11/21/2024

We shall start from the beginning

Cos`the beginning`s a good place to start

- Propositional Logic
- Atomic entities, the *propositions*
- Forming phrases or formulas through *connectives*
- Are they false? Are they true?’
- Are they always true? Are they always false?

Why Prop?

Because it works?

- Well, often!
- Benefit from well-developed algorithm
- Used in many, many applications
- From Theophrastus’ *modus ponens*
- To Boole’s calculus
- To SAT,   WSAT, Model Counting, BDDs, Circuits,...

Finite Domains

- WE have N variables X_i
- Where each takes a value in a domain $X_i \in D_j$
- And a function $F(X_1 \ldots X_n)$
- We want to satisfy the constraints: $F(X_1 \ldots X_n) = \mathbf{T}$
- Or find the best solution: $\text{maxarg} F(X_1 \ldots X_n)$

Constraint Programing

- Initially used constraint solvers: SICStusProlog, ECLIPSe;
- Others work as libraries: Gecode, and OR-Tools in C++, Choco and JaCoP in Java
- MiniZinc is a huge effort to construct a constraint programming language
- Both MiniZinc and OR-Tools use SAT solvers
- Translations from Zhou and from BEE

An Example

```
%   S E N D
% + M O R E
% ---------
% M O N E Y
```

Define the Problem

```
main(Letters) :-
     Letters =[S,E,N,D,M,O,R,Y] ,
    foldl(domain(0,9),Letters),
    plus(D,E,Y),
    carry(D,E,C0),
    plus(C0,N,R,E),
    carry(C0,N,R,C1),
    plus(C1,E,O,N),
    carry(C1,E,O,C2),
    plus(C2,S,M,O),
    carry(C1,E,O,C2),
     eq(C3,M).
```

Spec: Notes

- We have to specify:
  - Entities
  - Relationships
  - Notice we describe the computation of the addition

The booleans

- Translate each variable into 10 boolean variables:
- $E_0 \vee E_1 \vee E_2 \vee E_3 \vee E_4 \vee E_5 \vee E_6 \vee E_7 \vee E_8 \vee E_9$
- Besides we know

$$\forall E_i, E_j : E_i \rightarrow \neg E_j$$

- Or
- $\forall E_i, E_j : \neg E_i \vee \neg E_j$

The Equations

- The letters S E N D M O R Y must be different, eg:
- $E_i \rightarrow \neg M_i$
- Arithmetic
- We cannot write all posib;e suns directly,
- We just write down the arithmetic

$$(D_0 \wedge E_0 \rightarrow Y_0) \wedge (D_0 \wedge E_1 \rightarrow Y_1) \wedge \dots$$

Arithmetic

- We should also consider carry-in e carry-out
- $(c^0 \land D_0 \land E_0 \rightarrow Y_0 \land c_1) \land (c^0 \land D_0 \land E_1 \rightarrow Y_1 \land c^1) \land \dots$
- Notice the carry is a single bool per column
- $(c^0 \land D_0 \land Y_0 \rightarrow E_0 \land c_1) \land (c^0 \land D_0 \land Y_1 \rightarrow E_1 \land c^1) \land \dots$

Implementation

```
main :-
    Letters =[S,E,N,D,M,O,R,Y],
    foldl(domain(0,9),Letters),
    plus(C0,D,E,Y,C1),
    plus(C1,N,R,E,C2),
    C2=1,
    C0=0.
```

```
domain(I,J,Input,TVs*DVs) :-
   NVs is J+1-i,
   length(Vs,NVs),
   at_least_one_true(Vs,TVs),
   all_different(Vs, DVs).

at_least_one_true([V],V)
at_least_one_true([V,V2|Vs],(V+Vs)) :-
       at_least_one_true([V2|Vs],Vs).

all_different([V|Vs],F*Fs) :-
    differs(V,Vs,F),
all_different(   Vs,Fs).

differs(V,[V1],xor(V,V1)).
differs(V,[V1|Vs],xor(V,V1)*F1):-
   differs(V,Vs,F1),
```

Implementation

```
enumerate(C1,IX,IY,IZ,C2,(IZ+ -IX + -IY + -C0)*
                          C1+ -IX + -IY + -C0)):-
    member(Ci,[0,1]),
    member(IX,[0,1,2,3,4,5,6,7,8,9] ),
    member(IY,[0,1,2,3,4,5,6,7,8,9] ),
    IZ is (Ci+IX+IY) mod 10,
    C2 is (Ci+IX+IY)//10,
    IX\=IY,IZ \= IX, IY\=Z.
```

An Example

PySAT

- Interface to glucose solver: `G = Glucose3()`
- $A \to B$: `G.add_clause([-1,2])`
- $B \to C$ `G.add_clause([-2,3])`
- `G.solve()`
- `G.get_model()`

How does it work ?

Davis-Putnam-Logemann-Loveland (DPLL) Algorithm (1962)x

- Explores the search space of potential models, and backtracks
- We will define the algorithm as building up a partial model •
- A partial model assigns truth values to only some variables; a partial function •
- We will represent partial models by finite sets of literals (e.g.) •
- The algorithm returns either (sat,M) or unsat •
- in the former case, M will be a model for the input formula T

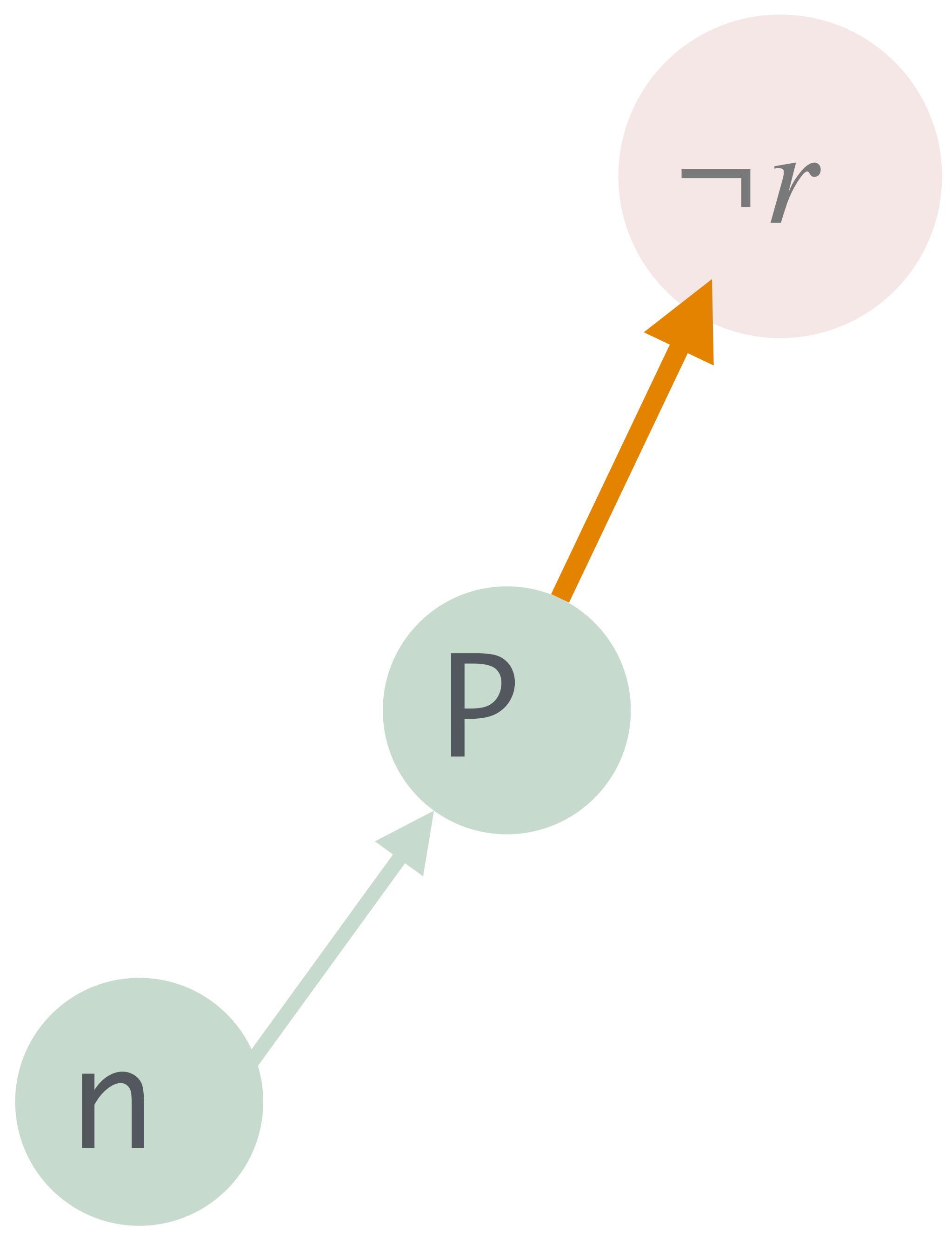
Propagation in DPLL

- If Λ is $\top$ then return (sat,M)
- If Λ contains an empty clause then return unsat
- Pure literal rule: If p occurs only positively (negatively) in Λ, delete clauses of Λ in which occurs, update to (to)
- Unit propagation:
  - If I is a unit clause $M \to \{I\} \cup M$
  - remove all clauses from Λ which have $\neg I$ as a disjunct, and
  - update all clauses in Λ containing I as a disjunct by removing that disjunct

Propagation

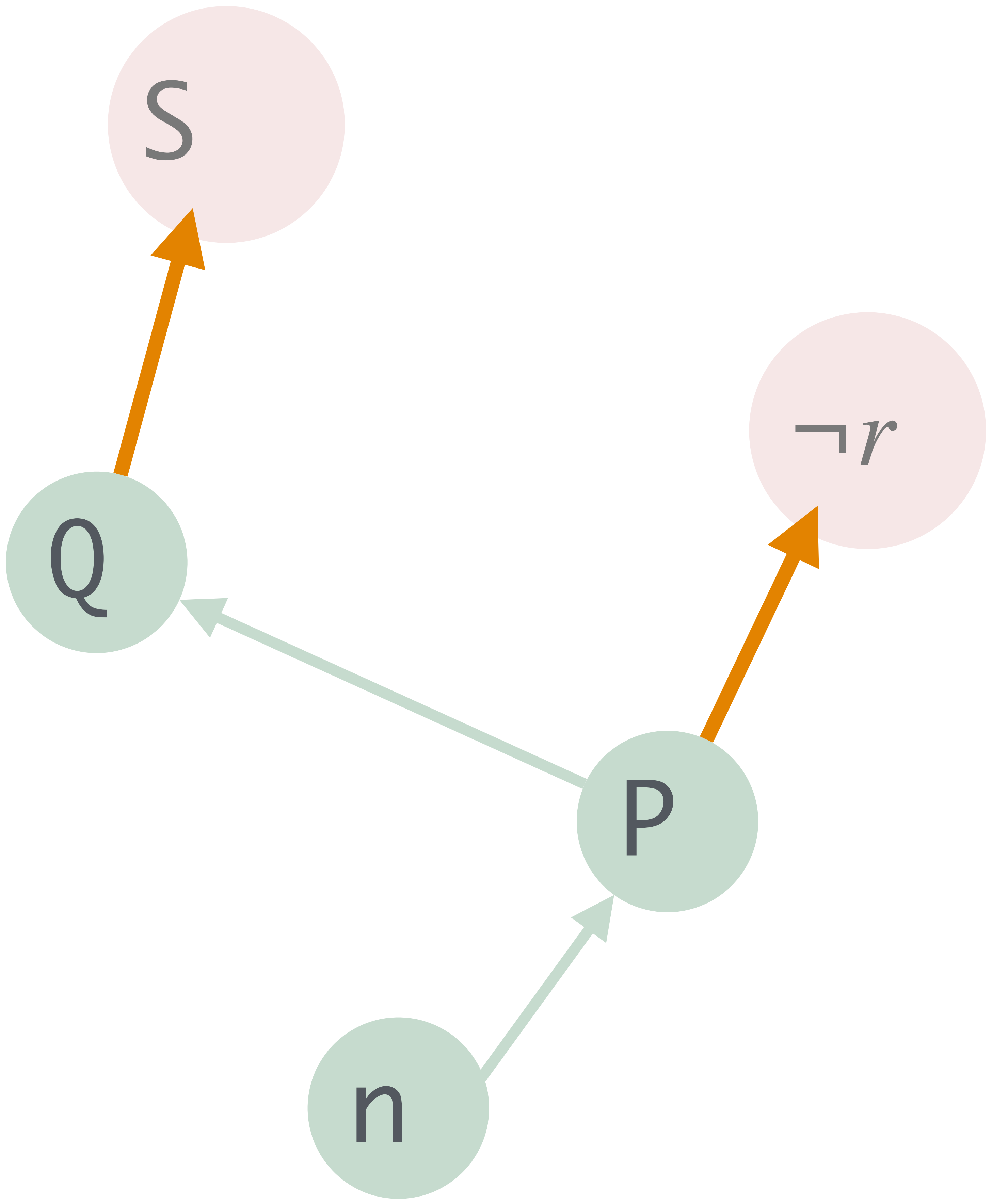
n+p
−n+p+s
−p+−r
−u+t
q+r+t
−q+s
−p+t+u
−p+−t+−u
r+−t+u

Propagation



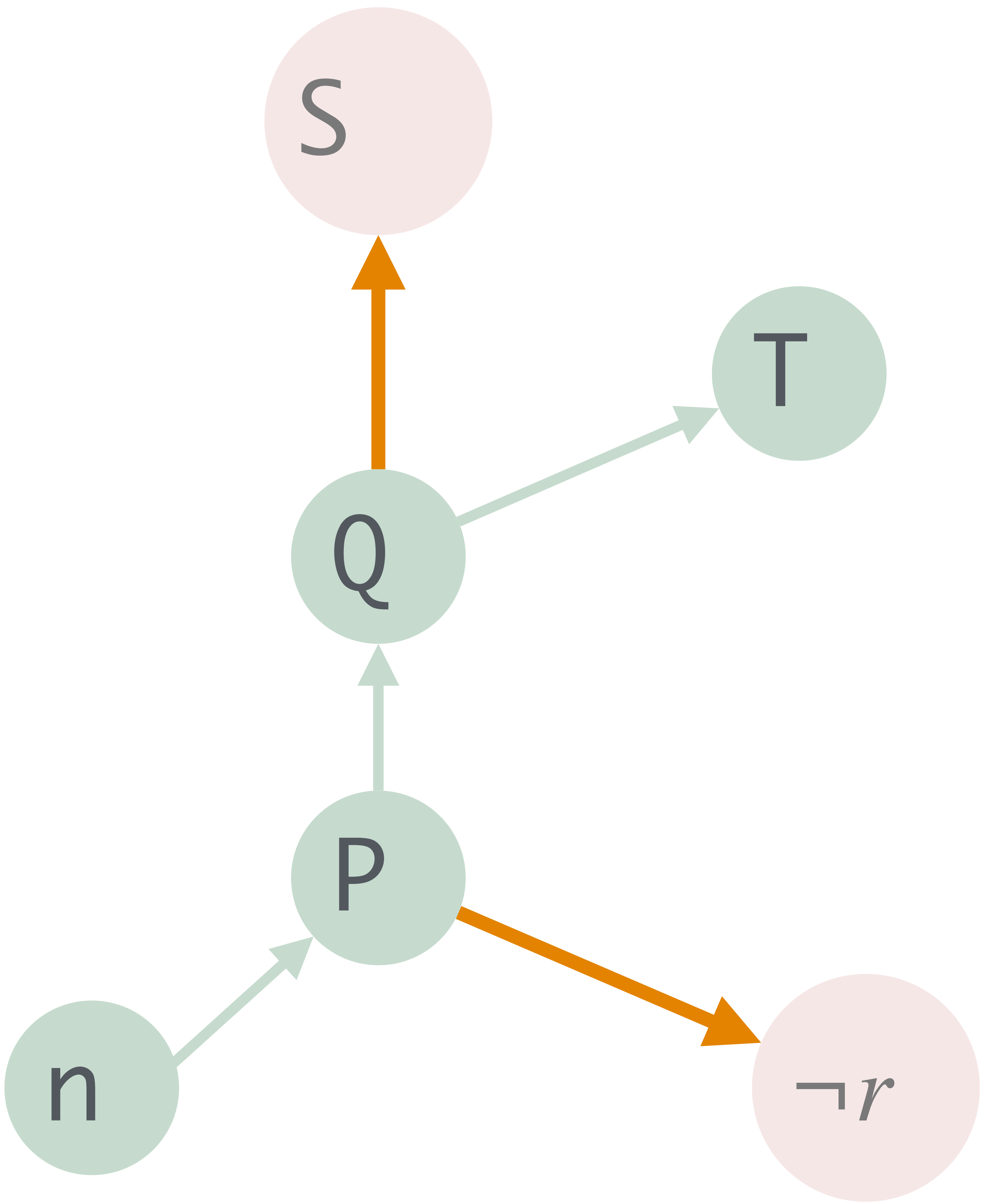
p+s
−p+−r
−u+t
q+r+t
−q+s
−p+t+u
−p+−t+−u
r+−t+u

Propagation



−u+t
−q+s
t+u
−t+−u
−t+u

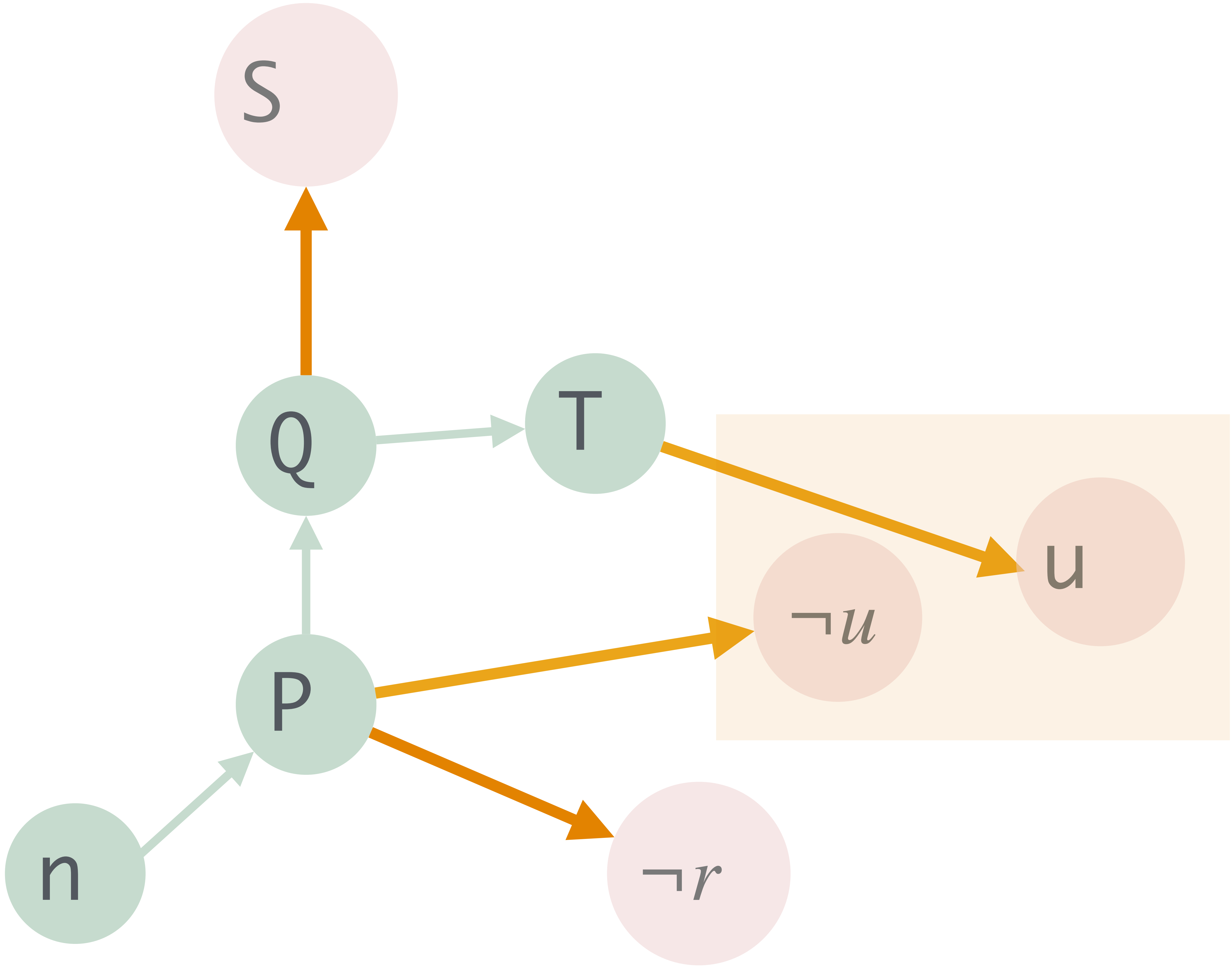
Propagation



CONFLICT!!

Conflict Resolution

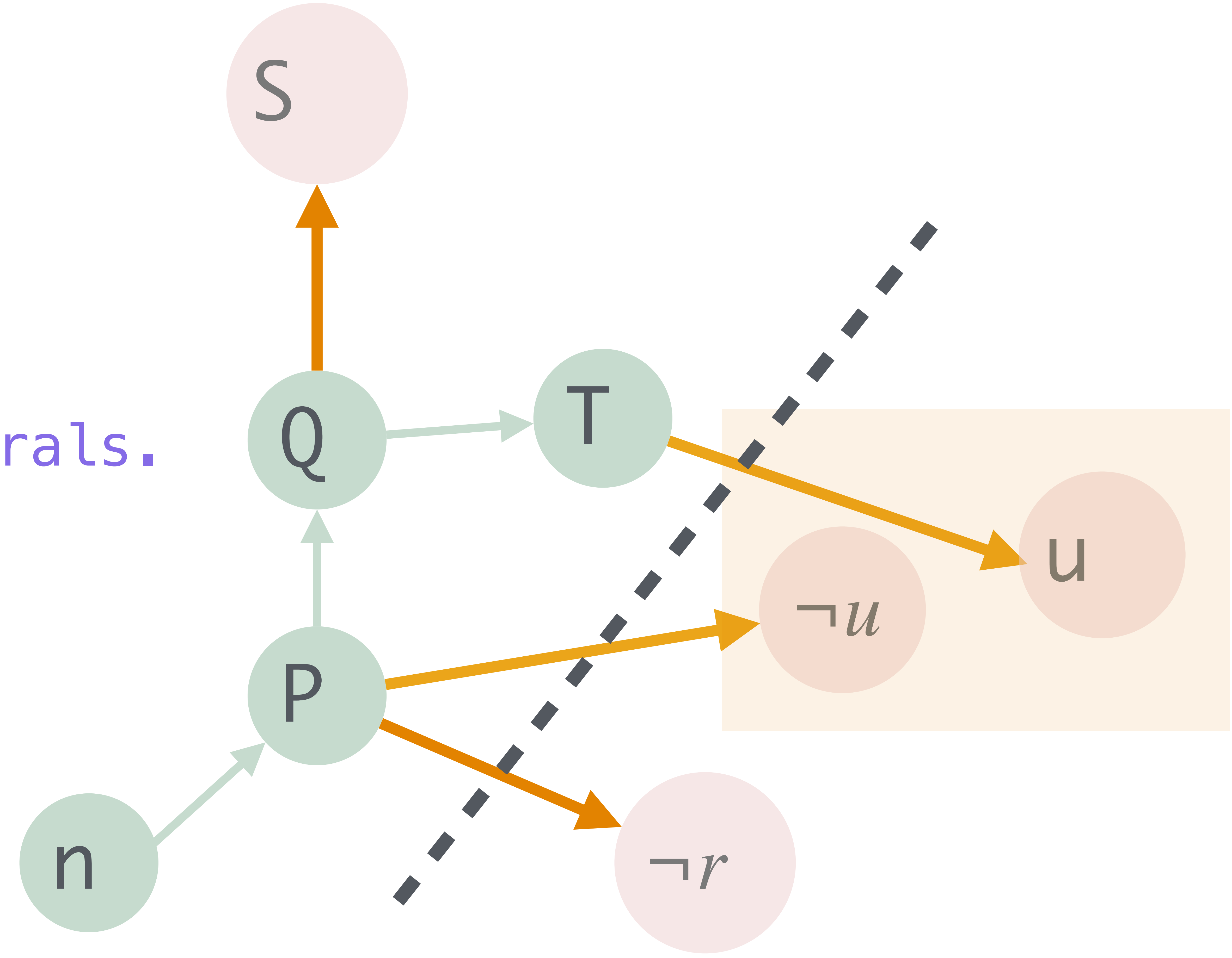
- Conflict between $\neg t \vee \neg p \vee \neg u$ and $r \vee \neg t \vee u$
- Involves three variables: t, p, r
- $\neg r$ follows from p
- DPLL must retry t or p
- Non-chronological backtracking
- `Pop t and q`
- `Try -t`
- `If it fails, -p`



Conflict Resolution

Learning

- To avoid the confict
- Cut the conflict
- Add disj of negated border literals.
- Eg: ¬P + ¬T



CDCL

Conflict-Driven Clause Learning

- DPLL plus heuristics for conflict handling
- Different ways to do clause learning:
  - “1-UIP strategy”: working backwards from conflicting literals, find the first (“latest”) node which is on all paths from the decision literal to be changed.
  - Learned rules may lead to earlier propagation
  - They may grow too quickly

SAT Research

SAT 2024 and CAV2024

- Symmetry
- Satzilla: using ML in SAT solving
- Compilation to OBDD, decision-DNNF
- Scalability and Parallelism
- Quantum Computing
- NeuralNetworks

ML of Boolean Networks

- Learning Local Search Heuristics for Boolean Satisfiability
- Boolean Decision Rules via Column Generation
- A boolean task algebra for reinforcement learning
- Boolformer: Symbolic Regression of Logic Functions with Transformers
- On Quantifying Literals in Boolean Logic and its Applications to Explainable AI (Extended Abstract)
-

Other representations

- And-Or Trees
  - Popular in Bayes Networks
  - Exploit independence
  - Exploit exclusiness
  - See work by Darwiche
  - We'll focus on Binary Decision Diagrams

