Foundations of Hardware Verification

John Wickerson

14 December 2023

This lecture

- Temporal logic
- Binary decision diagrams

Temporal logic

- SymbiYosys can verify *immediate* assertions like:

```
assert (count > 0);
```

- It can also handle *concurrent* assertions like:

```
assert (wen > 0 |=> waddr > 0);
```

- These can be generalised to formulas in *temporal logic*.

Linear Temporal Logic

ϕ ::= true
| false
| p // atom
| $\neg\,\phi$ // negation
| $\phi \wedge \phi$ // conjunction
| $\phi \vee \phi$ // disjunction
| $\phi \rightarrow \phi$ // implication
| **X** ϕ // ne**X**t state
| **F** ϕ // some **F**uture state
| **G** ϕ // all future states (**G**lobally)
| ϕ **U** ϕ // **U**ntil
| ϕ **W** ϕ // **W**eak-until

count > 0

count > 0 $\rightarrow$ waddr $\neq$ raddr

en $\rightarrow$ **X** (count > 0)

go $\rightarrow$ **F** (heat)

dead $\rightarrow$ **G** (dead)

G (req $\rightarrow$ **F** (ack))

liftmoving **U** (floor = 5)

motor **W** targetreached

Amir Pnueli
(1941–2009)

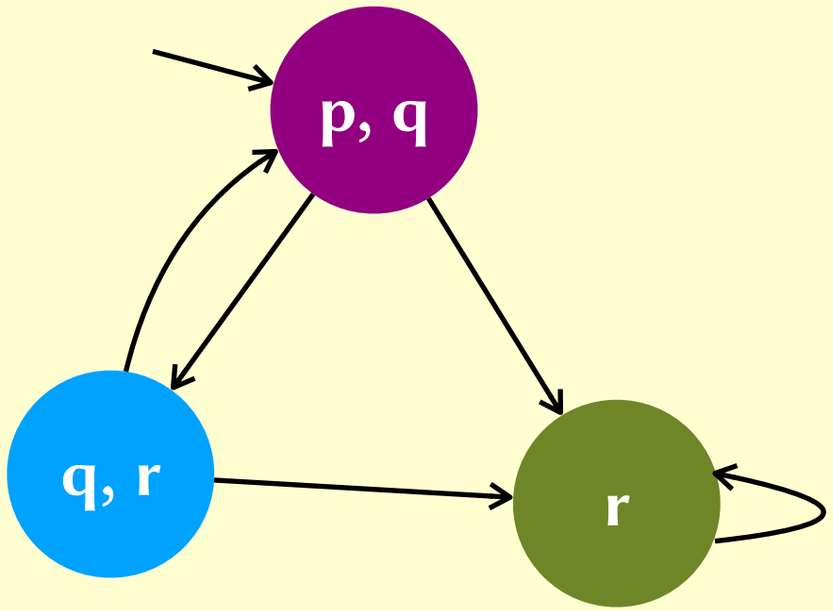
Paths

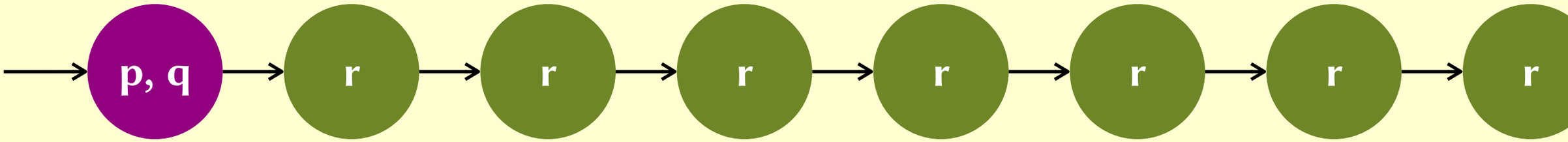
$p \rightarrow q$ ✓

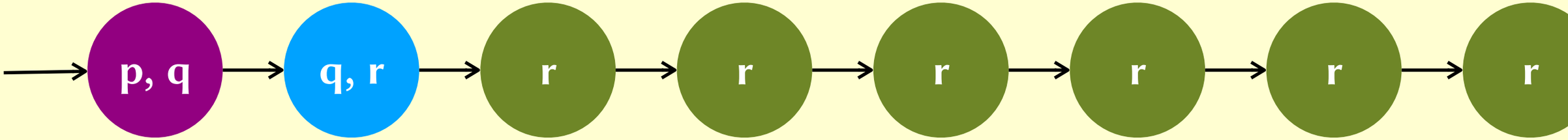
$\mathbf{G}\ (p \rightarrow q)$ ✓

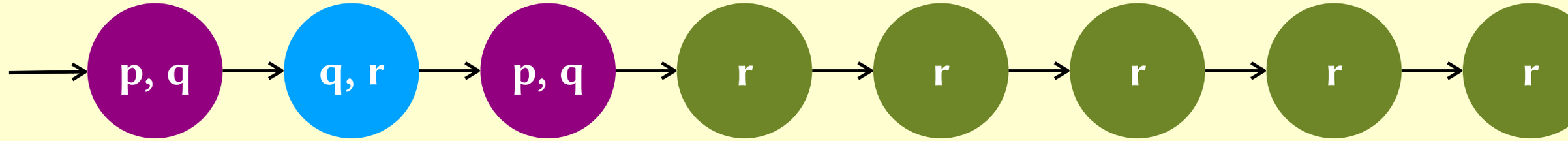
$\mathbf{X}\ r$ ✓ $\mathbf{F}\ r$ ✓ $\mathbf{G}\ \mathbf{F}\ r$ ