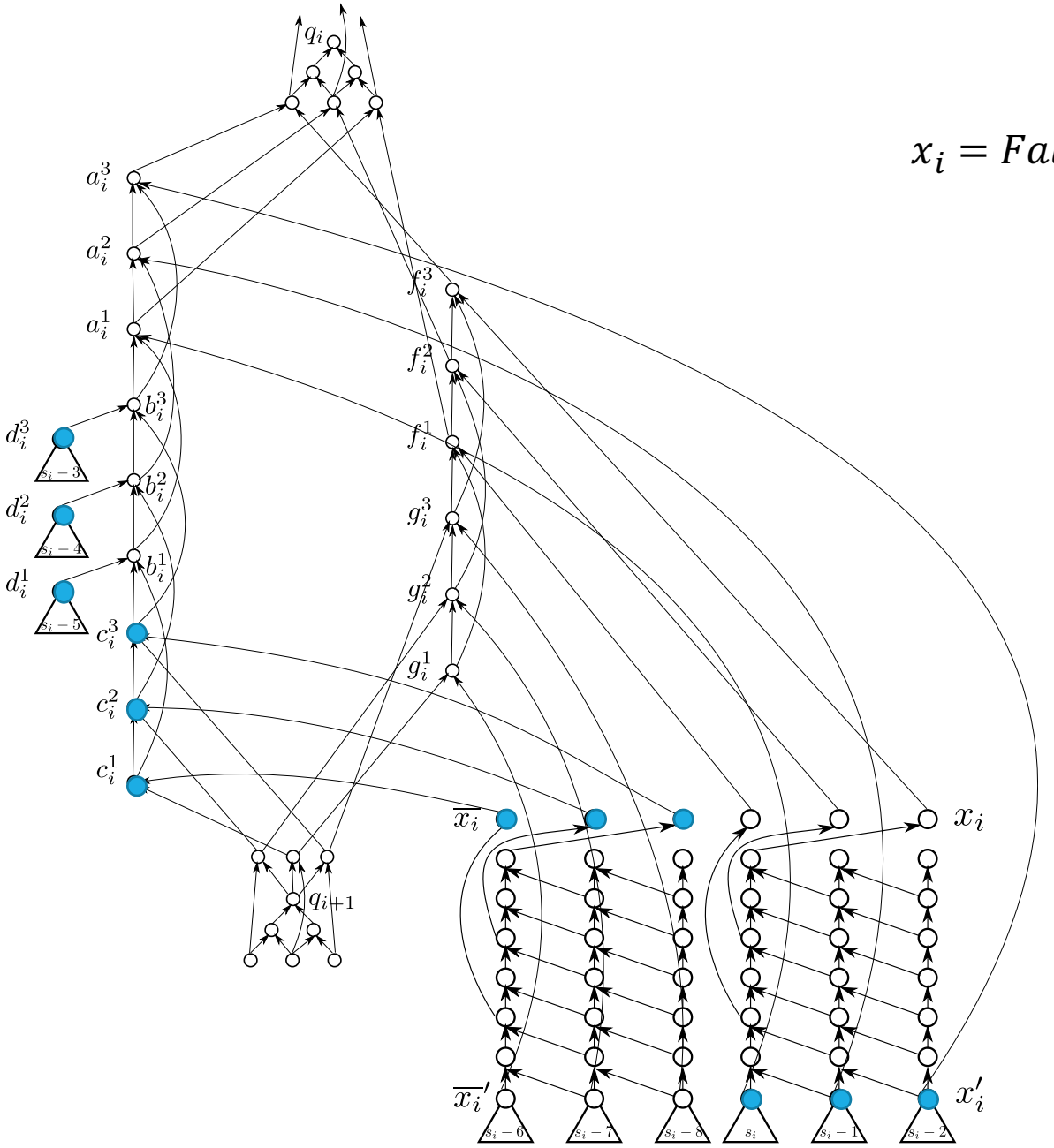
Universal Quantifier Block

$x_i = False;\ \overline{x_i} = True$



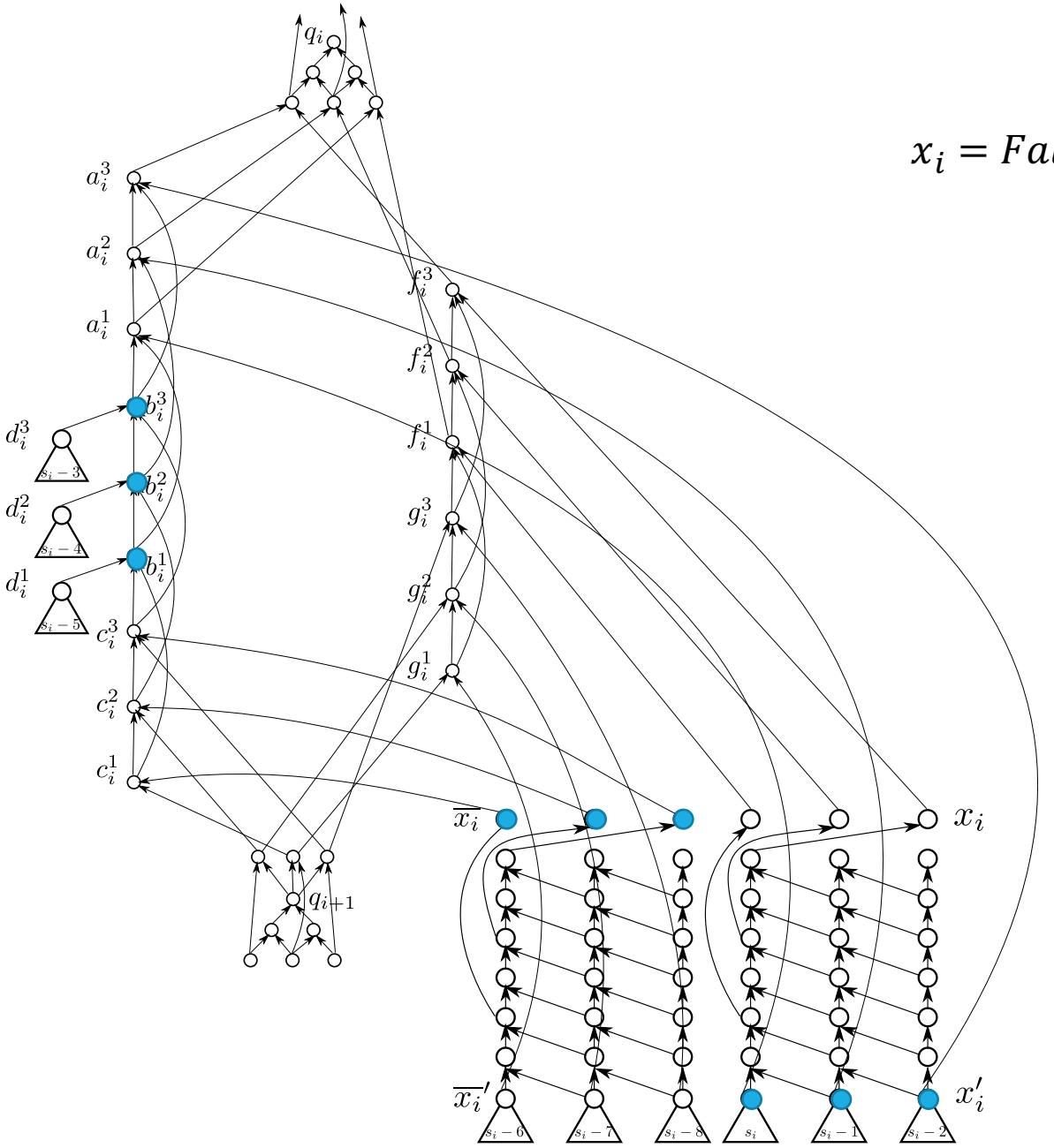
Universal Quantifier Block

$x_i = False;\ \overline{x_i} = True$



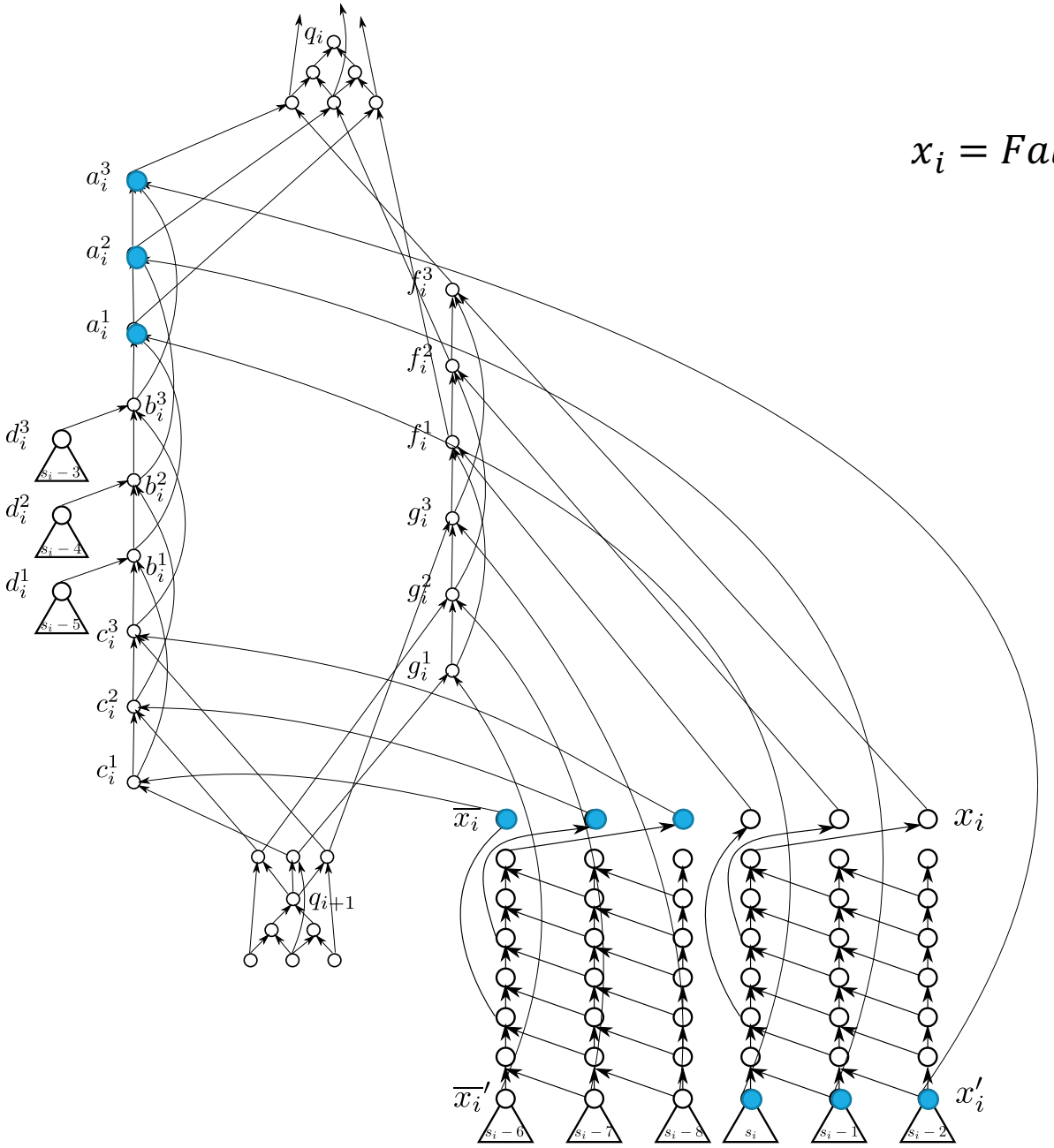
Universal Quantifier Block

$x_i = False;\ \overline{x_i} = True$



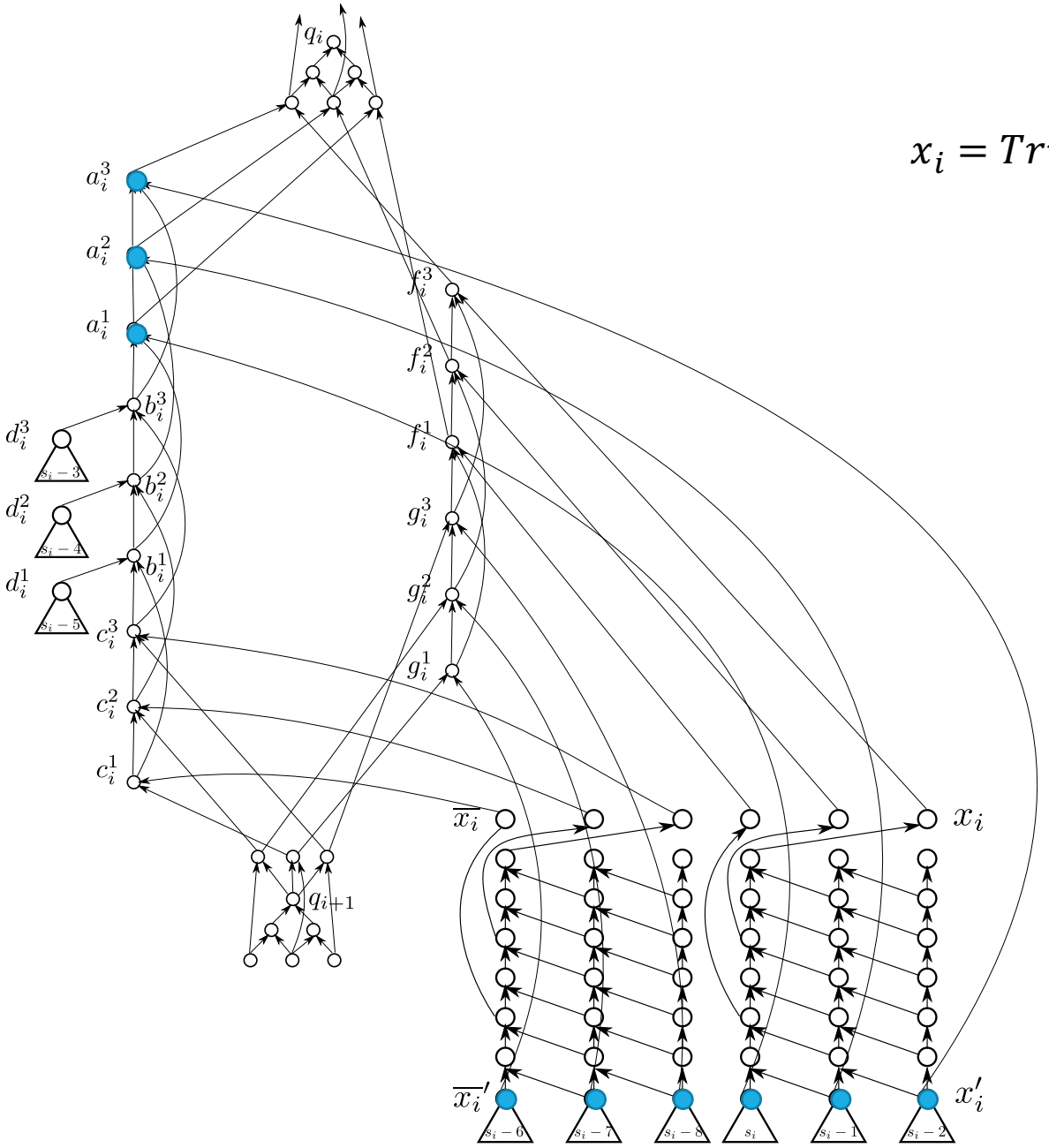
Universal Quantifier Block

$x_i = False;\ \overline{x_i} = True$



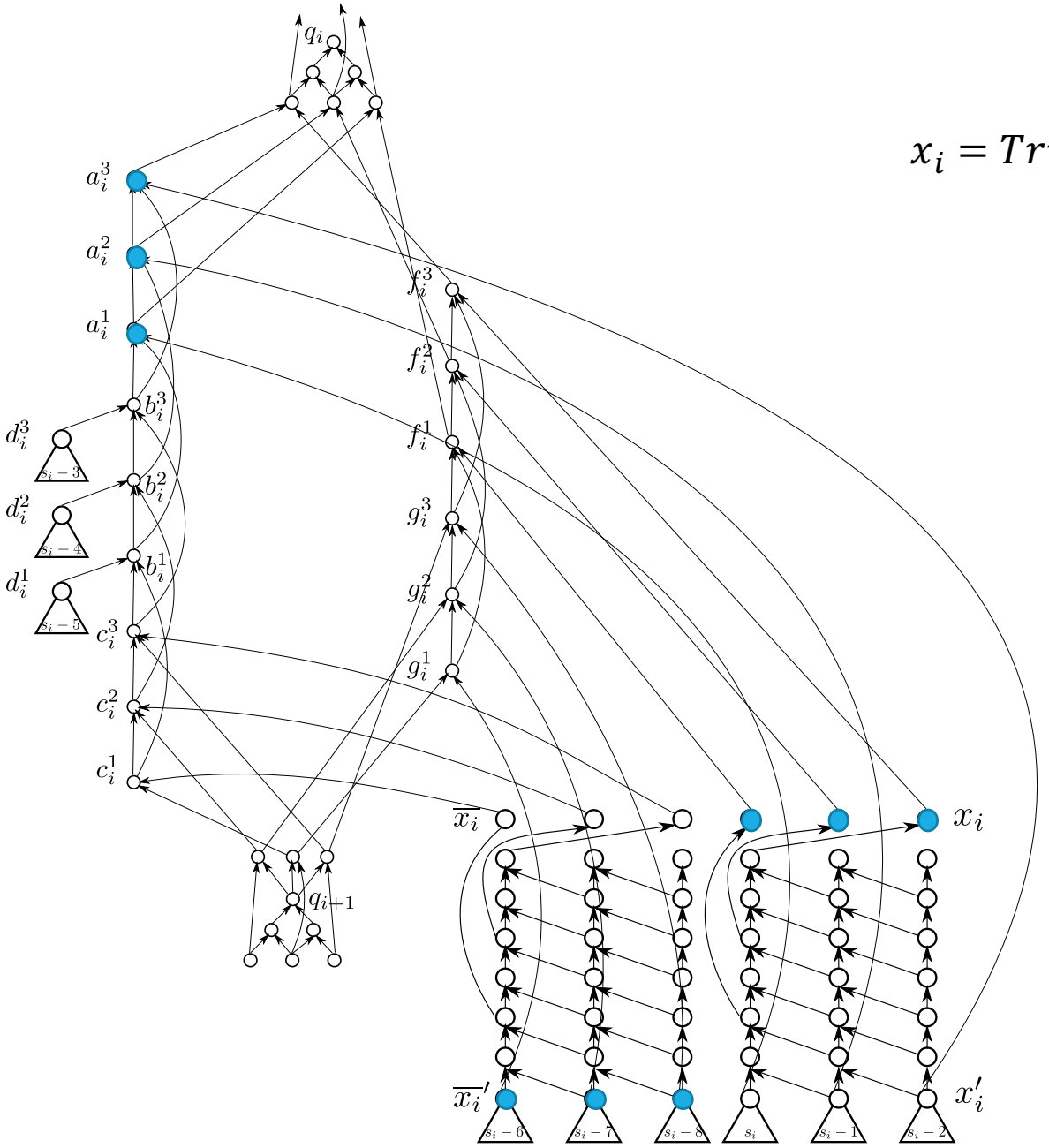
Universal Quantifier Block

$x_i = True;\ \overline{x_i} = False$



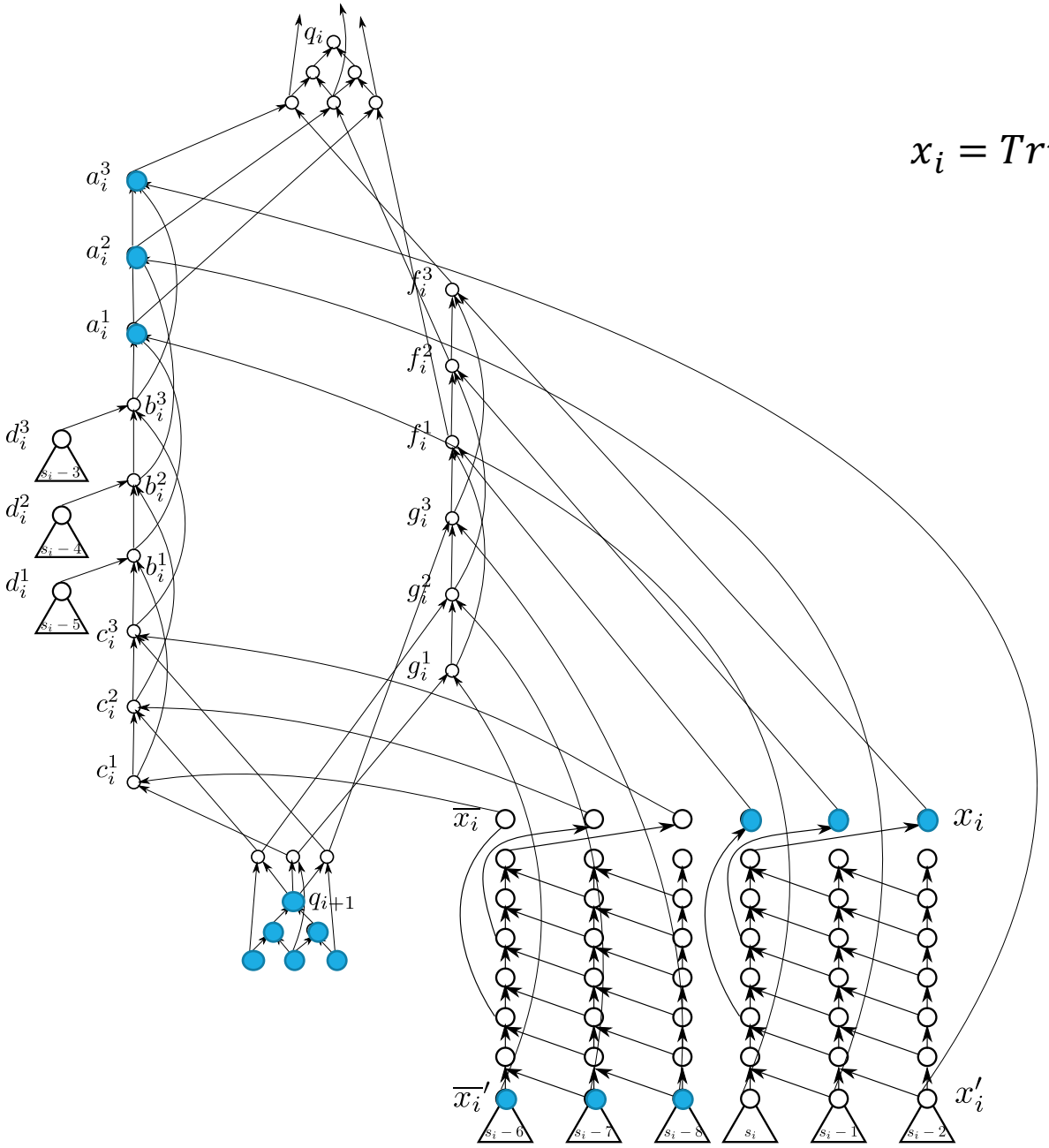
Universal Quantifier Block

$x_i = True;\ \overline{x_i} = False$



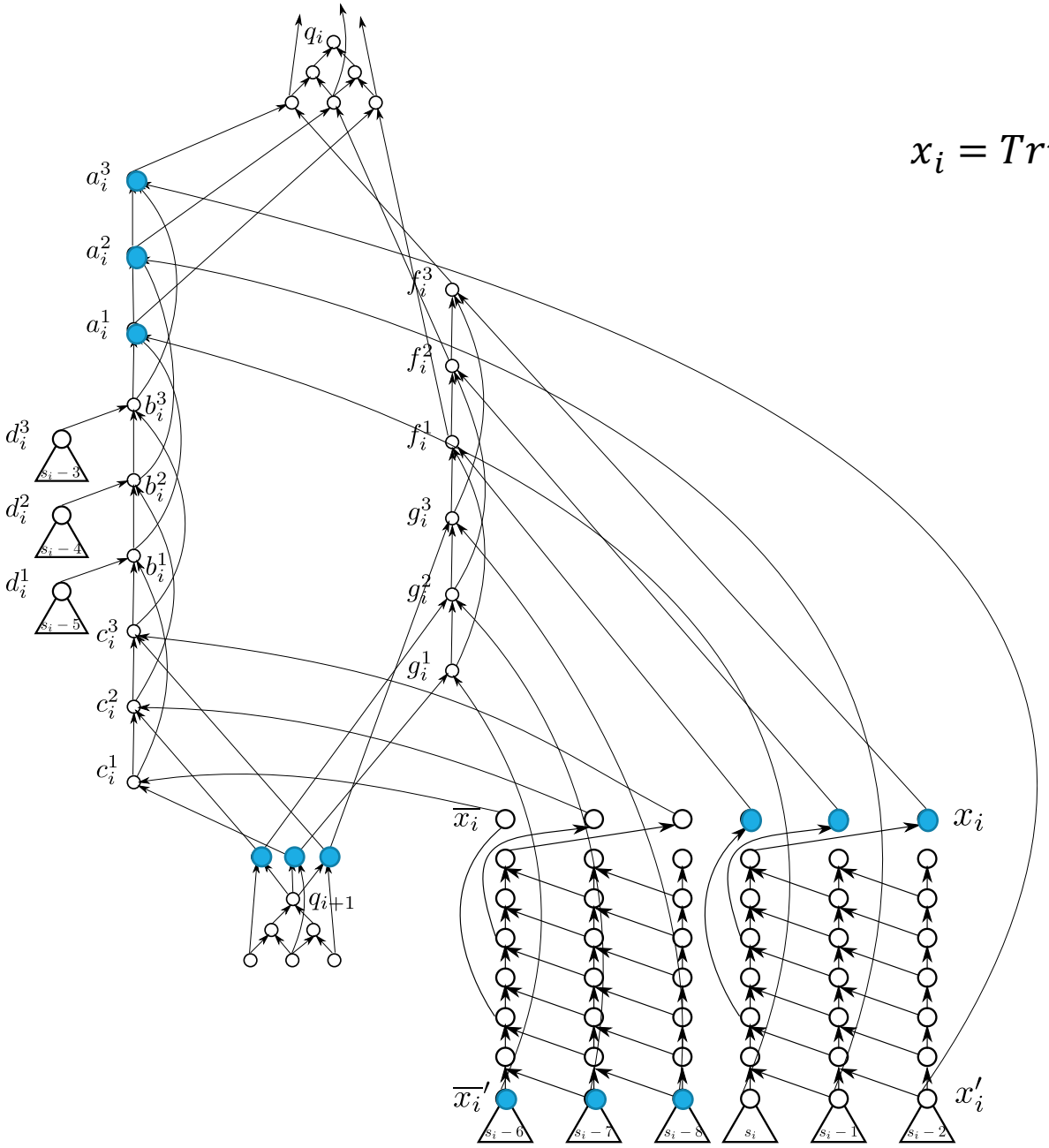
Universal Quantifier Block

$x_i = True;\ \overline{x_i} = False$



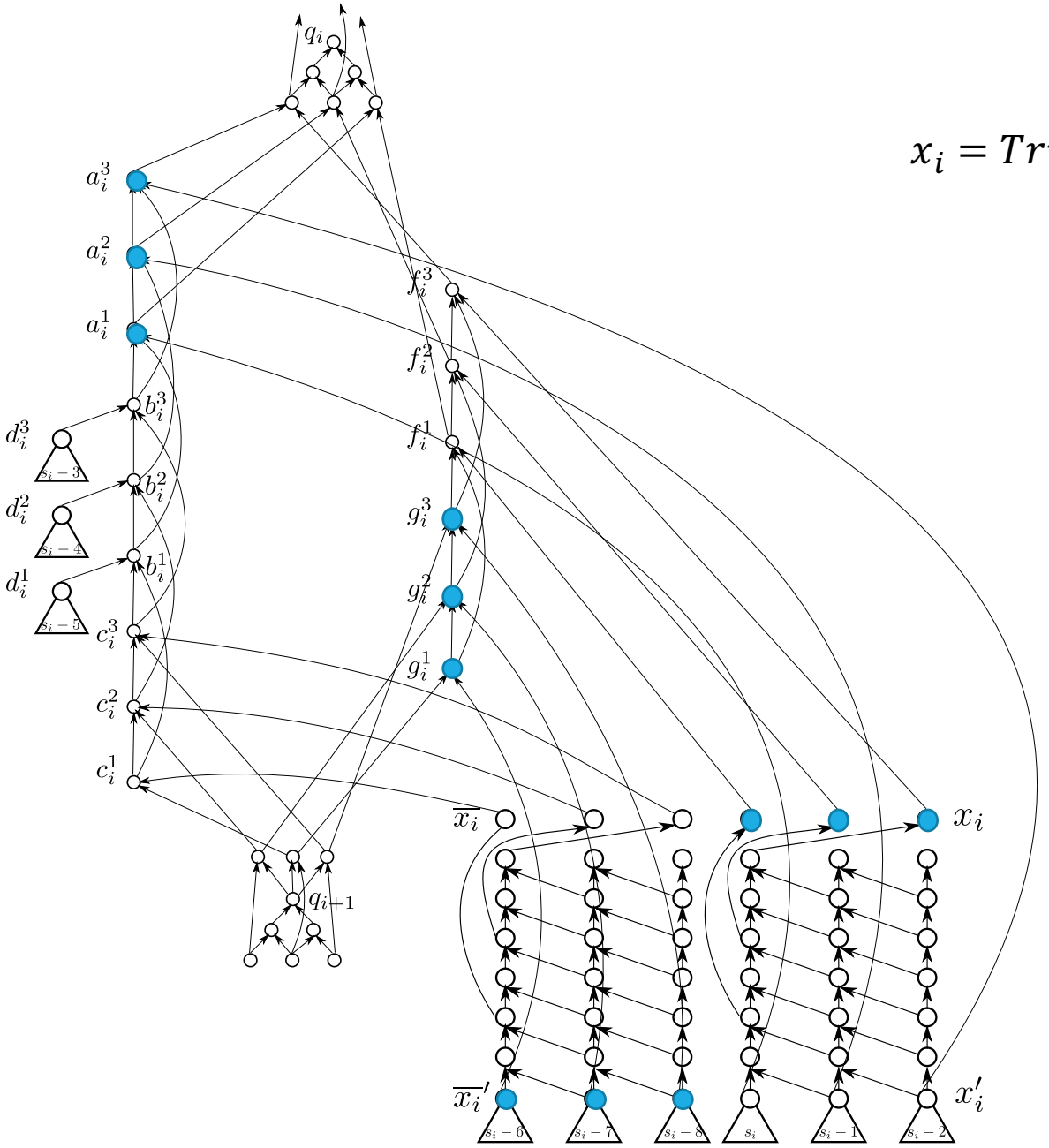
Universal Quantifier Block

$x_i = True;\ \overline{x_i} = False$



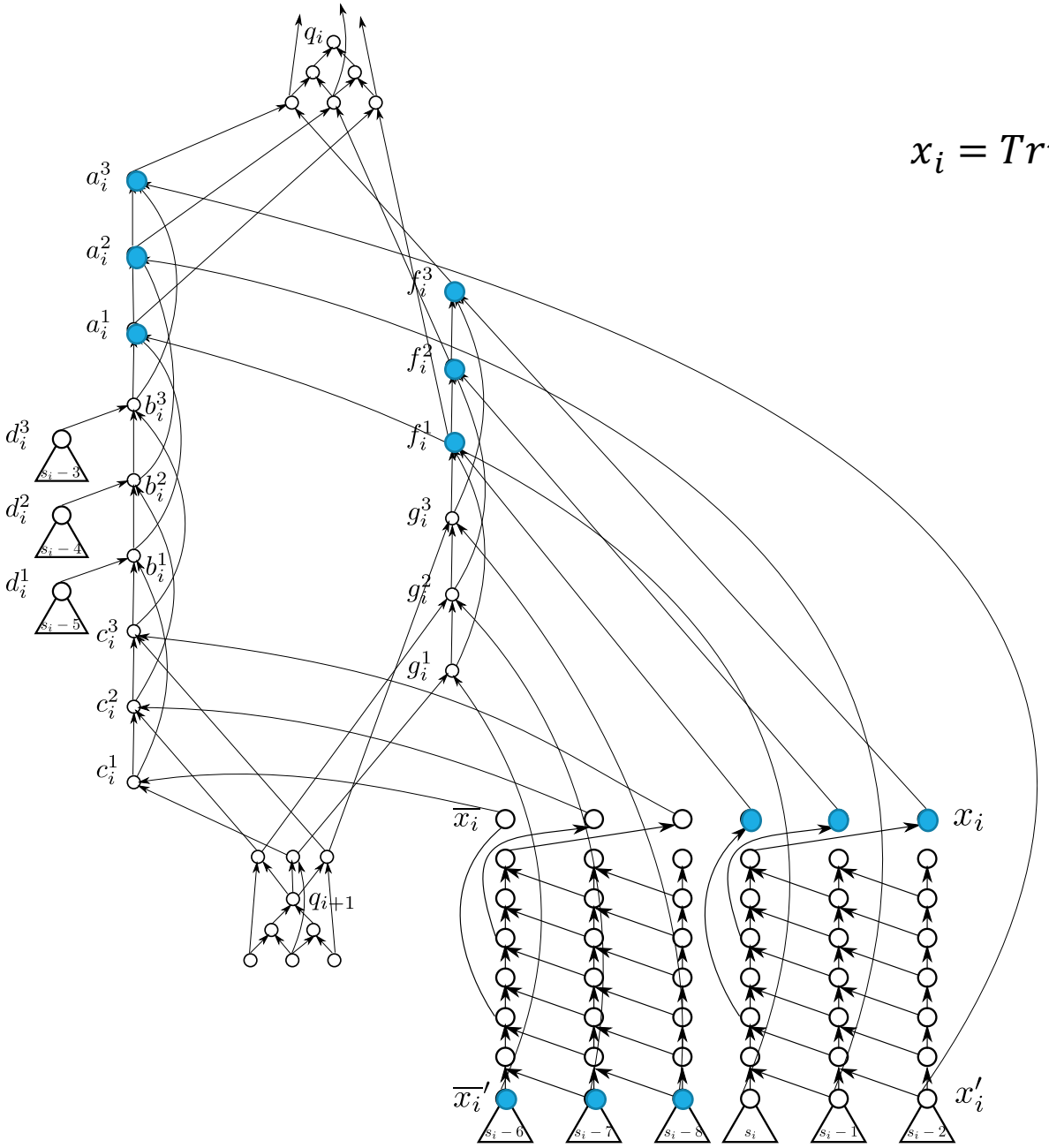
Universal Quantifier Block

$x_i = True;\ \overline{x_i} = False$



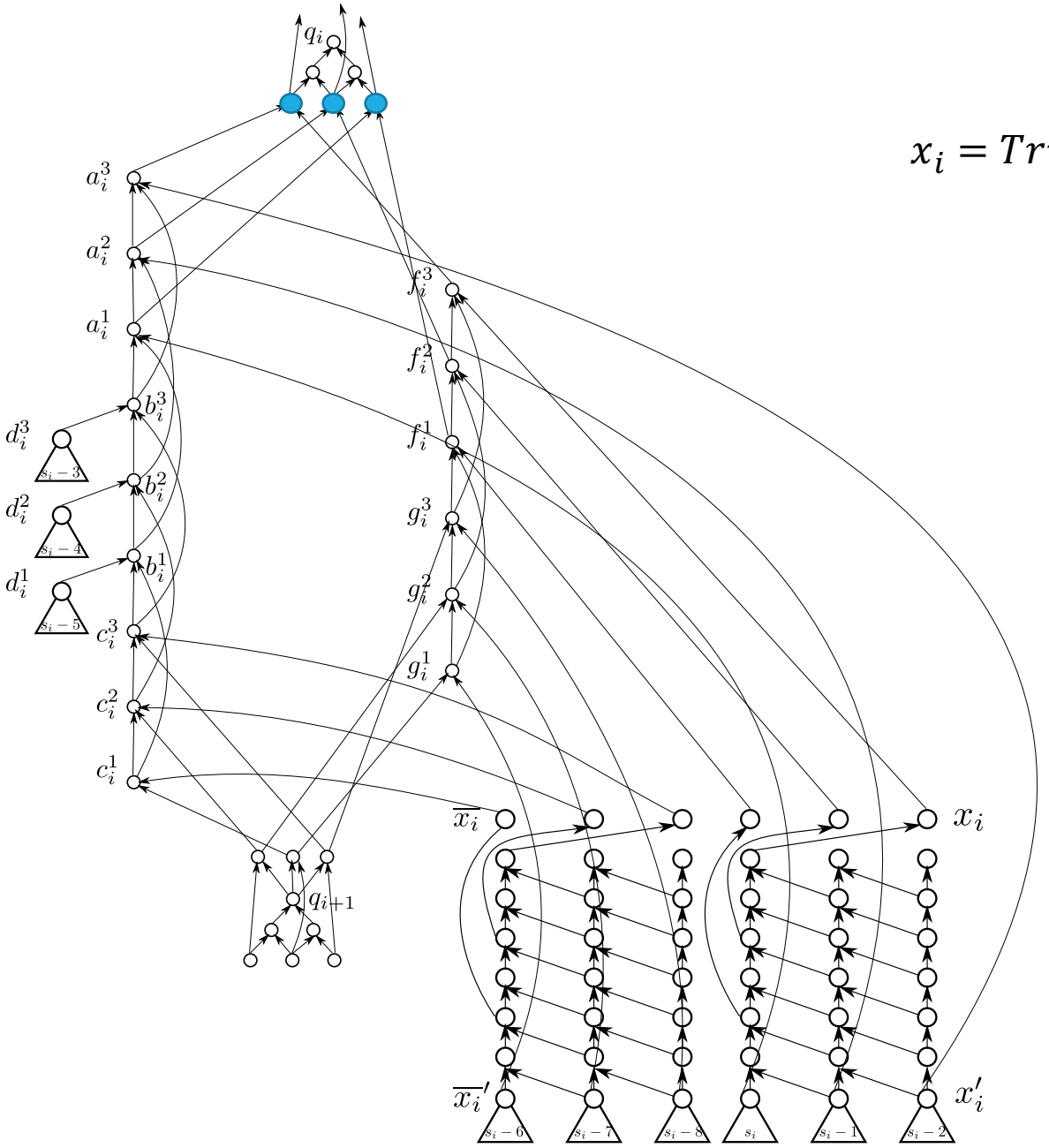
Universal Quantifier Block

$x_i = True;\ \overline{x_i} = False$



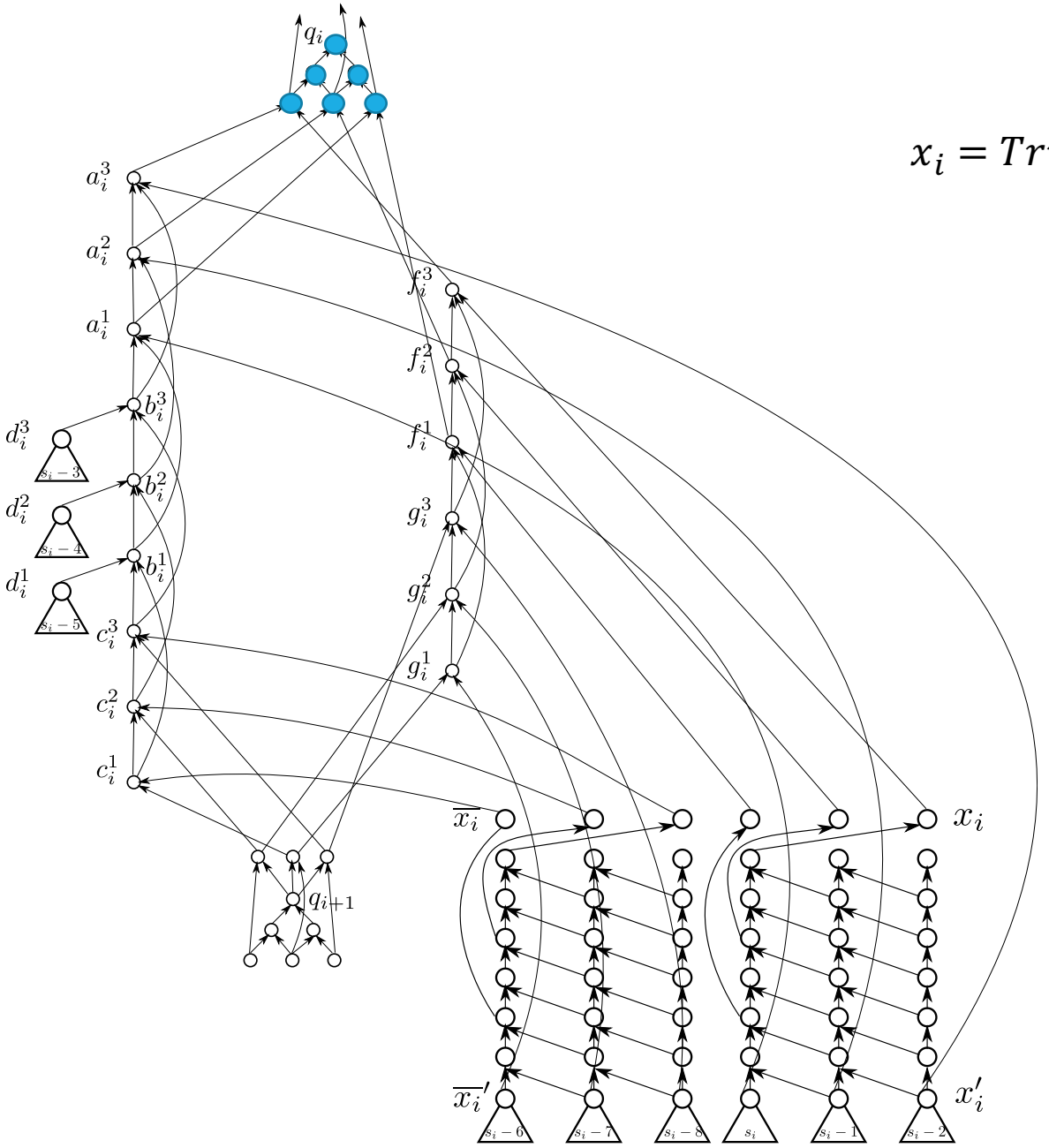
Universal Quantifier Block