OBDD

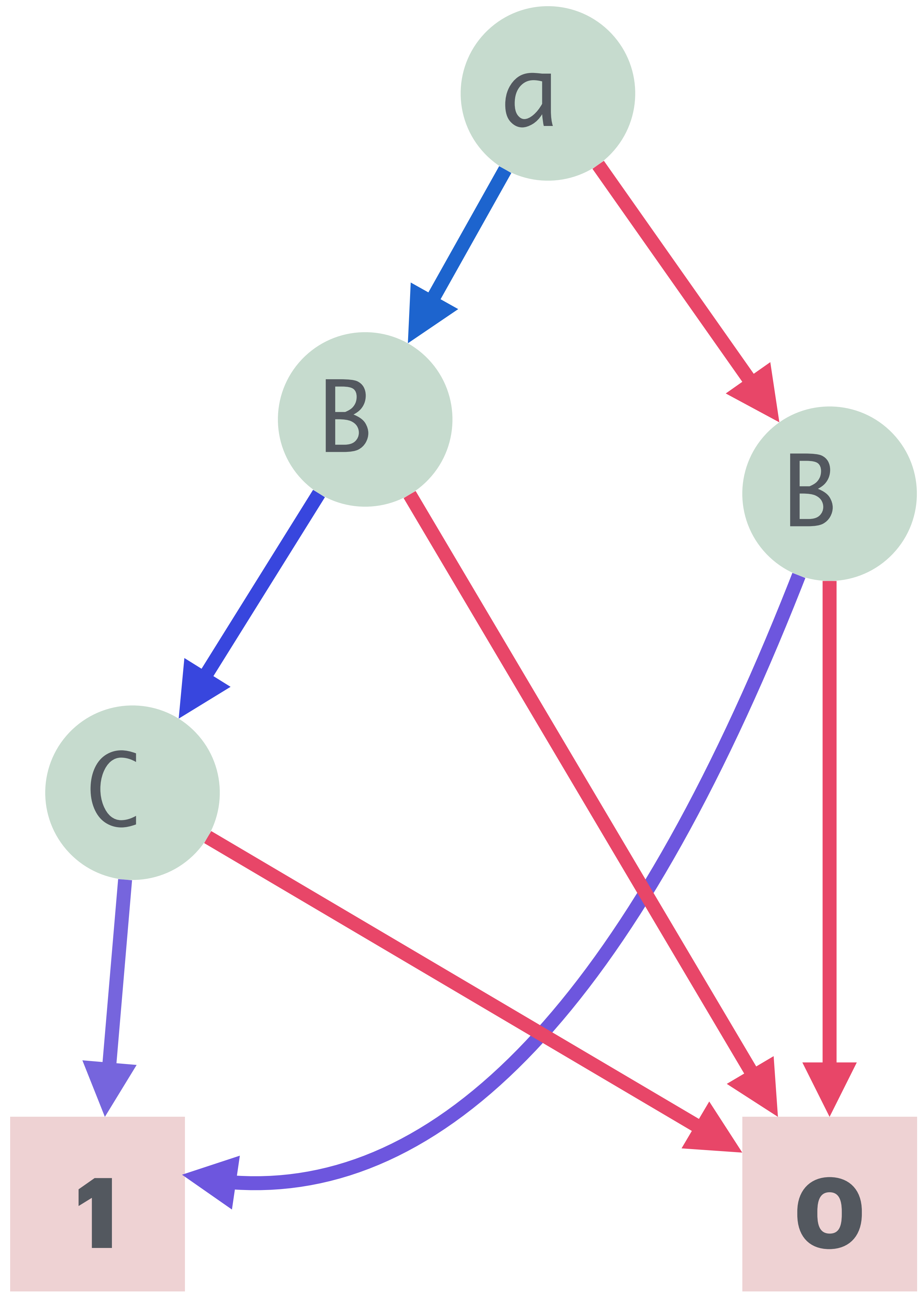
Work by Bryant

- Consider $a \land (\neg b \lor c) \lor b \land c \lor \neg a \land \neg c \land b$
- Same as $a \land (\neg b \lor c) \lor (a \lor \neg a) \land b \land c \lor \neg a \land \neg c \land b$
- Or $a \land (\neg b \lor c \lor b \land c) \lor \neg a \land (\neg c \land b \lor b \land c)$
- We can take c: $a \land (b \land c \lor \neg b) \lor \neg a \land (b)$

OBDD

$a \wedge (\neg b \vee c) \vee b \wedge c \vee \neg a \wedge \neg c \wedge b$

$a \wedge (b \wedge c \vee \neg b) \vee \neg a \wedge (b)$



OBDD

How?

- Split on variables until reaching true or false
- For each node T with left-subtree VT and right-subtree FT:
- $V = T \wedge VT \vee \neg T \wedge FT$
- Also, also choose the same order for selecting variables, independent of branch
- Plus, merge equal bottom subtrees

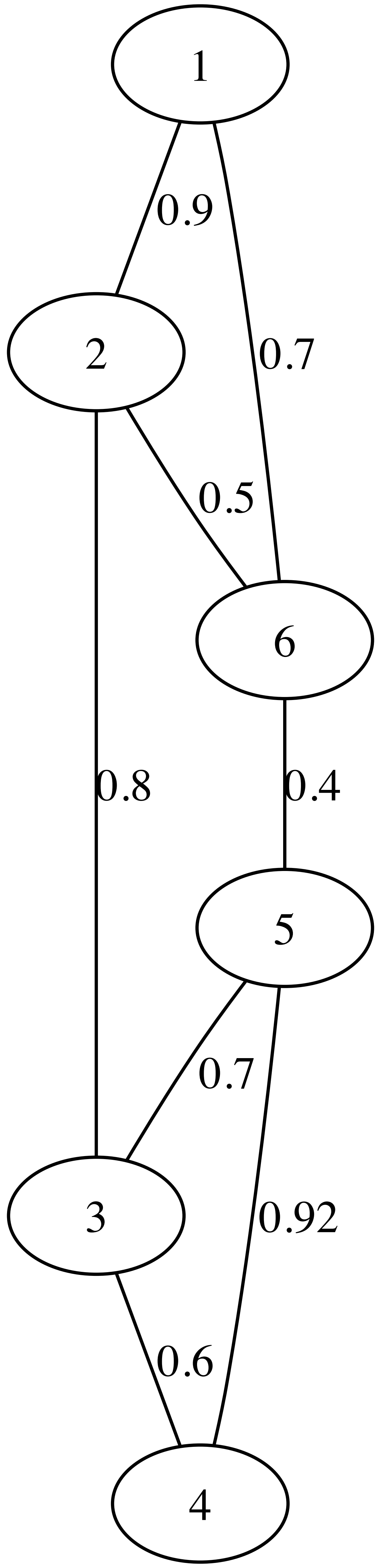
OBDD

Why

- Compiled Representation of theory
- $a \neg b c$ is true?
- $\neg a \neg b \neg c$ is true?
- Just follow the path in the graph
- Is the theory satisfiable?
- Just look for a path from top to 1, or 1 to top
- Is the theory a tautology?
-

A Problog program for path finding

```
% An example graph:
0.9::edge(1,2).
0.8::edge(2,3).
0.6::edge(3,4).
0.7::edge(1,6).
0.5::edge(2,6).
0.4::edge(6,5).
0.7::edge(5,3).
0.2::edge(5,4).
```



Probabilities

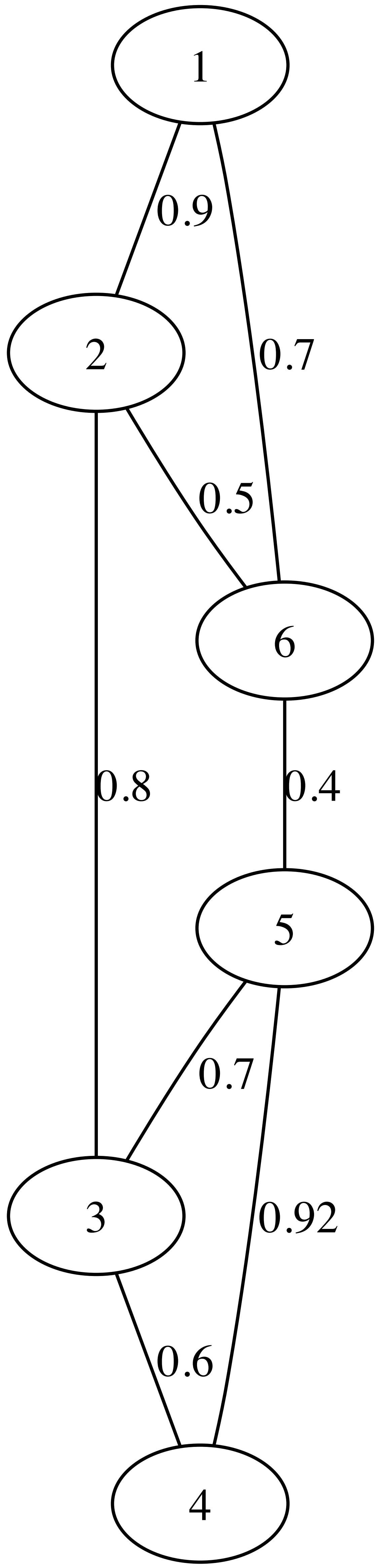
Prolog

```
%%%%
% definition of acyclic path
%using list of visited nodes
path(X,Y) :- path(X,Y,[X],_).

path(X,X,A,A).
path(X,Y,A,R) :-
  X\==Y,
  edge(X,Z),
  absent(Z,A),
  path(Z,Y,[Z|A],R).

% using directed edges in both
% directions
edge(X,Y) :- dir_edge(Y,X).
edge(X,Y) :- dir_edge(X,Y).

% checking whether node hasn't
% been visited before
absent(_,[]).
absent(X,[Y|Z]):-X \= Y, absent(X,Z).
```

Computing Probabilities

Prob of reaching node 4 from node 1

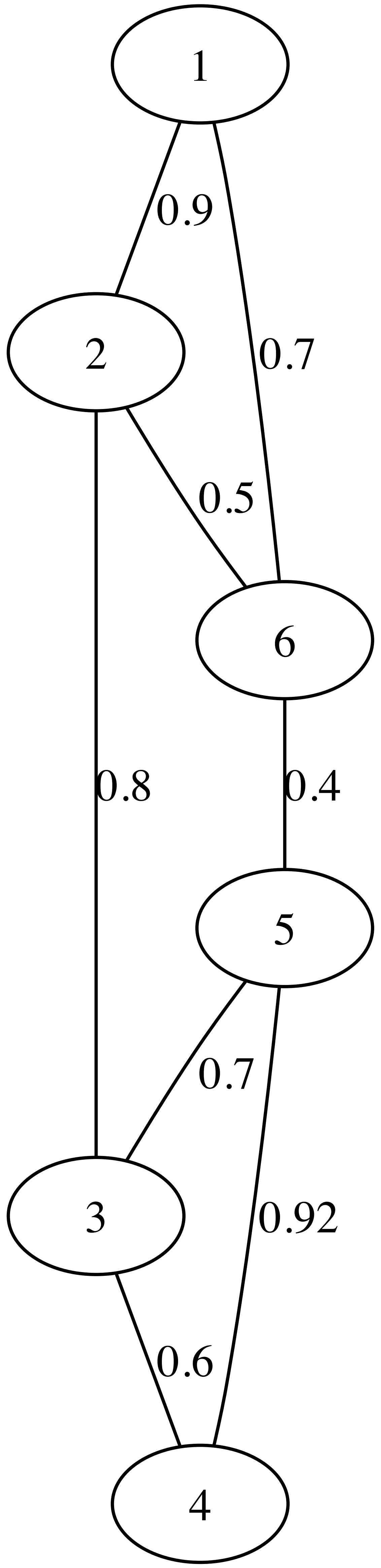
- Pr(1…4) is the union of
  - Pr(1,2,3,4) =Pr(1–2)Pr(2--3)Pr(3--4)
  - Pr(1,2,3,5,4) = Pr(1–2)Pr(2--3)Pr(3--5)Pr(5--4)
  - Pr(1,2,6,5,3,4) = Pr(1--2)Pr(2--6)Pr(6--5)Pr(5--3)Pr(3--4)
  - Pr(1,2,6,5,4) = Pr(1--2)Pr(2--6)Pr(6--5)Pr(5--4)
  - Pr(1,6,5,3,4) = Pr(1--6)Pr(6--5)Pr(5--3)Pr(3--4)
  - Pr(1,6,5,4) = Pr(1--6)Pr(6--5)Pr(5--4)
- **Union is a Sum-Product**

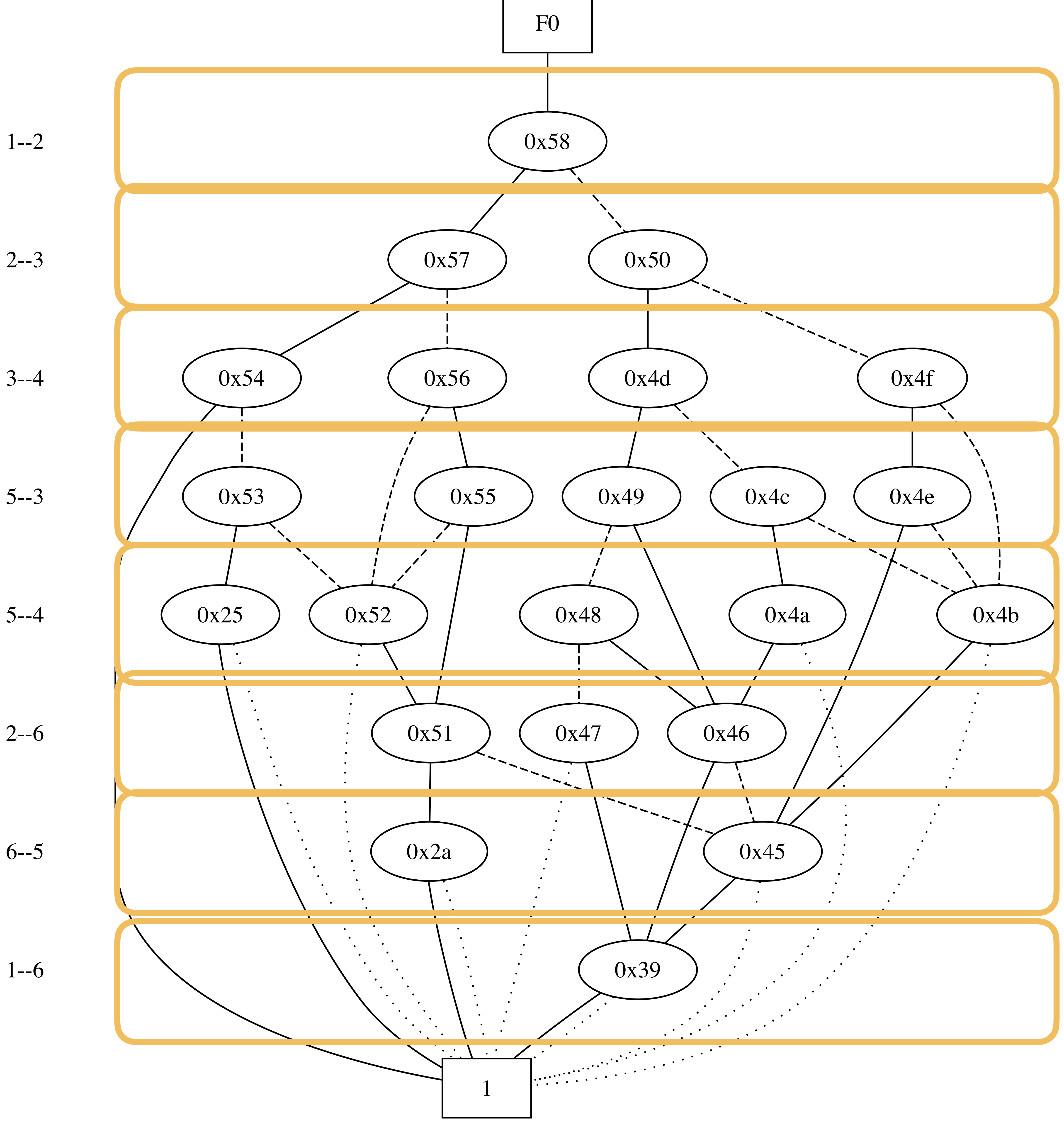


BDD to the rescue

Pr(1—4)?

- BDDs split on a variable
  - $:V \wedge T \vee \neg V \wedge F$
  - The splits in the same order
  - Generate layers

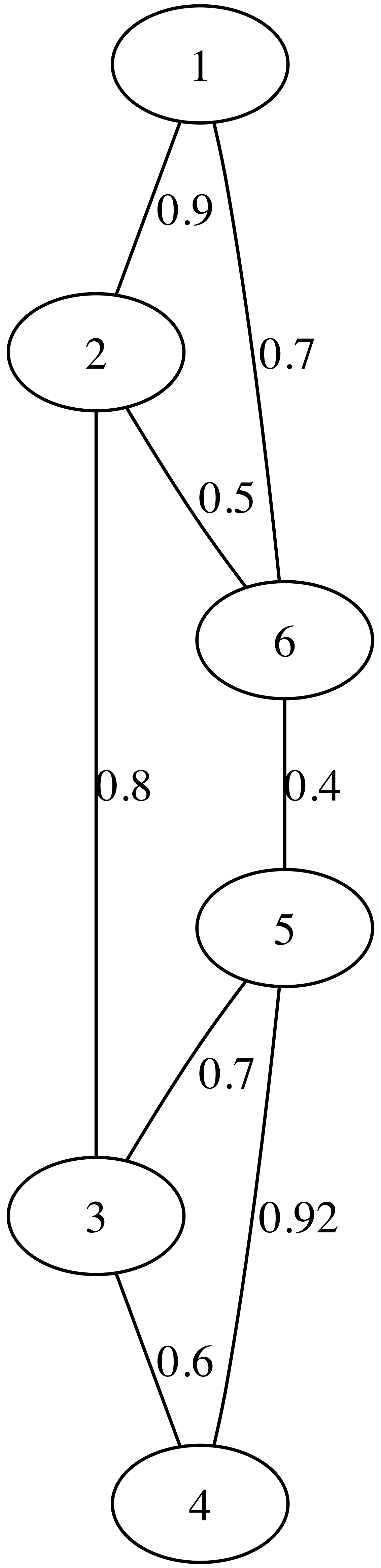


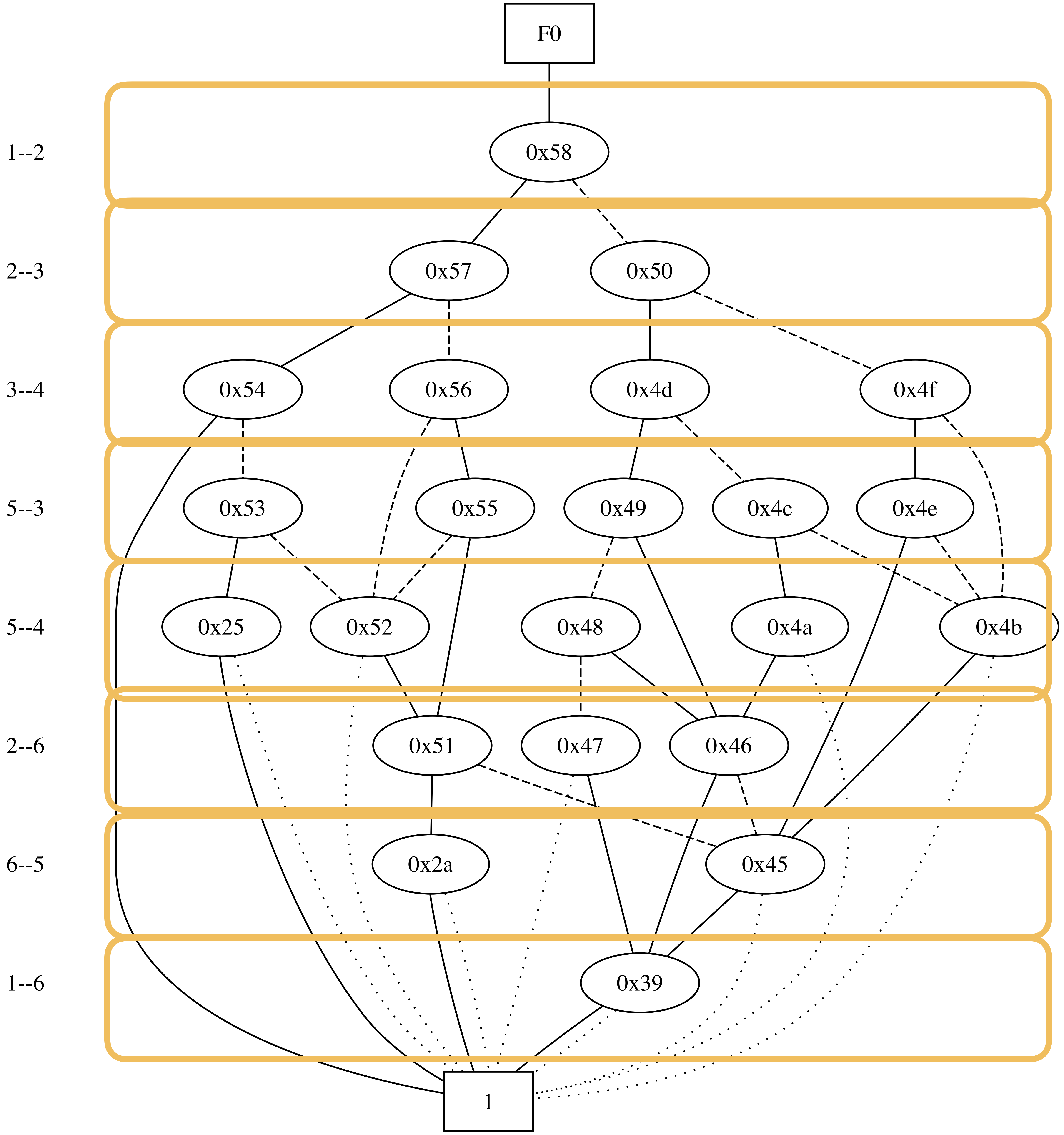


BDD to the rescue

Pr(1—4)?

- BDDs split on a variable
  - $:V \wedge T \vee \neg V \wedge F$
- Problog softens split
  - $Pr(T)\theta_v + Pr(F)(1-\theta_V)$
  - Compute bottom up

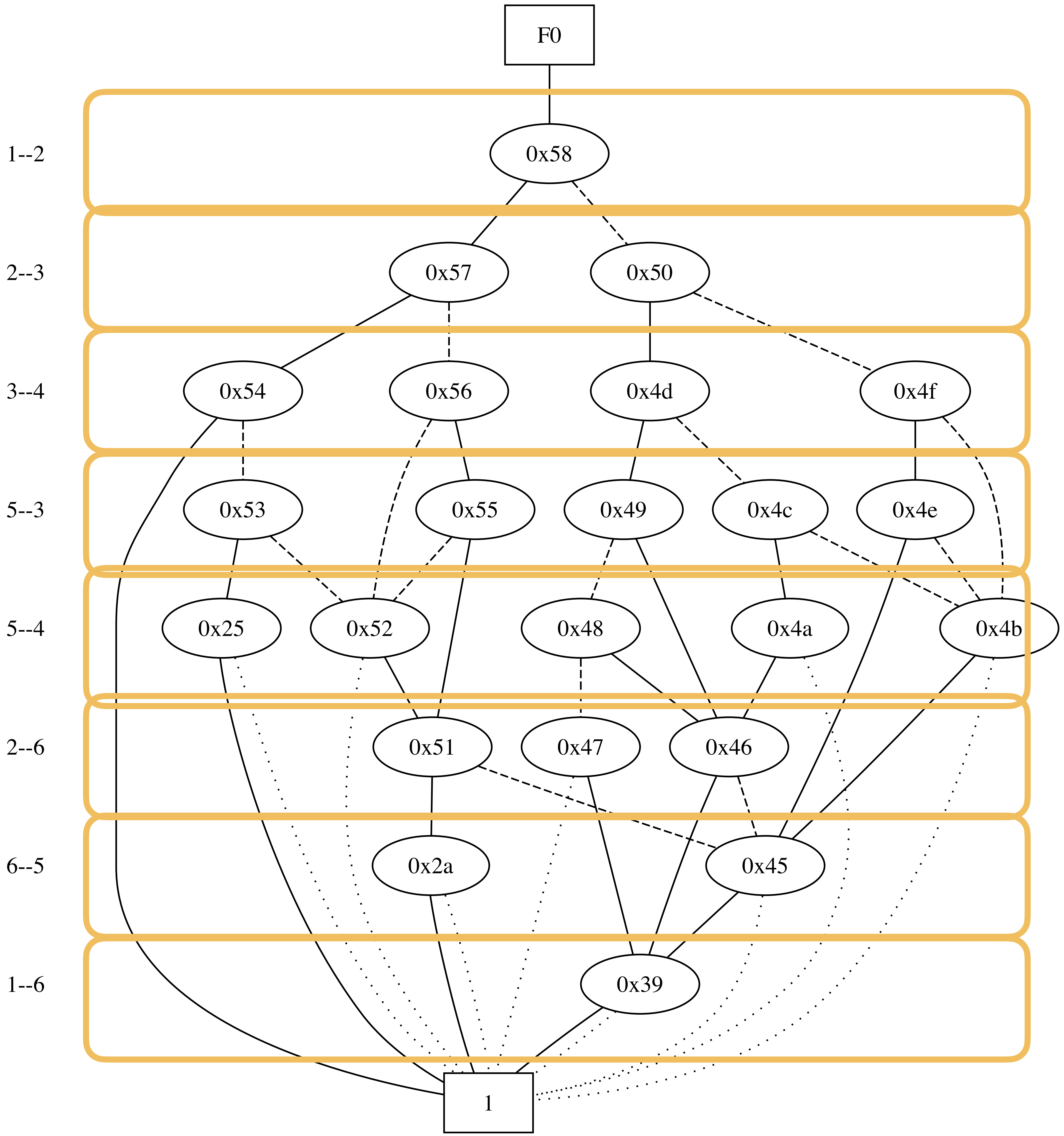




BDD to the rescue

Pr(1—4)?

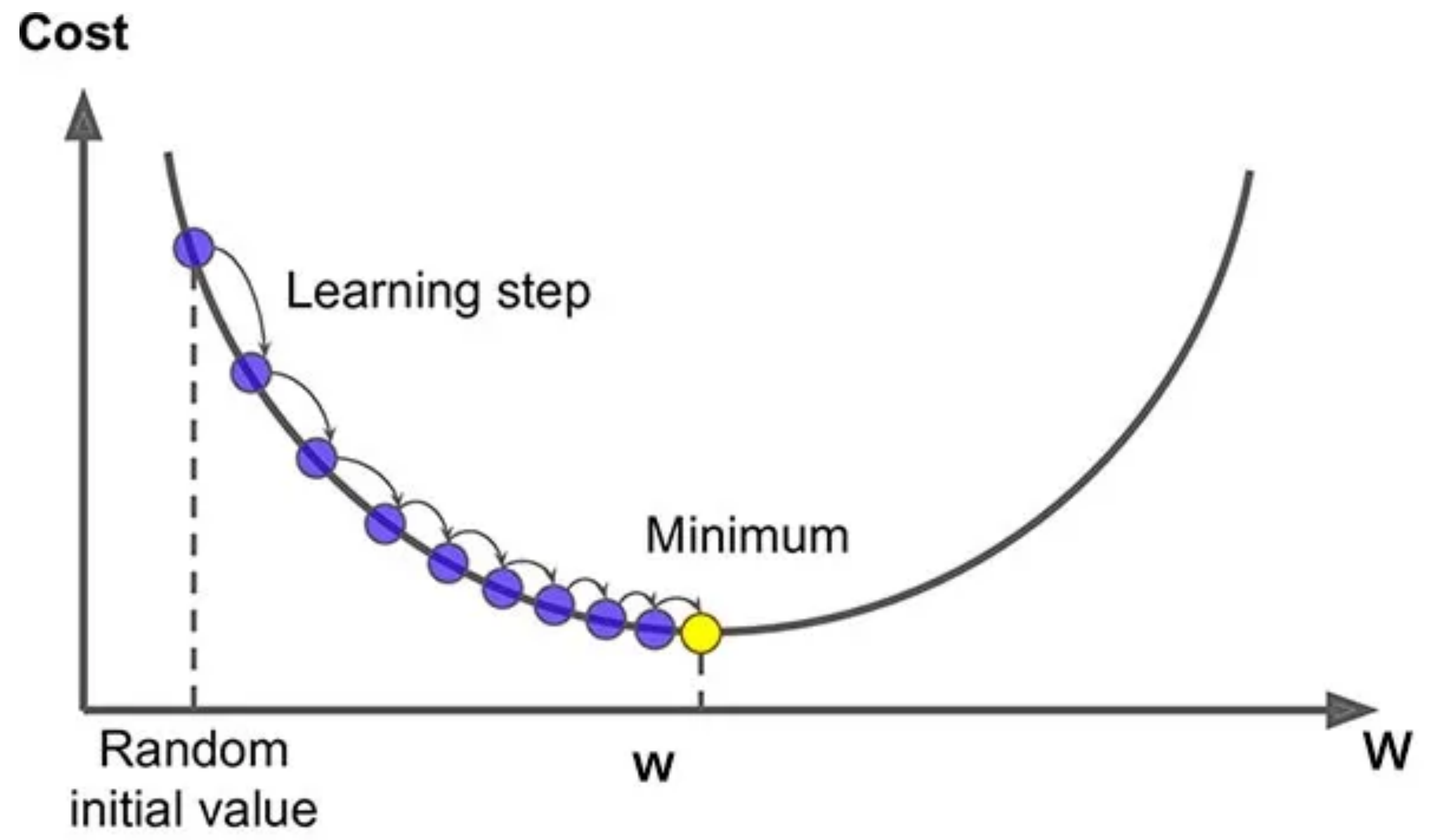
- Problog softens splilt
- $Pr(Node) = \theta_v Pr(T) + (1 - \theta_V) Pr(F)$
- $Pr(0x2a) = \theta_{56} + (1 - \theta_{56})(1 - 1) = \theta_{56}$
- $Pr(0x39) = \theta_{16} + (1 - \theta_{16})(1 - 1) = \theta_{16}$
- $Pr(0x45) = \theta_{16}\theta_{56} + (1 - \theta_{56})(1 - 1) = \theta_{16}\theta_{56}$
- Pr(0x51)...
- Until we reach F



