Online Planning in MDPs

Tom Silver

Robot Planning Meets Machine Learning

Princeton University

Fall 2025

Recap and Preview

- Last lecture, we considered computing *policies* for MDPs.
  - A policy assigns an action to *every* state in the MDP.
- Complexity of computing/storing a policy is at least linear in the number of states. For large MDPs, this is a dealbreaker.
- This time, we will suppose that *an initial state is known*.
  - How can we use knowledge of an initial state to reduce complexity?
  - *Partial policies*: partial assignment of states to actions

The Factory and the Wild

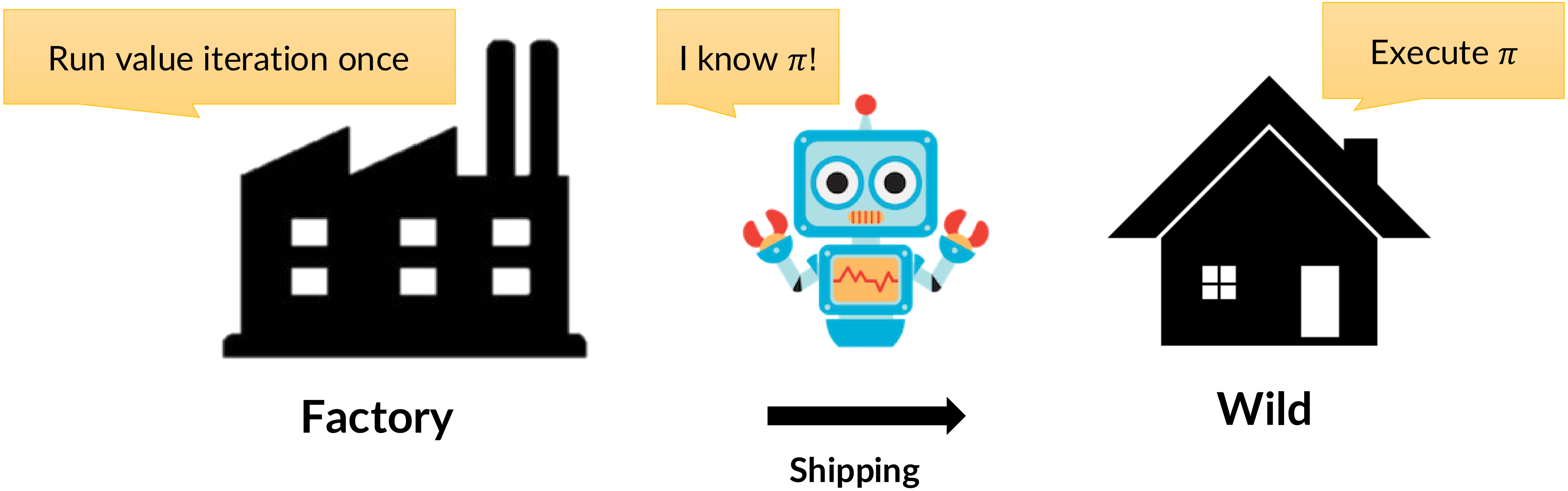
Let's make a robot startup!

Planning Offline (In the Factory)

If known wild MDP, then we can run value iteration **offline** (in the factory) and compute π.

Planning Offline (In the Factory)

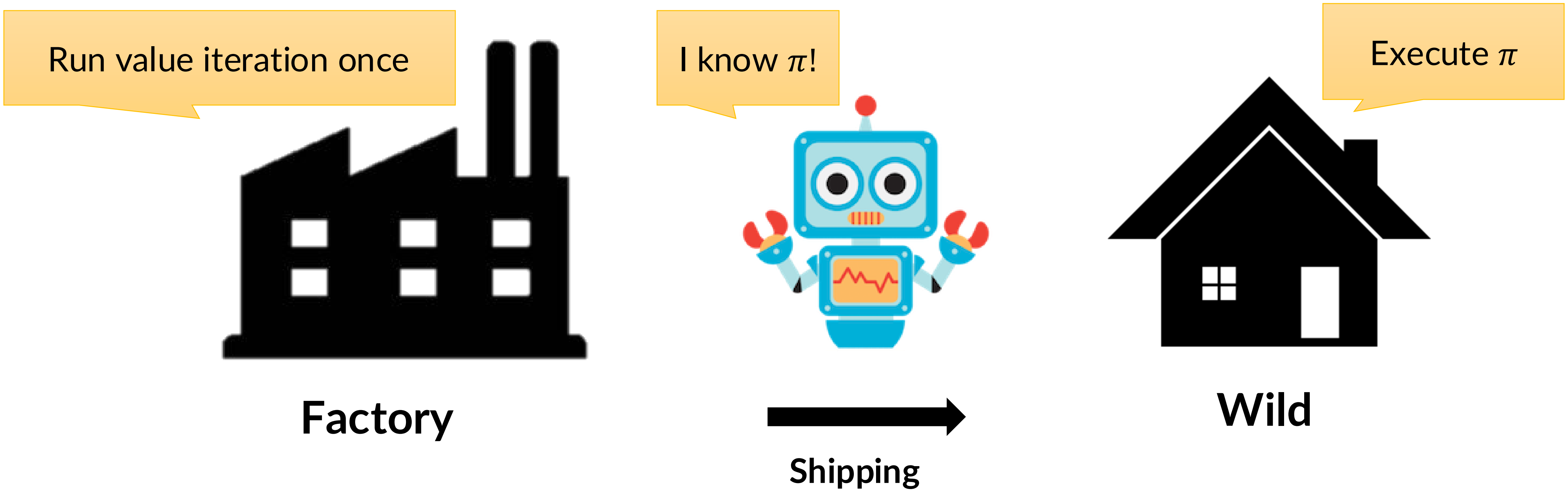
If known wild MDP, then we can run value iteration **offline** (in the factory) and compute π.



Planning Offline (In the Factory)

If known wild MDP, then we can run value iteration **offline** (in the factory) and compute π.

When might this be a good or bad idea?

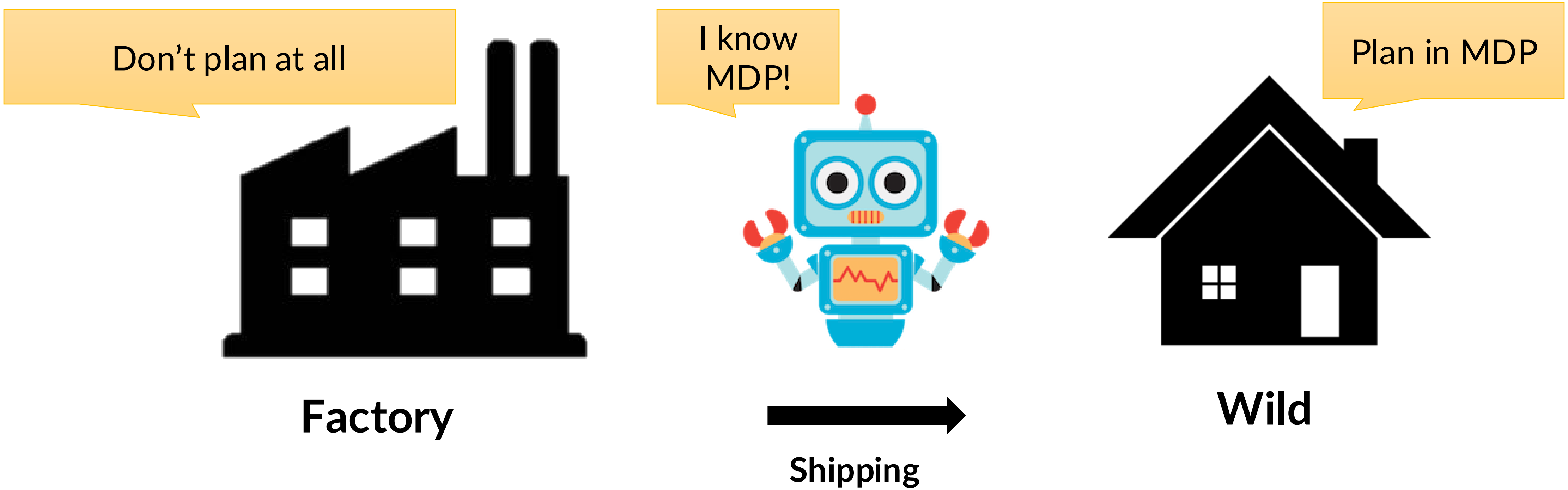


Planning Online (In the Wild)

Alternatively, we could ship the robot with the MDP, and have it plan **online** (in the house).

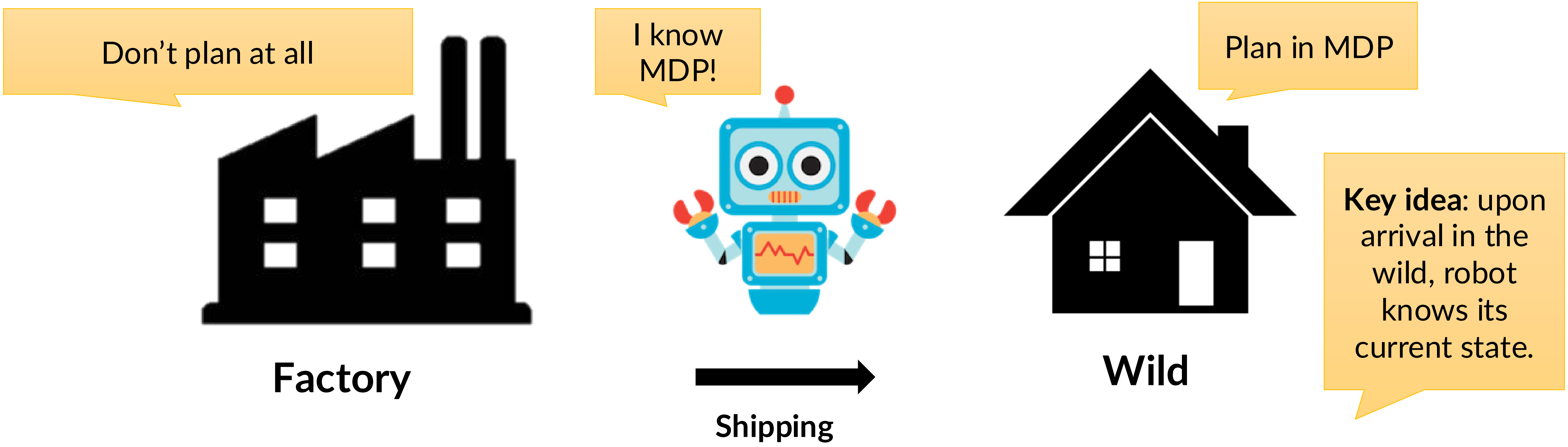
Planning Online (In the Wild)

Alternatively, we could ship the robot with the MDP, and have it plan **online** (in the house).



Planning Online (In the Wild)

Alternatively, we could ship the robot with the MDP, and have it plan **online** (in the house).



Leveraging Known Current State

Assume that we know current state $s_0 \in \mathcal{S}$

Key idea: only some states are *reachable* from s_0.

Leveraging Known Current State

Assume that we know current state $s_0 \in \mathcal{S}$

Key idea: only some states are *reachable* from s_0.

A state s is **reachable** at depth T from state s_0 if there exists actions $(a_0, a_1, \dots, a_{T-1})$ and states $(s_1, s_2, \dots, s_{T-1}, s)$ where

$$\prod P(s_{t+1} \mid s_t, a_t) > 0.$$

Leveraging Known Current State

Assume that we know current state $s_0 \in \mathcal{S}$

Key idea: only some states are *reachable* from s_0.

A state s is **reachable** at depth T from state s_0 if there exists actions $(a_0, a_1, \ldots, a_{T-1})$ and states $(s_1, s_2, \ldots, s_{T-1}, s)$ where

$$\prod P(s_{t+1} \mid s_t, a_t) > 0.$$

Alternatively: a state is reachable if there is some chance that taking T random actions will land us in that state.

Leveraging Known Current State

Assume that we know current state $s_0 \in \mathcal{S}$

Key idea: only some states are *reachable* from s_0.

A state s is **reachable** at depth T from state s_0 if there exists actions $(a_0, a_1, \ldots, a_{T-1})$ and states $(s_1, s_2, \ldots, s_{T-1}, s)$ where

$$\prod P(s_{t+1} \mid s_t, a_t) > 0.$$

Alternatively: a state is reachable if there is some chance that taking T random actions will land us in that state.

How can we efficiently compute reachable states?

How can we leverage reachable states for planning?

And-Or Directed Acyclic Graphs

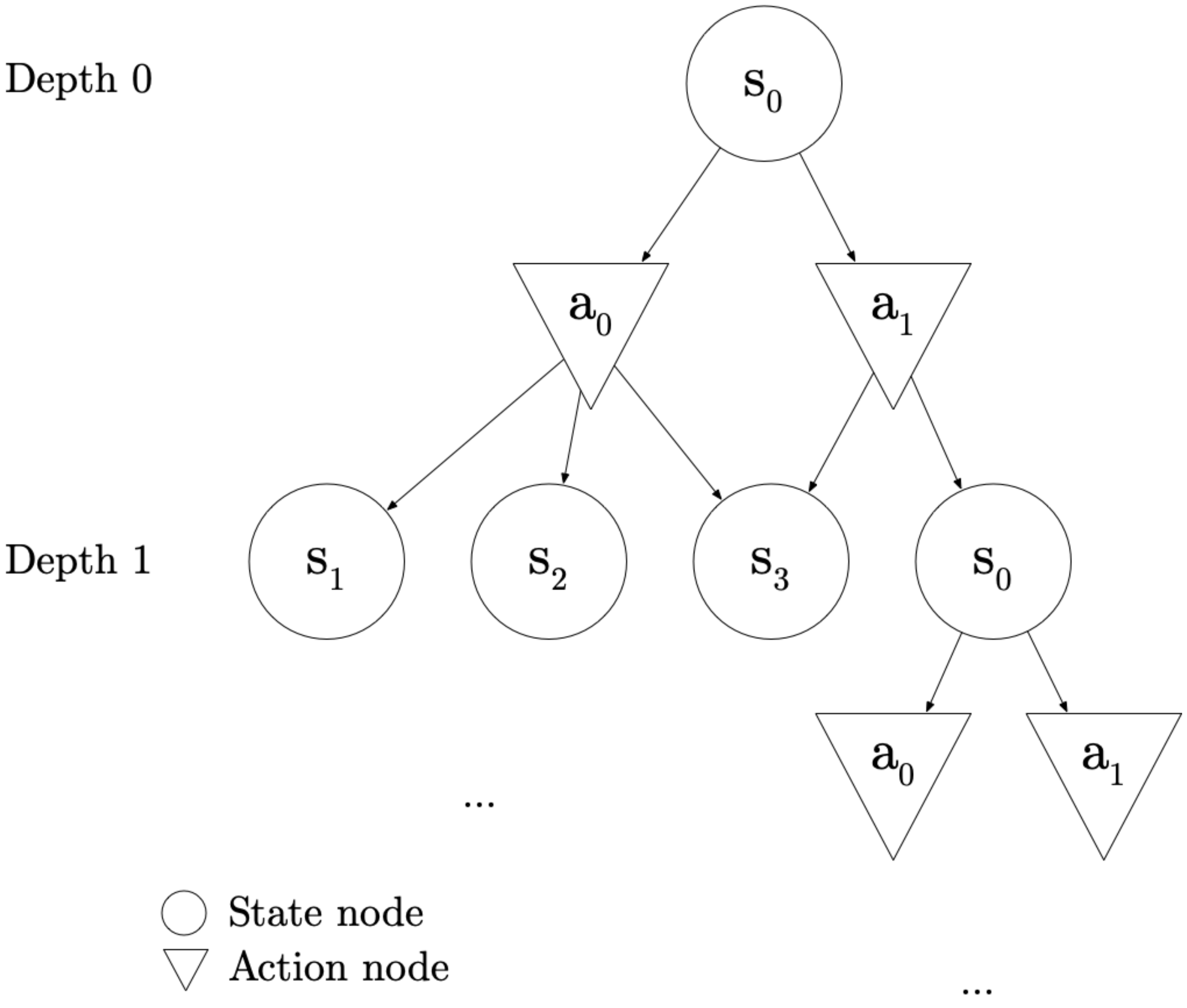
And-Or DAGs (AODAGs)

- State nodes
- Action nodes

Children of state nodes: one per action

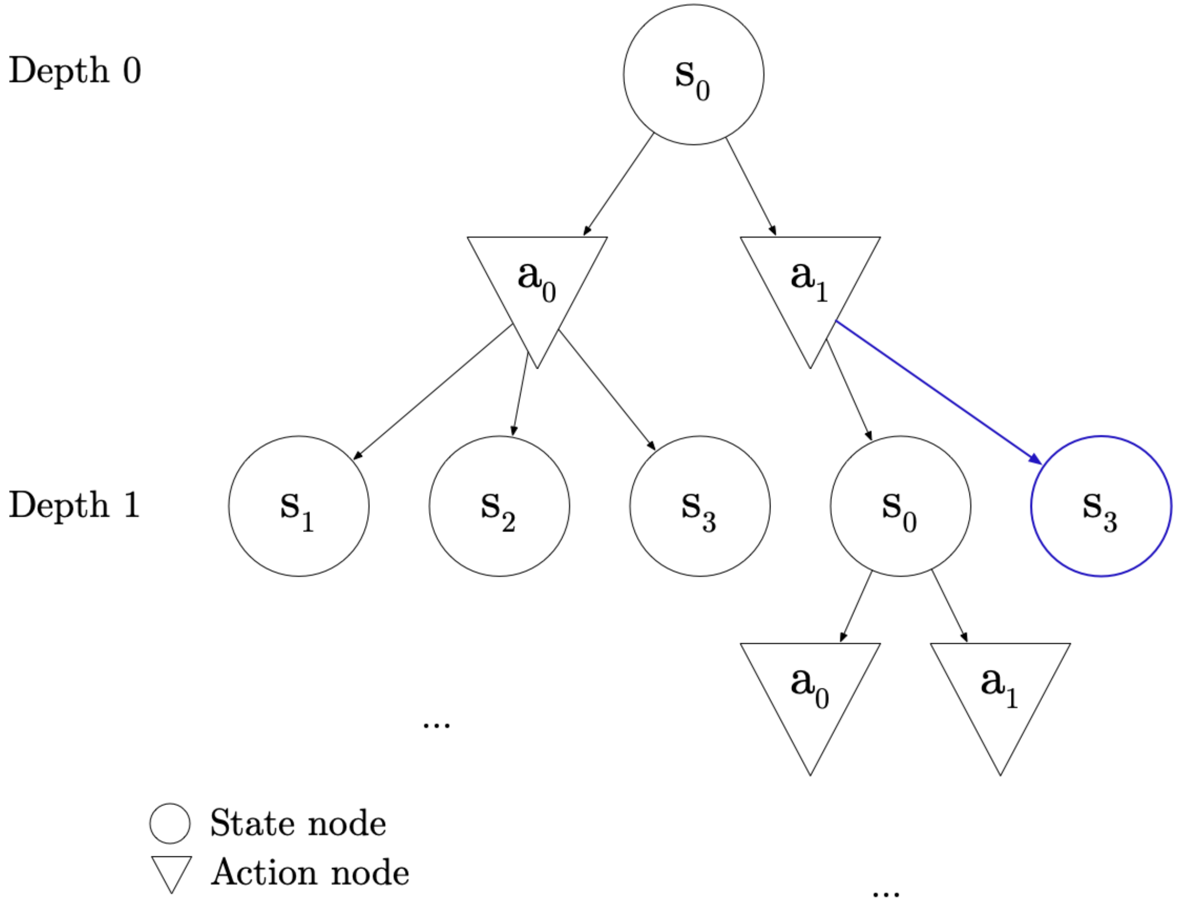
Children of action nodes: one per *possible* next state

Nonzero probability



Warning: I Just Made Up AODAGs

- Usually, just a tree
- But then we could have duplicate (state, depth)
- Their subtrees would be duplicated too
- Everything will be fine – we may just do redundant computation



Finding Reachable States: Finite Depth

To compute states reachable at depth T from state s_0 :

1. Construct AODAG to depth T
2. Read off states from depth T

Finding Reachable States: Finite Depth

To compute states reachable at depth T from state s_0 :

1. Construct AODAG to depth T
2. Read off states from depth T

True or False: if a state is reachable at depth 1, it must also be reachable at depths > 1.

Using Reachable States: Finite Depth

Using Reachable States: Finite Depth

Key observation: to choose an optimal action for s_0, we only need to know $Q_0^*(s_0, a)$ for all $a \in \mathcal{A}$.

We *don't* need to compute all optimal values.

Using Reachable States: Finite Depth

Key observation: to choose an optimal action for s_0, we only need to know $Q_0^*(s_0, a)$ for all $a \in \mathcal{A}$.

We *don't* need to compute all optimal values.

Idea: modify our DP algorithm for computing value function so that it *only considers the reachable states at each depth*.