$x_i = True;\ \overline{x_i} = False$



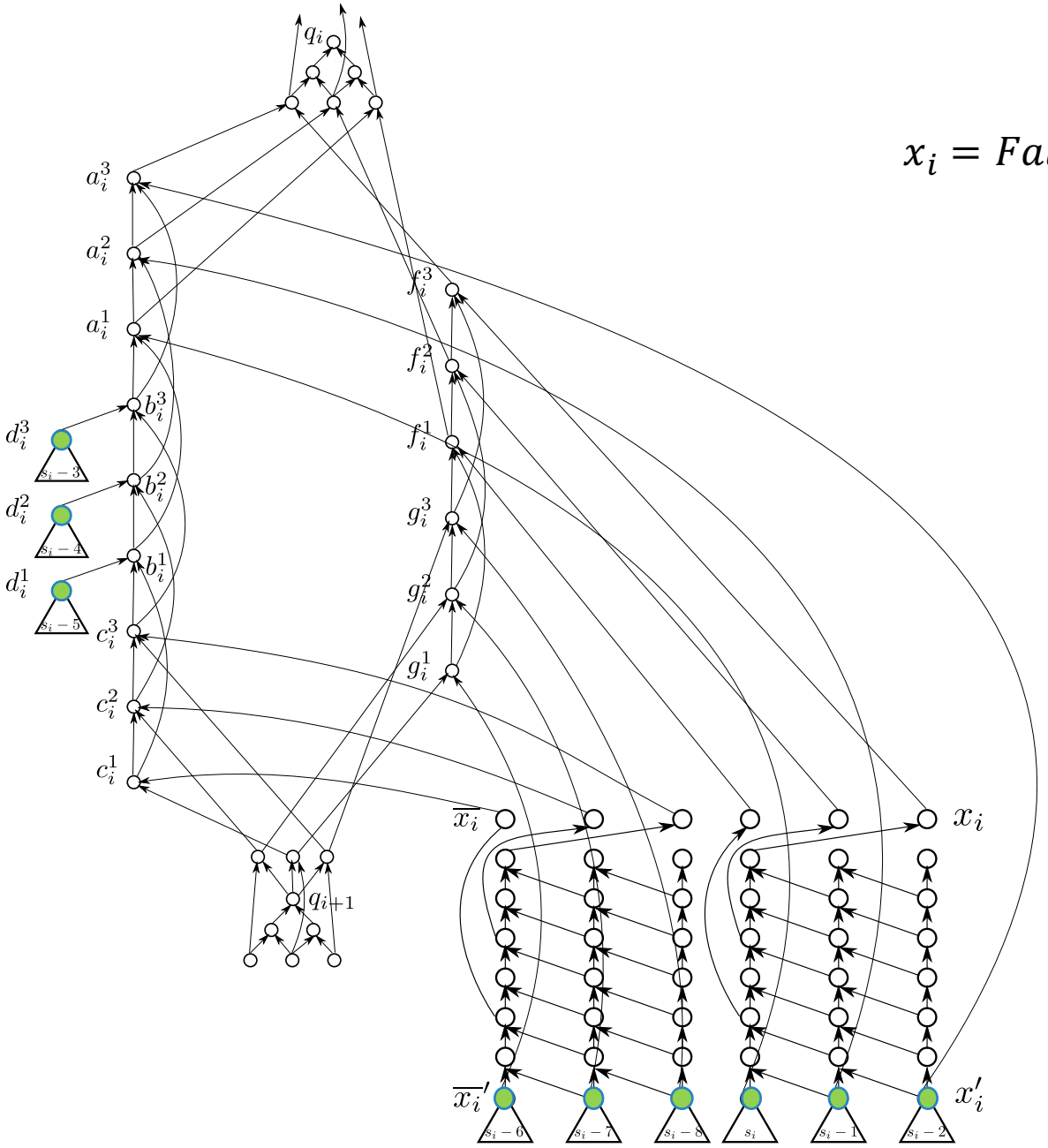
Universal Quantifier Block

$x_i = True;\ \overline{x_i} = False$



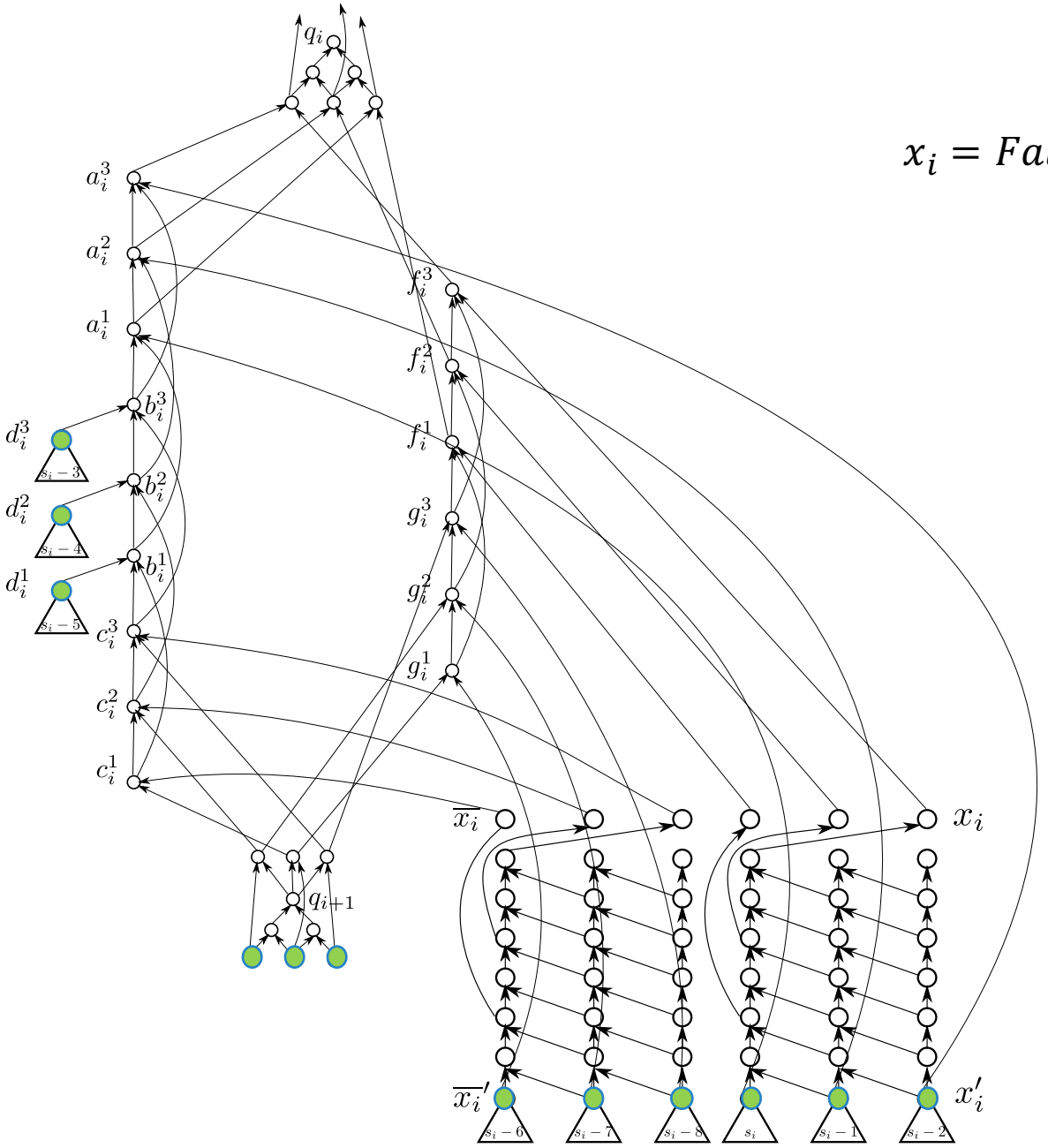
Universal Quantifier Block

$x_i = False;\ \overline{x_i} = False$



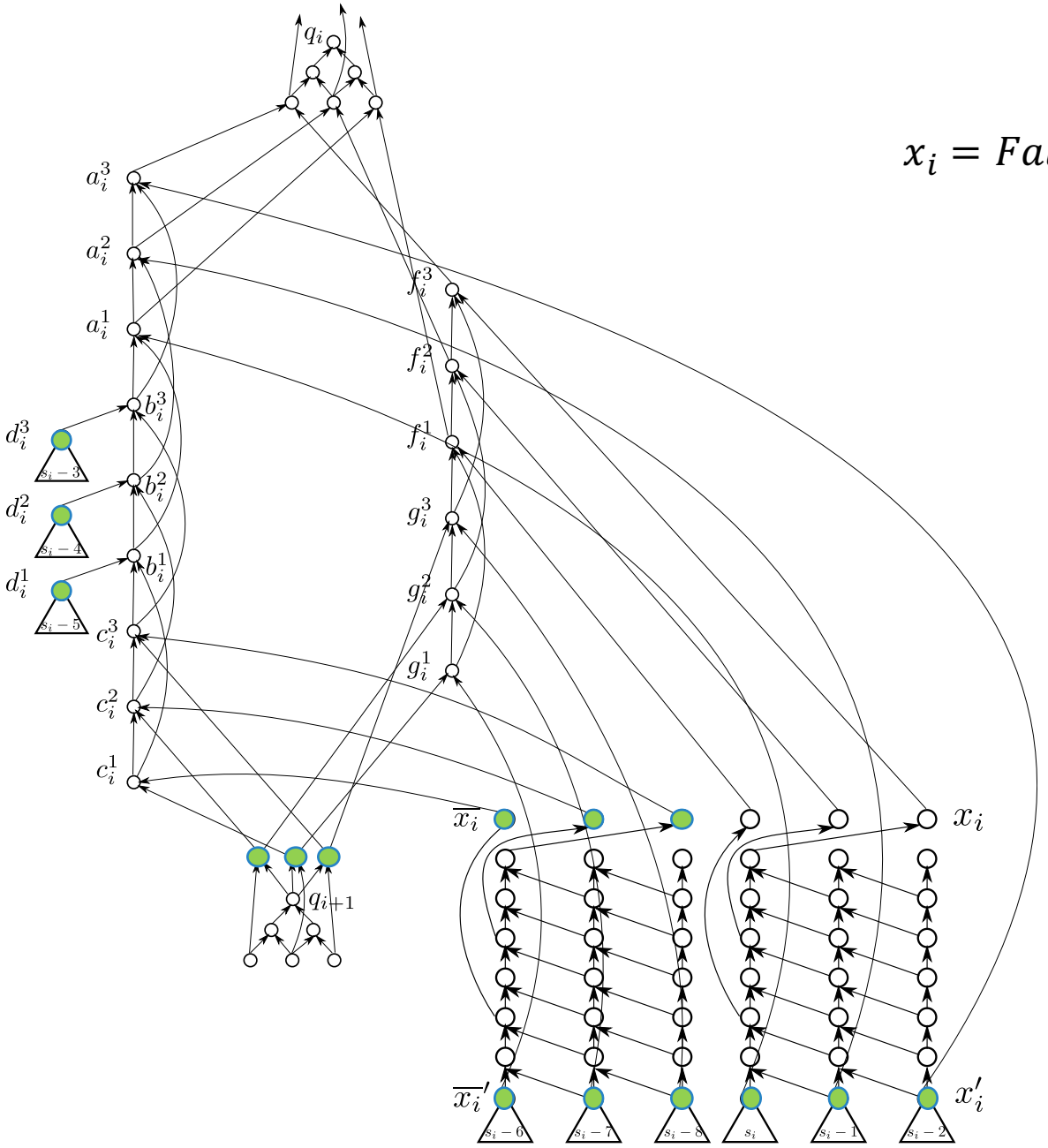
Universal Quantifier Block

$x_i = False;\ \overline{x_i} = False$



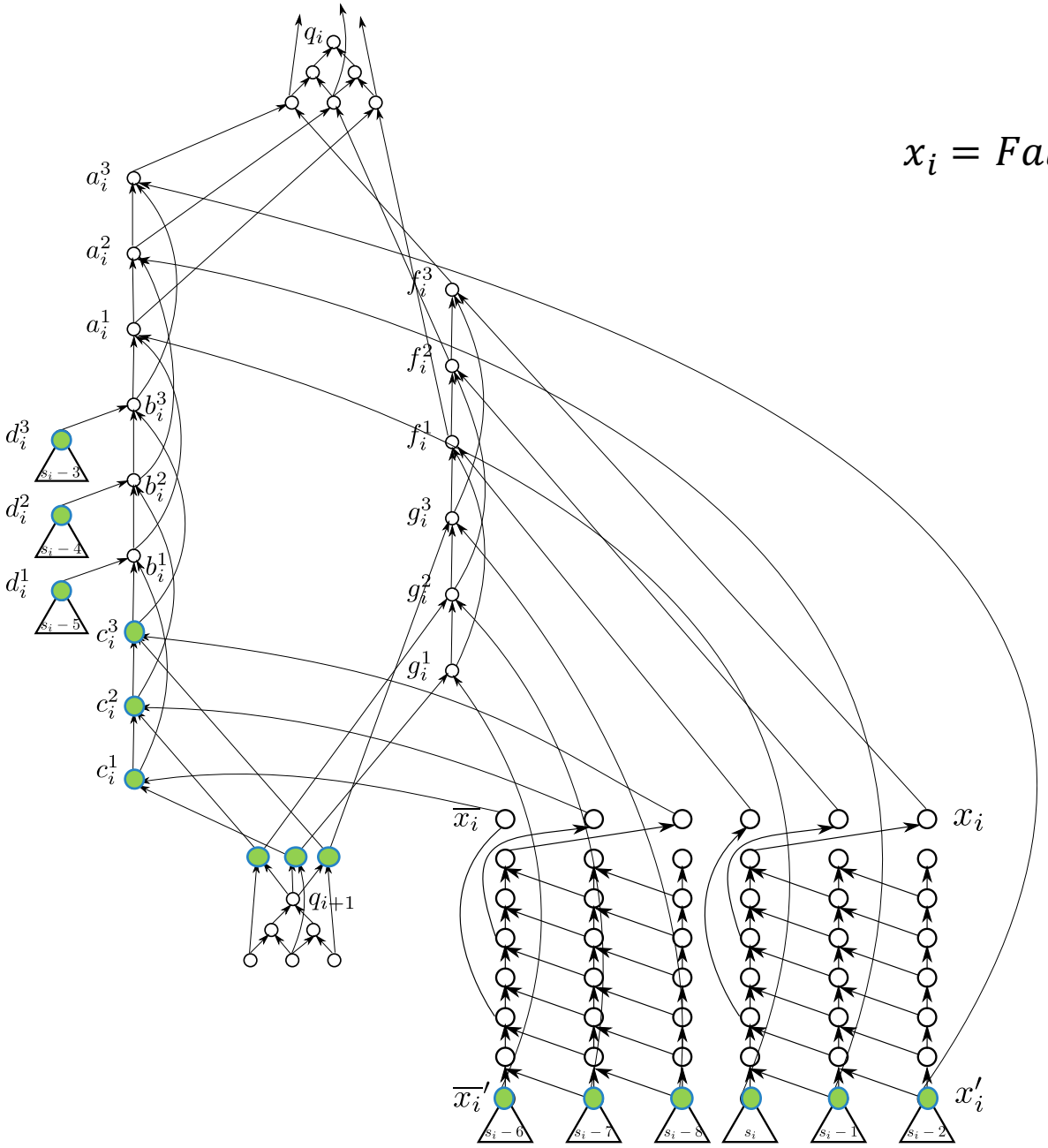
Universal Quantifier Block

$x_i = False;\ \overline{x_i} = False$



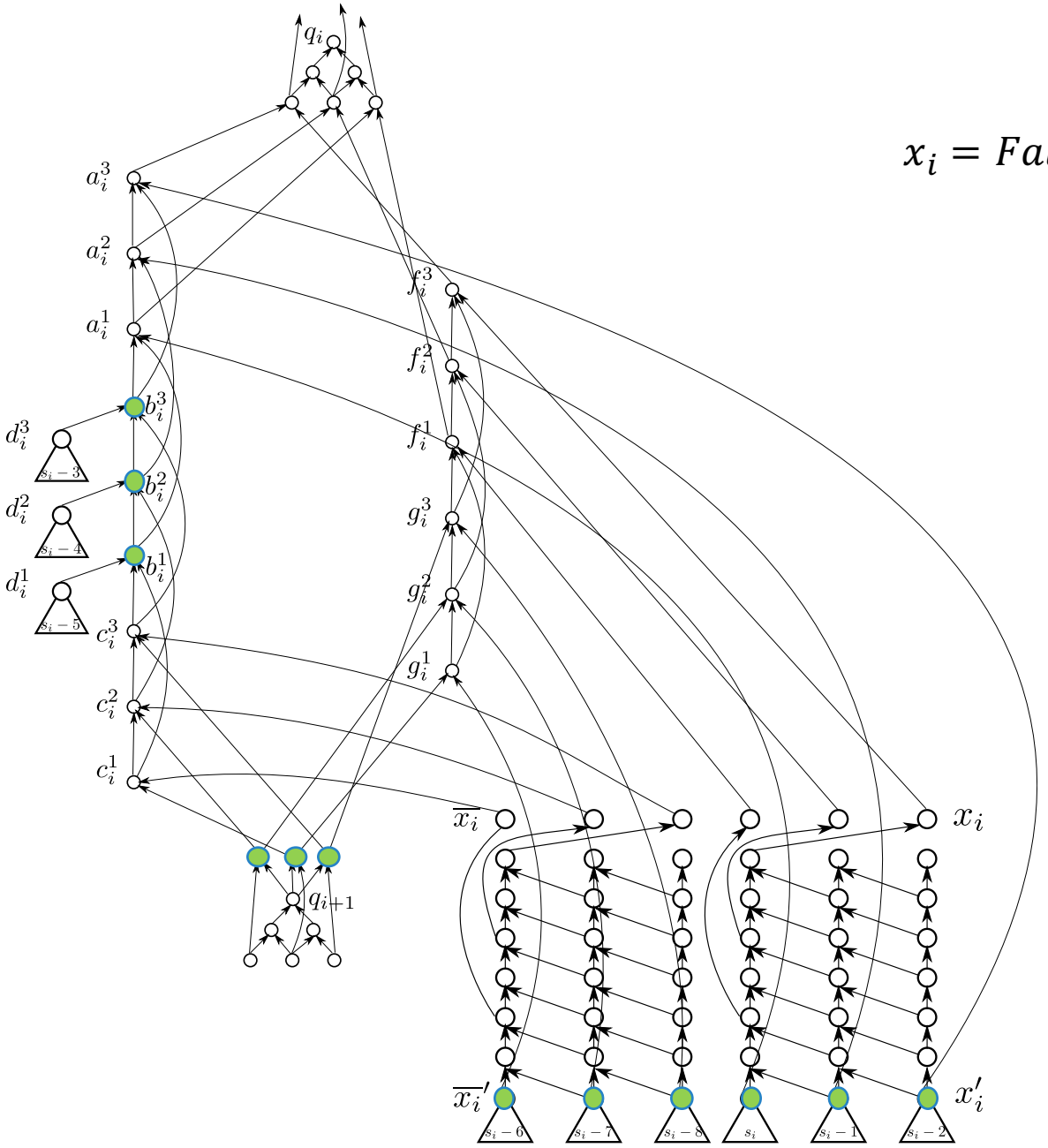
Universal Quantifier Block

$x_i = False;\ \overline{x_i} = False$



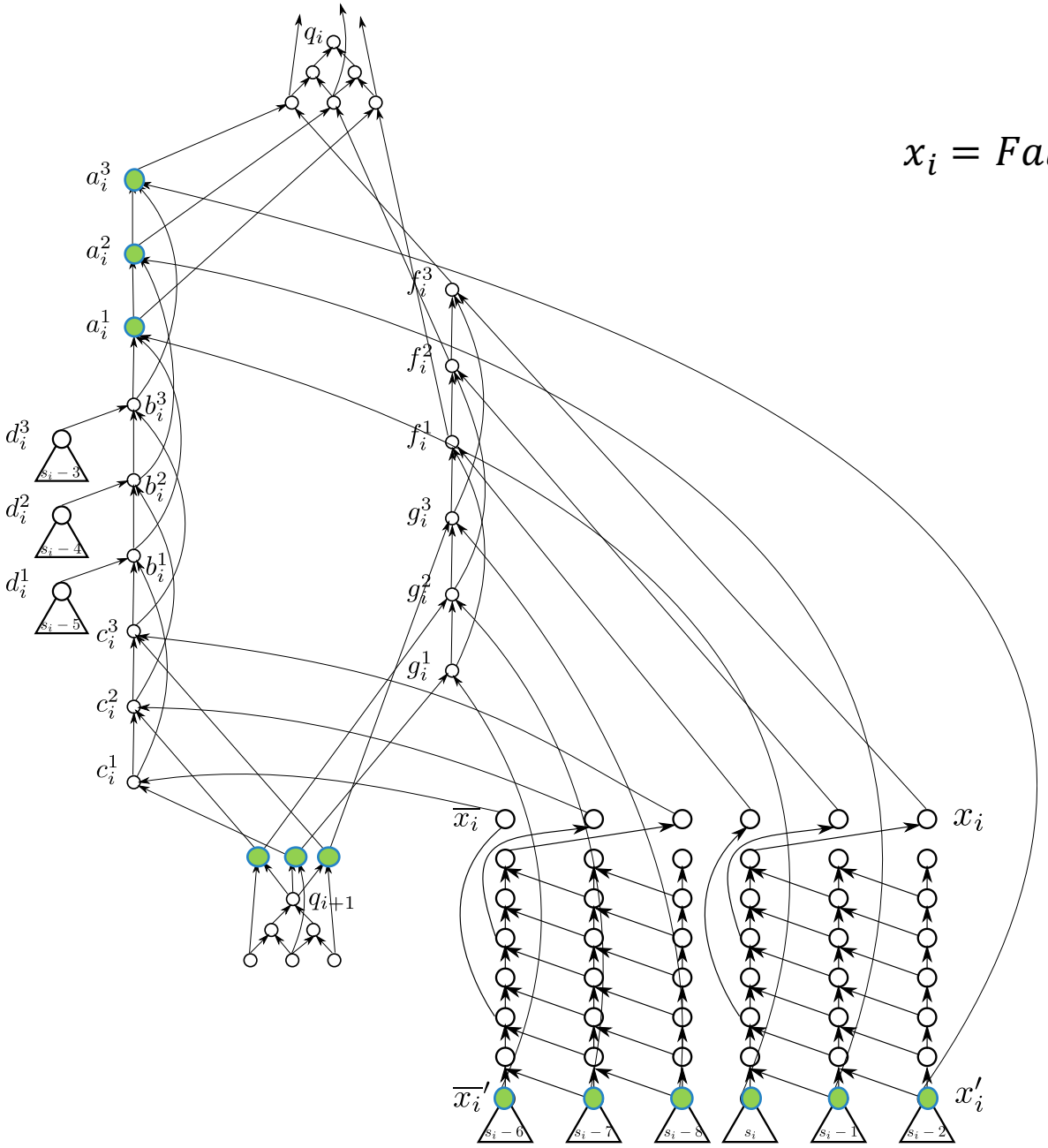
Universal Quantifier Block

$x_i = False;\ \overline{x_i} = False$



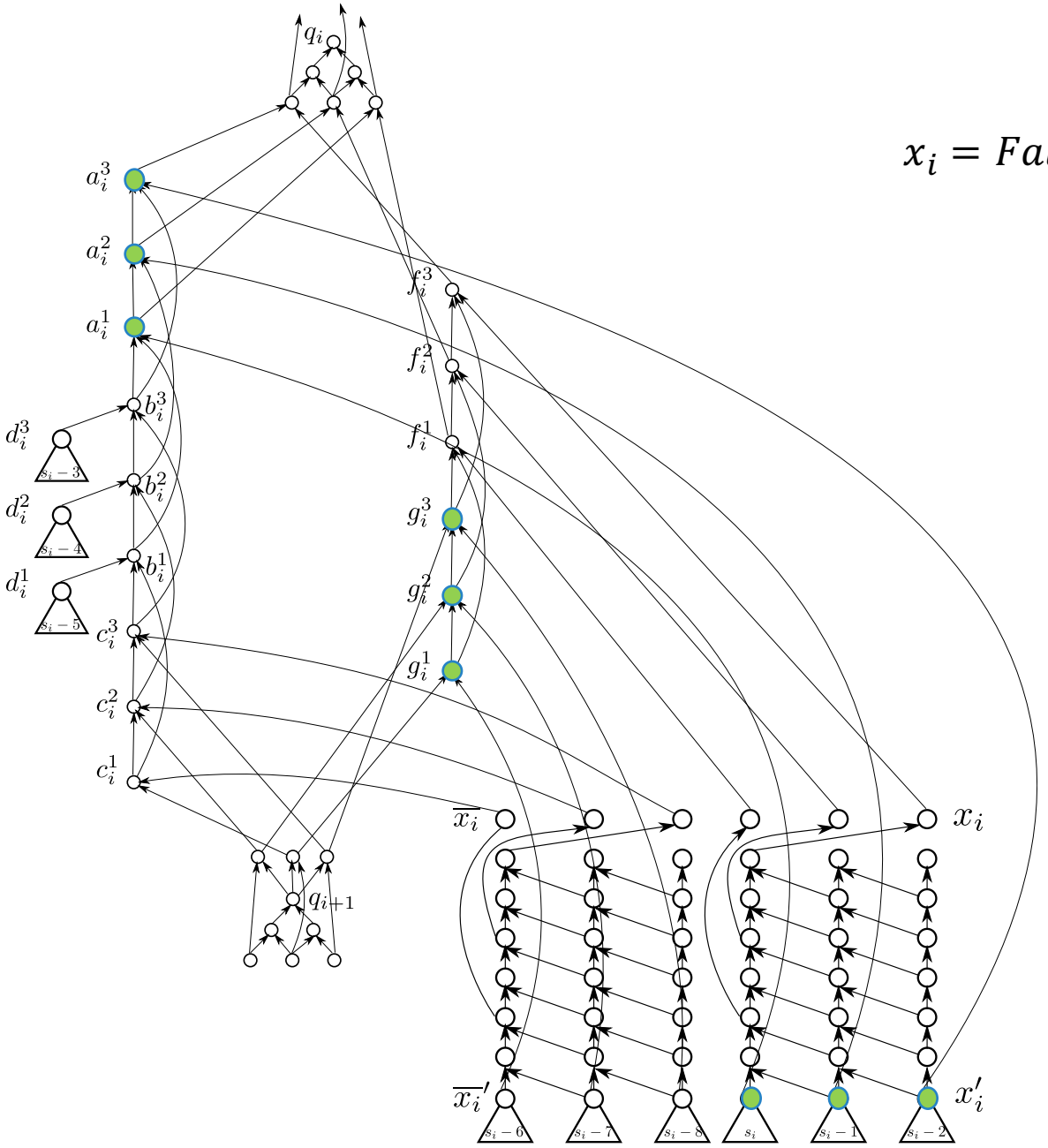
Universal Quantifier Block

$x_i = False;\ \overline{x_i} = False$



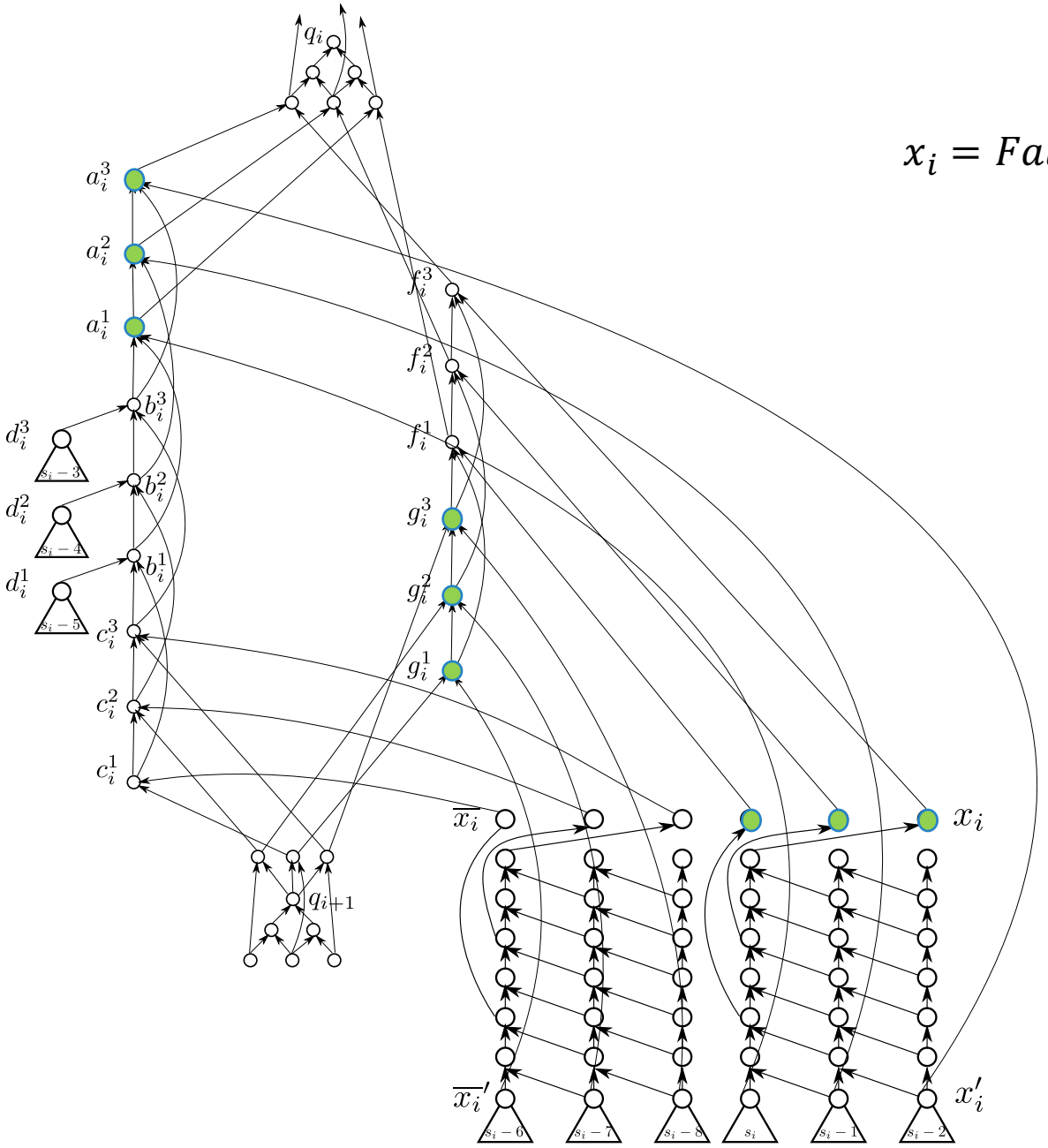
Universal Quantifier Block

$x_i = False;\ \overline{x_i} = False$



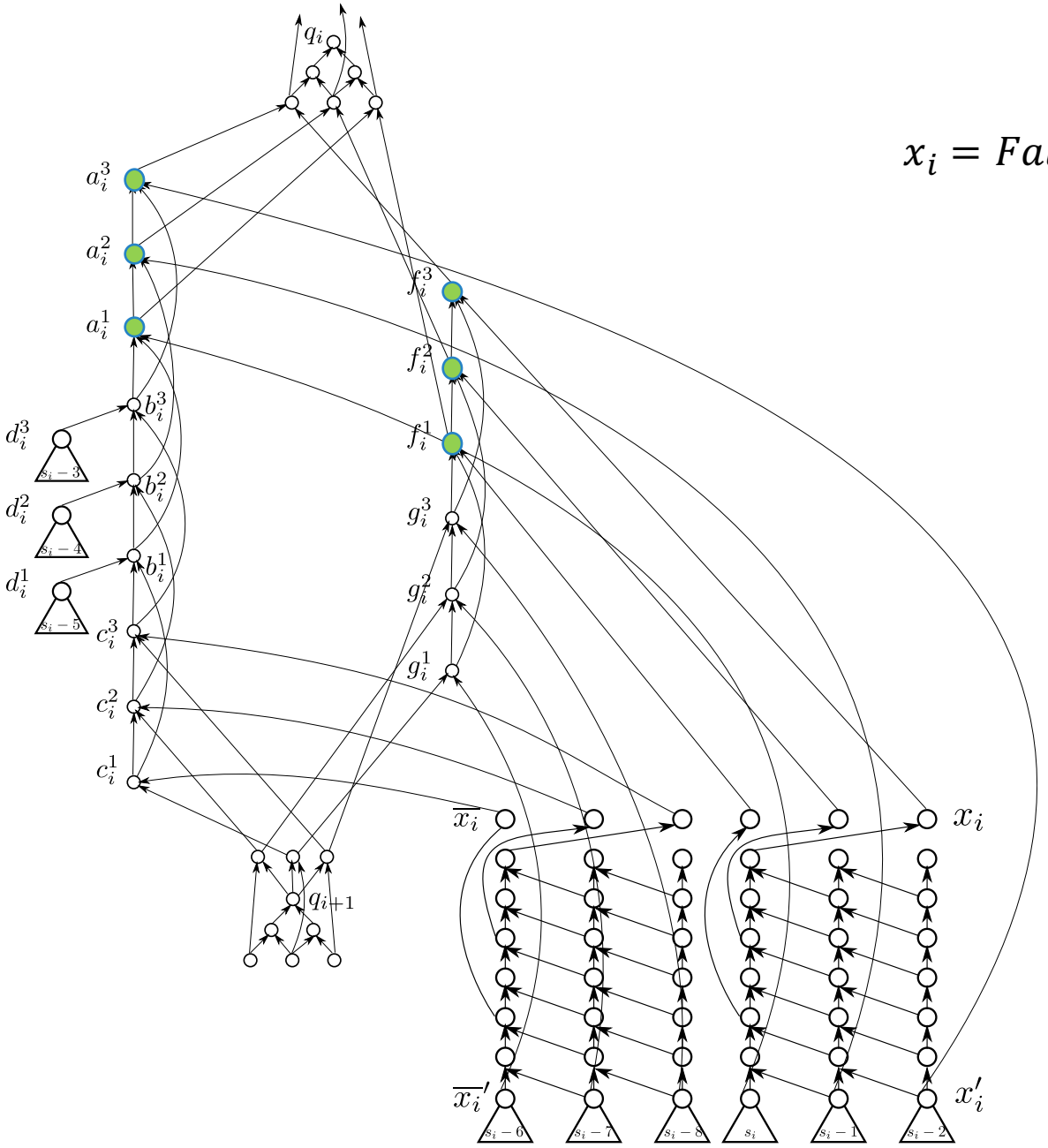
Universal Quantifier Block

$x_i = False;\ \overline{x_i} = False$



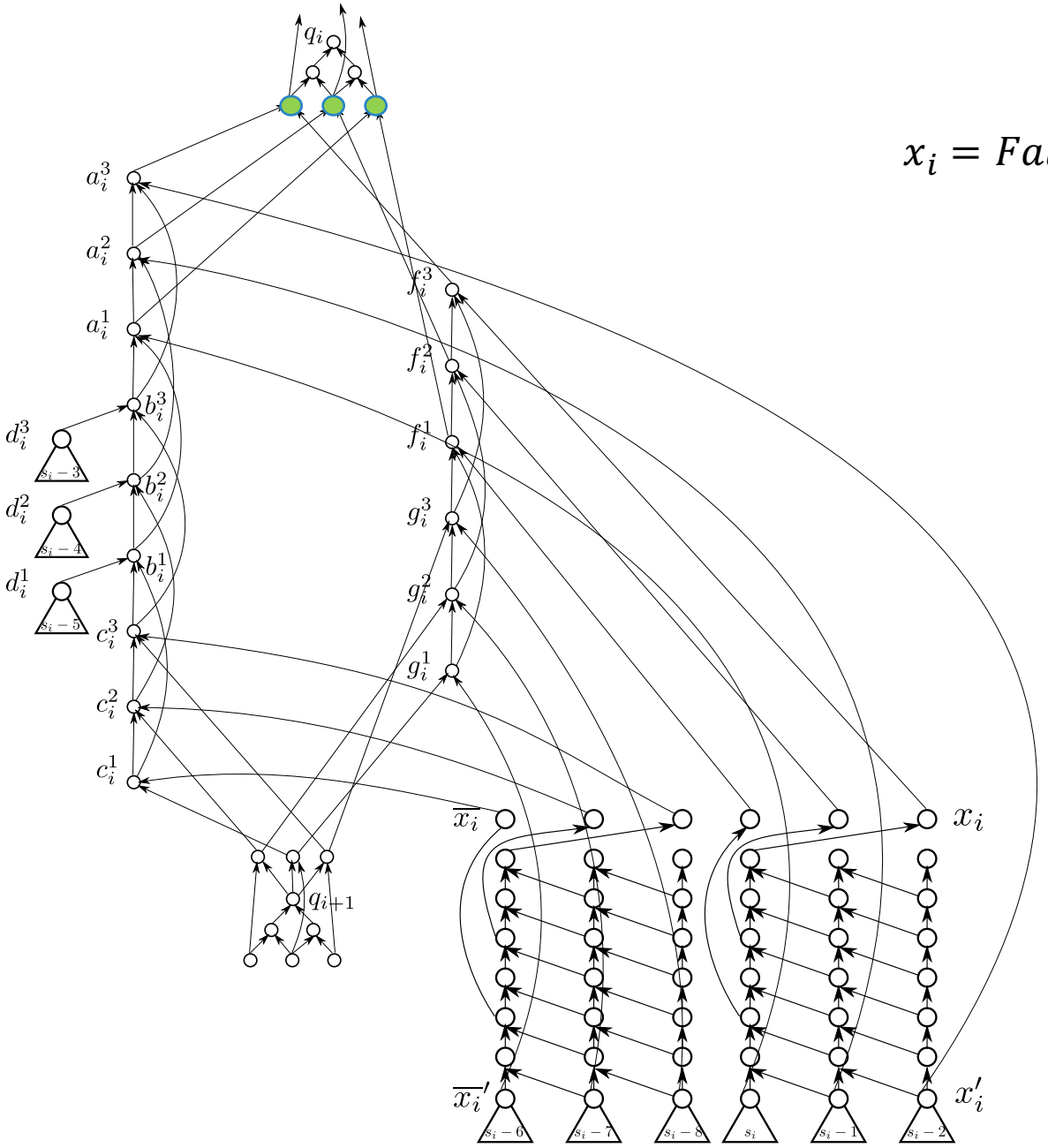
Universal Quantifier Block

$x_i = False;\ \overline{x_i} = False$