Parameter Learning

Probabilties from ex

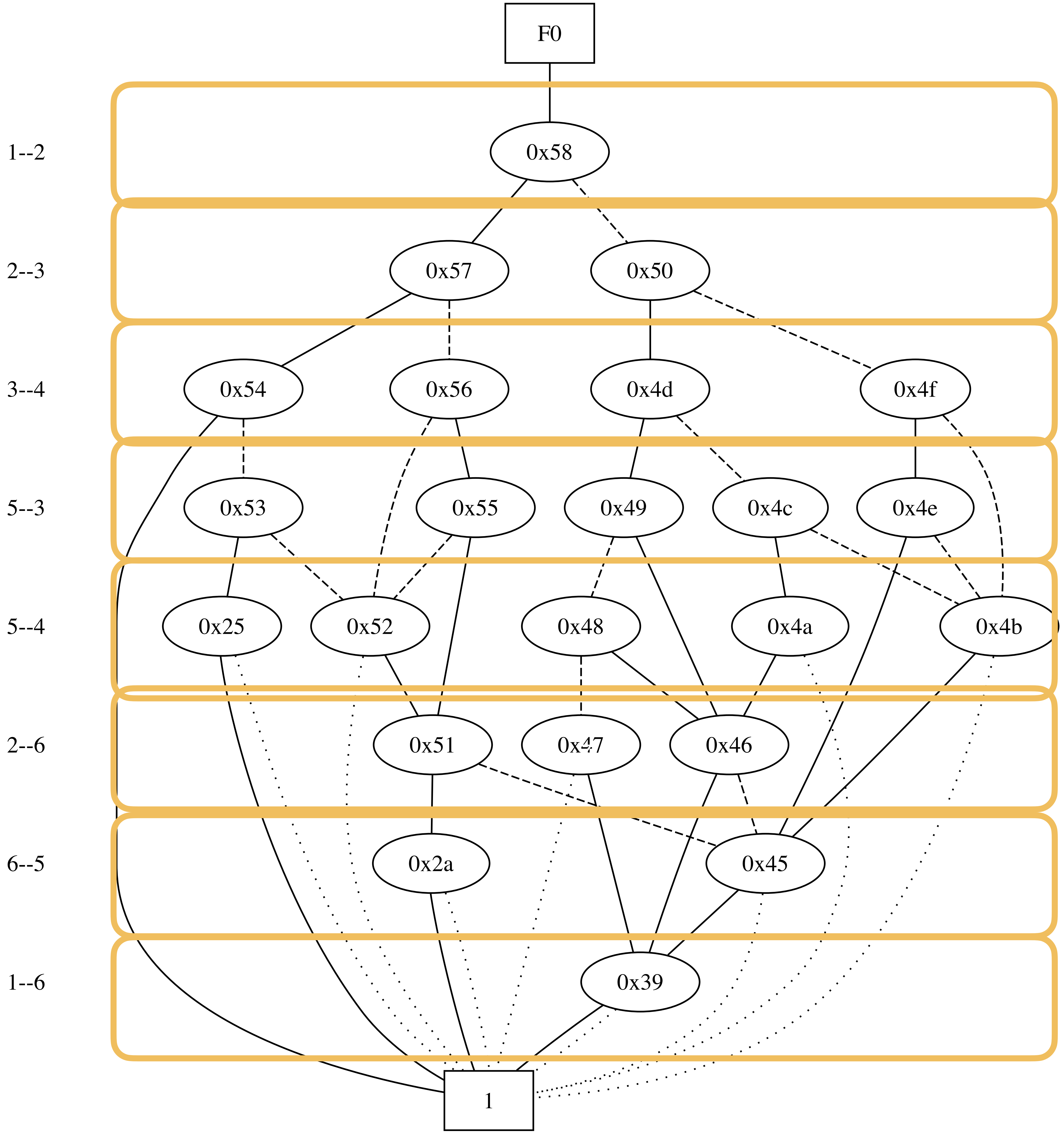
- We want to minimise $\sum_i |\hat{F}(\mathbf{x_i}) - y_i|$ ::
- To use gradient descent
- solve $\sum_i \frac{\delta \hat{F}(\mathbf{x_i})}{\delta \theta_{ij}} = 0$
- How to compute a derivative for a tree?