Using Reachable States: Finite Depth

```
ExpectimaxSearch(s_0, 𝒮, 𝒜, P, R, H)
// a.k.a. ComputeFiniteHorizonValueFunctionOnline
// Represent values as dictionary V[t][s] = V_t^*(s).
V = dict()
// Base case: final values are 0
for each s ∈ reachable states at depth H
    V[H][s] = 0
// Recursive step: compute backwards in time
for t = H − 1, H − 2, . . . , 0
    for each s ∈ reachable states at depth t
        V[t][s] = BellmanBackup(s, V, 𝒮, 𝒜, P, R, t)
        for each a ∈ 𝒜
return V
```

a.k.a. expectimax search, forward search

Warning: don’t use this implementation. (See later slides)

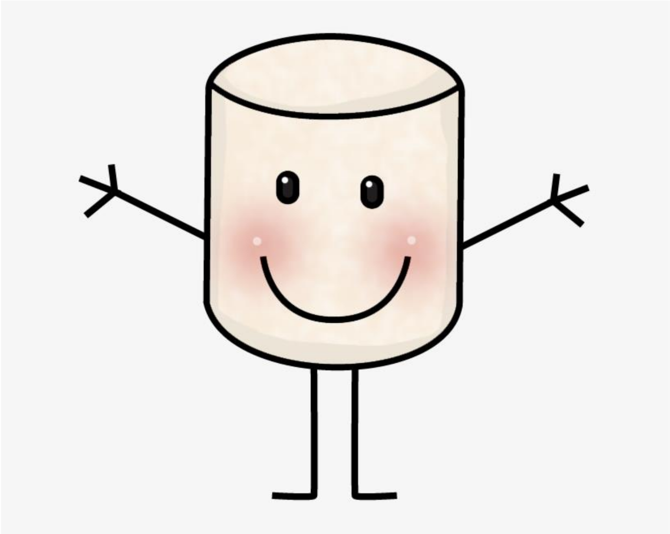
Expectimax Search on AODAGs

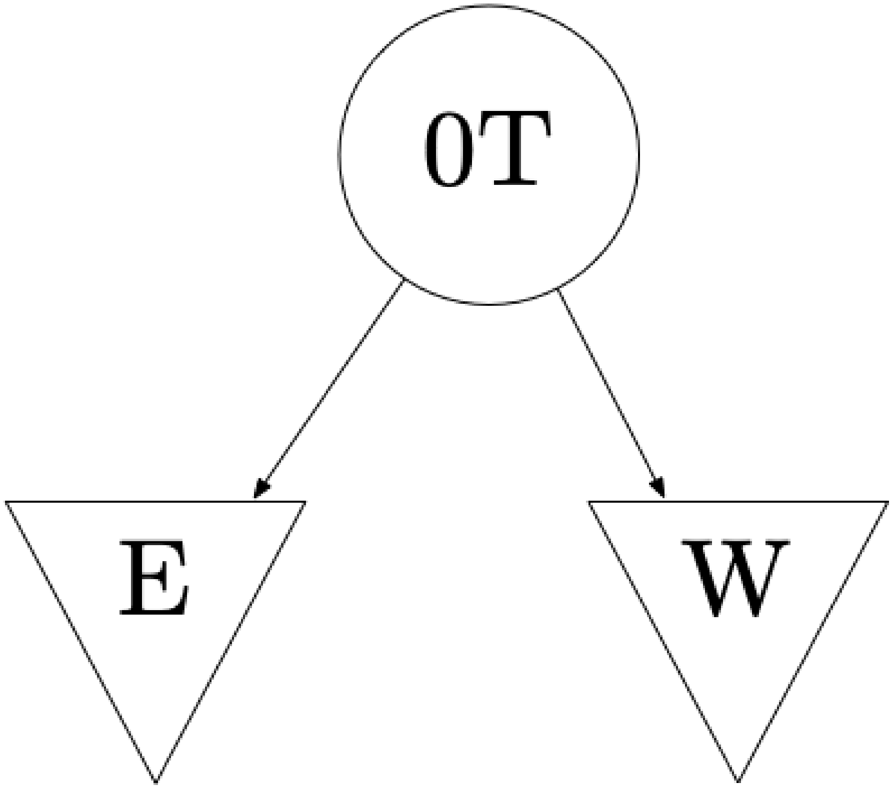
Key ideas:

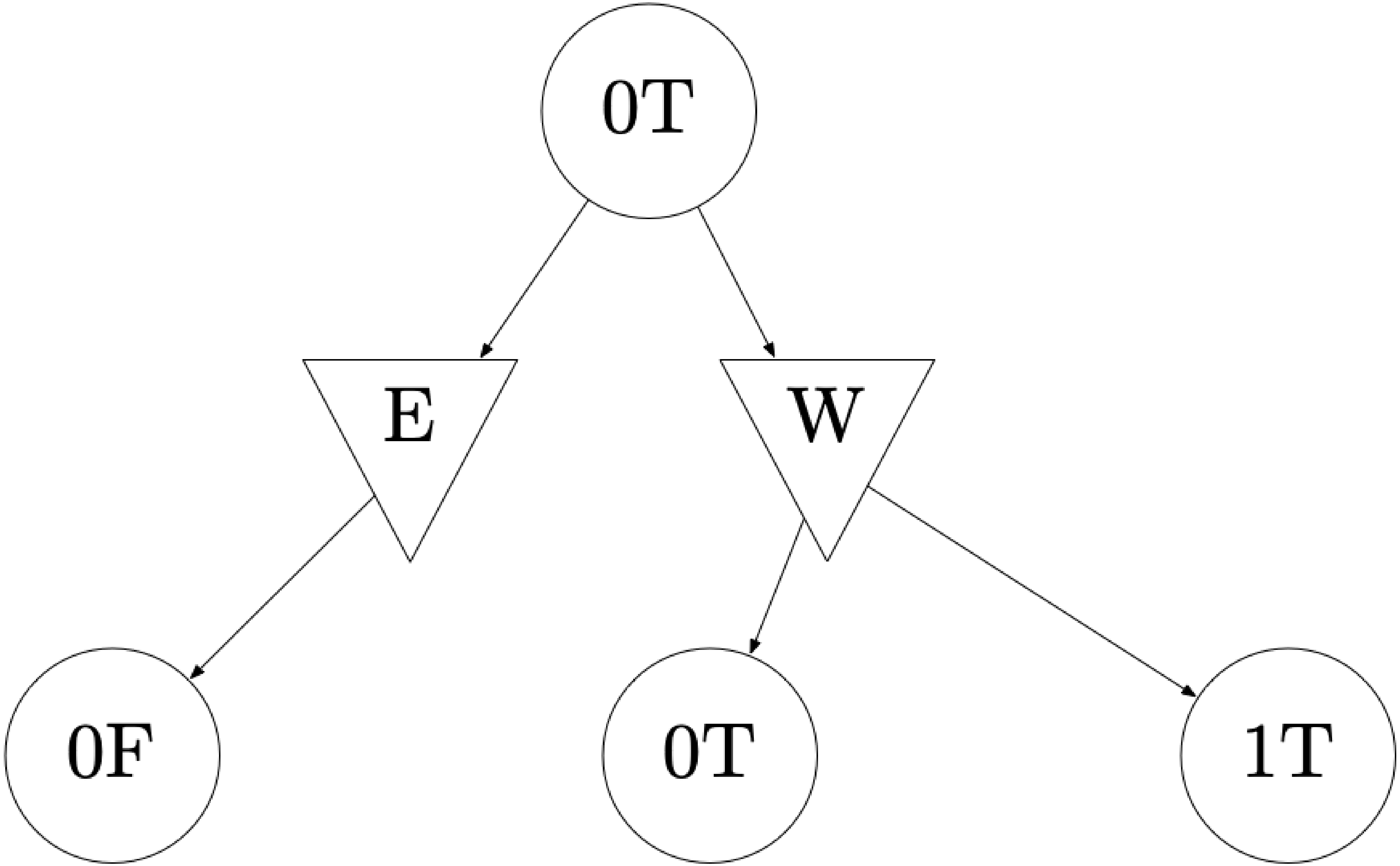
1. Start at the leaves, work up to the root
2. Annotate states with value function
3. Annotate actions with Q function

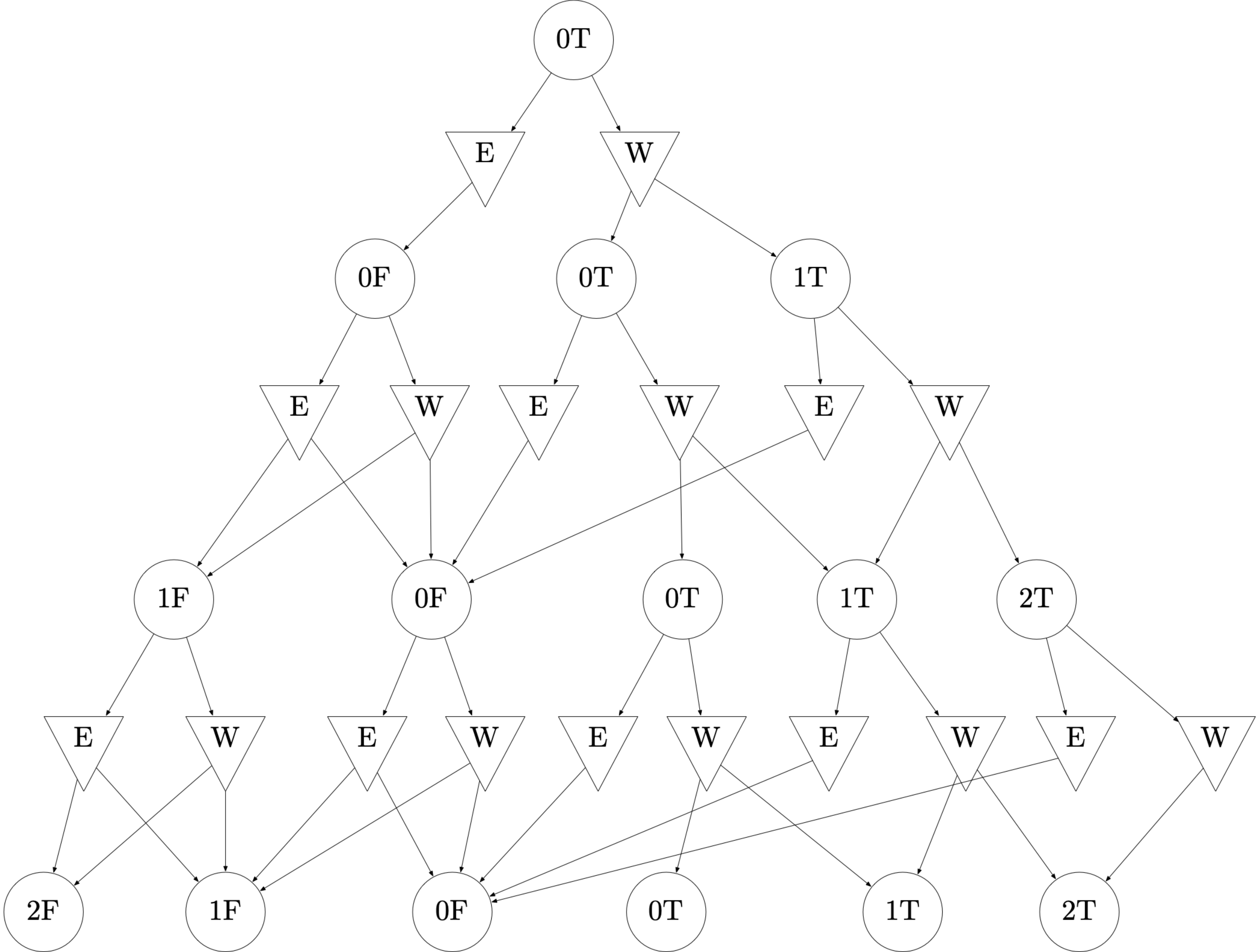
Example: Marshmallows

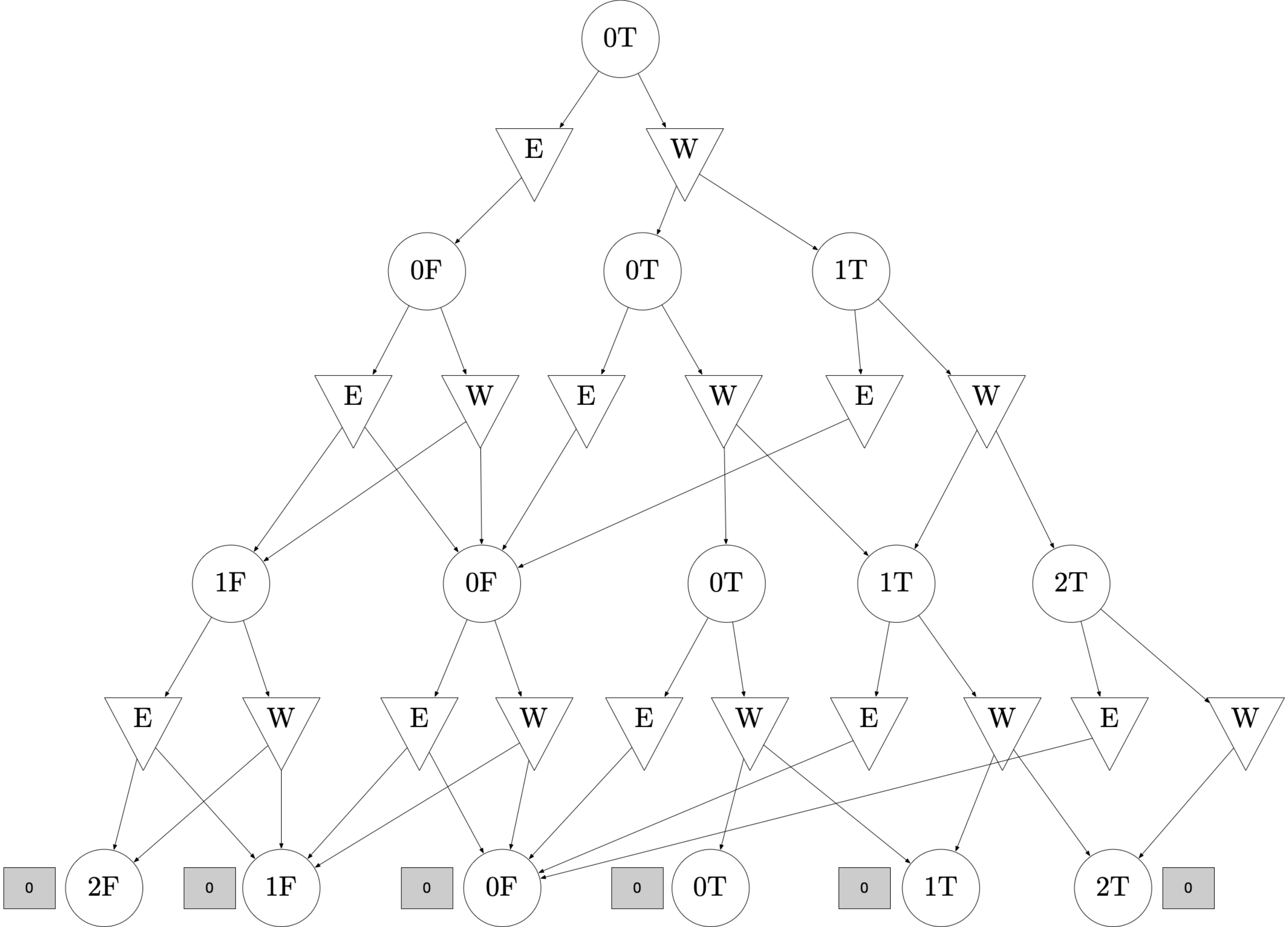
- **States**: (hunger level, marshmallow remains)
  - Hunger level: 0, 1, 2 (higher is hungrier)
  - Marshmallow remains: True or False
- **Actions**: *eat* marshmallow, or *wait*
- **Horizon:** finite (horizon $H = 4$)
- **Rewards:** Negative hunger level squared (on next state)
- **Transition distribution**:
  - Marshmallow remains updated in obvious way
  - If *wait:*
    - With probability 0.25, hunger level increases by 1
    - Otherwise, hunger level stays the same
  - If *eat* (and marshmallow remains):
    - With probability 1, hunger level set to 0
  - If *eat* (and marshmallow gone):
    - Same as waiting




0T
E
W


0T
E
W
0F
0T
1T


0T
E
W
0F
0T
1T
E
W
E
W
E
W
1F
0F
0T
1T
2T
E
W
E
W
E
W
E
W
E
W
2F
1F
0F
0T
1T
2T



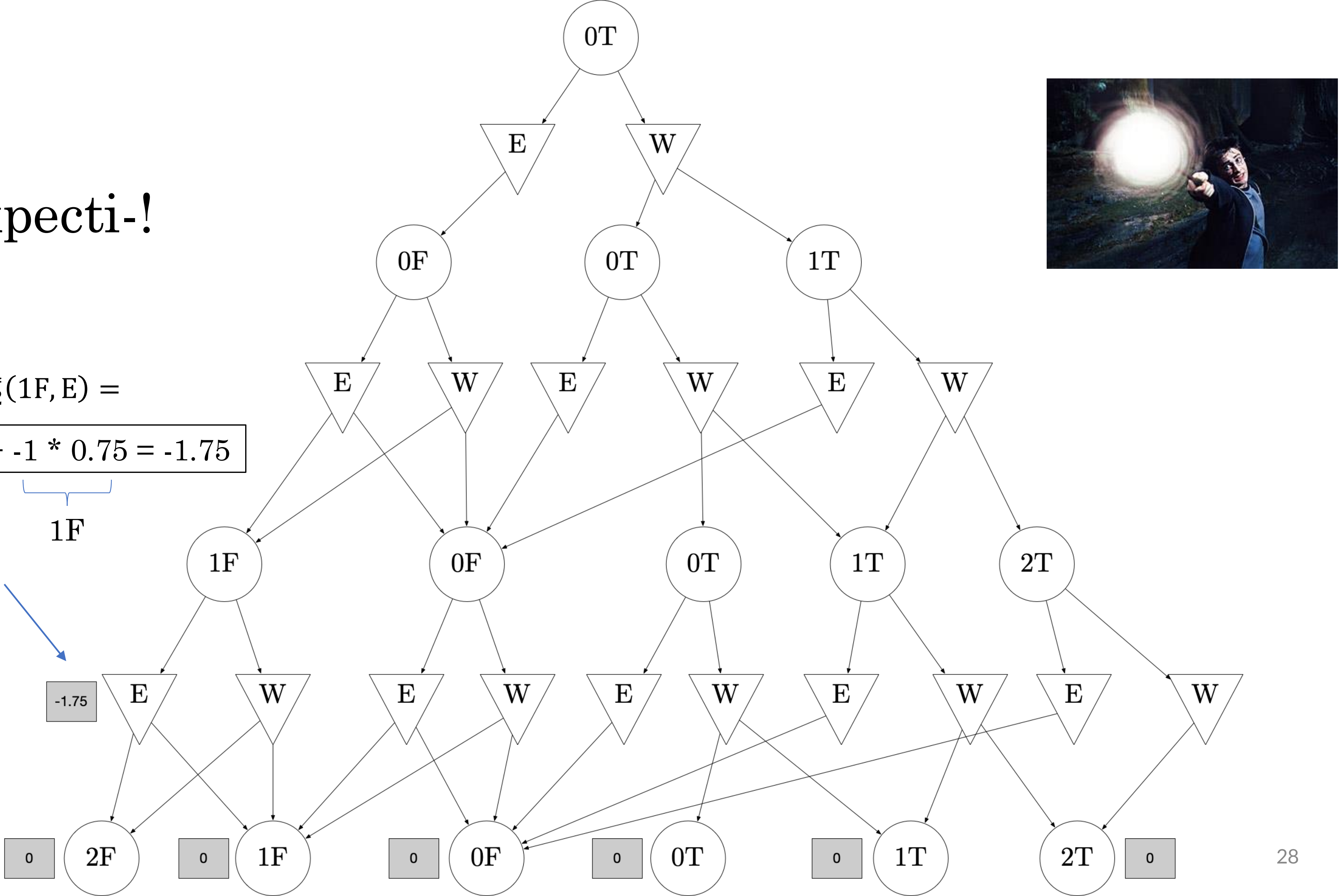
0T
E
W
0F
0T
1T
E
W
E
W
E
W
1F
0F
0T
1T
2T
E
W
E
W
E
W
E
W
E
W
0
2F
0
1F
0
0F
0
0T
0
1T
2T
0

Expecti-!

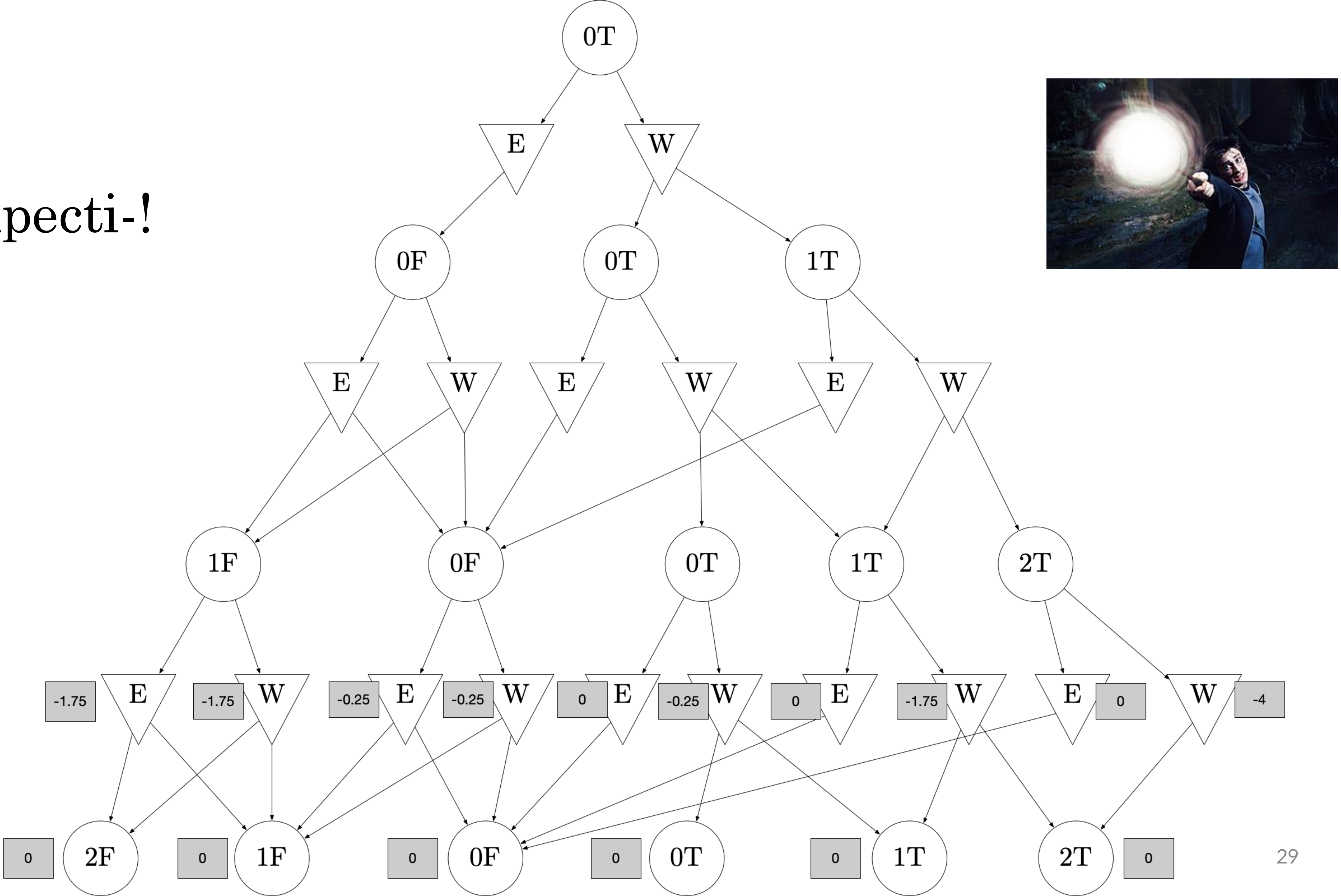
$Q_2^*(1\text{F}, \text{E}) =$

-4 * 0.25 + -1 * 0.75 = -1.75

2F 1F



Expecti-!

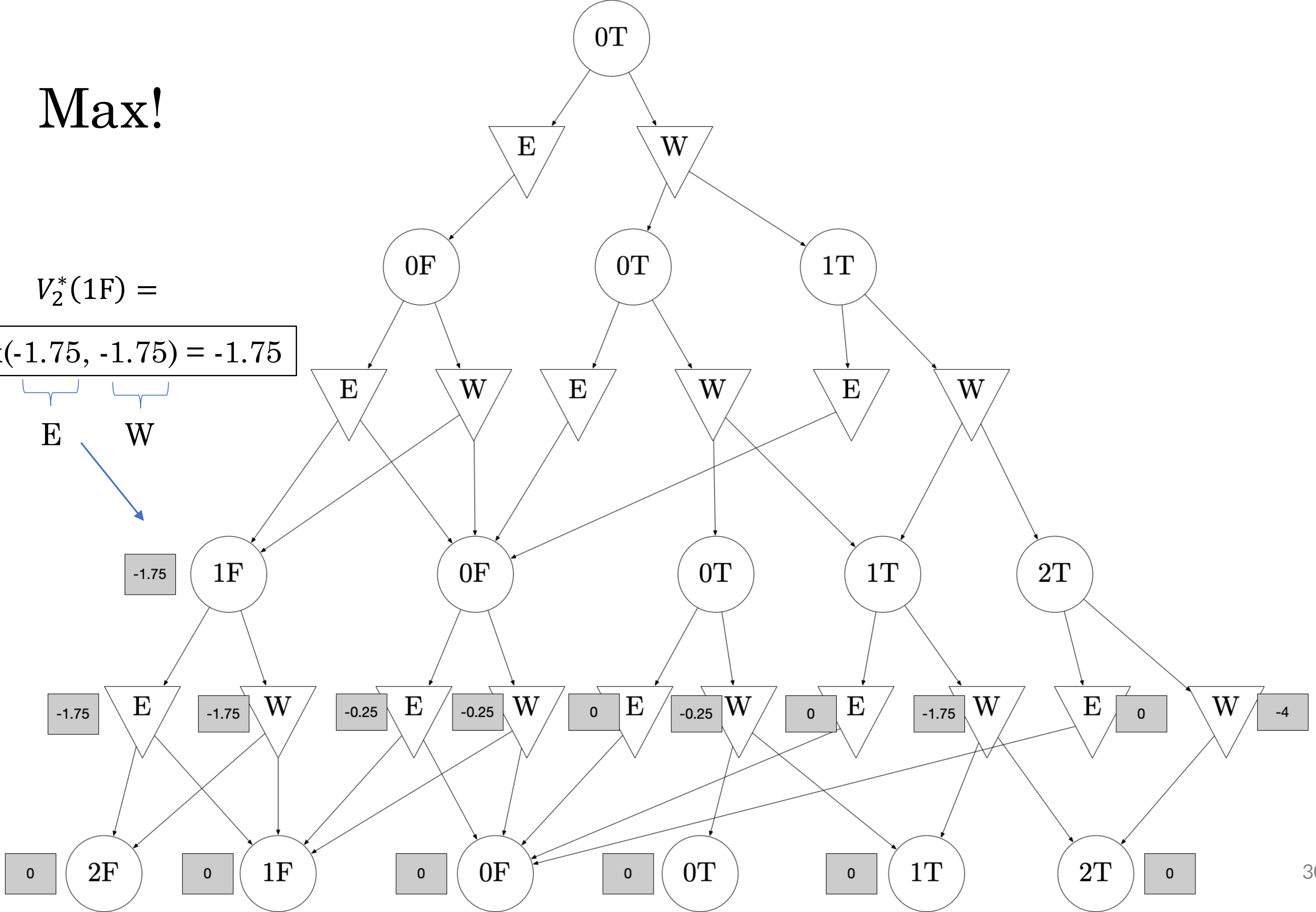

0T
E
W
0F
0T
1T
E
W
E
W
E
W
1F
0F
0T
1T
2T
-1.75
E
-1.75
W
-0.25
E
-0.25
W
0
E
-0.25
W
0
E
-1.75
W
E
0
W
-4
0
2F
0
1F
0
0F
0
0T
0
1T
2T
0

Max!

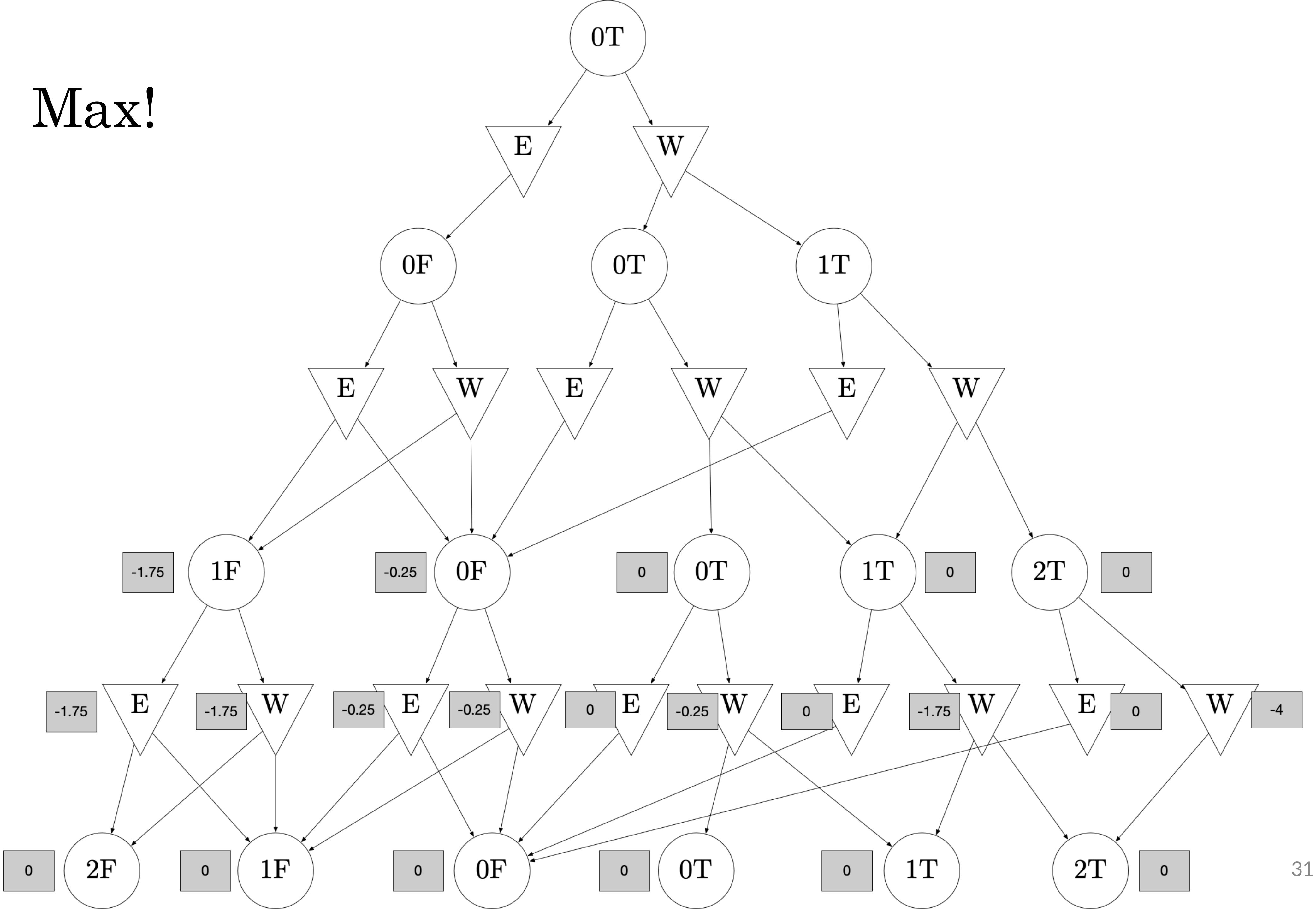
$V_2^*(1F) =$

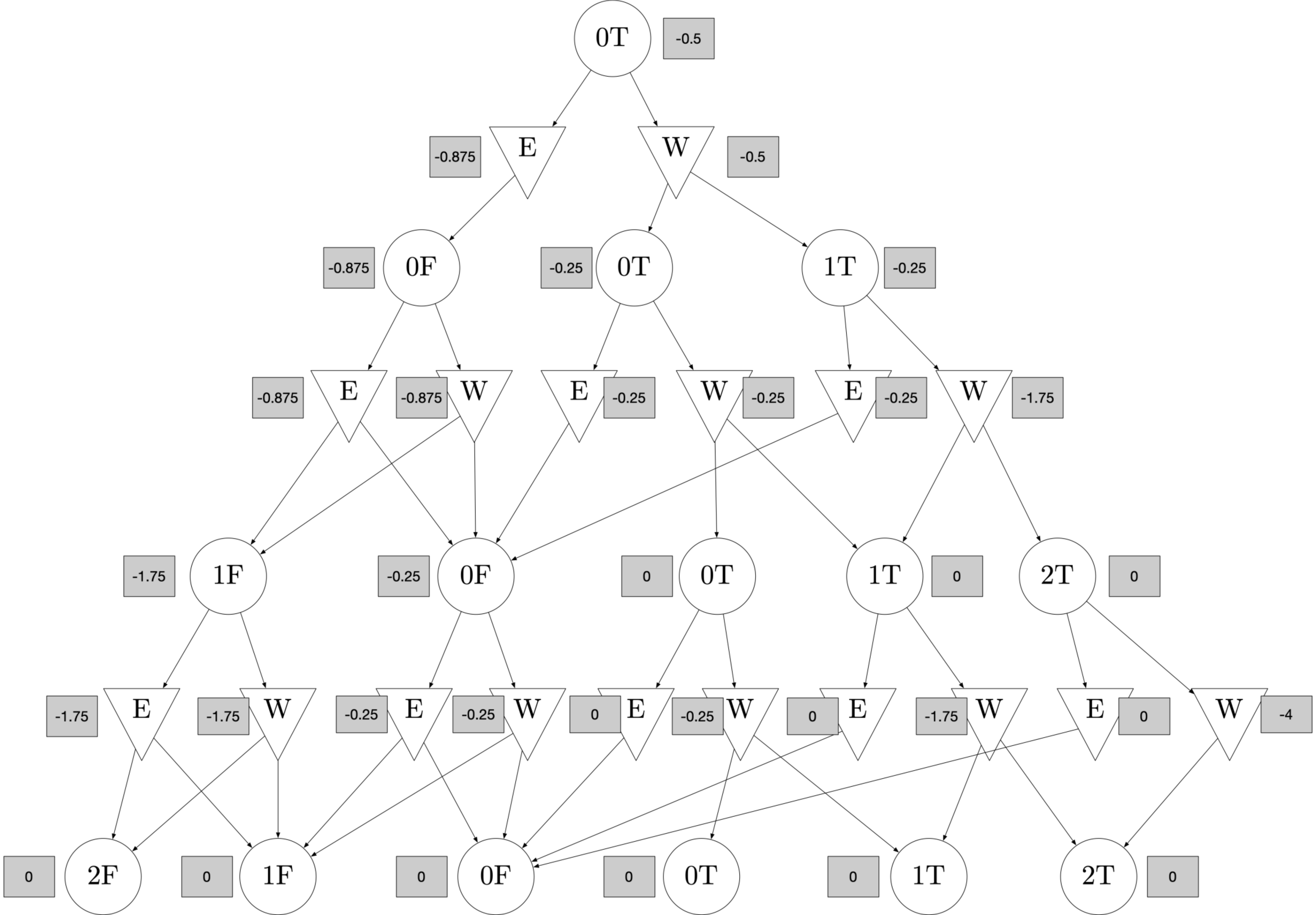
max(-1.75, -1.75) = -1.75

E W



Max!


0T
E
W
0F
0T
1T
E
W
E
W
E
W
-1.75
1F
-0.25
0F
0
0T
1T
0
2T
0
-1.75
E
-1.75
W
-0.25
E
-0.25
W
0
E
-0.25
W
0
E
-1.75
W
E
0
W
-4
0
2F
0
1F
0
0F
0
0T
0
1T
2T
0


0T
-0.5
-0.875
E
W
-0.5
-0.875
0F
-0.25
0T
1T
-0.25
-0.875
E
-0.875
W
E
-0.25
W
-0.25
E
-0.25
W
-1.75
-1.75
1F
-0.25
0F
0
0T
1T
0
2T
0
-1.75
E
-1.75
W
-0.25
E
-0.25
W
0
E
-0.25
W
0
E
-1.75
W
E
0
W
-4
0
2F
0
1F
0
0F
0
0T
0
1T
2T
0

Interleaving Planning and Execution

In the wild, we can "plan a little", "execute a little", repeat.

Interleaving Planning and Execution

In the wild, we can “plan a little”, “execute a little”, repeat.

Receding horizon control (RHC):

1. Plan to H time steps in the future (even if infinite horizon)
2. Execute for $T_{replan} \leq H$ time steps (often $T_{replan} = 1$)
3. Repeat

Interleaving Planning and Execution

In the wild, we can "plan a little", "execute a little", repeat.

Receding horizon control (RHC):

1. Plan to H time steps in the future (even if infinite horizon)
2. Execute for $T_{replan} \leq H$ time steps (often $T_{replan} = 1$)
3. Repeat

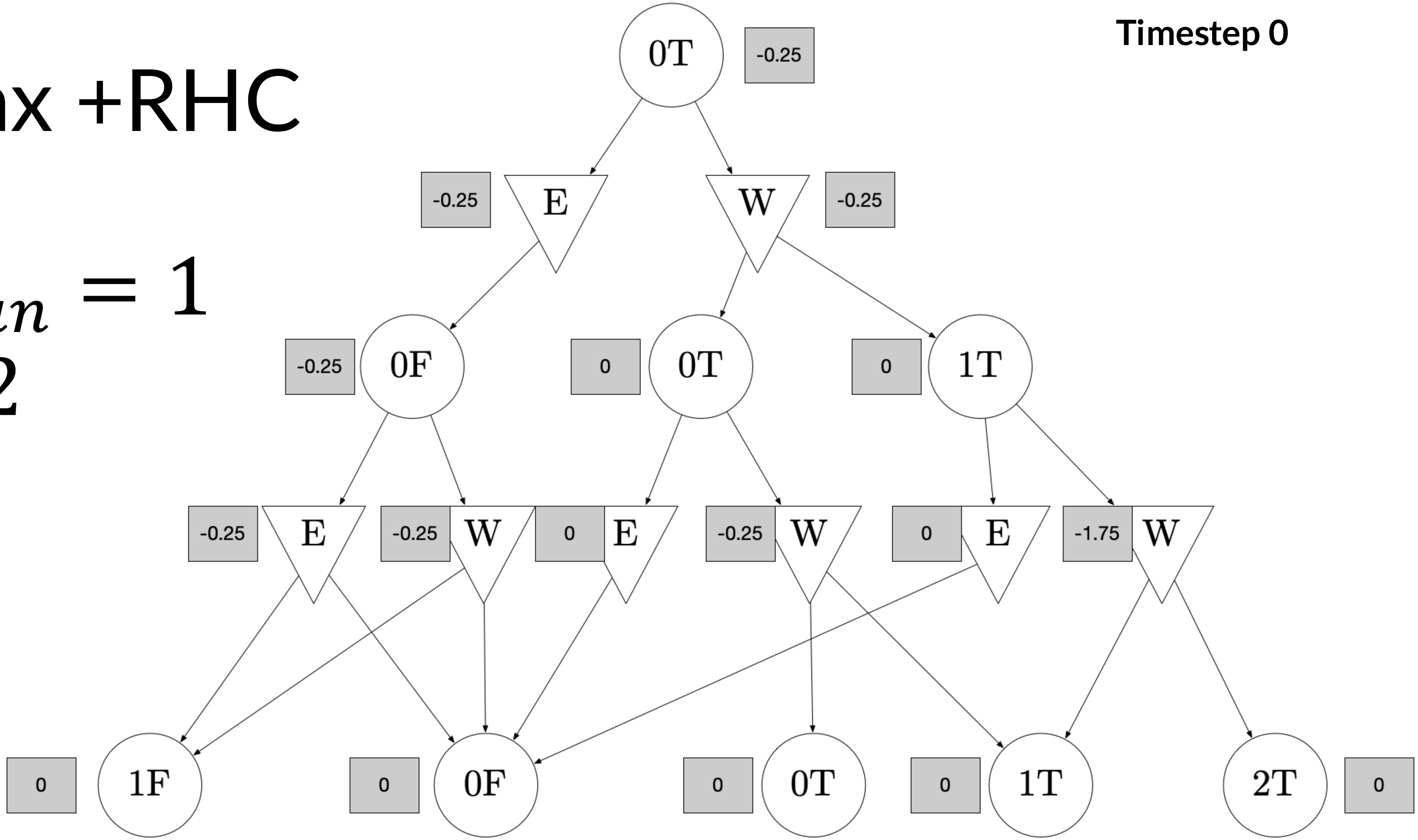
Execution monitoring:

- Replan only when some criteria are met
- Example: replan if you encounter an "unexpected" state (for different possible definitions of unexpected).

Example: Expectimax +RHC

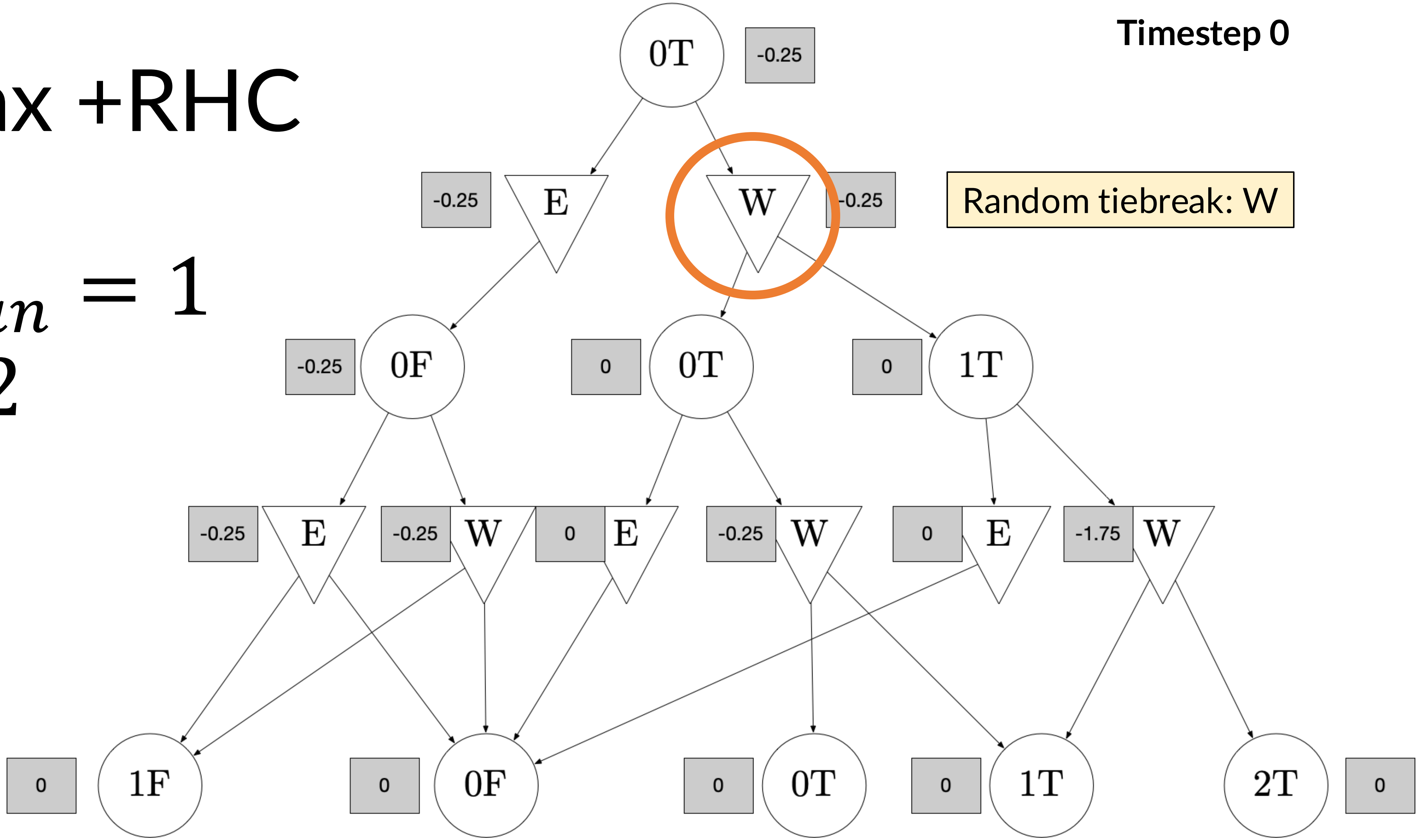
$$T_{replan} = 1$$
$$H = 2$$

Timestep 0



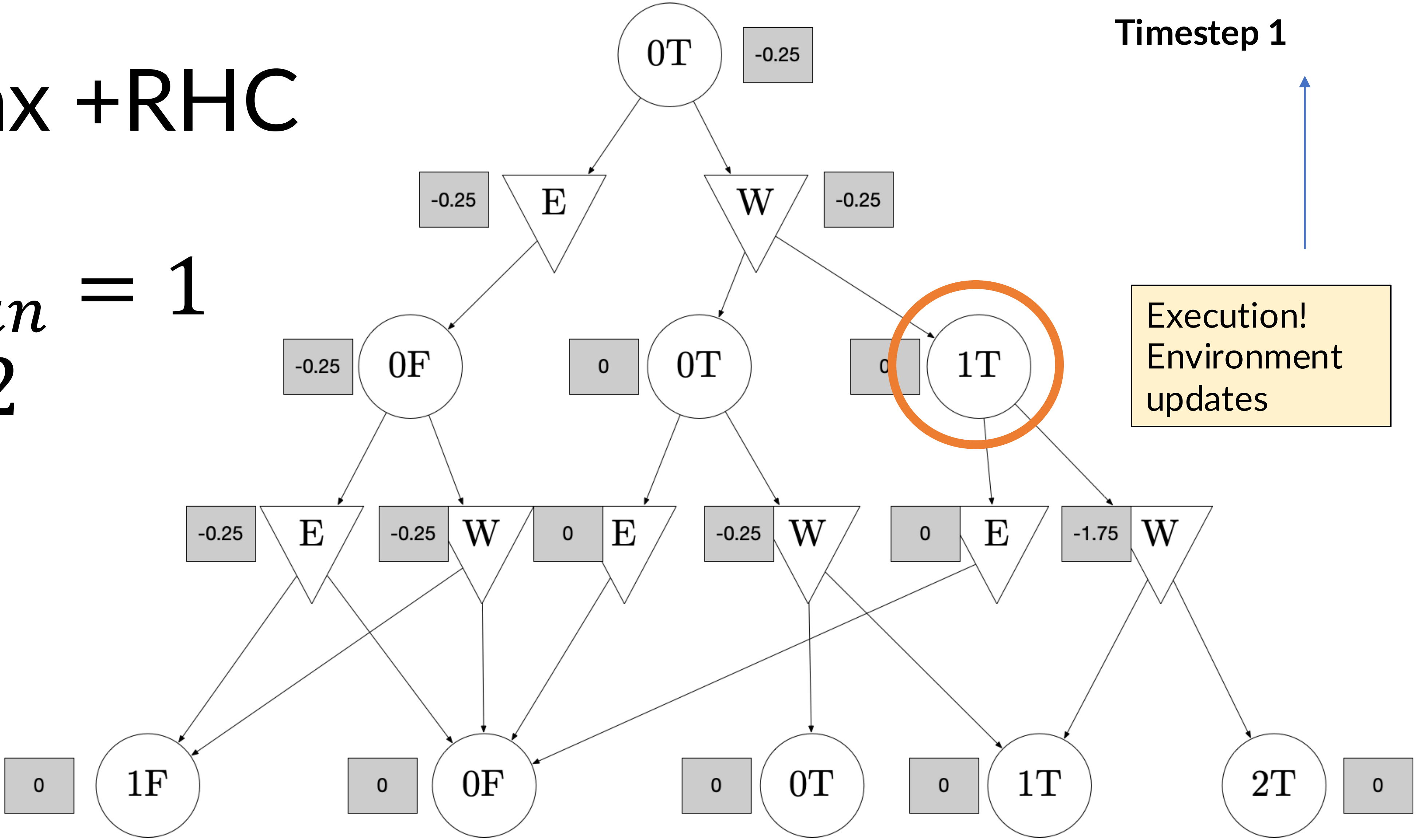
Example: Expectimax +RHC

$$T_{replan} = 1$$
$$H = 2$$



Example: Expectimax +RHC

$$T_{replan} = 1$$
$$H = 2$$



Timestep 1

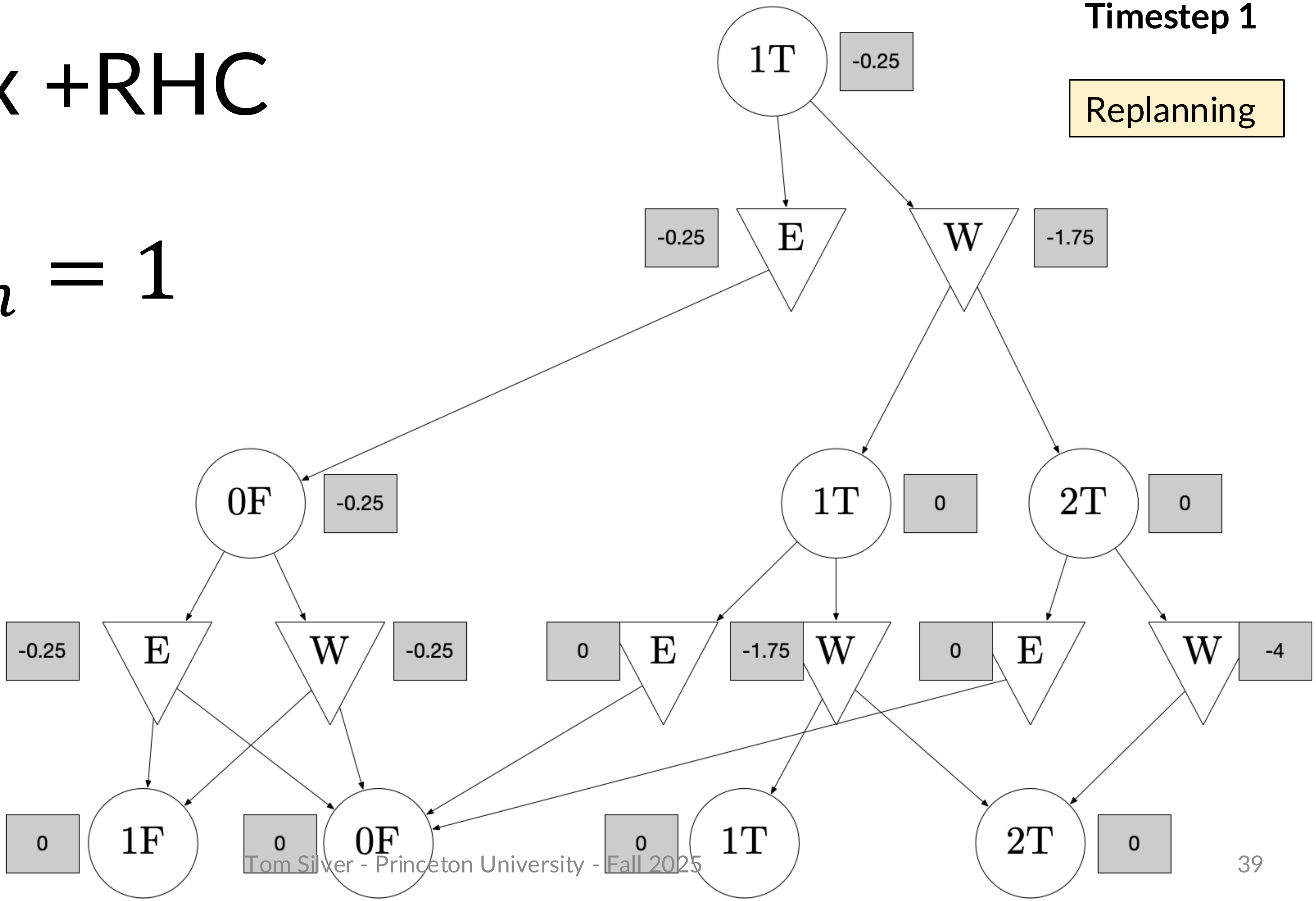
Execution! Environment updates

Example: Expectimax +RHC

$$T_{replan} = 1$$
$$H = 2$$

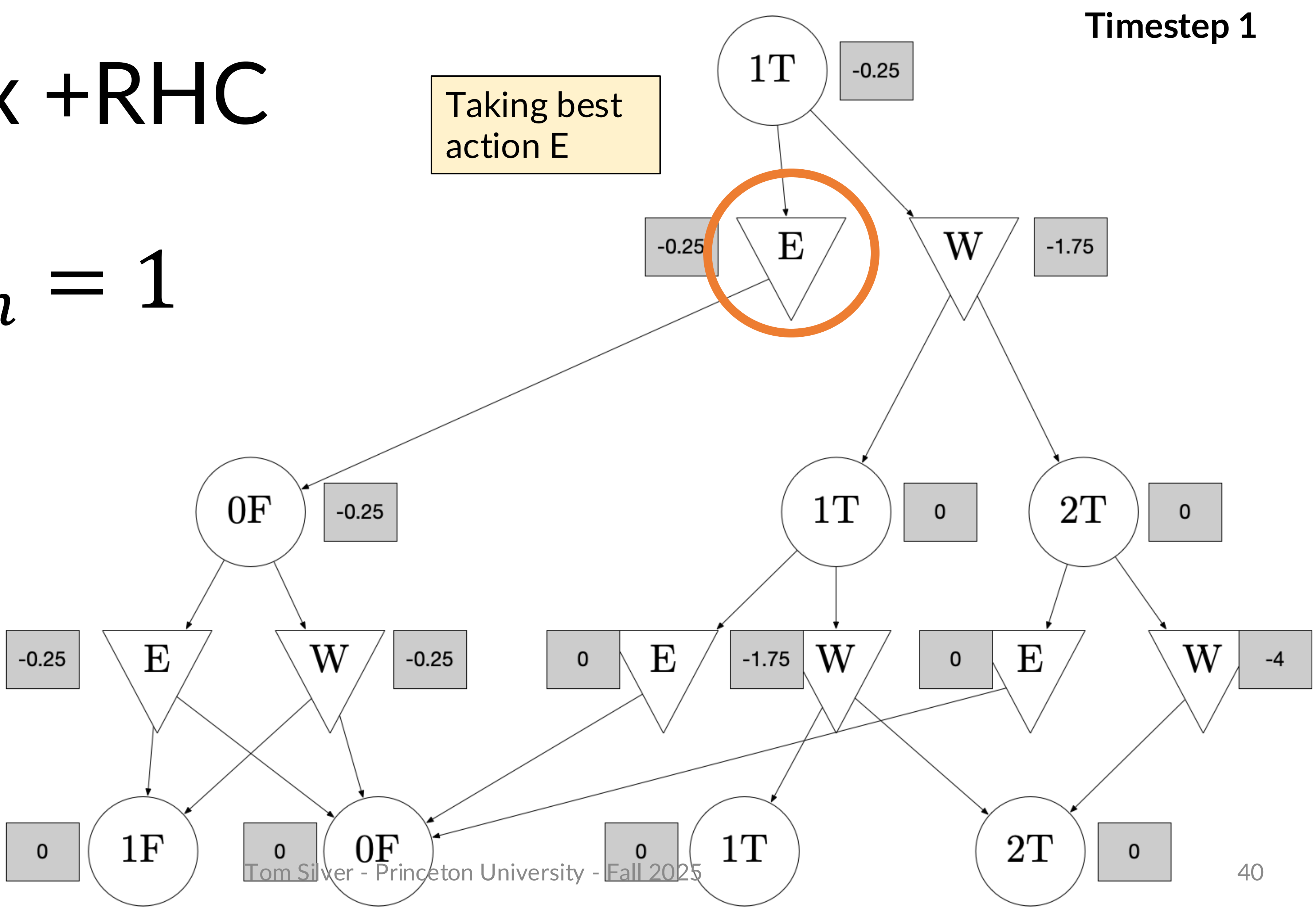
Timestep 1

Replanning



Example: Expectimax +RHC

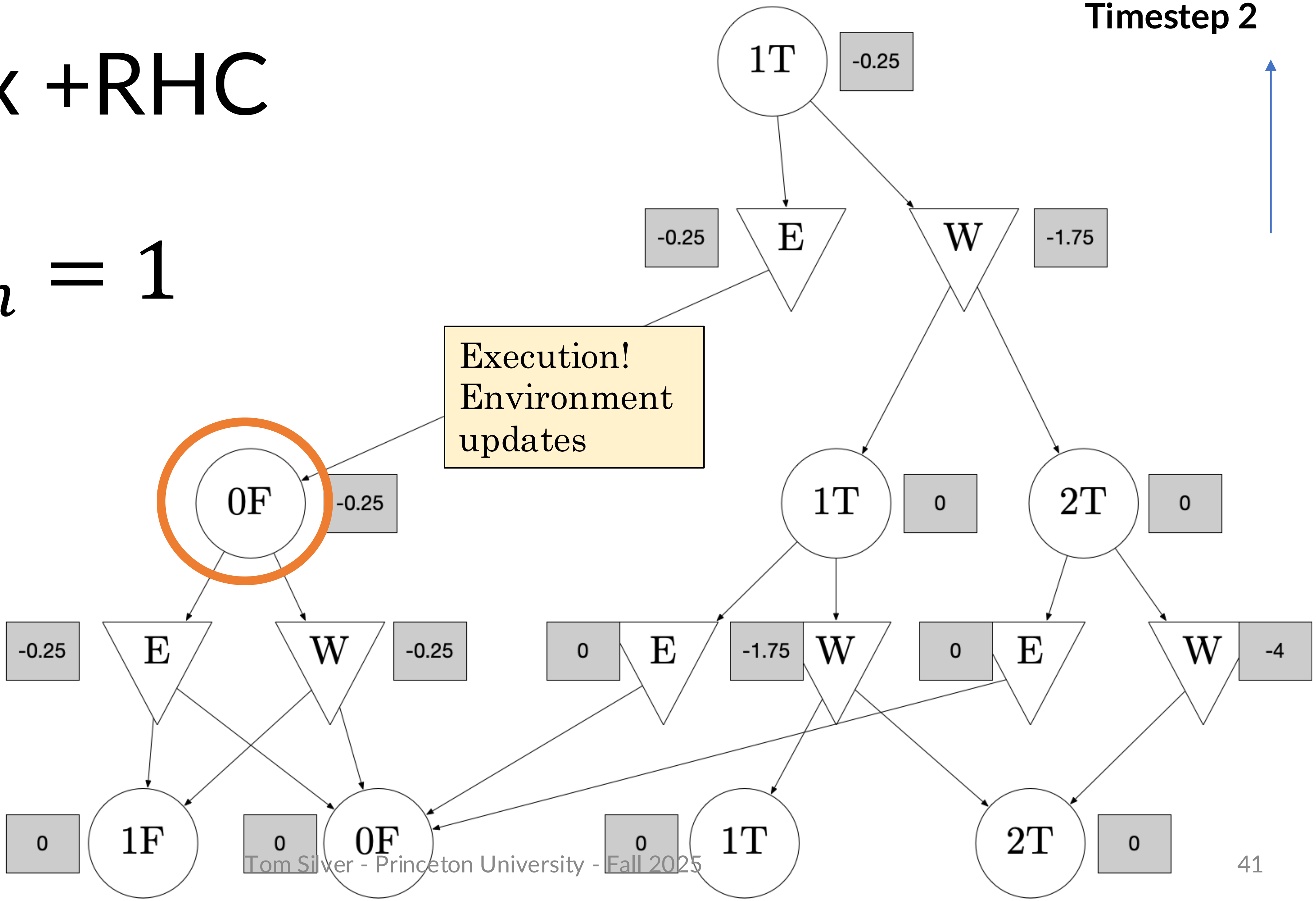
$$T_{replan} = 1$$
$$H = 2$$



Example: Expectimax +RHC

$$T_{replan} = 1$$
$$H = 2$$

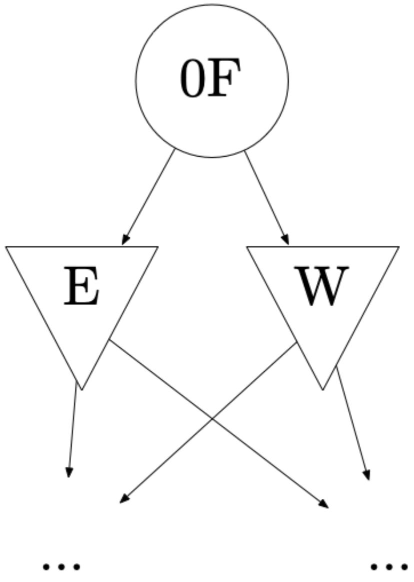
Timestep 2



Example: Expectimax +RHC

Timestep 2

$$T_{replan} = 1$$
$$H = 2$$



Etc.

Expectimax Search: Alternative Implementation

Construct *tree* and compute values simultaneously.

$\textsc{ExpectimaxSearch}(s_0, \mathcal{S}, \mathcal{A}, P, R, H)$

// Alternative implementation; performs DFS over tree
return $\mathrm{argmax}_{\mathtt{a}}\, \mathrm{Q}(s_0, \mathtt{a}, 0, \mathcal{S}, \mathcal{A}, P, R, H)$

$\mathrm{Q}(\mathtt{s}, \mathtt{a}, \mathtt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$

return $\sum_{\mathtt{ns}} P(\mathtt{ns} \mid \mathtt{s}, \mathtt{a})(R(\mathtt{s}, \mathtt{a}, \mathtt{ns}) + \mathrm{V}(\mathtt{ns}, \mathtt{t}+1, \mathcal{S}, \mathcal{A}, P, R, H))$

$\mathrm{V}(\mathtt{s}, \mathtt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$

if $t = H$
    return 0
return $\max_{\mathtt{a}} \mathrm{Q}(\mathtt{s}, \mathtt{a}, \mathtt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$

Expectimax Search: Alternative Implementation

Construct *tree* and compute values simultaneously.

$\textsc{ExpectimaxSearch}(s_0, \mathcal{S}, \mathcal{A}, P, R, H)$

// Alternative implementation; performs DFS over tree
return $\text{argmax}_{\mathsf{a}}\, \text{Q}(s_0, \mathsf{a}, 0, \mathcal{S}, \mathcal{A}, P, R, H)$

$\text{Q}(\mathsf{s}, \mathsf{a}, \mathsf{t}, \mathcal{S}, \mathcal{A}, P, R, H)$

return $\sum_{\mathsf{ns}} P(\mathsf{ns} \mid \mathsf{s}, \mathsf{a})(R(\mathsf{s}, \mathsf{a}, \mathsf{ns}) + \text{V}(\mathsf{ns}, \mathsf{t}+1, \mathcal{S}, \mathcal{A}, P, R, H))$

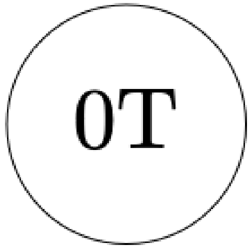
$\text{V}(\mathsf{s}, \mathsf{t}, \mathcal{S}, \mathcal{A}, P, R, H)$

if $t = H$
    return 0
return $\max_{\mathsf{a}} \text{Q}(\mathsf{s}, \mathsf{a}, \mathsf{t}, \mathcal{S}, \mathcal{A}, P, R, H)$

This is the more common implementation of expectimax that you should probably use!

Returns first action only. Assumes replanning

Note that without caching, this computes a *tree*, not AODAG. (Possible duplicates.)



$\textsc{ExpectimaxSearch}(s_0, \mathcal{S}, \mathcal{A}, P, R, H)$

// Alternative implementation; performs DFS over tree

return $\operatorname{argmax}_{\mathtt{a}} \mathrm{Q}(s_0, \mathtt{a}, 0, \mathcal{S}, \mathcal{A}, P, R, H)$

$\mathrm{Q}(\mathtt{s}, \mathtt{a}, \mathtt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$

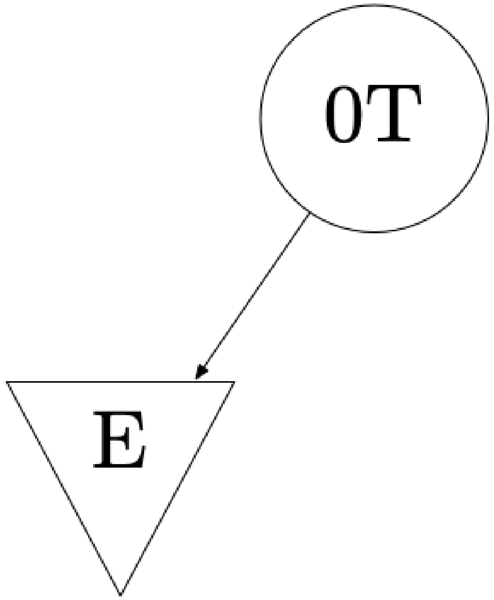
return $\sum_{\mathtt{ns}} P(\mathtt{ns} \mid \mathtt{s}, \mathtt{a})(R(\mathtt{s}, \mathtt{a}, \mathtt{ns}) + \mathrm{V}(\mathtt{ns}, \mathtt{t}+1, \mathcal{S}, \mathcal{A}, P, R, H))$

$\mathrm{V}(\mathtt{s}, \mathtt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$

if $t = H$

  return 0

return $\max_{\mathtt{a}} \mathrm{Q}(\mathtt{s}, \mathtt{a}, \mathtt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$



$\textsc{ExpectimaxSearch}(s_0, \mathcal{S}, \mathcal{A}, P, R, H)$

// Alternative implementation; performs DFS over tree

return $\text{argmax}_{\mathtt{a}}\, \mathrm{Q}(s_0, \mathtt{a}, 0, \mathcal{S}, \mathcal{A}, P, R, H)$

$\mathrm{Q}(\mathtt{s}, \mathtt{a}, \mathtt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$

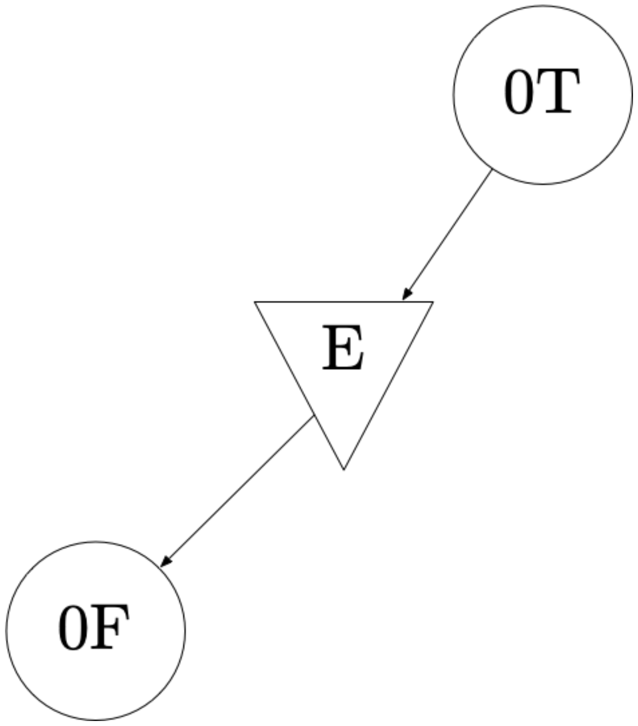
return $\sum_{\mathtt{ns}} P(\mathtt{ns} \mid \mathtt{s}, \mathtt{a})(R(\mathtt{s}, \mathtt{a}, \mathtt{ns}) + \mathrm{V}(\mathtt{ns}, \mathtt{t}+1, \mathcal{S}, \mathcal{A}, P, R, H))$

$\mathrm{V}(\mathtt{s}, \mathtt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$

if $t = H$

    return 0

return $\max_{\mathtt{a}} \mathrm{Q}(\mathtt{s}, \mathtt{a}, \mathtt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$



EXPECTIMAXSEARCH$(s_0, \mathcal{S}, \mathcal{A}, P, R, H)$

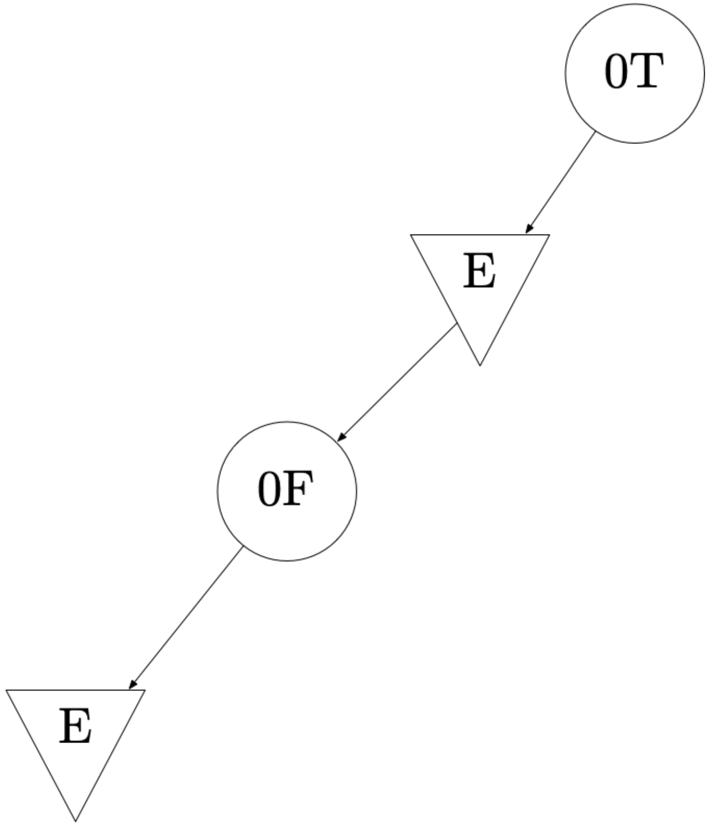
// Alternative implementation; performs DFS over tree
return $\text{argmax}_{\texttt{a}}\ \text{Q}(s_0, \texttt{a}, 0, \mathcal{S}, \mathcal{A}, P, R, H)$

$\text{Q}(\texttt{s}, \texttt{a}, \texttt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$

return $\sum_{\texttt{ns}} P(\texttt{ns} \mid \texttt{s}, \texttt{a})(R(\texttt{s}, \texttt{a}, \texttt{ns}) + \text{V}(\texttt{ns}, \texttt{t} + 1, \mathcal{S}, \mathcal{A}, P, R, H))$

$\text{V}(\texttt{s}, \texttt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$

if $t = H$
    return 0
return $\max_{\texttt{a}} \text{Q}(\texttt{s}, \texttt{a}, \texttt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$



$\textsc{ExpectimaxSearch}(s_0, \mathcal{S}, \mathcal{A}, P, R, H)$

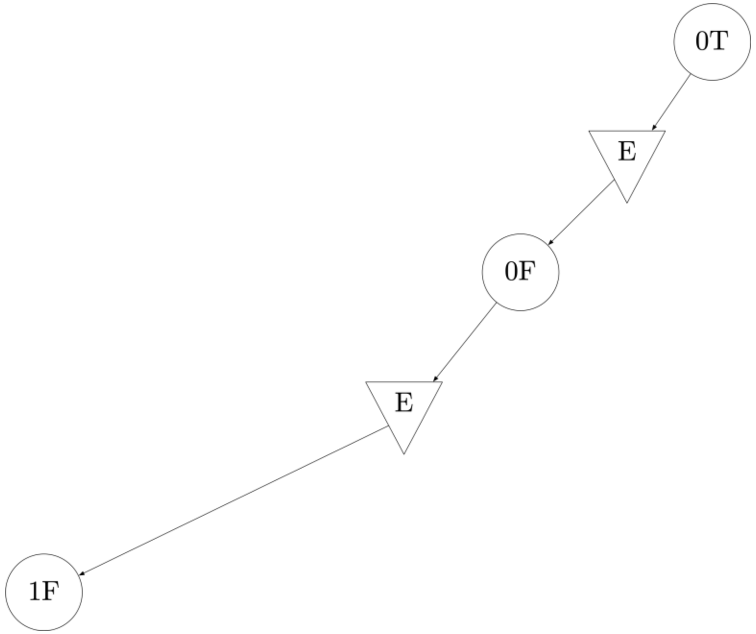
// Alternative implementation; performs DFS over tree
return $\text{argmax}_{\mathtt{a}}\, \text{Q}(s_0, \mathtt{a}, 0, \mathcal{S}, \mathcal{A}, P, R, H)$

$\text{Q}(\mathtt{s}, \mathtt{a}, \mathtt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$

return $\sum_{\mathtt{ns}} P(\mathtt{ns} \mid \mathtt{s}, \mathtt{a})(R(\mathtt{s}, \mathtt{a}, \mathtt{ns}) + \text{V}(\mathtt{ns}, \mathtt{t} + 1, \mathcal{S}, \mathcal{A}, P, R, H))$

$\text{V}(\mathtt{s}, \mathtt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$

if $t = H$
  return 0
return $\max_{\mathtt{a}} \text{Q}(\mathtt{s}, \mathtt{a}, \mathtt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$



EXPECTIMAXSEARCH$(s_0, \mathcal{S}, \mathcal{A}, P, R, H)$

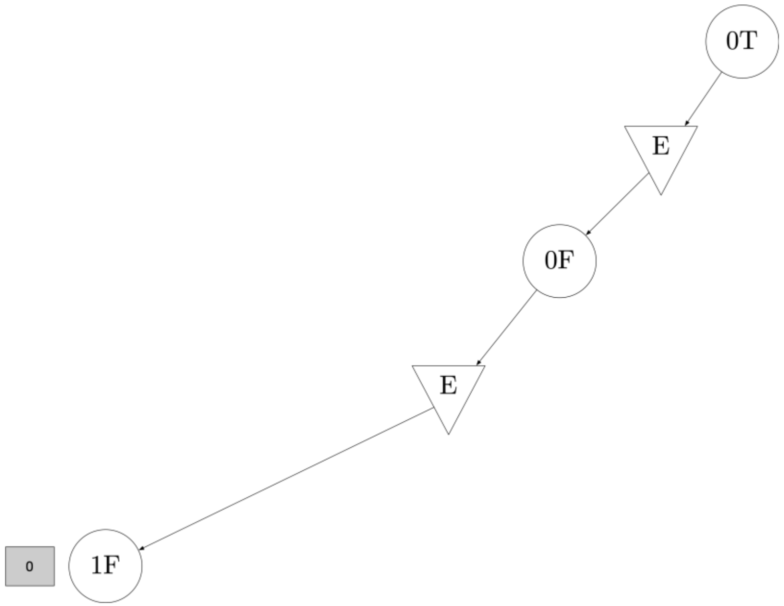
// Alternative implementation; performs DFS over tree
return $\text{argmax}_{\mathtt{a}}\ \mathrm{Q}(s_0, \mathtt{a}, 0, \mathcal{S}, \mathcal{A}, P, R, H)$

$\mathrm{Q}(\mathtt{s}, \mathtt{a}, \mathtt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$

return $\sum_{\mathtt{ns}} P(\mathtt{ns} \mid \mathtt{s}, \mathtt{a})(R(\mathtt{s}, \mathtt{a}, \mathtt{ns}) + \mathrm{V}(\mathtt{ns}, \mathtt{t}+1, \mathcal{S}, \mathcal{A}, P, R, H))$

$\mathrm{V}(\mathtt{s}, \mathtt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$

if $t = H$
    return 0
return $\max_{\mathtt{a}} \mathrm{Q}(\mathtt{s}, \mathtt{a}, \mathtt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$



$\textsc{ExpectimaxSearch}(s_0, \mathcal{S}, \mathcal{A}, P, R, H)$

// Alternative implementation; performs DFS over tree

return $\text{argmax}_{\mathtt{a}}\ \text{Q}(s_0, \mathtt{a}, 0, \mathcal{S}, \mathcal{A}, P, R, H)$

$\text{Q}(\mathtt{s}, \mathtt{a}, \mathtt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$

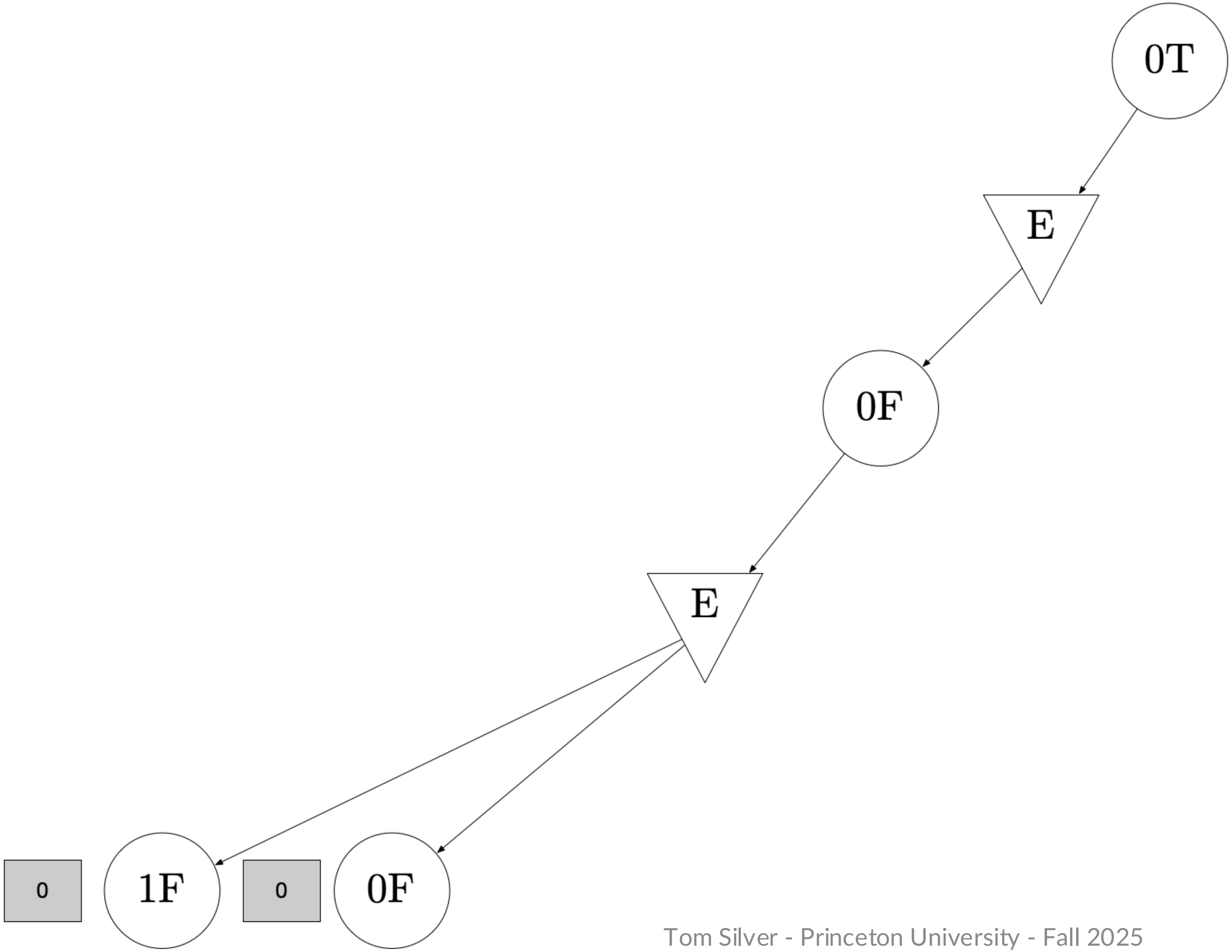
return $\sum_{\mathtt{ns}} P(\mathtt{ns} \mid \mathtt{s}, \mathtt{a})(R(\mathtt{s}, \mathtt{a}, \mathtt{ns}) + \text{V}(\mathtt{ns}, \mathtt{t}+1, \mathcal{S}, \mathcal{A}, P, R, H))$

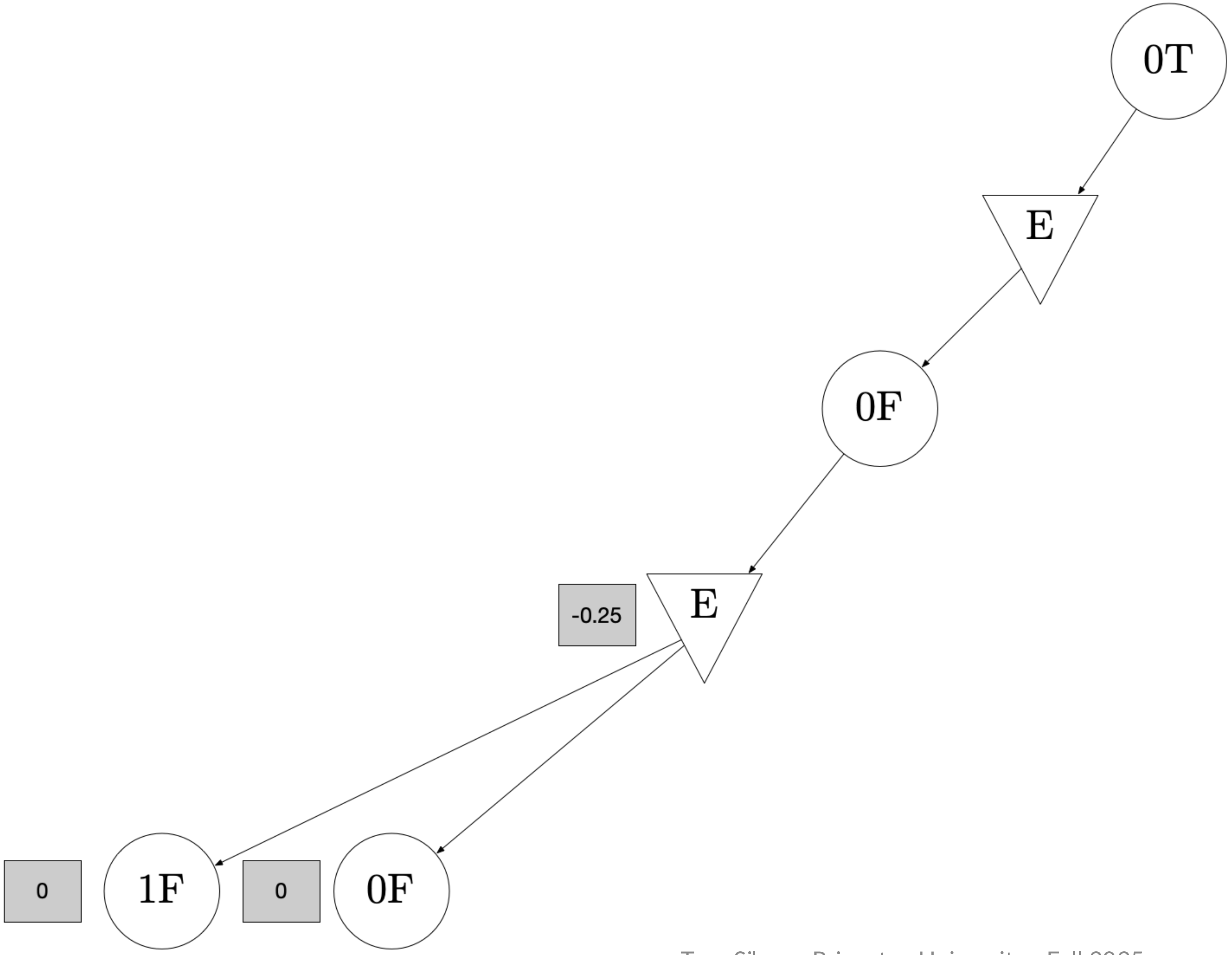
$\text{V}(\mathtt{s}, \mathtt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$

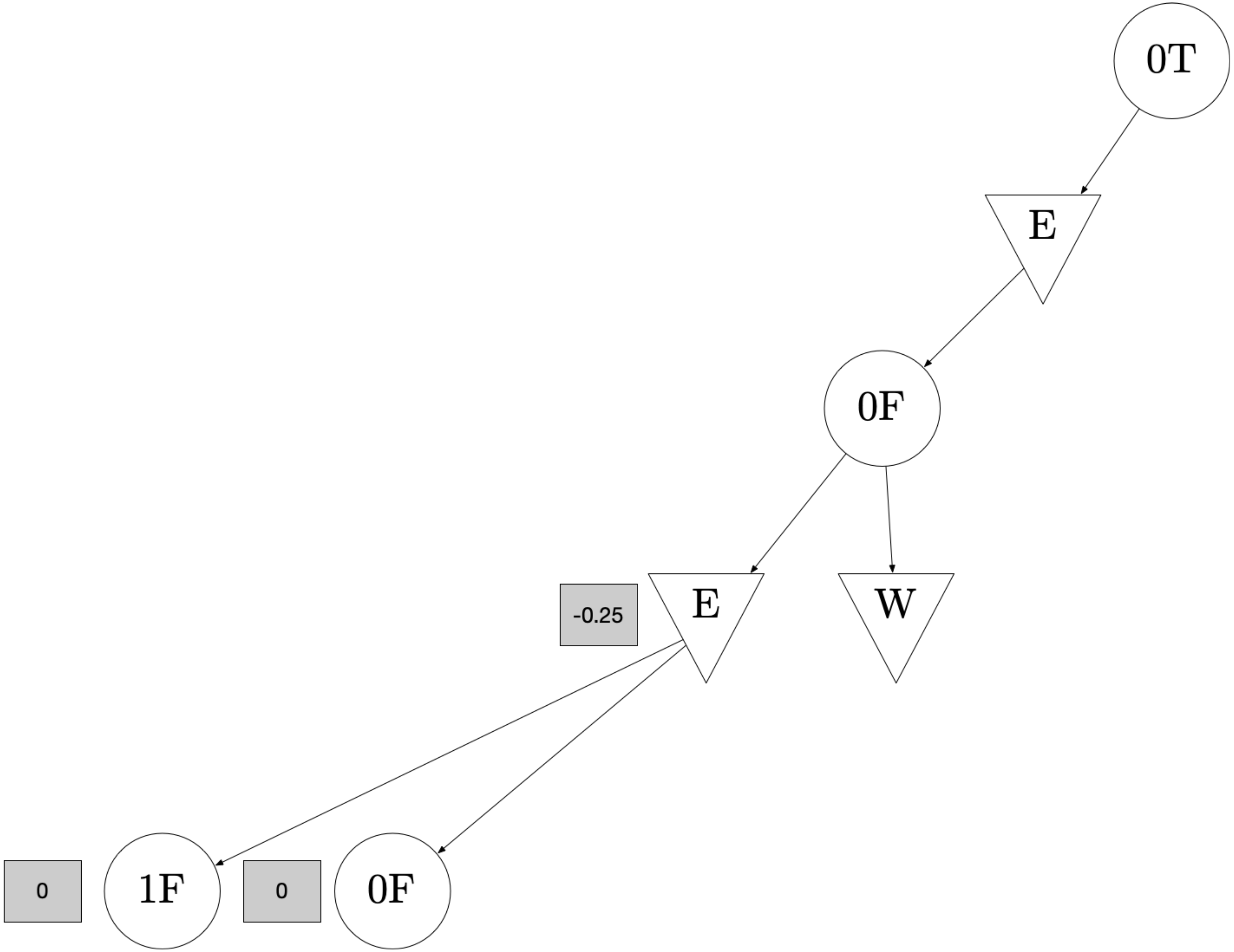
if $t = H$

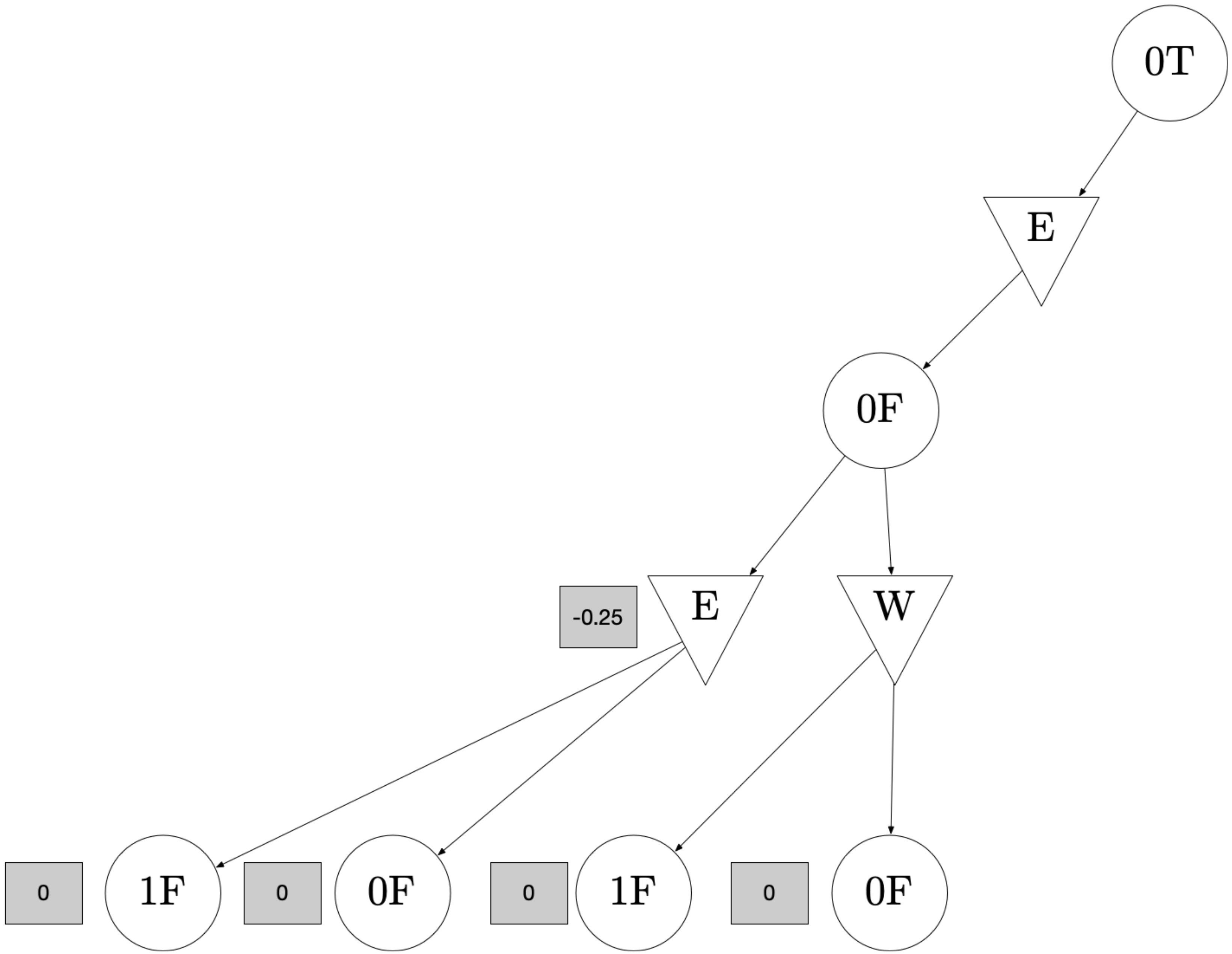
  return 0

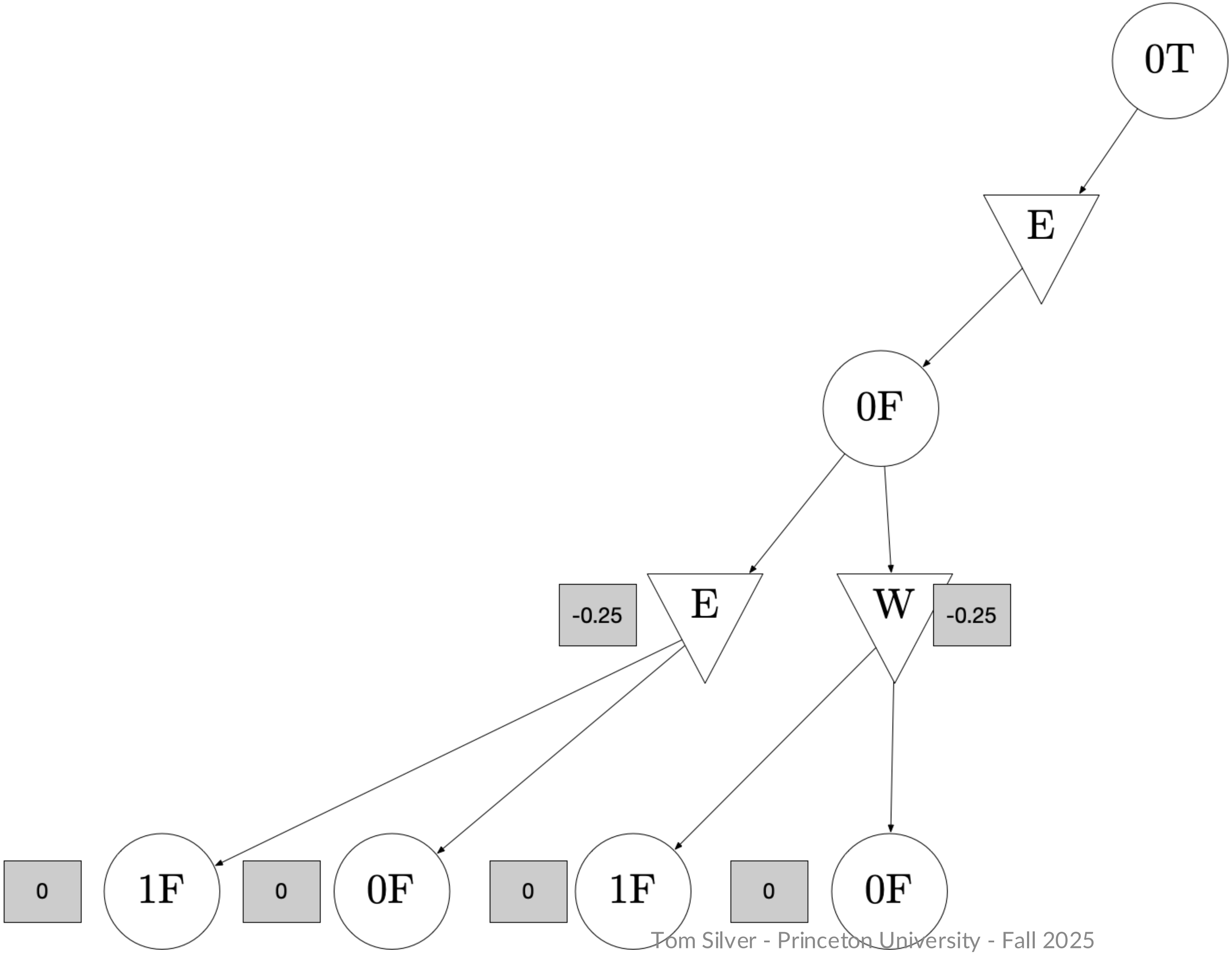
return $\max_{\mathtt{a}} \text{Q}(\mathtt{s}, \mathtt{a}, \mathtt{t}, \mathcal{S}, \mathcal{A}, P, R, H)$