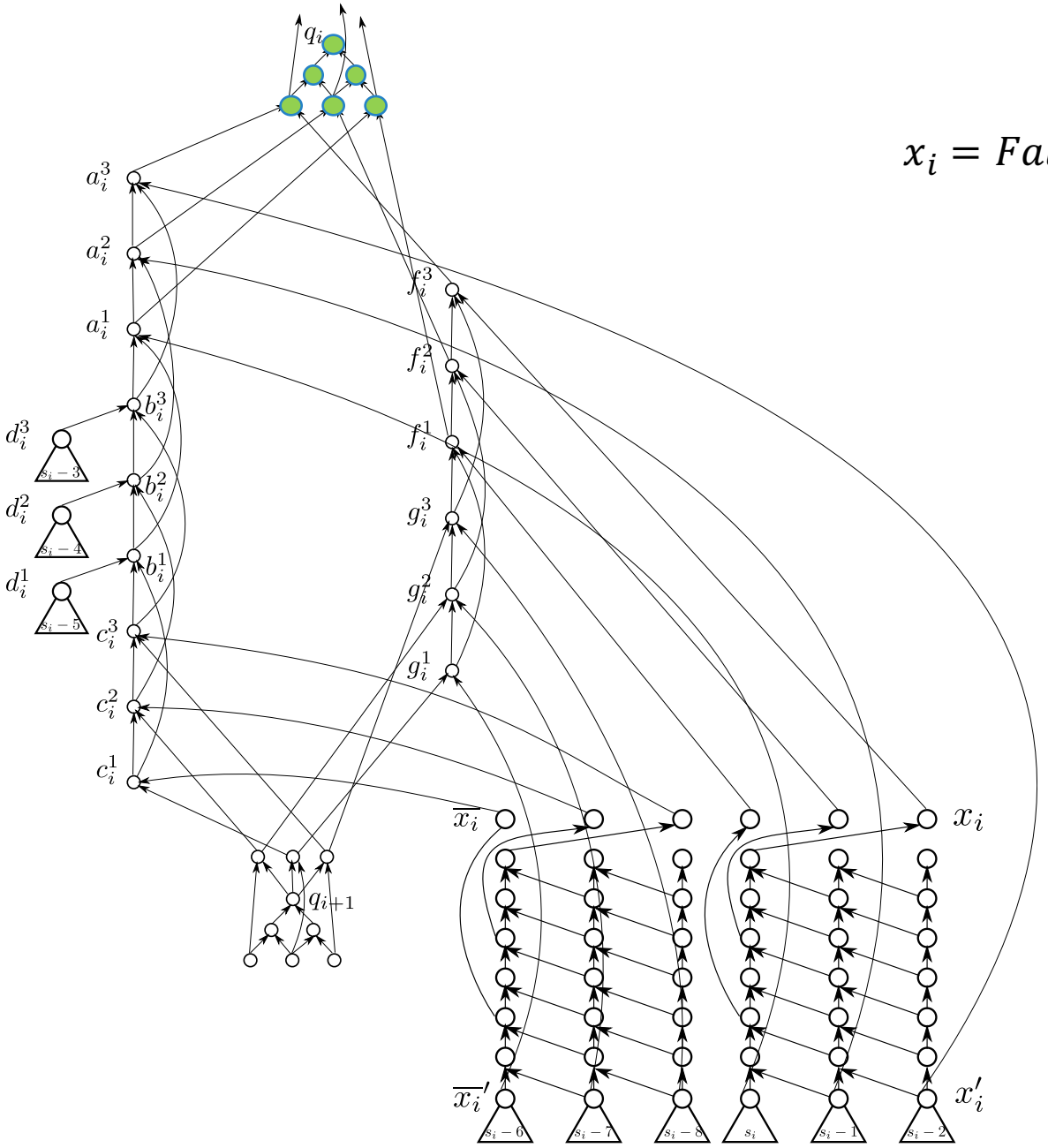
Universal Quantifier Block

$x_i = False;\ \overline{x_i} = False$



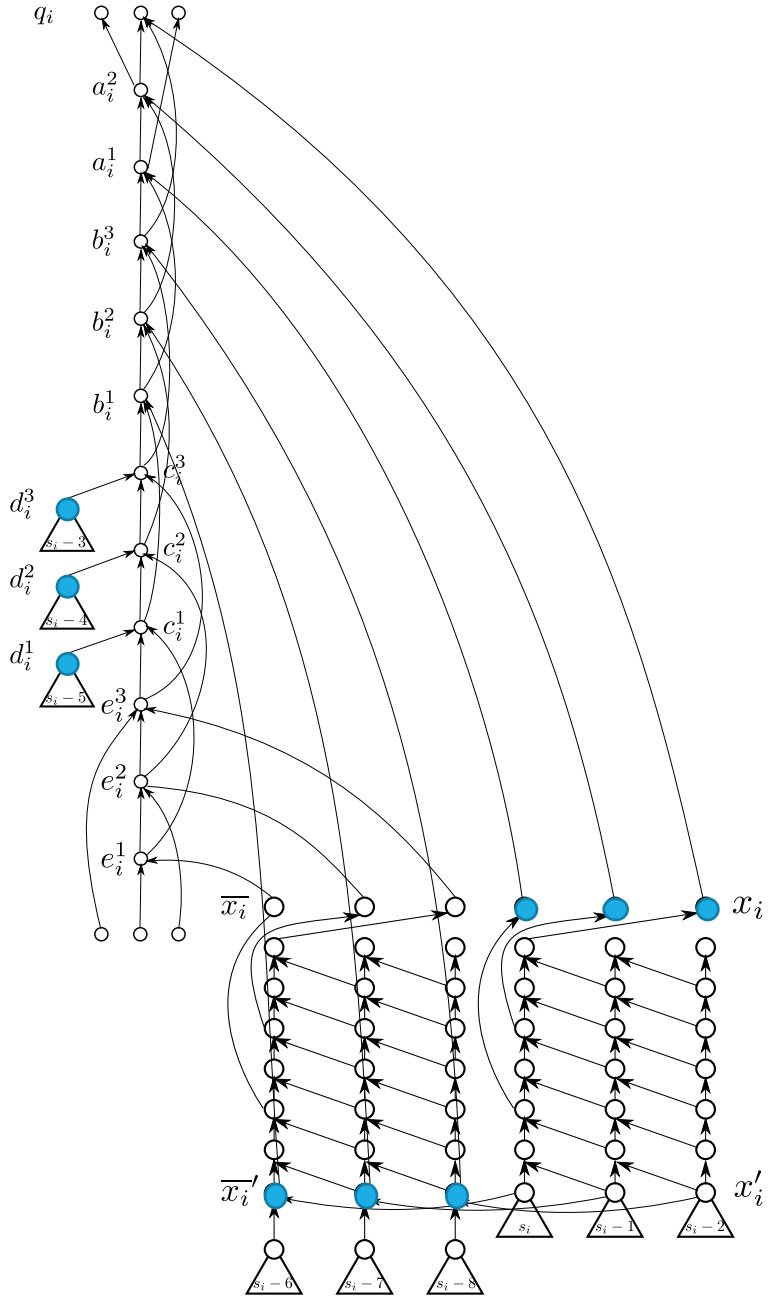
Universal Quantifier Block

$x_i = False;\ \overline{x_i} = False$



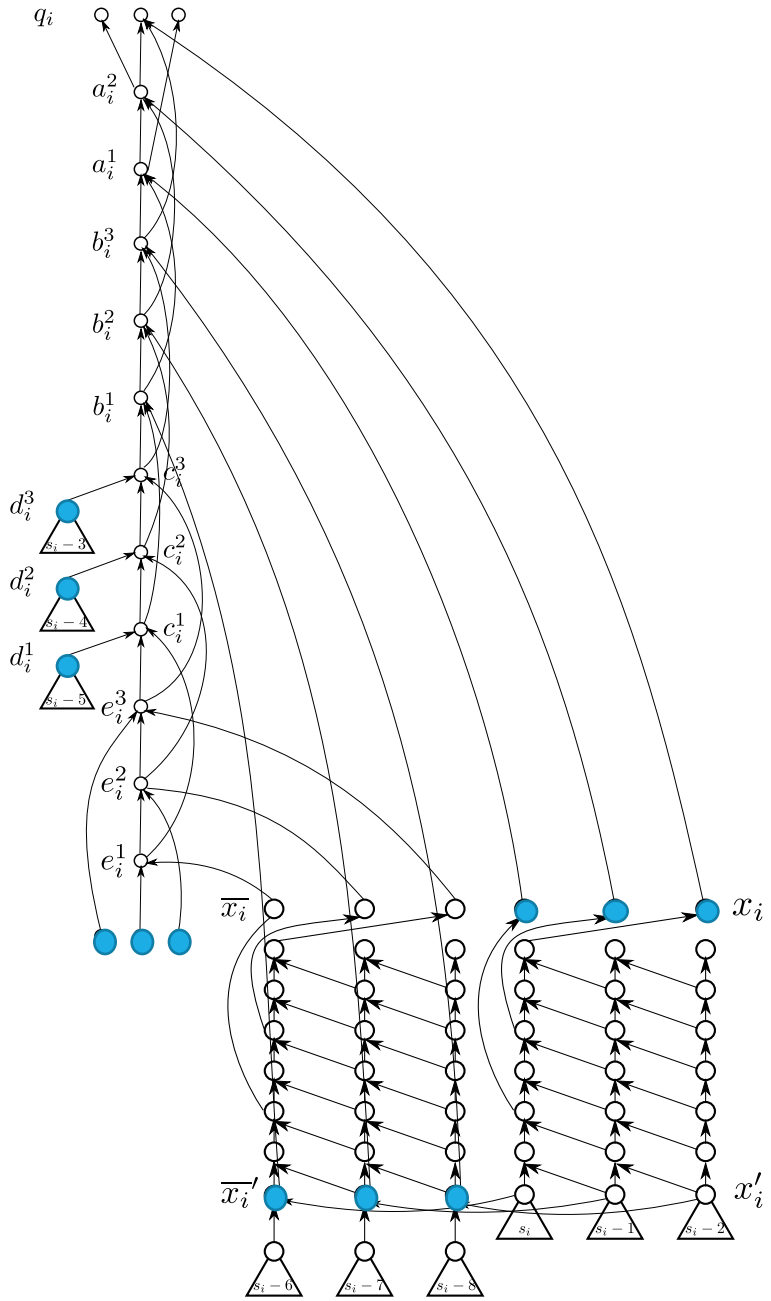
Existential Quantifier Block

$x_i = True;\ \overline{x_i} = False$



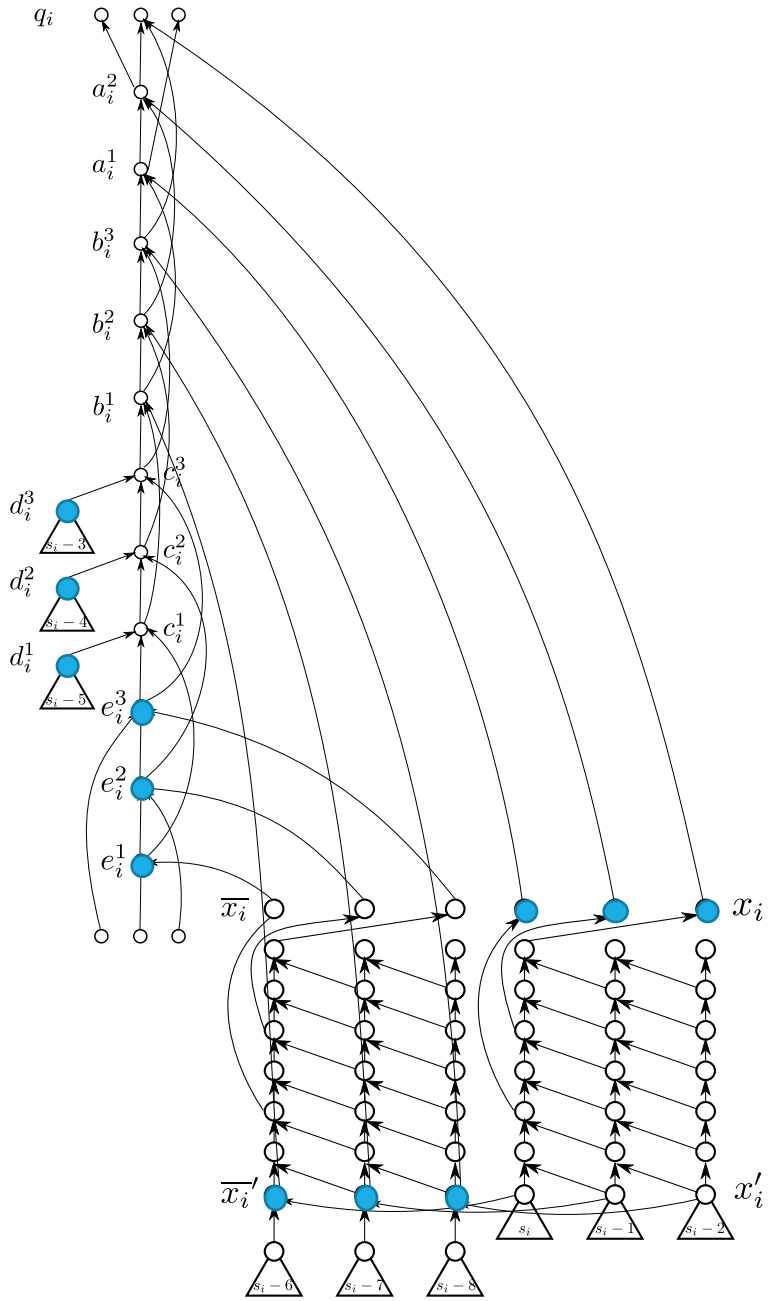
Existential Quantifier Block

$x_i = True;\ \overline{x_i} = False$



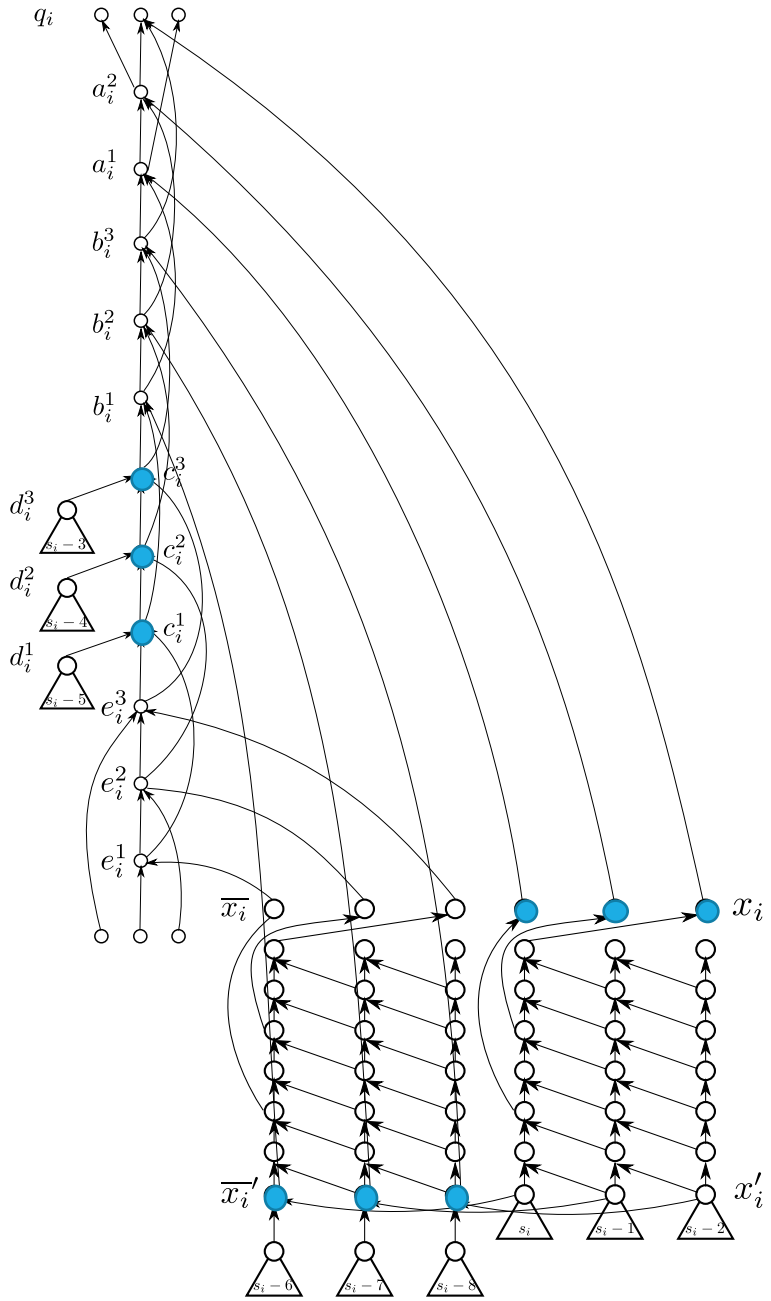
Existential Quantifier Block

$x_i = True;\ \overline{x_i} = False$



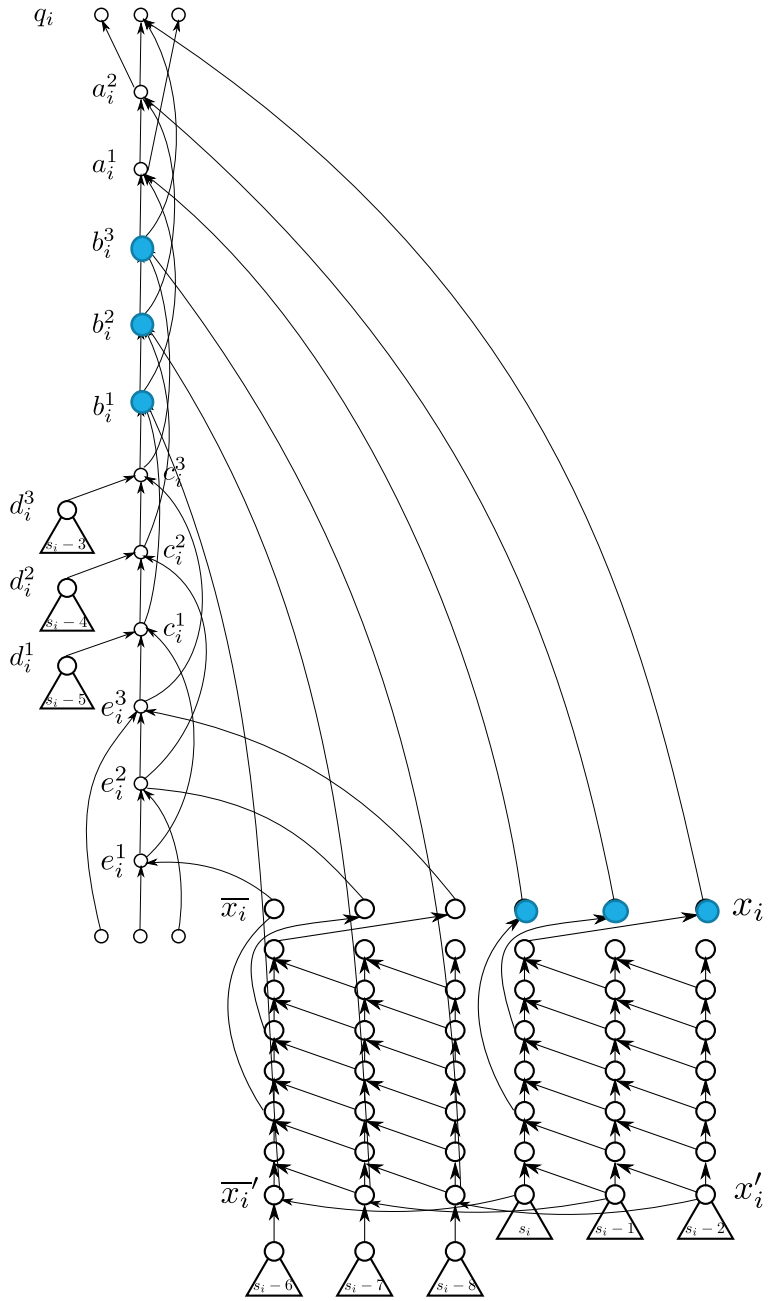