The Gradient

Let us obtain $\delta F / \delta\theta_{34}$

Using $Pr(T)\theta_v + Pr(F)(1 - \theta_V)$

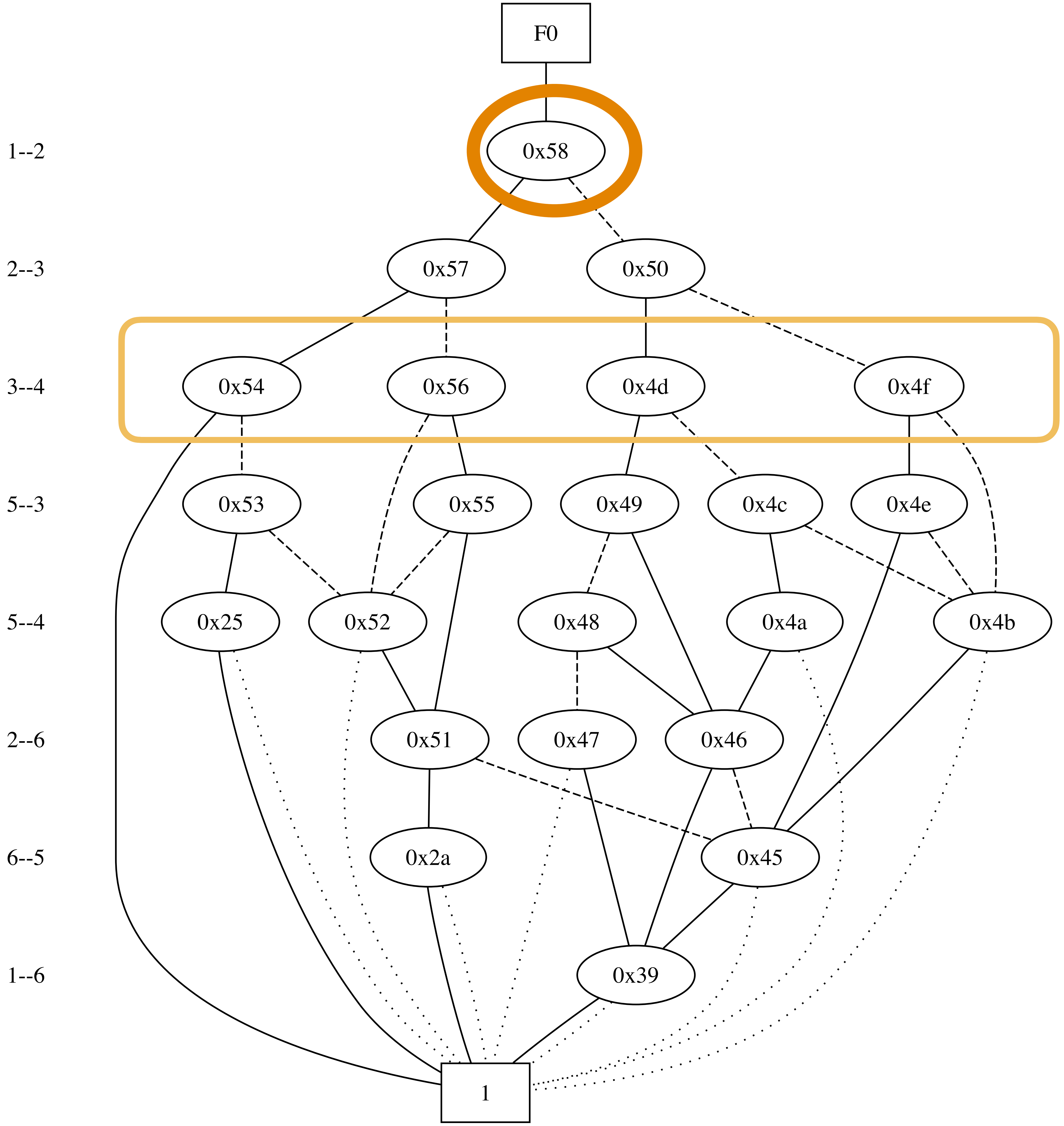


The Gradient

Let us obtain $\delta F / \delta \theta_{34}$

Using $Pr(T)\theta_v + Pr(F)(1 - \theta_V)$

$$F = \theta_{12}F(0x57) + (1 - \theta_{12})F(0x59)$$

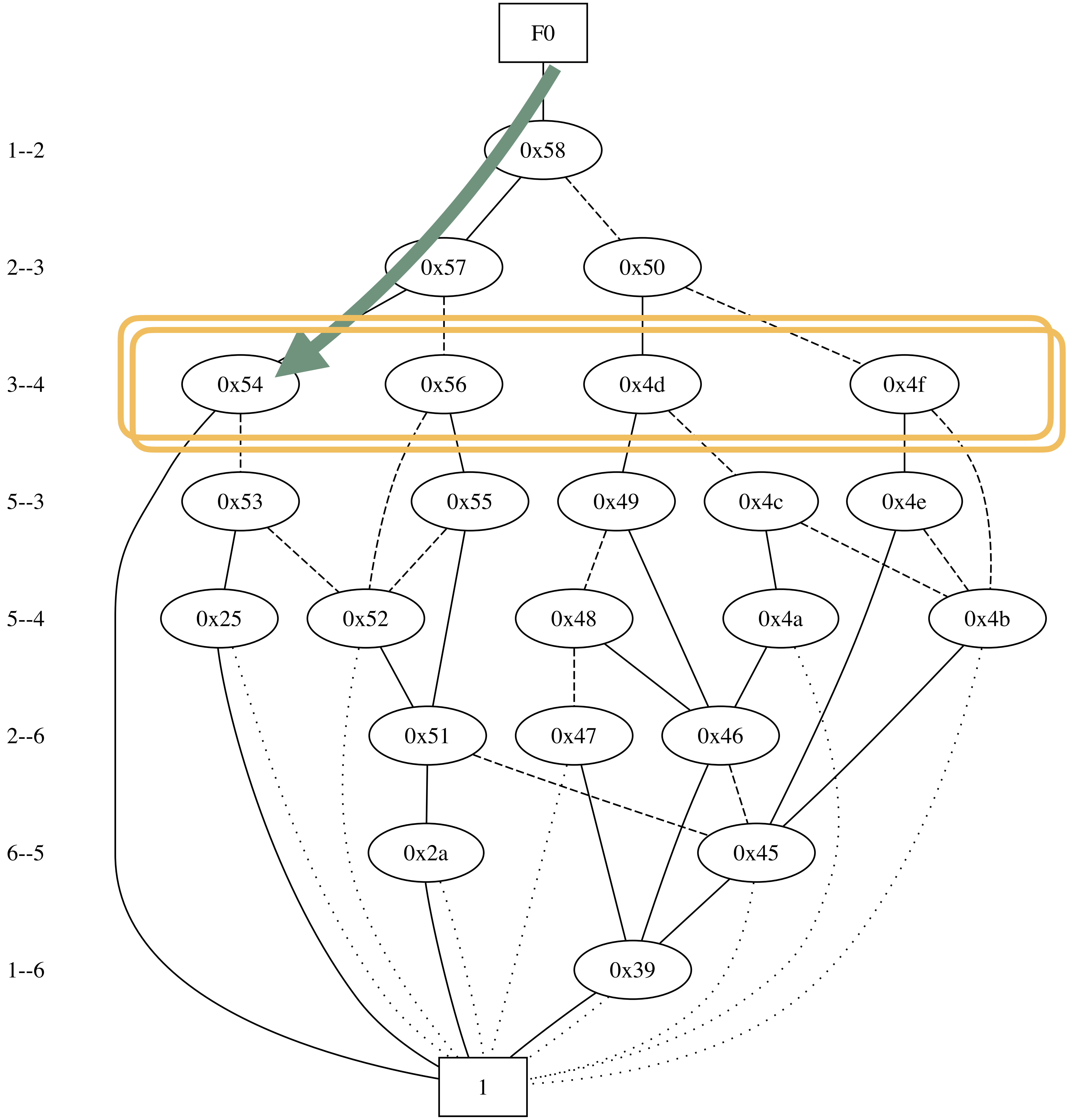


The Gradient

Let us obtain $\delta F / \delta \theta_{34}$

Using $Pr(T)\theta_v + Pr(F)(1 - \theta_V)$

$$F = \theta_{12}\theta_{23}F(0x54) + \ldots$$

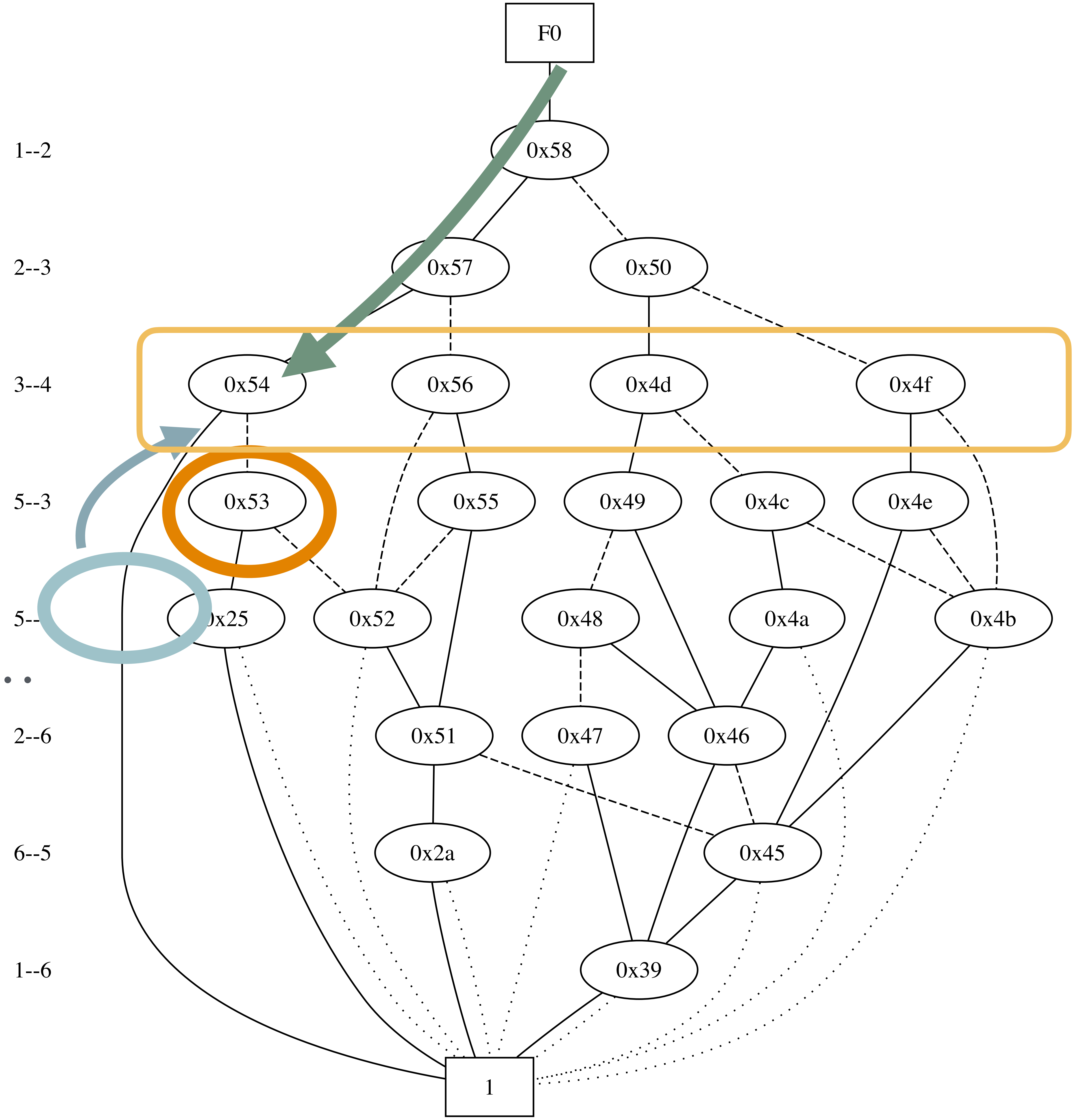


The Gradient

Let us obtain $\delta F / \delta\theta_{34}$

$$F = \theta_{12}\theta_{23}F(0x54) + \ldots$$

$$F = \theta_{12}\theta_{23}(\theta_{\mathbf{34}} \times F(1) + (1 - \theta_{\mathbf{34}})F(0x53)) + \ldots$$



BackPropagation

- First compute probabilities from bottom-up: Pr(0x39), Pr(0x2a)...
- Then compute the paths from top down
- Multiply and sum
- This BDD acts as a **Neural Networ**k
- We can see a node as a neuron
- Do we want to?

ProbLog-2

- Designed to learn from interpretations
- First grounds the program
- Uses sentental decision diagrams
- Idea should still work?
- But, can we interpret a BDD?