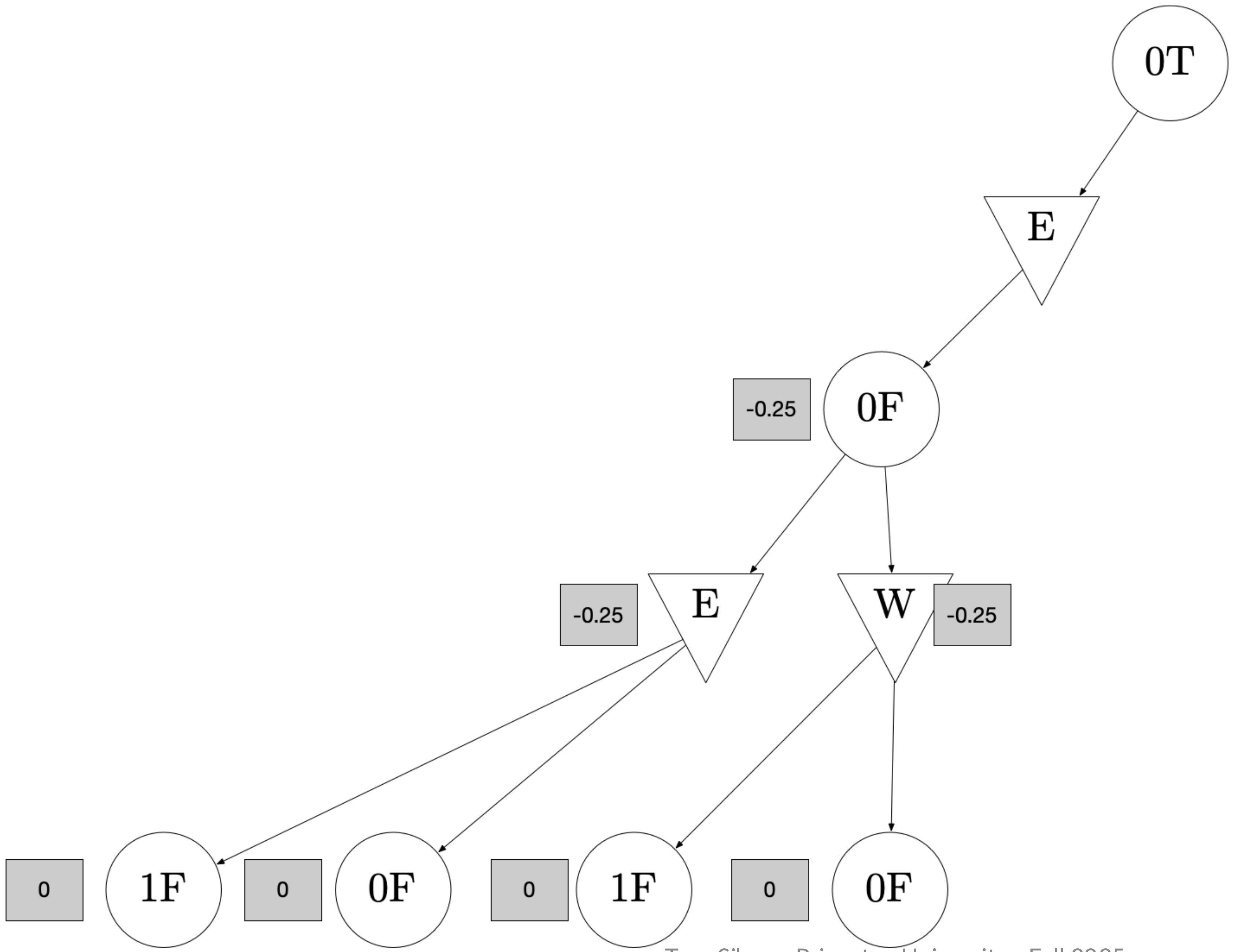

0T
E
0F
E
0
1F
0
0F

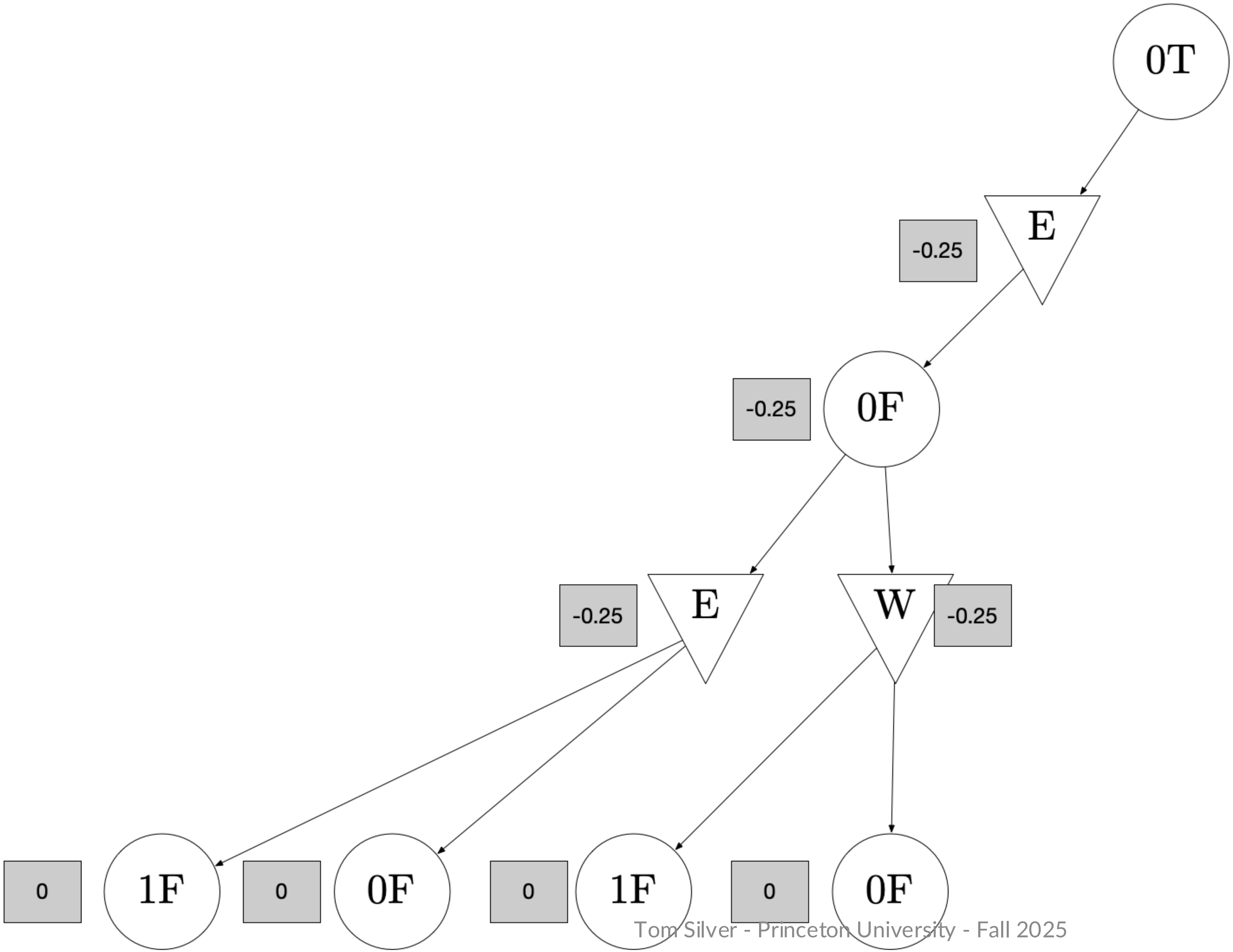

0T
E
0F
-0.25
E
0
1F
0
0F

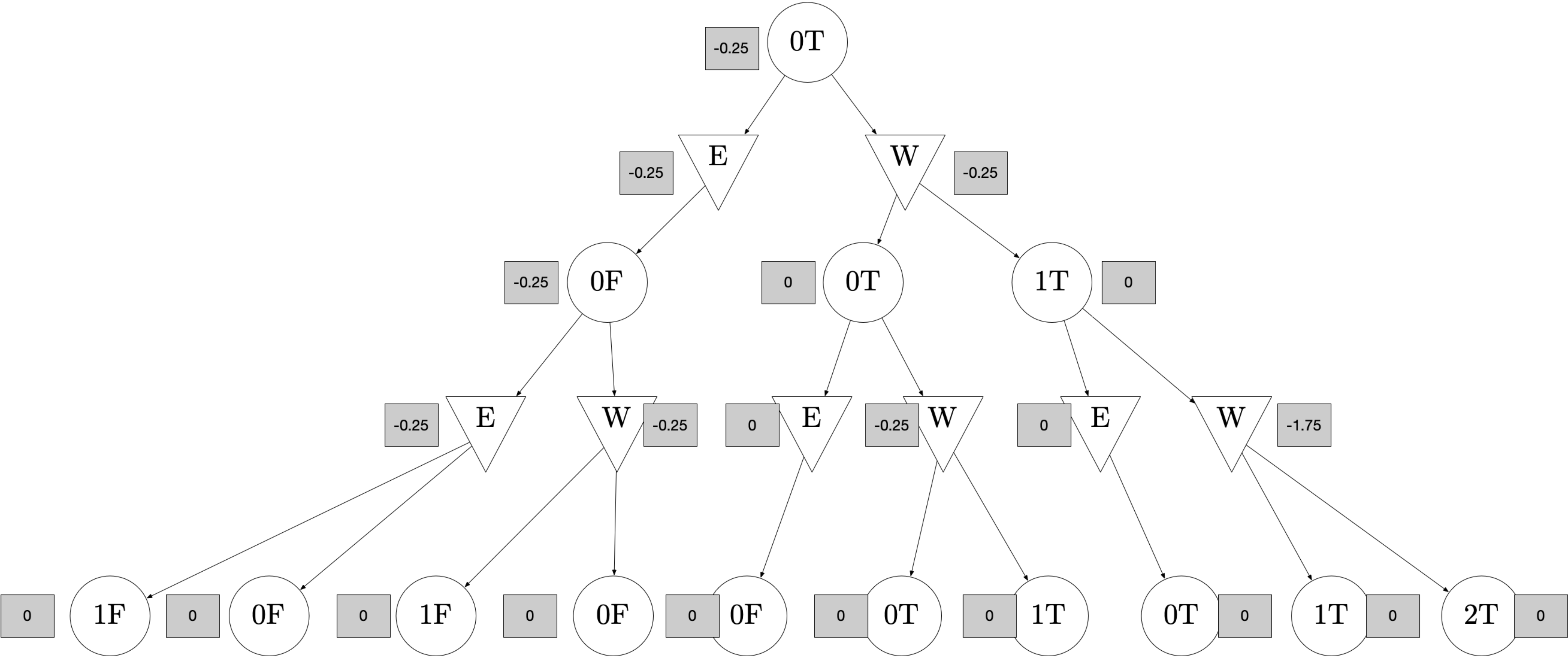

0T
E
0F
-0.25
E
W
0
1F
0
0F


0T
E
0F
-0.25
E
W
0
1F
0
0F
0
1F
0
0F


0T
E
0F
-0.25
E
W
-0.25
0
1F
0
0F
0
1F
0
0F


0T
E
-0.25
0F
-0.25
E
W
-0.25
0
1F
0
0F
0
1F
0
0F


0T
E
-0.25
-0.25
0F
-0.25
E
W
-0.25
0
1F
0
0F
0
1F
0
0F


-0.25
0T
-0.25
E
W
-0.25
-0.25
0F
0
0T
1T
0
-0.25
E
W
-0.25
0
E
-0.25
W
0
E
W
-1.75
0
1F
0
0F
0
1F
0
0F
0
0F
0
0T
0
1T
0T
0
1T
0
2T
0

Finding Reachable States: Infinite Depth

A state s is *reachable* from s_0 in the infinite-horizon case if there exists *some* T such that s is reachable at depth T.

How to find these reachable states?

Finding Reachable States: Infinite Depth

A state s is *reachable* from s_0 in the infinite-horizon case if there exists *some* T such that s is reachable at depth T.

How to find these reachable states?

Build out AODAG, but only create nodes for *new* states.

Using Reachable States: Infinite Depth

Idea: create an *abstract MDP* with only the reachable states.

$$(\mathcal{S}, \mathcal{A}, P, R, \gamma) \xrightarrow{\text{abstraction}} (\mathcal{S}_{\text{reachable}}, \mathcal{A}, P, R, \gamma)$$

Then, plan in the abstract MDP.

Using Reachable States: Infinite Depth

Idea: create an *abstract MDP* with only the reachable states.

$$(\mathcal{S}, \mathcal{A}, P, R, \gamma) \xrightarrow{\text{abstraction}} ({\color{blue}\mathcal{S}_{\text{reachable}}}, \mathcal{A}, P, R, \gamma)$$

Technically, domains of P, R also change.

Then, plan in the abstract MDP.

This is a simple MDP abstraction. Others exist; active research area.

When is Reachability Not Enough?

- Sometimes, the number of reachable states is just too large

When is Reachability Not Enough?

- Sometimes, the number of reachable states is just too large
- Receding horizon control may also fail
  - If all rewards within short horizons are trivial
  - Or if the rewards seen within short horizons are misleading

When is Reachability Not Enough?

- Sometimes, the number of reachable states is just too large
- Receding horizon control may also fail
  - If all rewards within short horizons are trivial
  - Or if the rewards seen within short horizons are misleading

- *Heuristics* to the rescue!

Heuristics for MDPs

- For MDPs, a **heuristic** $\hat{V}$ is an approximate value function:

$$\hat{V}(s) \approx V(s).$$

Heuristics for MDPs

- For MDPs, a **heuristic** $\hat{V}$ is an approximate value function:

$$\hat{V}(s) \approx V(s).$$

- A heuristic is **admissible** if $\forall s \in \mathcal{S}. \hat{V}(s) \geq V(s)$
  - Can also weaken this to "for all *reachable* states"

Expectimax + Heuristic Leaf Evals

Idea: do receding horizon control, with expectimax search.

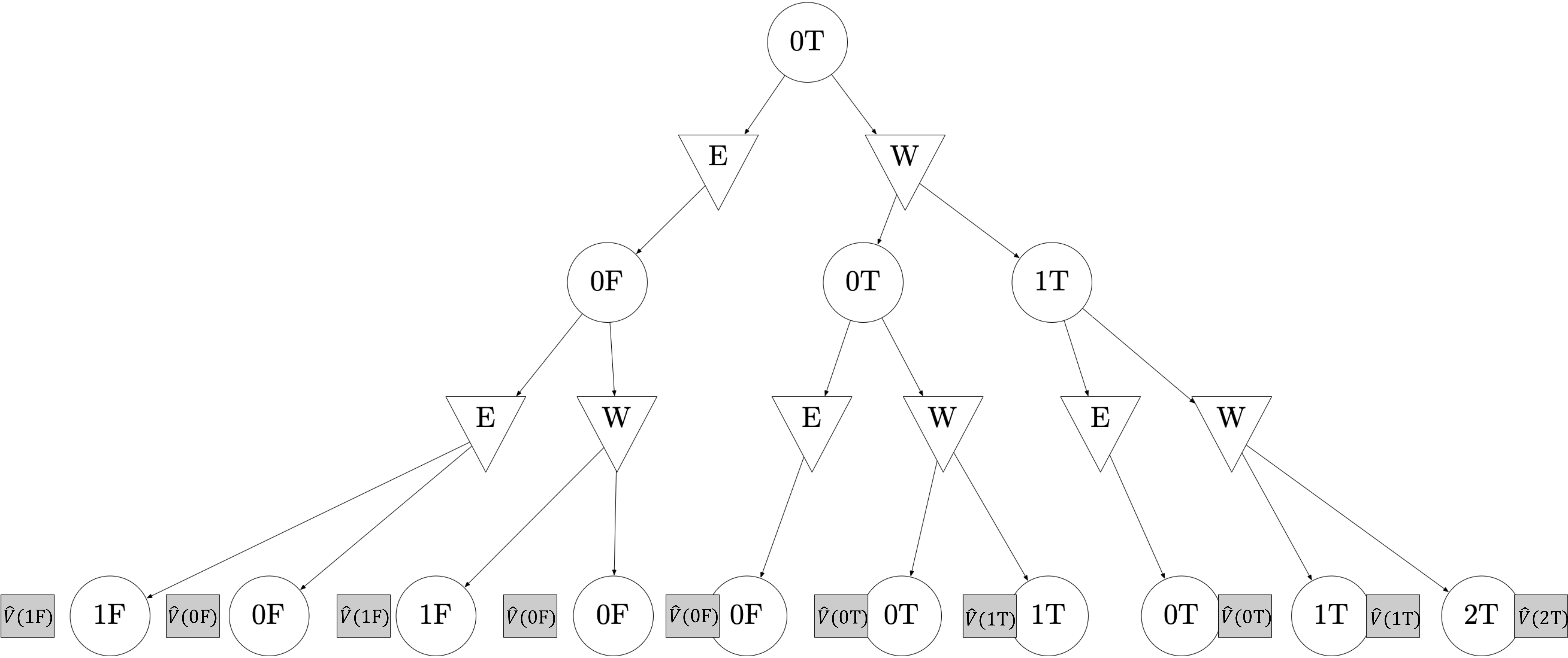
But! When we get to a leaf node, use $\hat{V}$ to evaluate it.

Expectimax + Heuristic Leaf Evals

Idea: do receding horizon control, with expectimax search.

But! When we get to a leaf node, use $\hat{V}$ to evaluate it.

If $\hat{V}$ were perfect, we could just do $H = 1$


0T
E
W
0F
0T
1T
E
W
E
W
E
W
$\hat{V}(1F)$
1F
$\hat{V}(0F)$
0F
$\hat{V}(1F)$
1F
$\hat{V}(0F)$
0F
$\hat{V}(0F)$
0F
$\hat{V}(0T)$
0T
$\hat{V}(1T)$
1T
0T
$\hat{V}(0T)$
1T
$\hat{V}(1T)$
2T
$\hat{V}(2T)$

Expectimax + Heuristic Leaf Evals

Idea: do receding horizon control, with expectimax search.

But! When we get to a leaf node, use $\hat{V}$ to evaluate it.

What are benefits / disadvantages to using a larger H?

Expectimax + Heuristic Leaf Evals

Idea: do receding horizon control, with expectimax search.

But! When we get to a leaf node, use $\hat{V}$ to evaluate it.

Limitation: we're exhaustively exploring the tree to depth H.

Wouldn't it be better to "focus" our computation?

Real-Time Dynamic Programming (RTDP)

Idea #1: Build only some parts of the AODAG, not all of it.

But which parts?

Real-Time Dynamic Programming (RTDP)

Idea #1: Build only some parts of the AODAG, not all of it.

But which parts?

Idea #2: Use running estimate of the value function, *initialized with the heuristic*, to choose parts to expand.

Expand the parts of the AODAG that seem “most promising.”

Real-Time Dynamic Programming (RTDP)

Real-Time Dynamic Programming (RTDP):

1. Use the heuristic to initialize a value function estimate.

Real-Time Dynamic Programming (RTDP)

Real-Time Dynamic Programming (RTDP):

1. Use the heuristic to initialize a value function estimate.
2. Sample a trajectory from the initial state. Select actions *greedily* with respect to value function estimate.

Real-Time Dynamic Programming (RTDP)

Real-Time Dynamic Programming (RTDP):

1. Use the heuristic to initialize a value function estimate.
2. Sample a trajectory from the initial state. Select actions *greedily* with respect to value function estimate.
3. Perform Bellman backups for all states in the trajectory

Best to start from the end and work backwards

Real-Time Dynamic Programming (RTDP)

Real-Time Dynamic Programming (RTDP):

1. Use the heuristic to initialize a value function estimate.
2. Sample a trajectory from the initial state. Select actions *greedily* with respect to value function estimate.
3. Perform Bellman backups for all states in the trajectory
4. Repeat from (2)

There are different possible formulations of RTDP. This one is based on "Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming." Bonet & Geffner (2003).

RTDP$(s_0, \hat{V}, \mathcal{S}, \mathcal{A}, P, R, \gamma)$

// Initialize value function estimate with heuristic
$\texttt{V}[\texttt{s}] = \hat{V}(\texttt{s})$ for each $\texttt{s} \in \mathcal{S}$ (can do this *lazily*)
// Sample trajectories and update until time budget runs out
repeat
  // Turn value function estimate into greedy policy
  $\pi(\texttt{s}) = \text{argmax}_{\texttt{a} \in \mathcal{A}} \sum_{\texttt{ns}} P(\texttt{ns} \mid \texttt{s}, \texttt{a})(R(\texttt{s}, \texttt{a}, \texttt{ns}) + \gamma \texttt{V[ns]})$
  // Collect a trajectory from s_0
  $\tau = \textsc{CollectTrajectory}(s_0, \pi, \mathcal{S}, \mathcal{A}, P, R, \gamma)$
  // Update value function estimate
  for $\texttt{s} \in \tau$
    $\texttt{V}[\texttt{s}] = \textsc{BellmanBackup}(\texttt{s}, \texttt{V}, \mathcal{S}, \mathcal{A}, P, R, \gamma)$
return $\texttt{V}$

$\text{RTDP}(s_0, \hat{V}, \mathcal{S}, \mathcal{A}, P, R, \gamma)$

// Initialize value function estimate with heuristic
$\texttt{V}[\texttt{s}] = \hat{V}(\texttt{s})$ for each $\texttt{s} \in \mathcal{S}$ (can do this *lazily*)
// Sample trajectories and update until time budget runs out
repeat
    // Turn value function estimate into greedy policy
    $\pi(\texttt{s}) = \text{argmax}_{\texttt{a} \in \mathcal{A}} \sum_{\texttt{ns}} P(\texttt{ns} \mid \texttt{s}, \texttt{a})(R(\texttt{s}, \texttt{a}, \texttt{ns}) + \gamma \texttt{V[ns]})$
    // Collect a trajectory from s_0
    $\tau = \text{CollectTrajectory}(s_0, \pi, \mathcal{S}, \mathcal{A}, P, R, \gamma)$
    // Update value function estimate
    for $\texttt{s} \in \tau$
        $\texttt{V}[\texttt{s}] = \text{BellmanBackup}(\texttt{s}, \texttt{V}, \mathcal{S}, \mathcal{A}, P, R, \gamma)$
return $\texttt{V}$

Note: we're in an infinite-horizon setting here, with stationary value functions. Finite-horizon versions also possible.

Value function approximation; initialized to $\hat{V}$

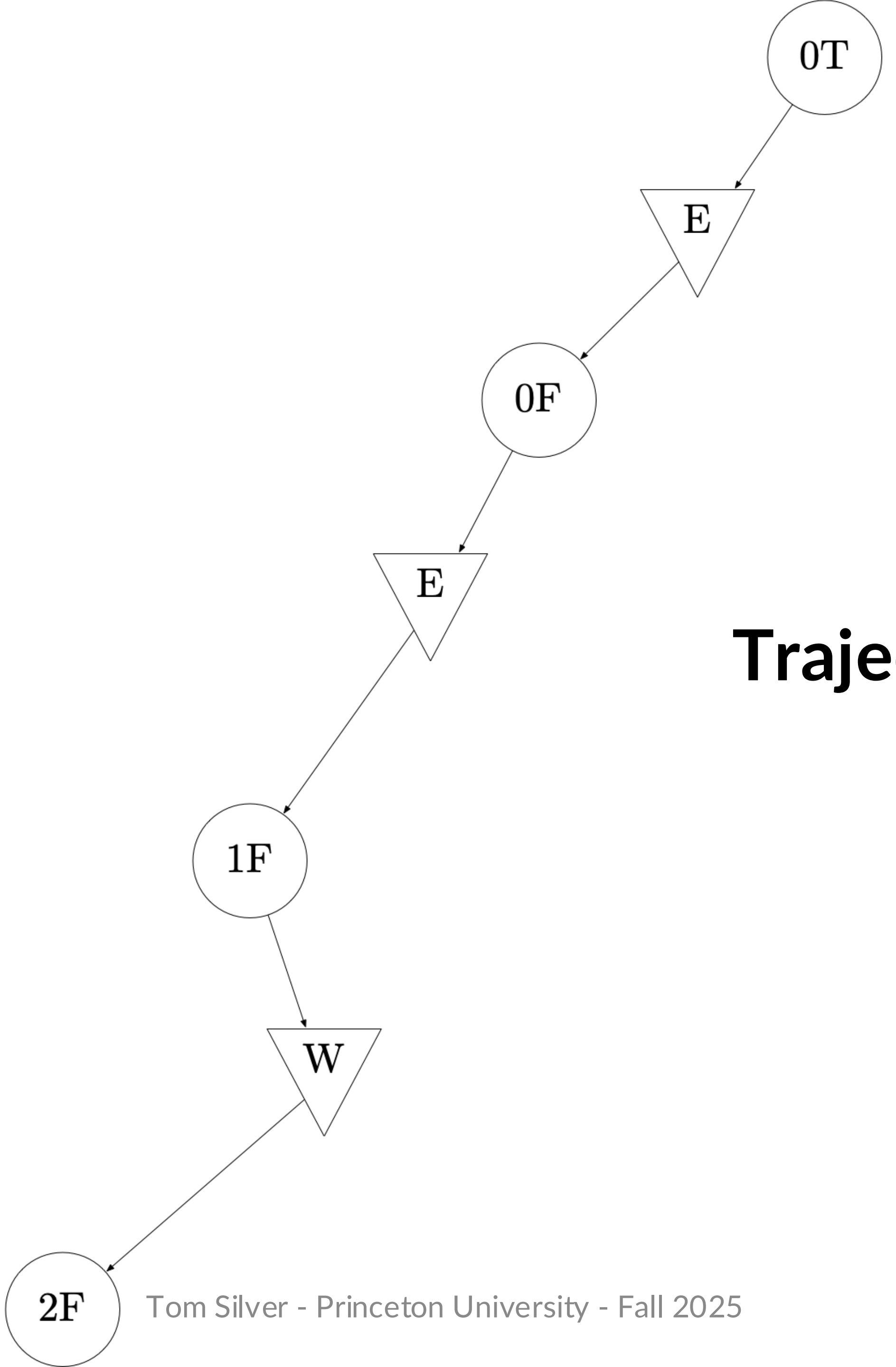
s	V
0T	-1
1T	-2
2T	-3
0F	-2
1F	-4
2F	-6

Greedy policy

s	V	π
0T	-1	E
1T	-2	E
2T	-3	E
0F	-2	E
1F	-4	W
2F	-6	E

Random tiebreaking for *F states

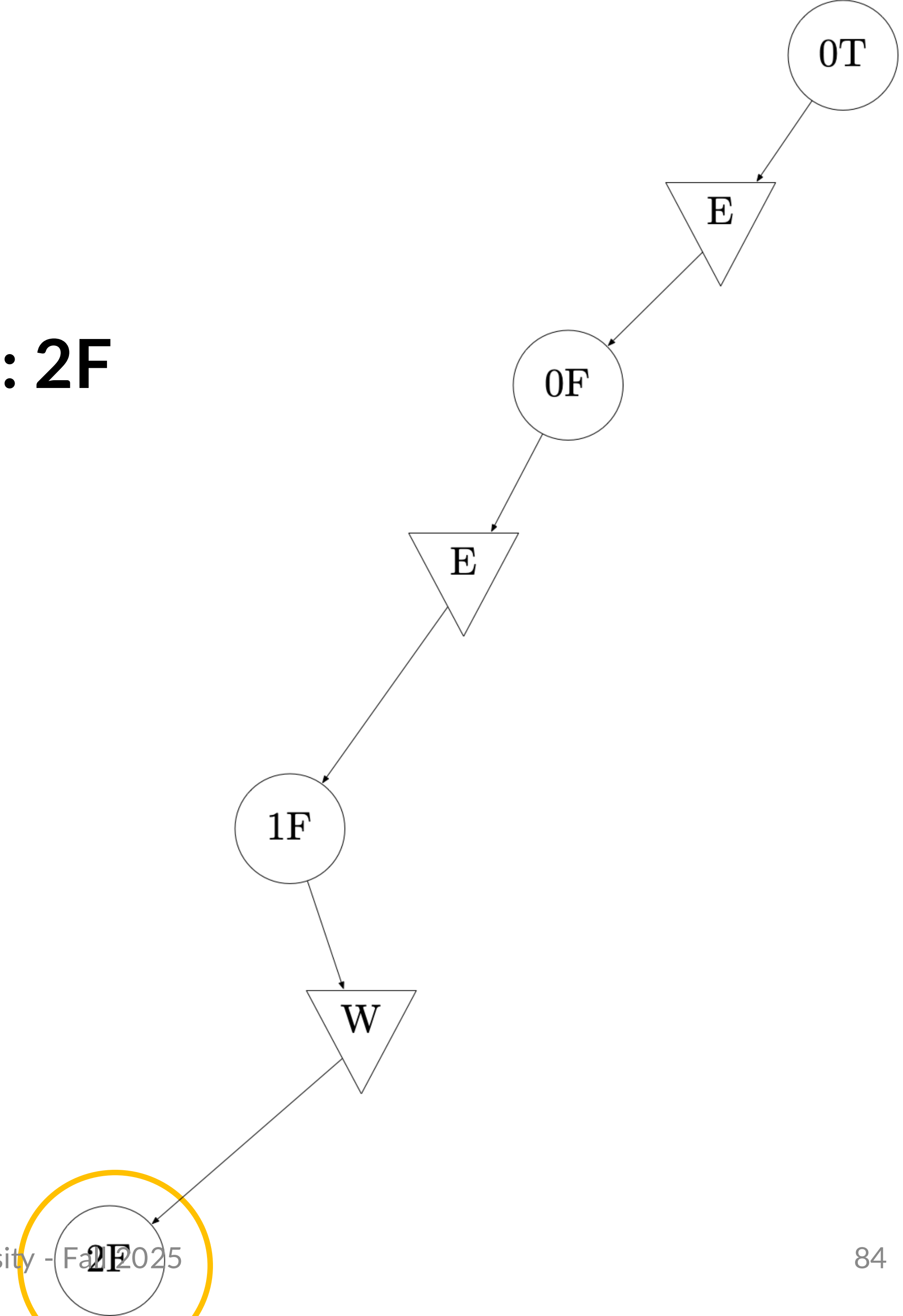
s	V	π
0T	-1	E
1T	-2	E
2T	-3	E
0F	-2	E
1F	-4	W
2F	-6	E