Existential Quantifier Block

$x_i = True;\ \overline{x_i} = False$



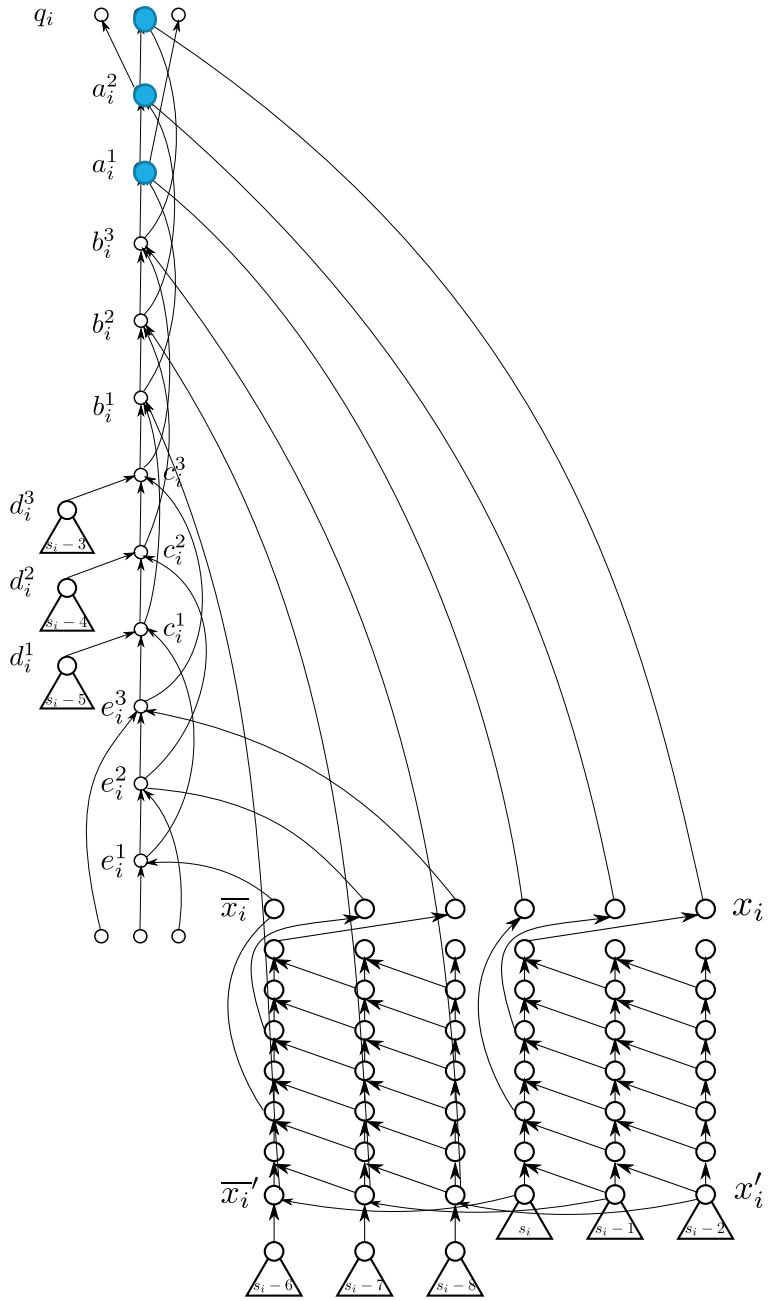
Existential Quantifier Block

$x_i = True;\ \overline{x_i} = False$



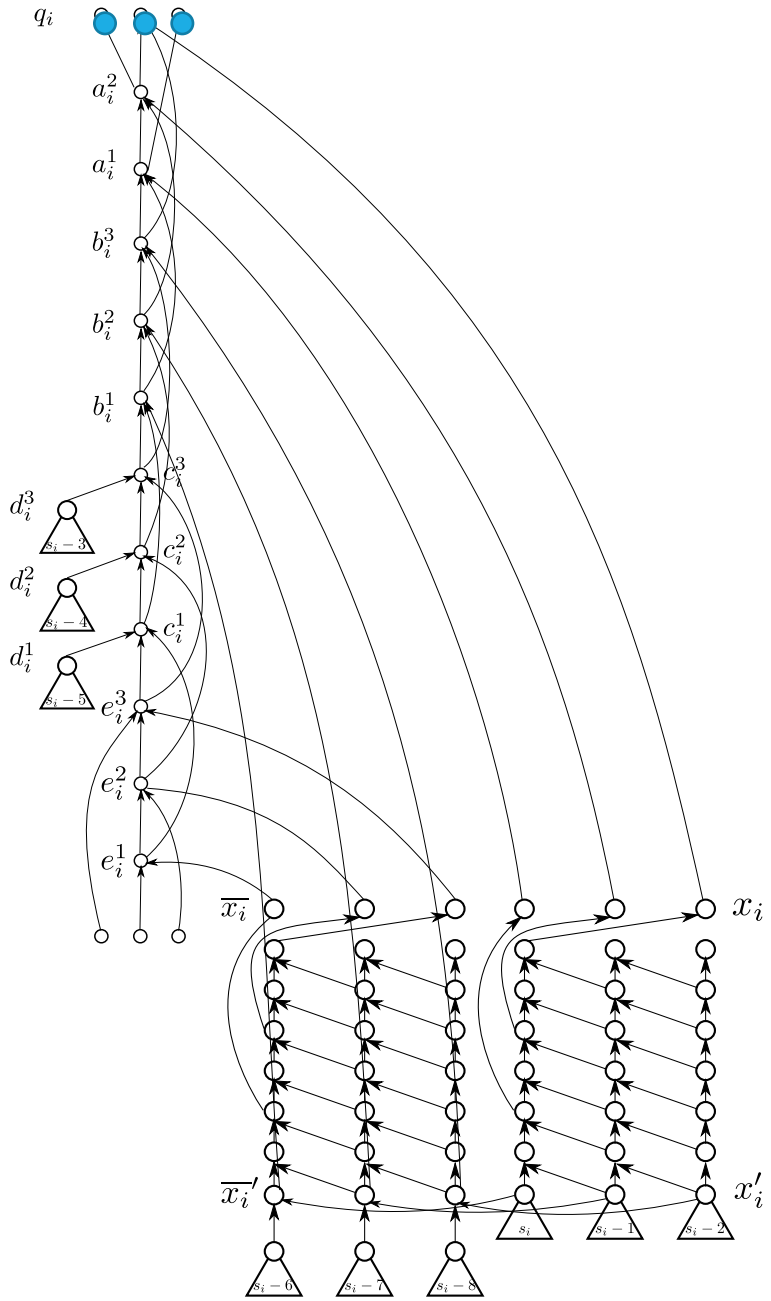
Existential Quantifier Block

$x_i = True;\ \overline{x_i} = False$

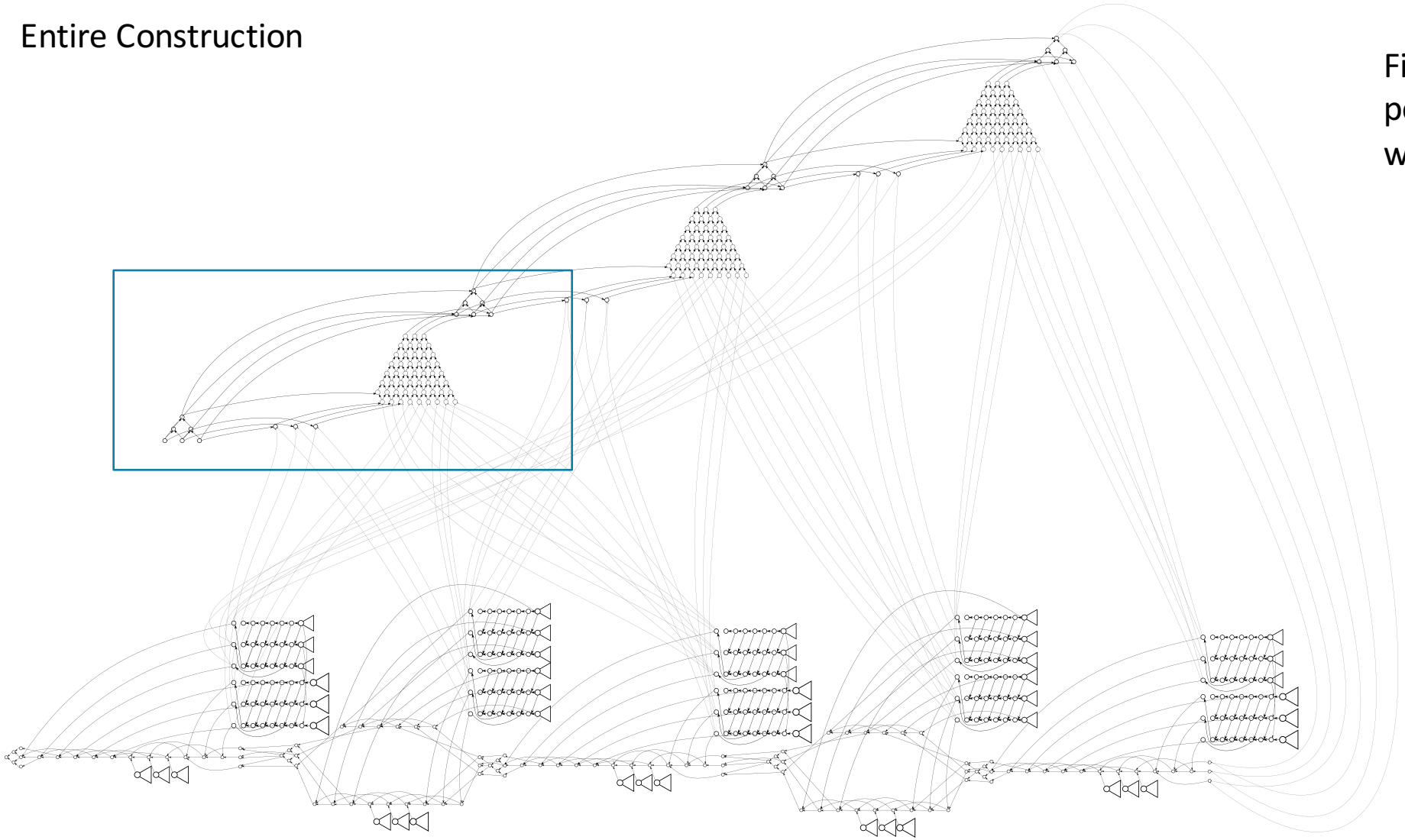


Existential Quantifier Block

$x_i = True;\ \overline{x_i} = False$

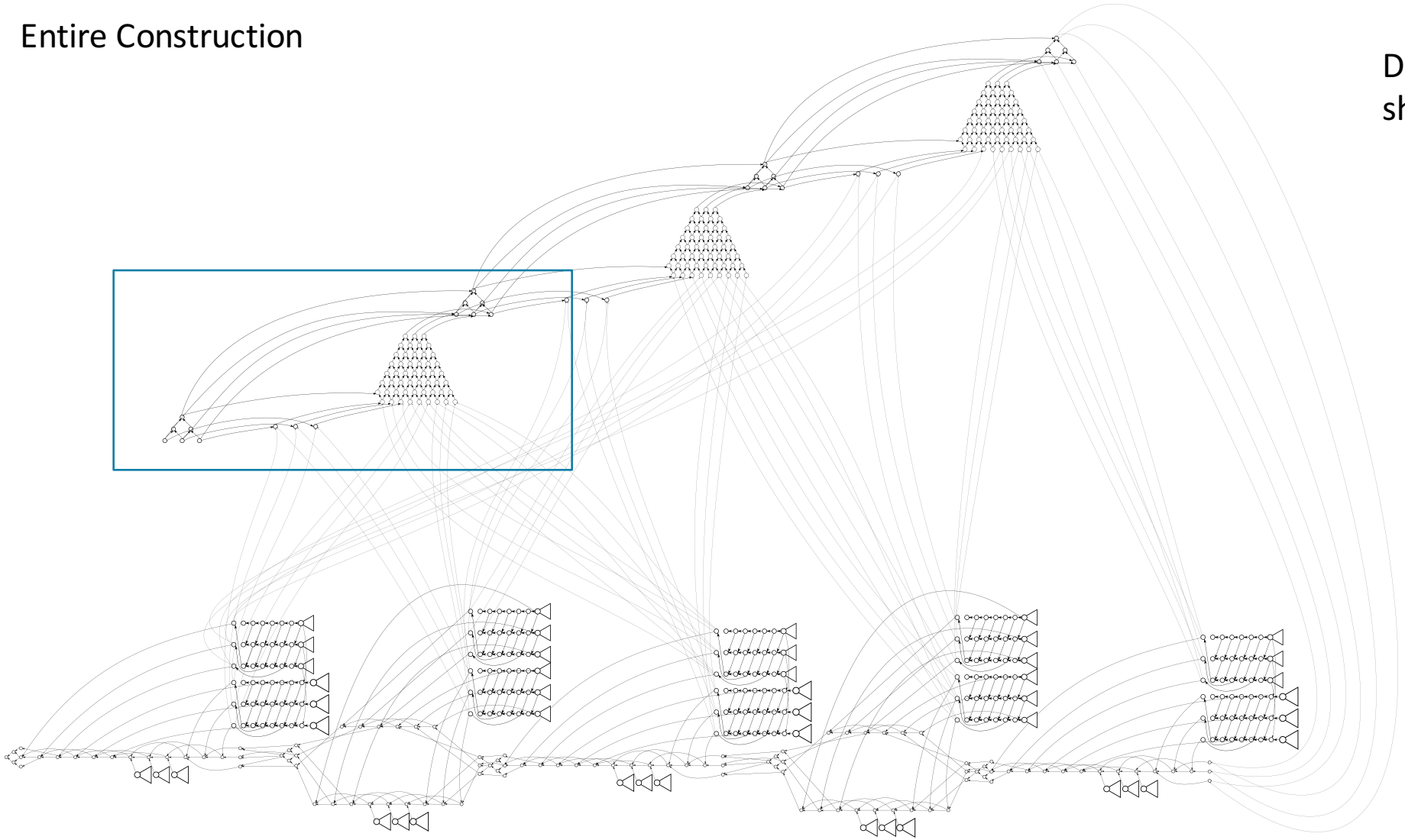


Entire Construction



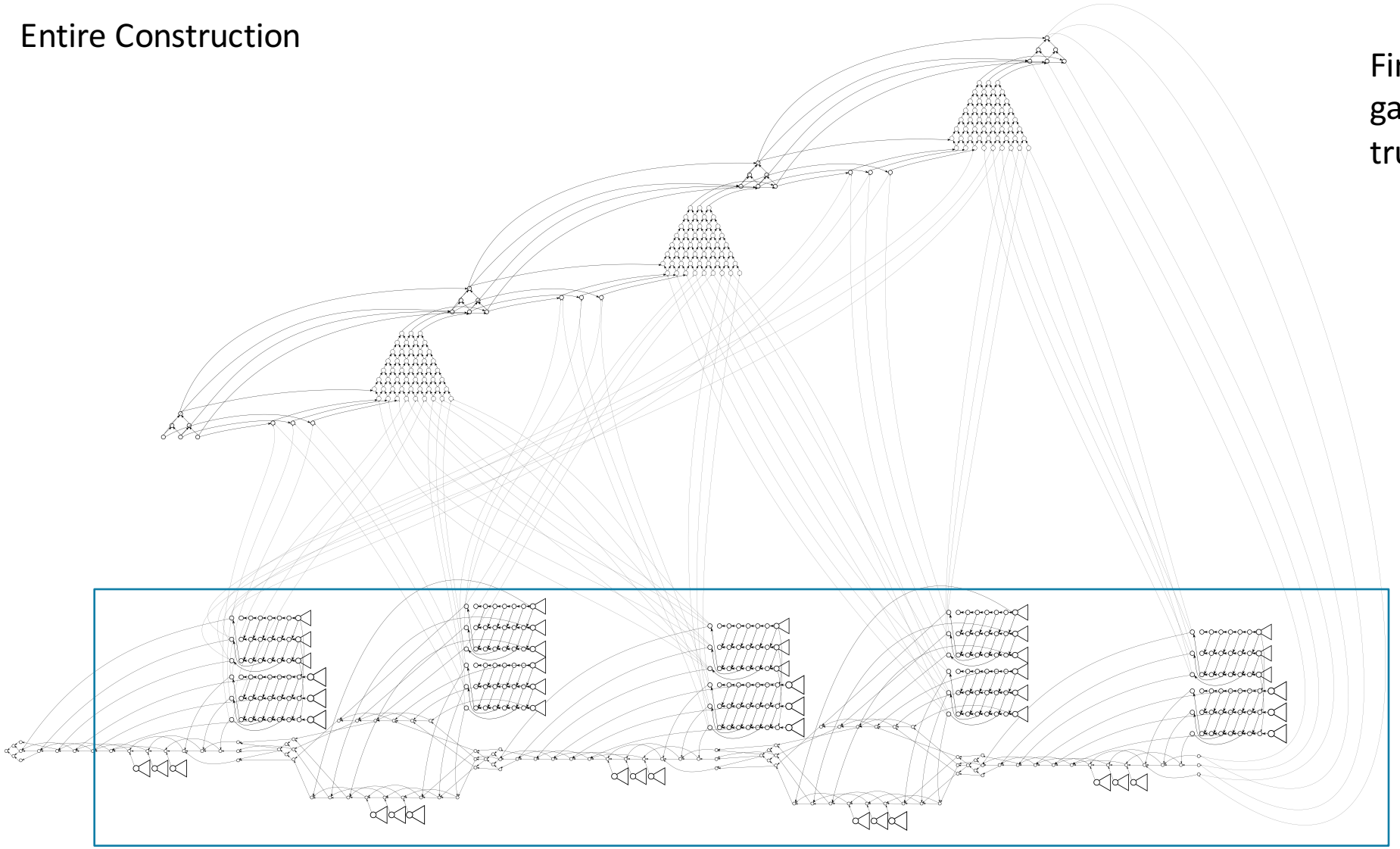
First clause can be pebbled regardless of whether it is true or false.

Entire Construction



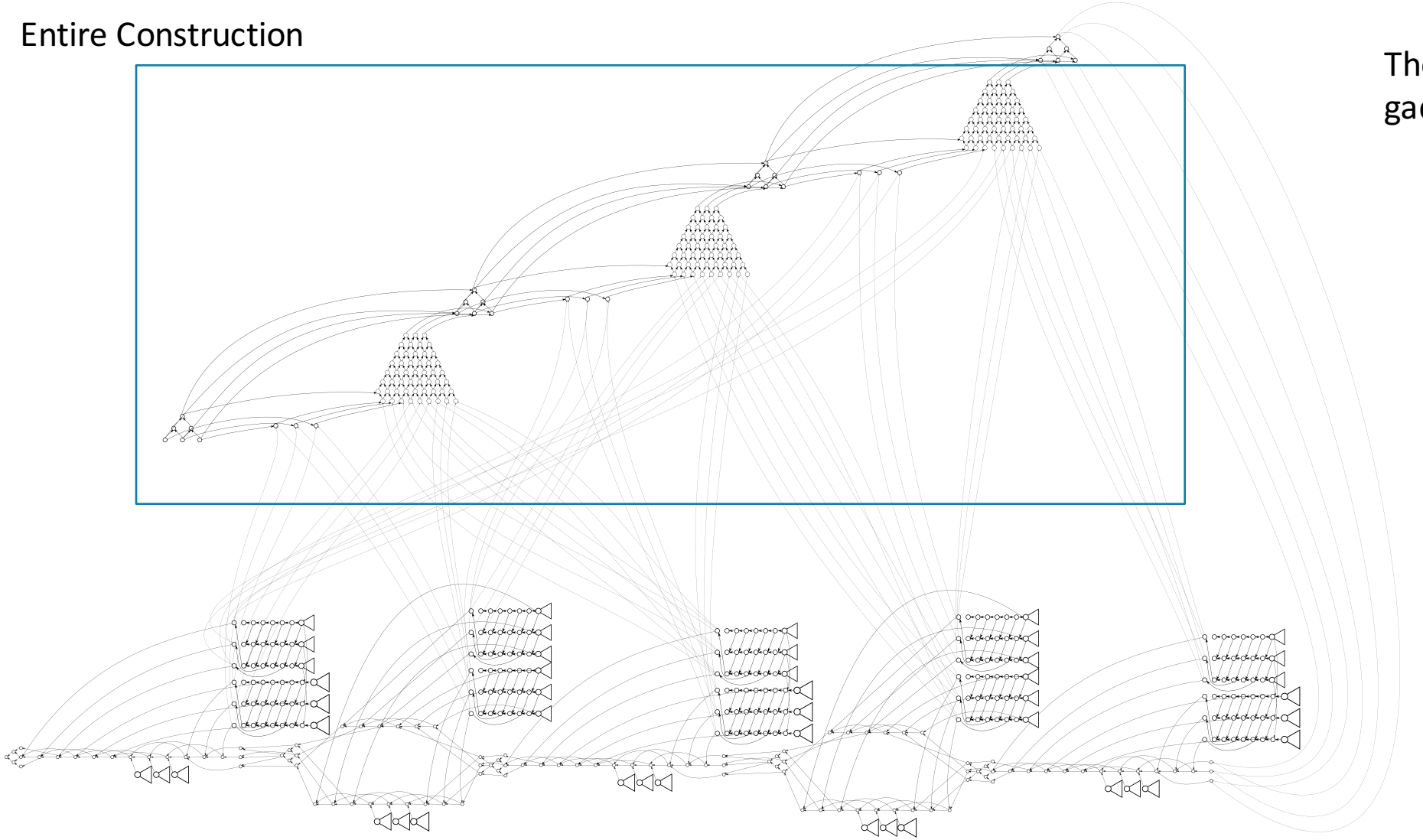
Duplicate first clause (not shown here)

Entire Construction



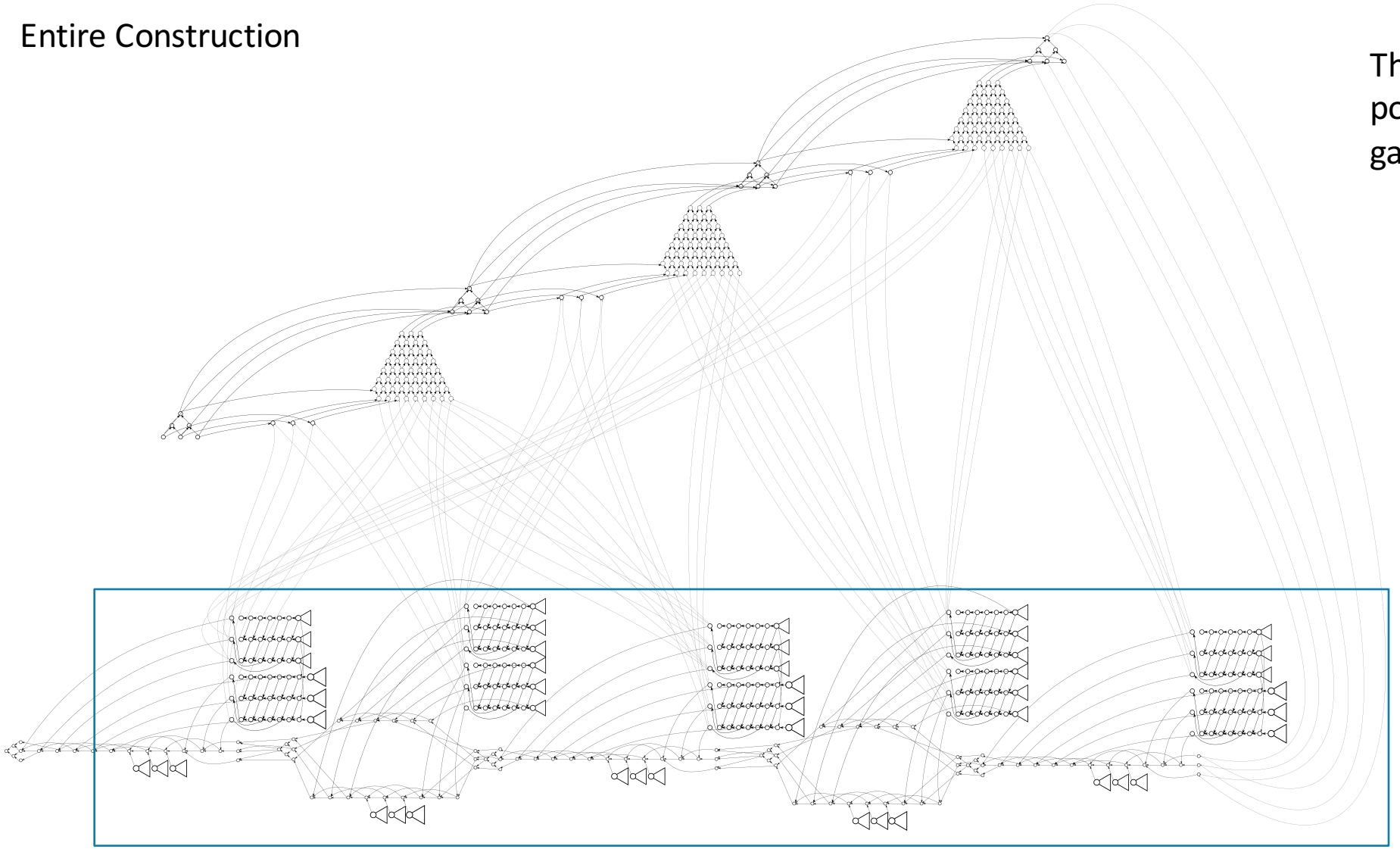
First pebble quantifier gadgets by assigning truth values to literals

Entire Construction



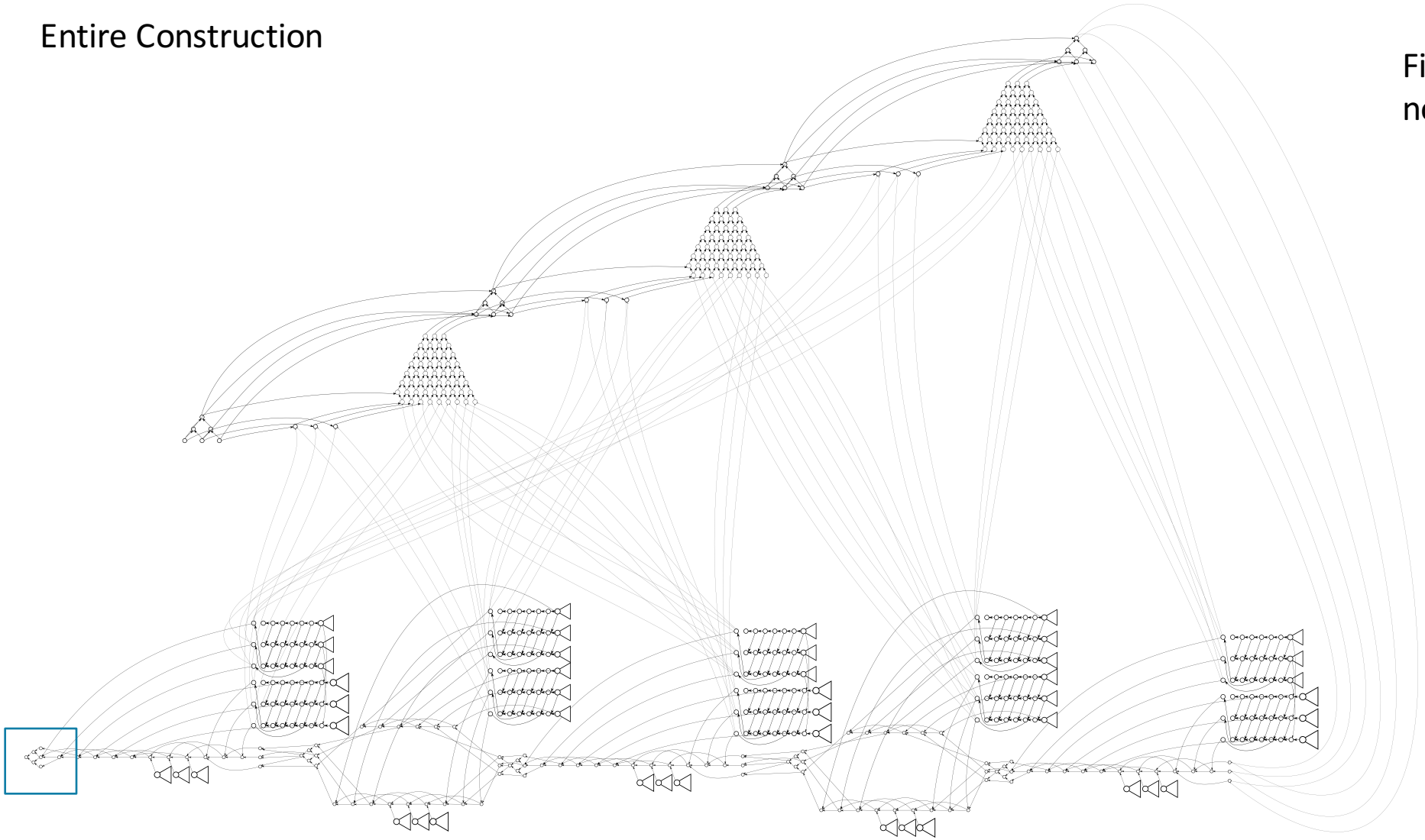
Then pebble clause gadgets

Entire Construction



Then pebble unpebbled portions of quantifier gadgets

Entire Construction



Finally pebble target node

PSPACE-hard to Approximate to Additive $n^{\frac{1}{3}-\epsilon}$ for $\epsilon > 0$

True instances B create G that requires $4Ku + 4K + 1$ pebbles

False instances B create G that requires $4Ku + 5K$ pebbles

Gap reduction with a gap of $K - 1$ pebbles

Total number of nodes in the graph is $n = O(K^3(u^3 + c))$

When $K \to \infty$, $K \approx n^{\frac{1}{3}}$, otherwise $K = O\left(n^{\frac{1}{3}}\right)$

Therefore, it is hard to approximate the number of pebbles required to pebble any DAG to an additive term $n^{\frac{1}{3}-\epsilon}$ for all $\epsilon > 0$

Hard to Pebble Graphs

The maximum time cost necessary to pebble any DAG is $O(n^k)$ where k is the minimum space cost

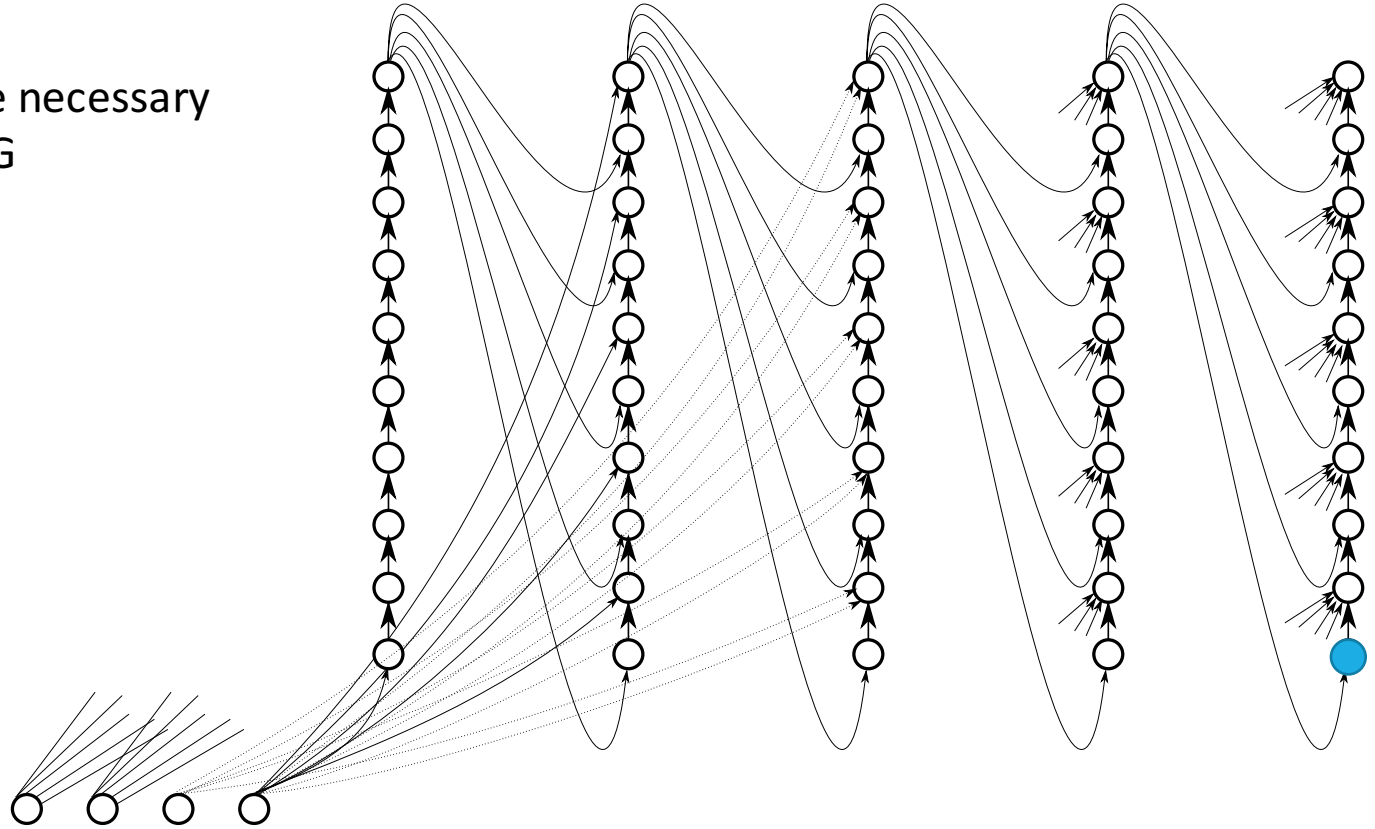
[AdRNV17] presented an independent explicit construction that require $\Omega(n^k)$ moves to pebble

We introduce an independent construction that have the following differences from the graph construction provided by [AdRNV17]:

- Steep tradeoff between exponential in k and linear cost in moves: useful for proofs of secure erasure (PoSE) and proofs of space
- Tradeoff also holds for some set of nonconstant k

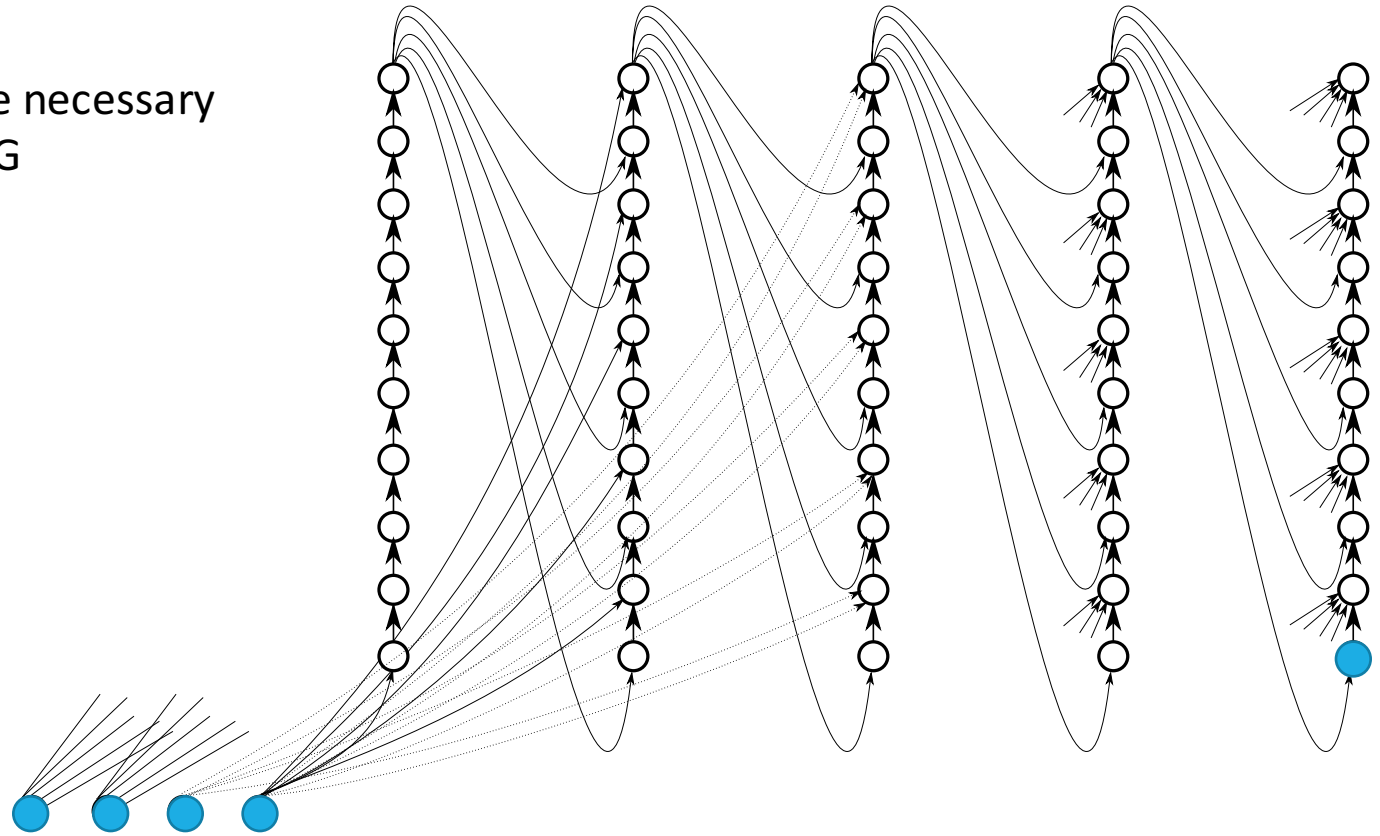
Hard to Pebble Graphs

$k = 5$ pebbles are necessary to pebble this DAG



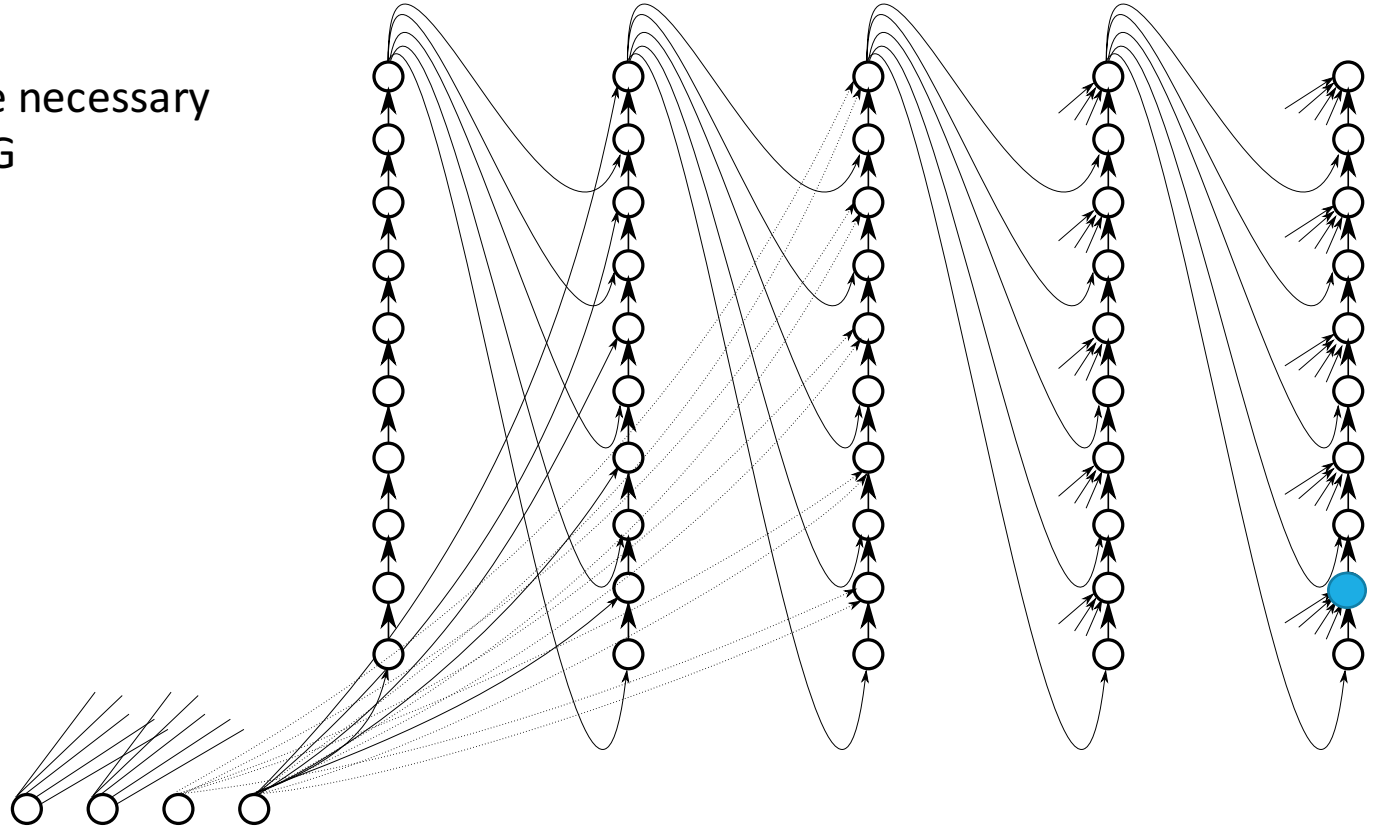
Hard to Pebble Graphs

$k = 5$ pebbles are necessary to pebble this DAG



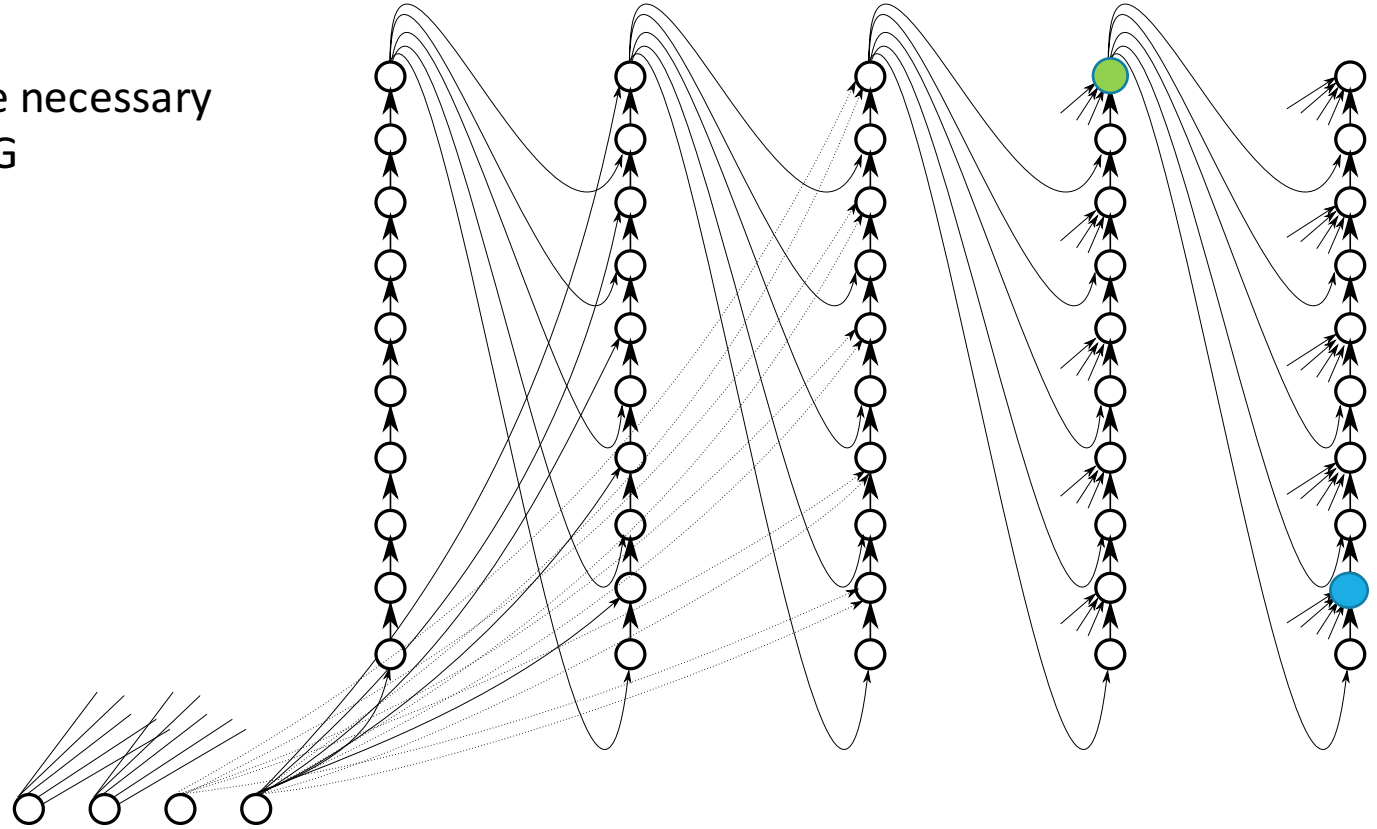
Hard to Pebble Graphs

$k = 5$ pebbles are necessary to pebble this DAG



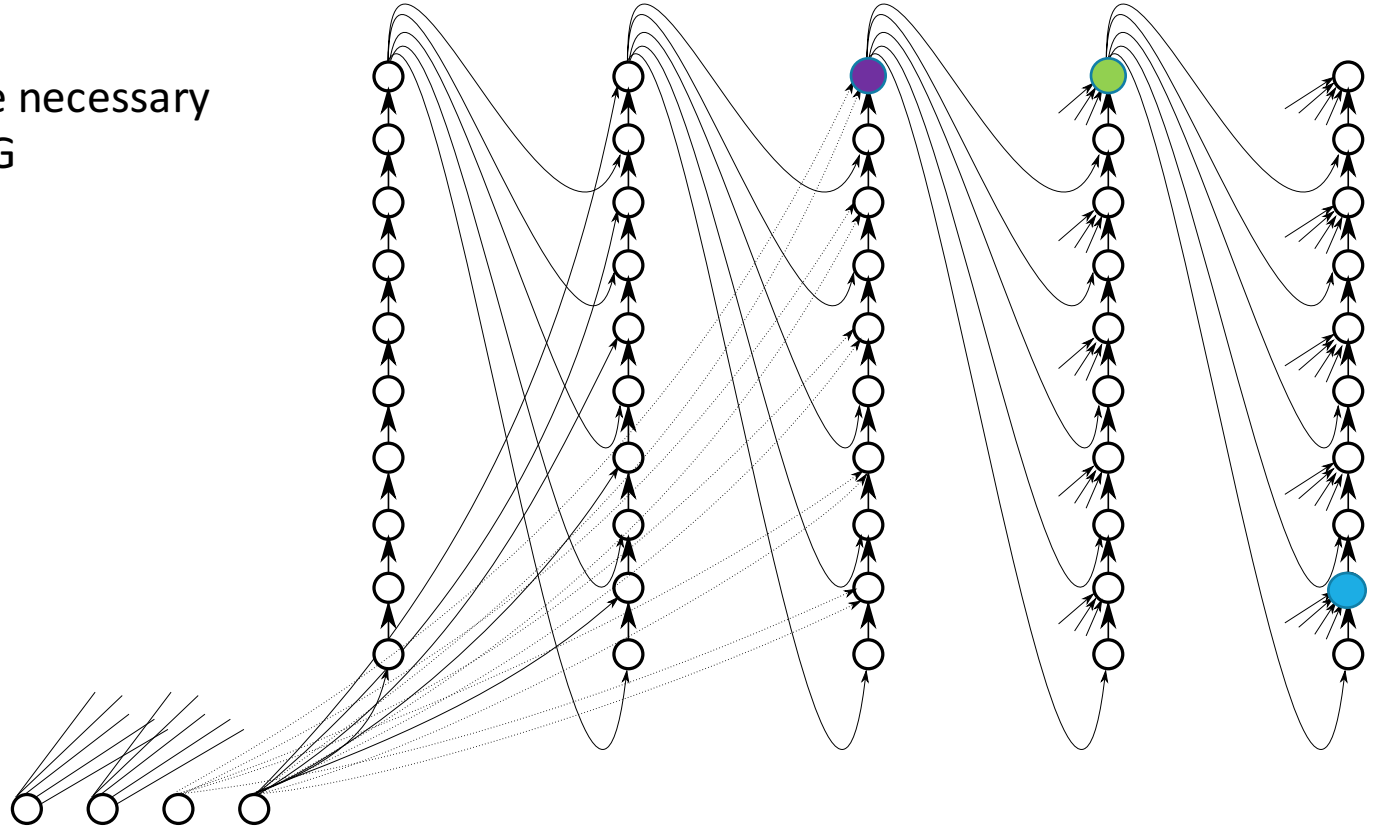
Hard to Pebble Graphs

$k = 5$ pebbles are necessary to pebble this DAG



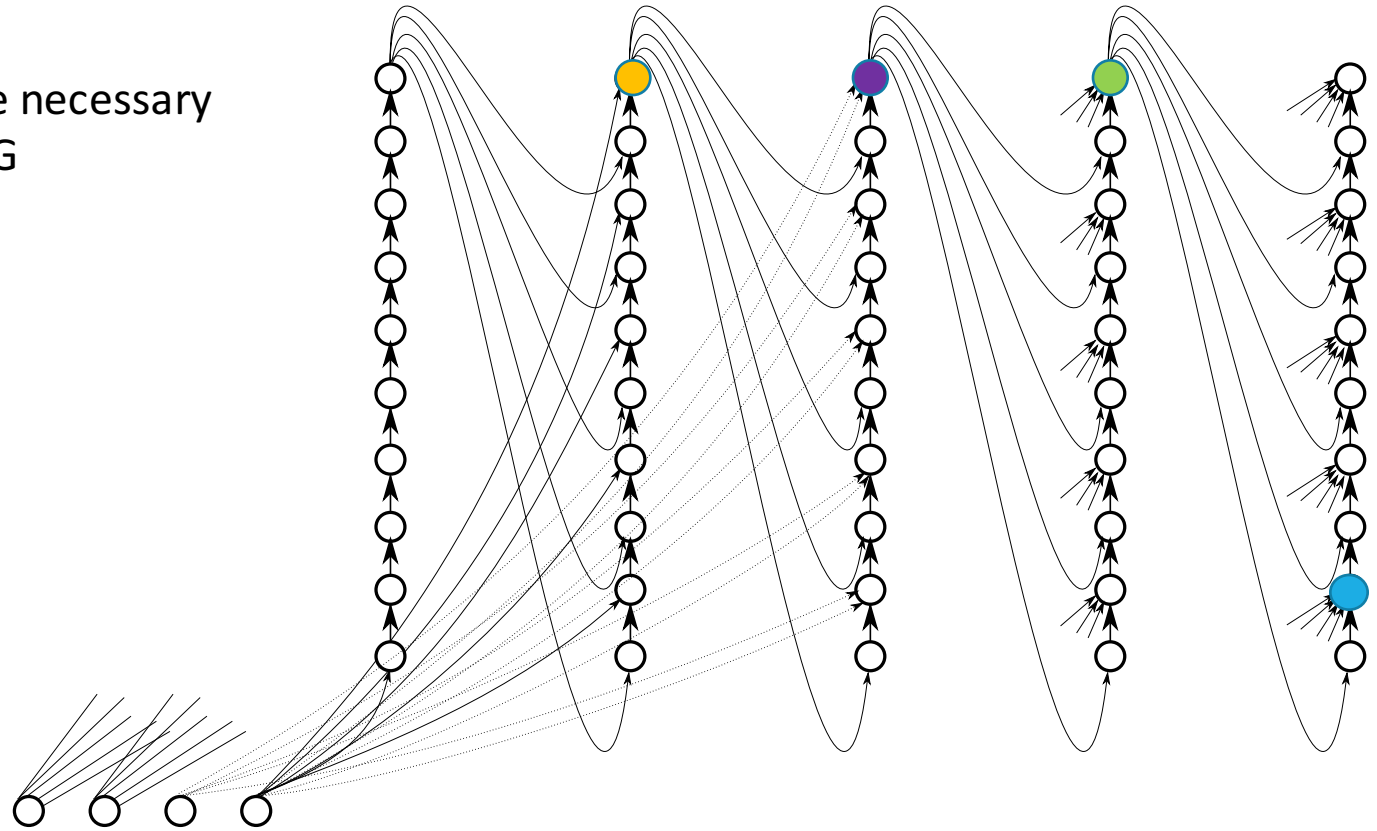
Hard to Pebble Graphs

$k = 5$ pebbles are necessary to pebble this DAG



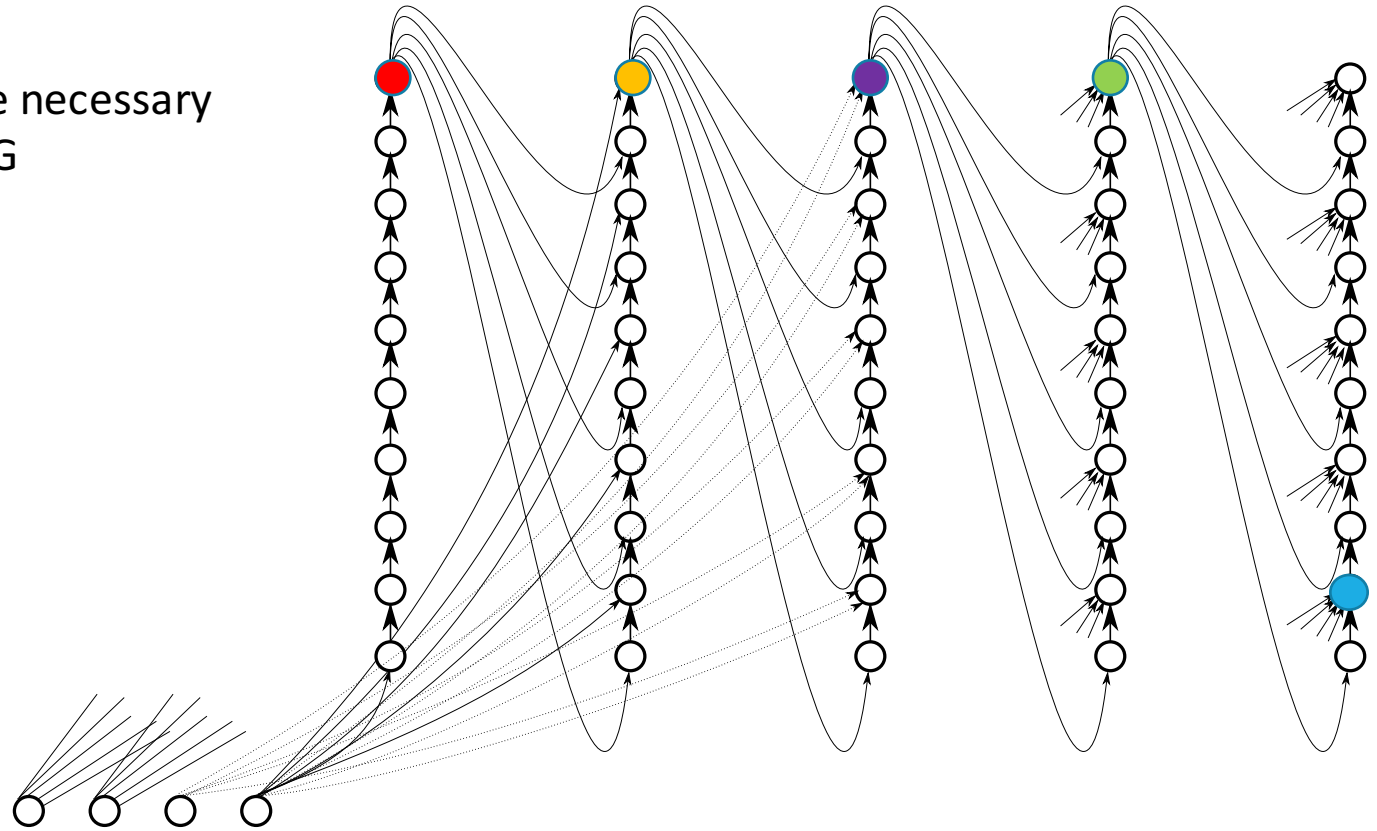
Hard to Pebble Graphs

$k = 5$ pebbles are necessary to pebble this DAG



Hard to Pebble Graphs

$k = 5$ pebbles are necessary to pebble this DAG



Hard to Pebble Graphs

In both the standard and black-white pebble games:

$$T(l) = \left(\frac{n-k}{2k}\right) T(l-1) + l\left(\frac{n-k}{2k}\right) \quad \text{where}$$

$$T(1) = \frac{n-k}{k}$$

Solving this recurrence gives:

$T(k) = \Omega(n^k)$ where k is constant for both standard and black-white games

Hard to Pebble Graphs

For max indegree-2 graphs, we can replace all indegree greater than 2 nodes by paths of pyramids or binary trees (see [DL17] for full description):

- Standard Pebble Game: $T(k) = \Omega\left(\left(\frac{n-k^2}{k^2}\right)^k\right)$ when $k < \sqrt{\frac{n}{2}}$
- Black-White Pebble Game: $T(k) = \Omega\left(\left(\frac{n-2^{2k-5}}{k^2}\right)^k\right)$ when $k = o(\log n)$

When k is constant $T(k) = \Omega(n^k)$ moves are necessary to pebble any member of this family

Open Questions

Using techniques presented, can we prove PSPACE-hardness of approximation for non-constant additive terms for the reversible and black-white pebble games?

Multiplicative hardness of approximation?

What is the exact pebbling cost of pyramids in the black-white pebble game?

Using ideas from construction, can we construct hard to pebble graphs for entire range of $0 < k \leq \frac{n}{\log(n)}$ with steep tradeoff?