Trajectory Collection

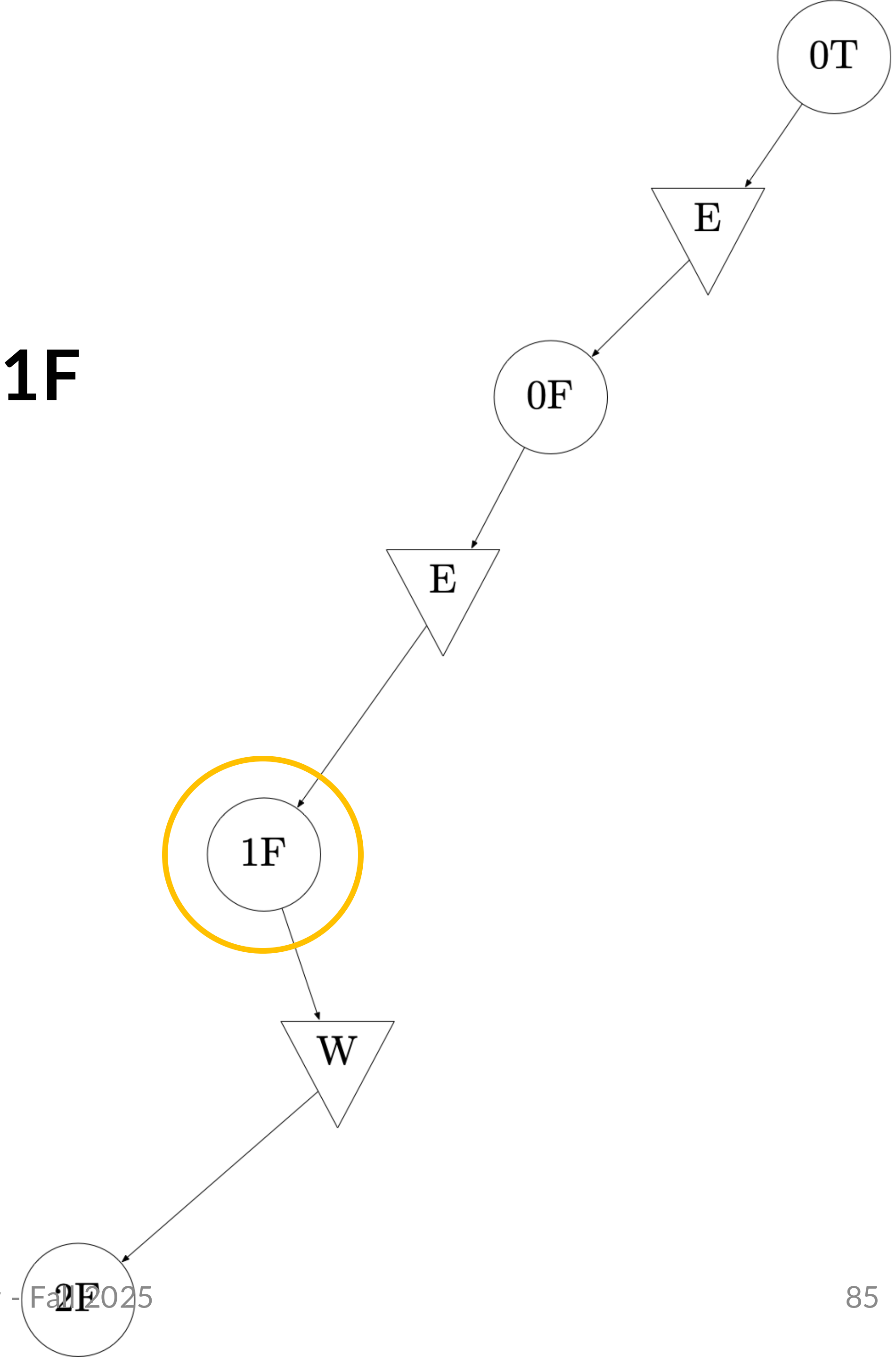
Bellman Backup: 2F

s	V	π
0T	-1	E
1T	-2	E
2T	-3	E
0F	-2	E
1F	-4	W
2F	*	E



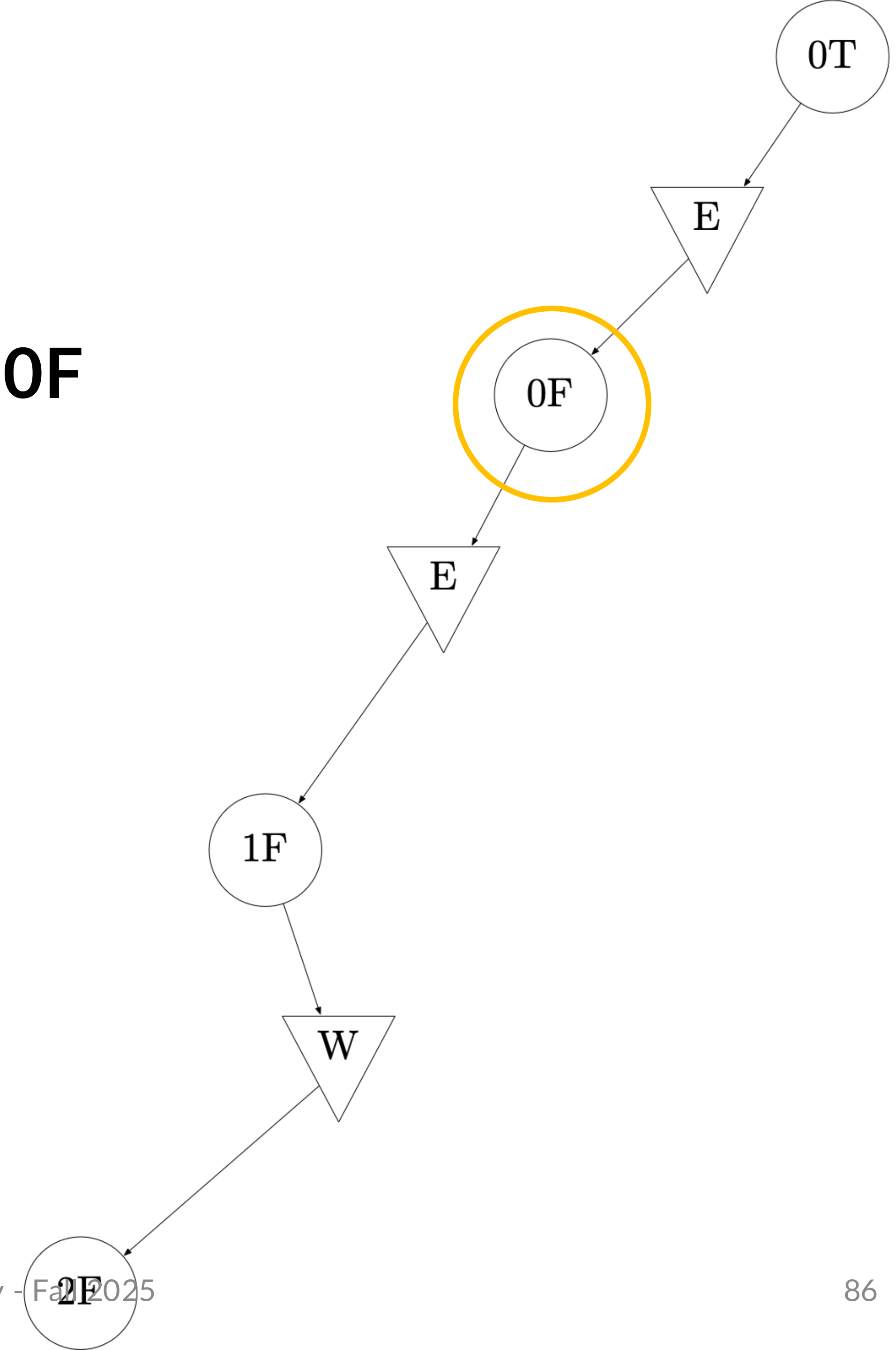
Bellman Backup: 1F

s	V	π
0T	-1	E
1T	-2	E
2T	-3	E
0F	-2	E
1F	*	W
2F	*	E



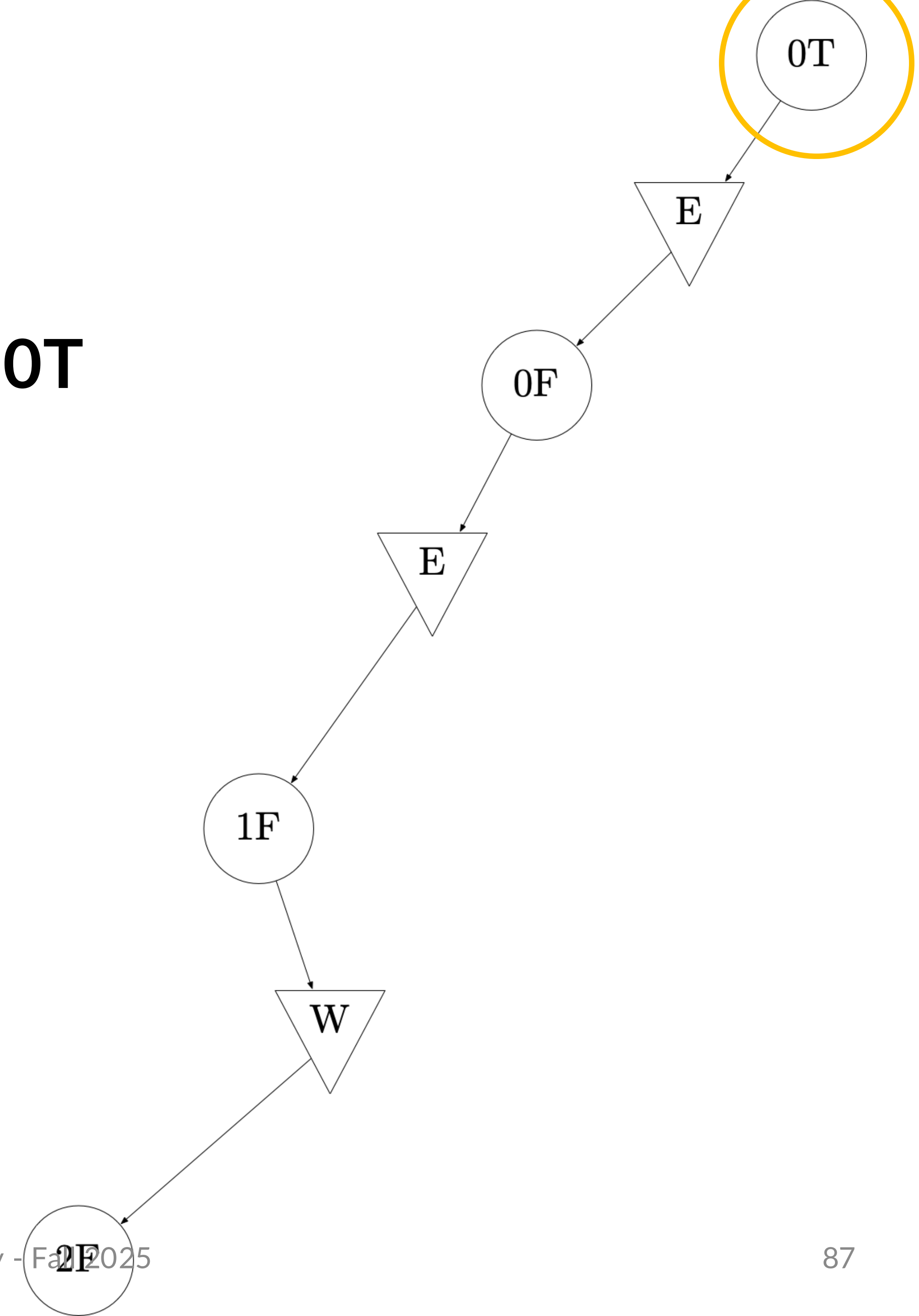
s	V	π
0T	-1	E
1T	-2	E
2T	-3	E
0F	*	E
1F	*	W
2F	*	E

Bellman Backup: 0F



Bellman Backup: 0T

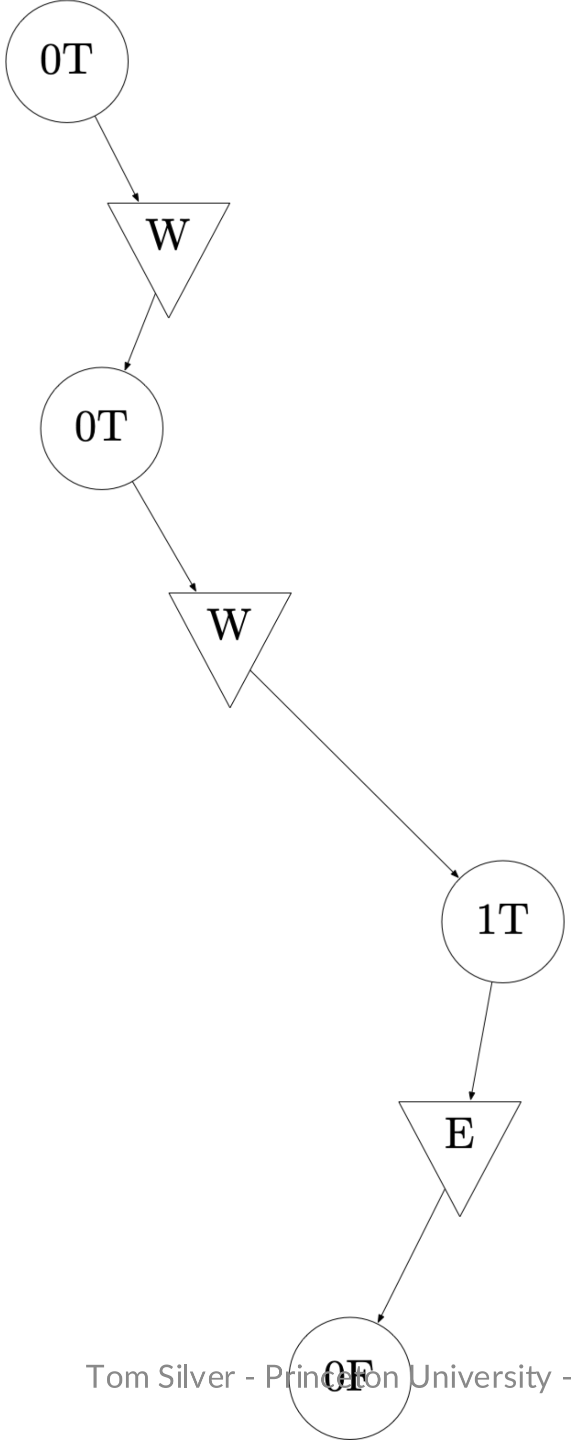
s	V	π
0T	*	E
1T	-2	E
2T	-3	E
0F	*	E
1F	*	W
2F	*	E



s	V	π
0T	*	**W**
1T	-2	**E**
2T	-3	**W**
0F	*	**E**
1F	*	**W**
2F	*	**E**

Greedy Policy Updates

s	*V*	*π*
0T	*	W
1T	-2	E
2T	-3	W
0F	*	E
1F	*	W
2F	*	E



Trajectory Collection

Etc.

RTDP Guarantees

- If heuristic is admissible, RTDP will converge to optimal policy (Barto, Bradtke, & Singh 1995; Bertsekas 1995)
- However, it may converge quite slowly, especially because it will repeatedly visit “solved” states
- Labelled RTDP (LRTDP) is an extension that avoids revisiting “solved” states (Bonet & Geffner 2003)

Stochastic Shortest Paths (SSPs)

Stochastic shortest path (SSP):

1. Rewards are nonpositive: $R(s, a, s') \leq 0$ for $s, s' \in \mathcal{S}, a \in \mathcal{A}$
2. There are done states $D \subseteq S$
3. There is at least one proper policy [1]

Can be understood as *goals*

Necessary because $H = \infty, \gamma = 1$.

[1] This is a technical condition that we will not define here.

Stochastic Shortest Paths (SSPs)

Stochastic shortest path (SSP):

1. Rewards are nonpositive: $R(s, a, s') \leq 0$ for $s, s' \in \mathcal{S}, a \in \mathcal{A}$
2. There are done states $D \subseteq S$
3. There is at least one proper policy [1]

As name suggests, we want to find the shortest path from the current state to a done state.

[1] This is a technical condition that we will not define here.

Determinization

Idea: given an SSP MDP, convert to deterministic problem.

Then, use heuristic to plan in the deterministic problem.

Determinization

Idea: given an SSP MDP, convert to deterministic problem.
Then, use heuristic to plan in the deterministic problem.

Caution! This strategy is approximate; we're losing info.

Determinization

Idea: given an SSP MDP, convert to deterministic problem.

Then, use heuristic to plan in the deterministic problem.

Caution! This strategy is approximate; we're losing info.

Good news: *we can use methods from for informed search, like A*, to plan in determinized problem.*

We'll return to these soon

Most-Likely Outcome Determinization

$$\langle(\mathcal{S}, \mathcal{A}, P, R, \mathcal{S}_{\mathcal{G}}), s_0\rangle \xrightarrow{\text{MLO determinize}} (\mathcal{S}, \mathcal{A}, \mathcal{S}_{\mathcal{G}}, s_0, f, c)$$

where:

- $f(s, a) = \text{argmax}_{s' \in \mathcal{S}} P(s' \mid s, a)$

"Most likely outcome"

- $c(s, a, s') = -R(s, a, s')$

Most-Likely Outcome Determinization

$$\langle(\mathcal{S}, \mathcal{A}, P, R, \mathcal{S}_{\mathcal{G}}), s_0\rangle \xrightarrow{\text{MLO determinize}} (\mathcal{S}, \mathcal{A}, \mathcal{S}_{\mathcal{G}}, s_0, f, c)$$

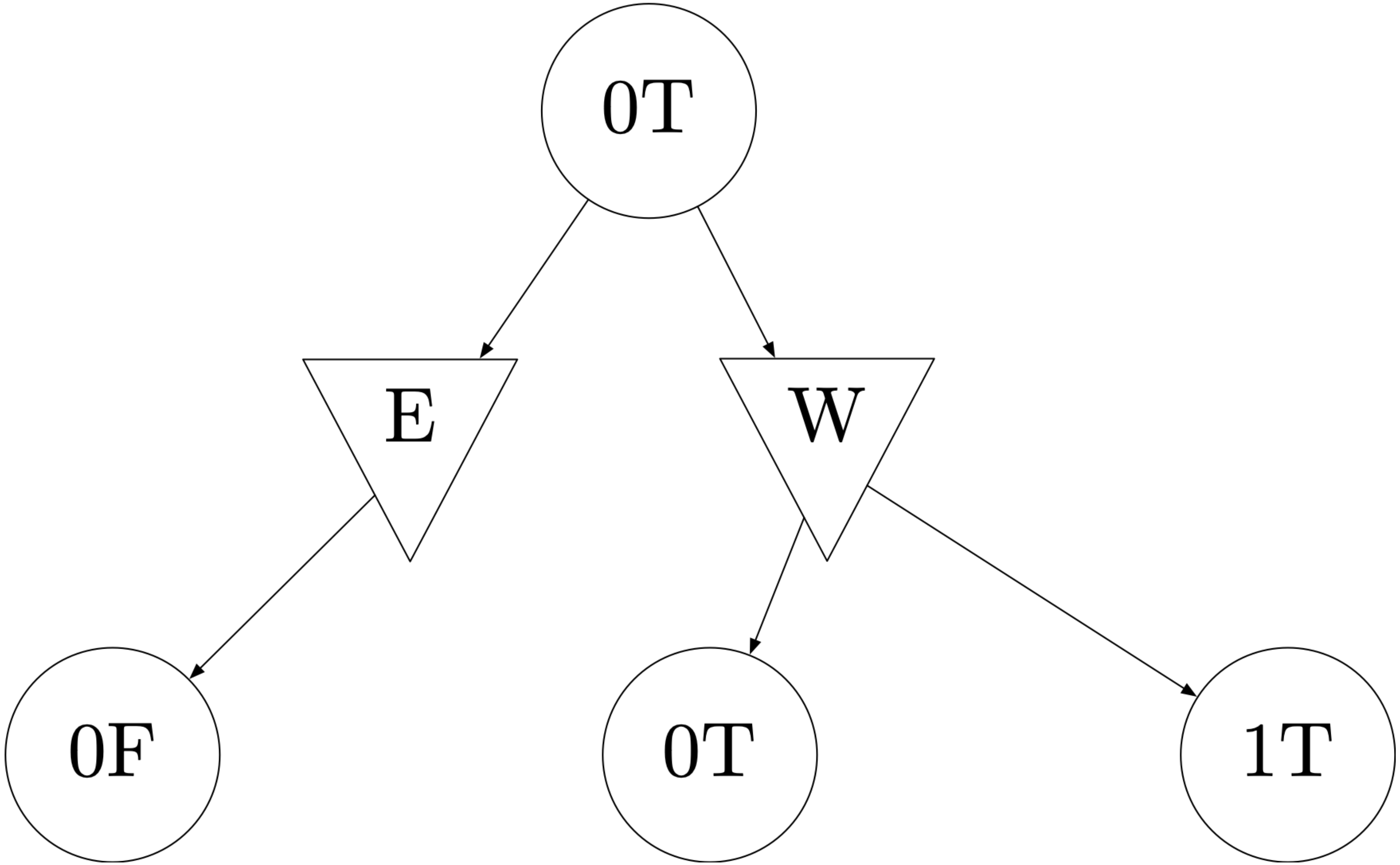
where:

- $f(s, a) = \text{argmax}_{s' \in \mathcal{S}} P(s' \mid s, a)$
- $c(s, a, s') = -R(s, a, s')$

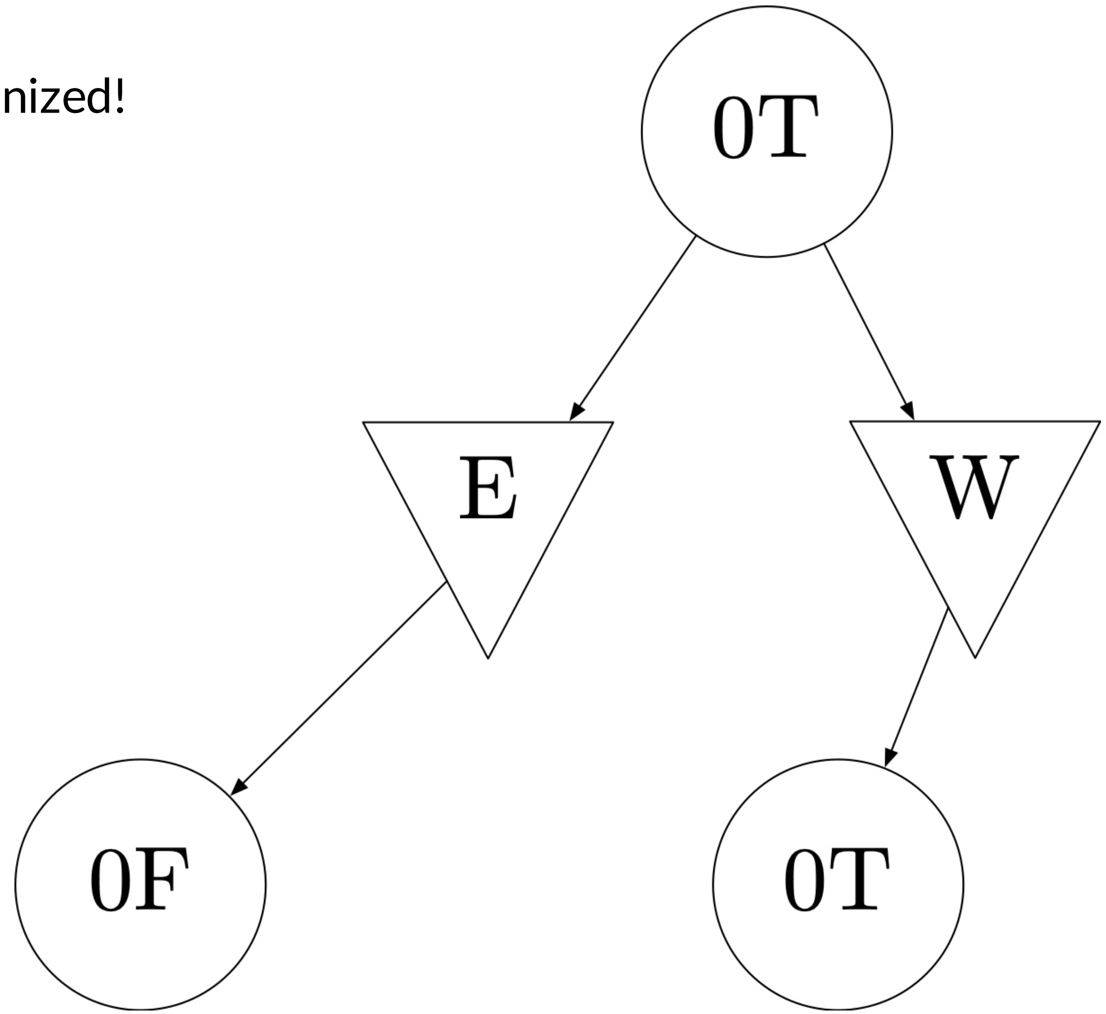
"Most likely outcome"

When might this go badly?

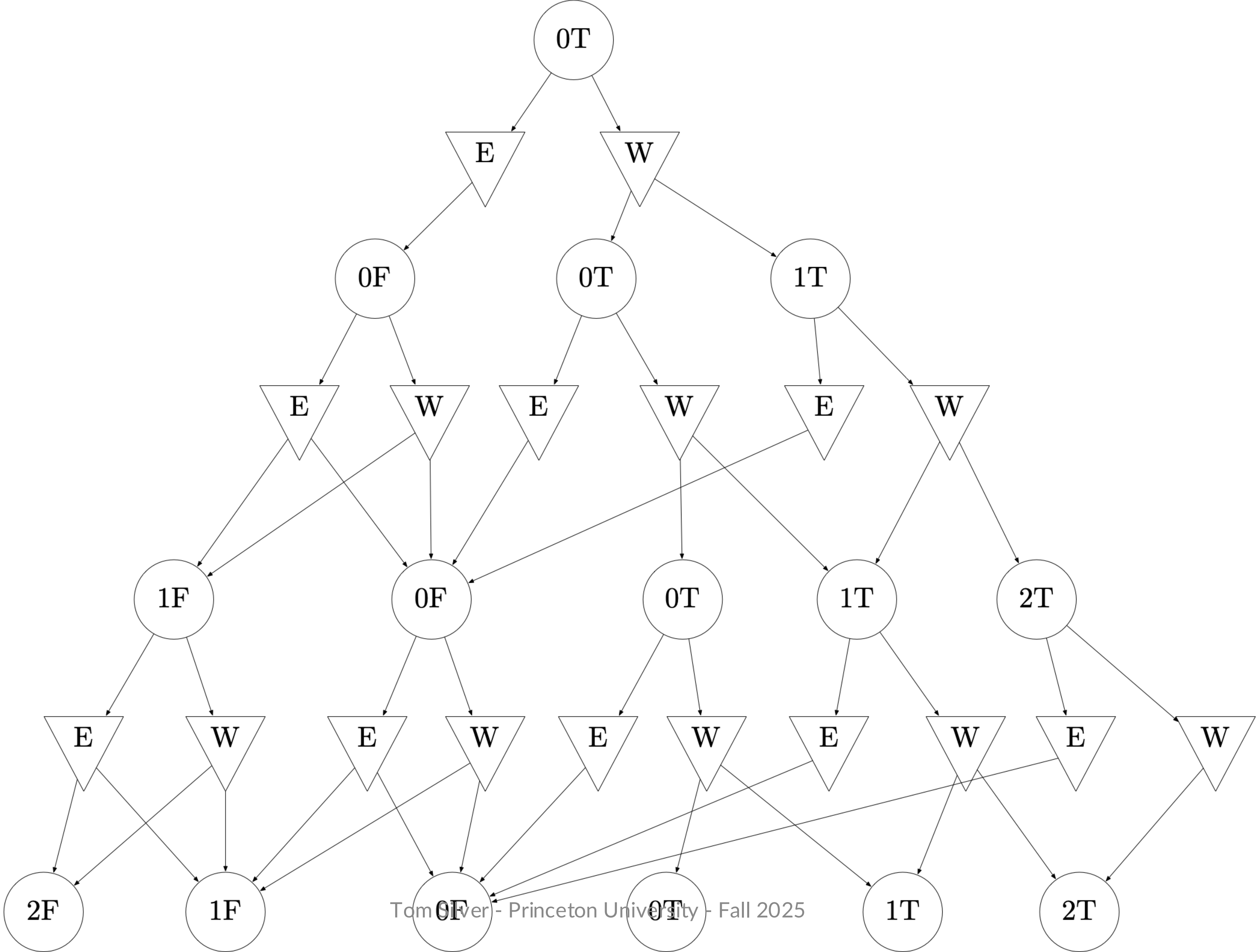
Original



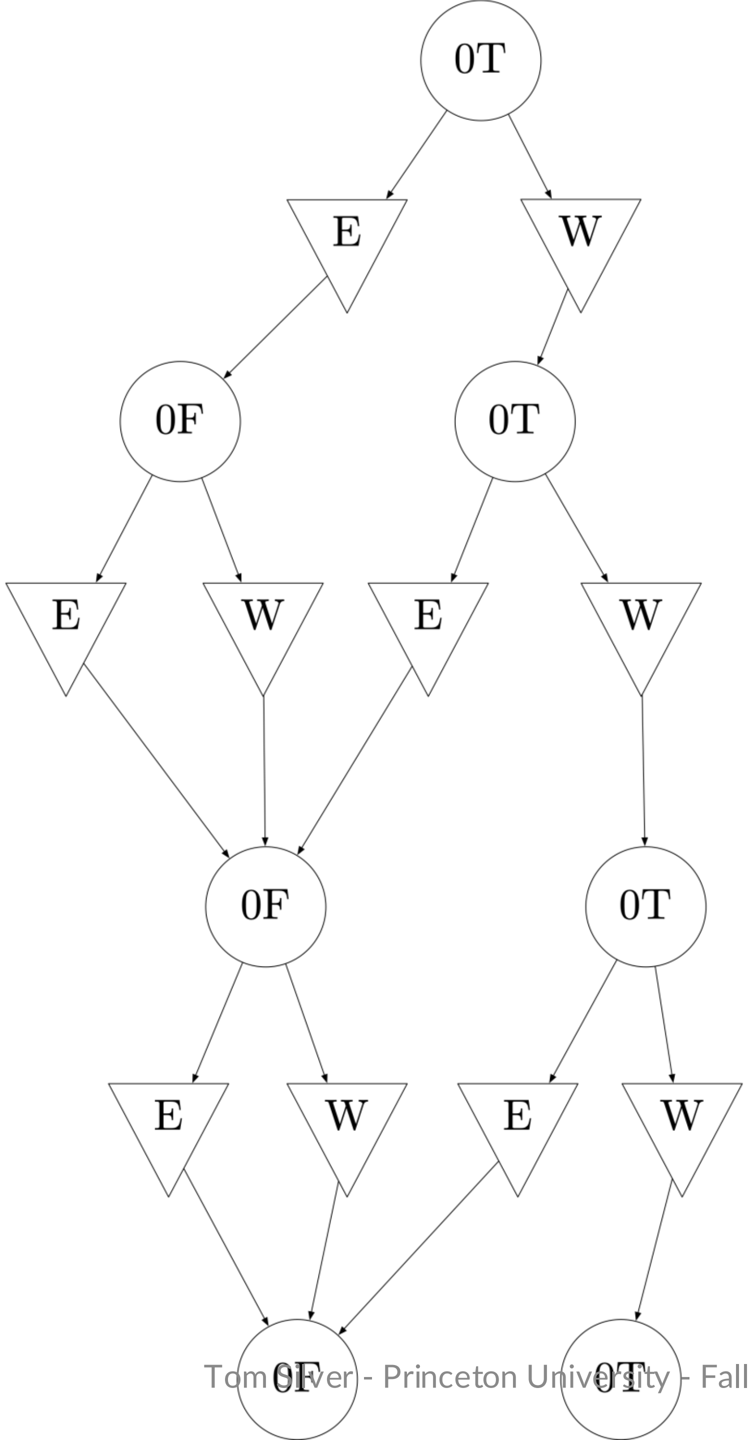
MLO determinized!



Original



MLO determinized!



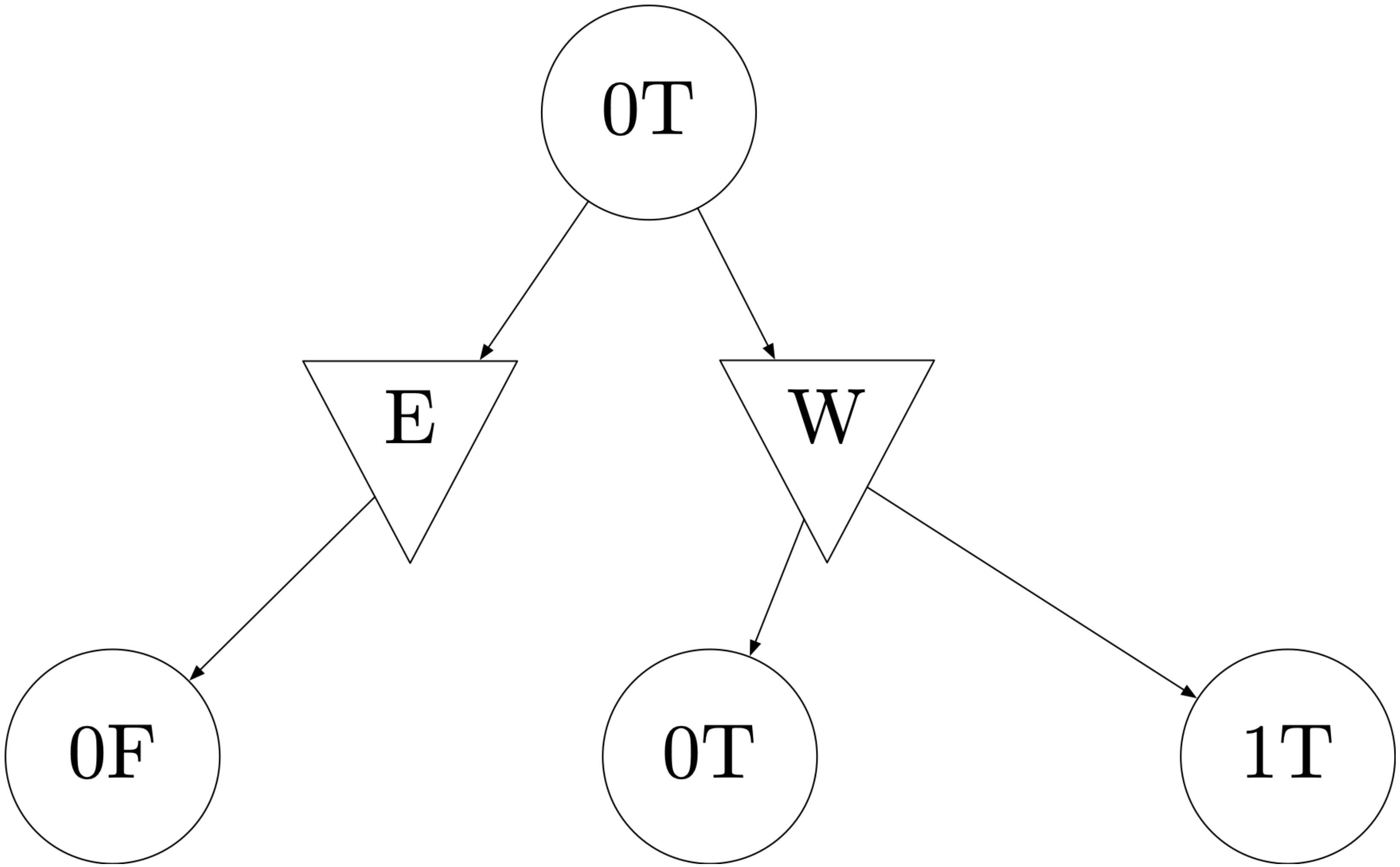
All Outcomes Determinization

$$\langle(\mathcal{S}, \mathcal{A}, P, R, \mathcal{S}_{\mathcal{G}}), s_0\rangle \xrightarrow{\text{AO determinize}} (\mathcal{S}, \mathcal{A}', \mathcal{S}_{\mathcal{G}}, s_0, f, c)$$

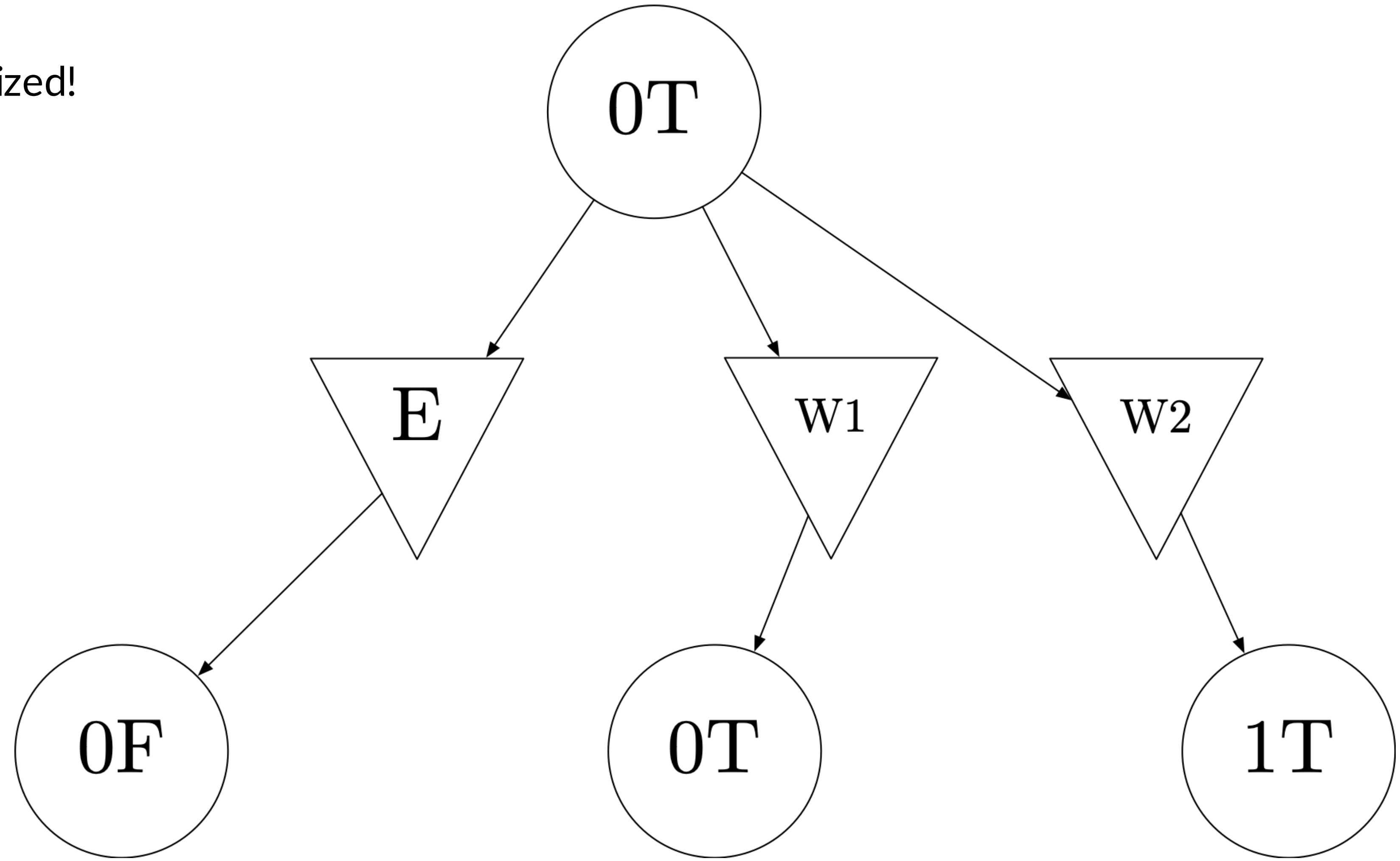
where:

- for each $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$ s.t. $P(s' \mid s, a) > 0$, there is an action $a' \in \mathcal{A}'$ s.t. $f(s, a') = s'$ and $c(s, a', s') = -R(s, a, s')$.

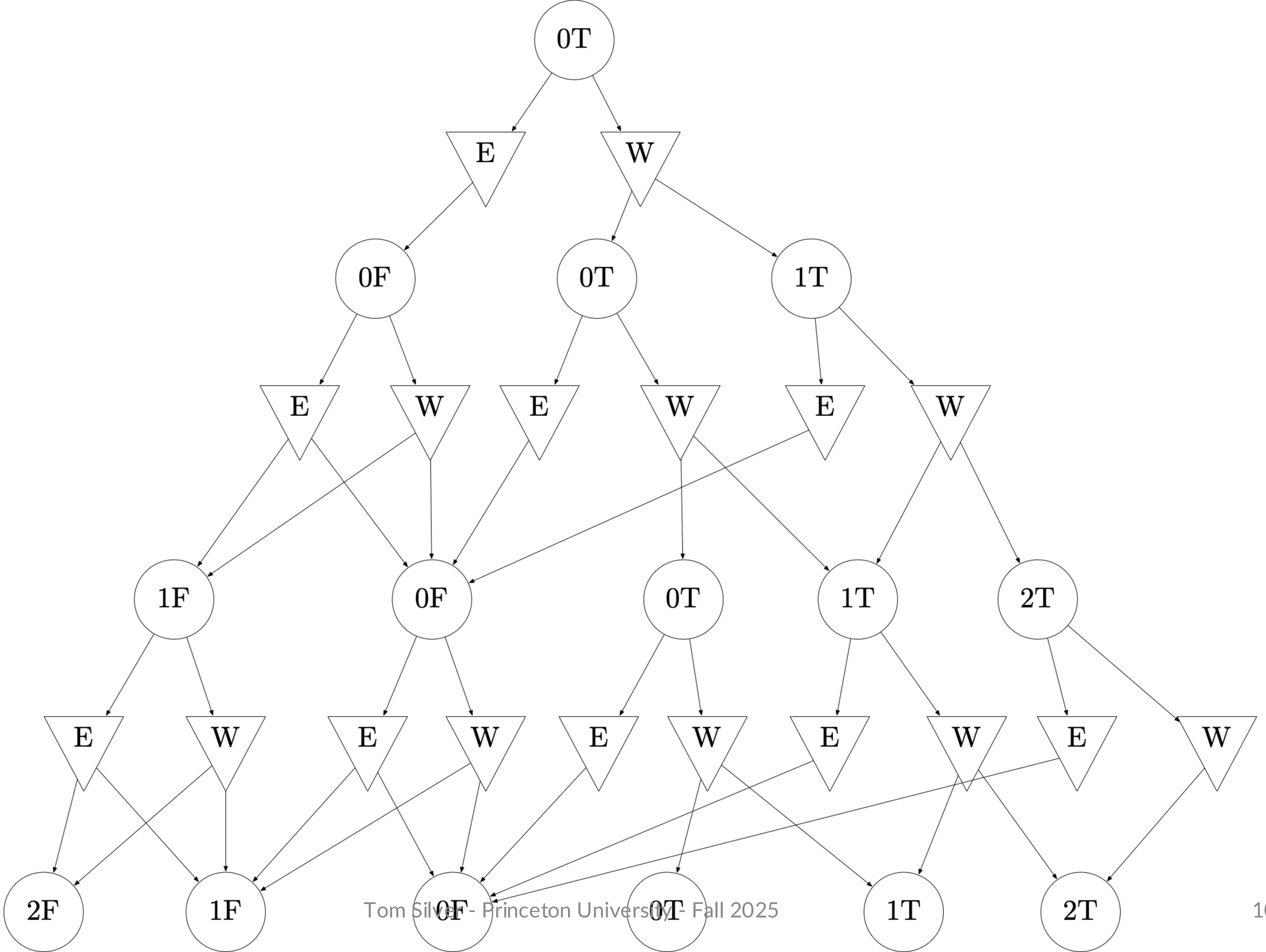
Original



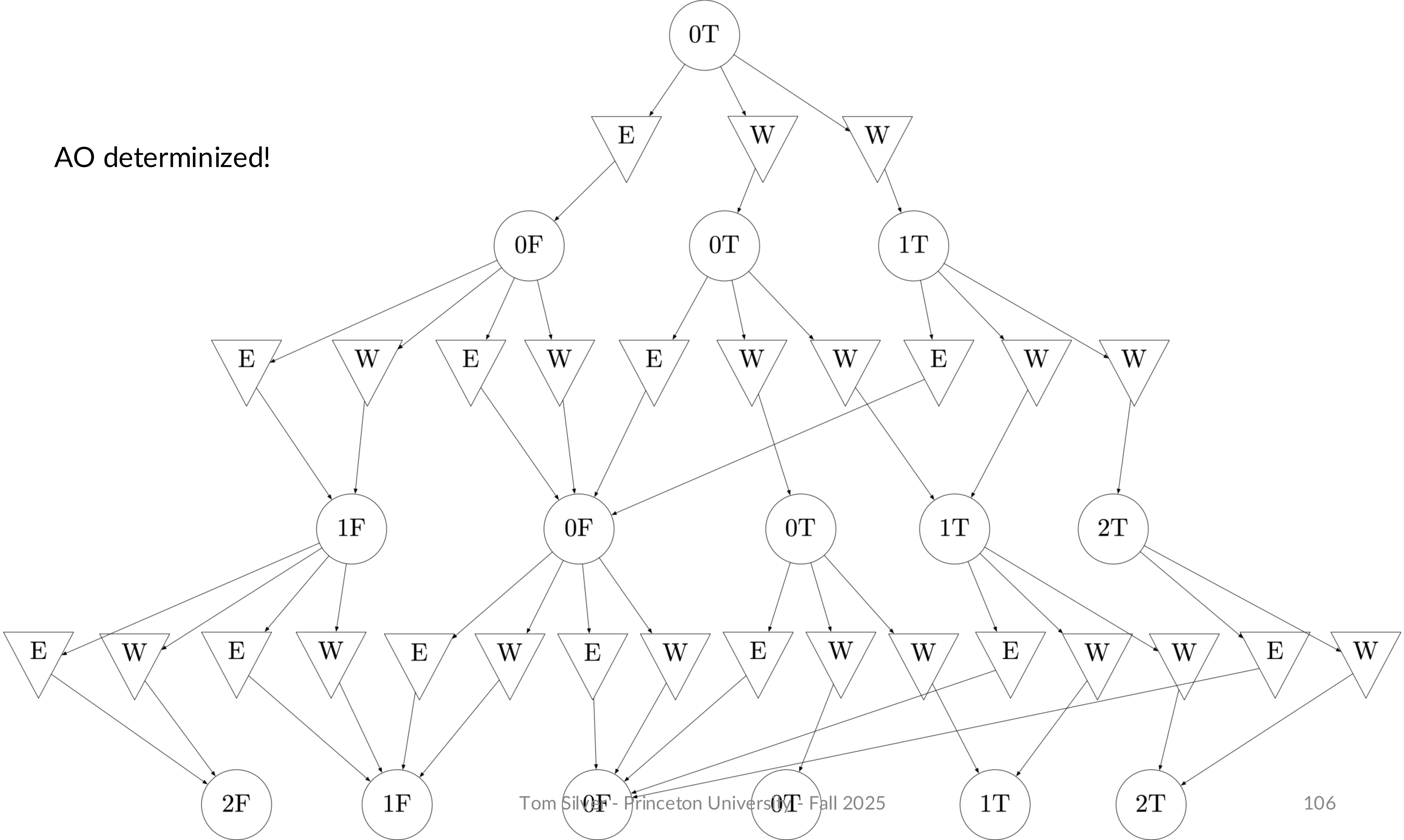
AO determinized!



Original



AO determinized!



All Outcomes Determinization

$$\langle(\mathcal{S}, \mathcal{A}, P, R, \mathcal{S}_{\mathcal{G}}), s_0\rangle \xrightarrow{\text{AO determinize}} (\mathcal{S}, \mathcal{A}', \mathcal{S}_{\mathcal{G}}, s_0, f, c)$$

where:

- for each $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$ s.t. $P(s' \mid s, a) > 0$, there is an action $a' \in \mathcal{A}'$ s.t. $f(s, a') = s'$ and $c(s, a', s') = -R(s, a, s')$.

When might this go badly?

Determinization

Other strategies convert transition probabilities into rewards, so the agent is discouraged from pursuing unlikely paths.

Summary

- Known current state? Only some states reachable.
- How to leverage reachability?
  - Finite horizon: expectimax search.
  - Infinite/indefinite horizon: reachability abstraction, or receding horizon control + expectimax search
- How to leverage heuristics?
  - Expectimax + heuristic leaf evals
- How to avoid exhaustive tree building?
  - RTDP
  - Determinization

Next Time

- Avoiding big Bellman backups (without determinizing)
- Scaling to larger state spaces
- Using sampling-based techniques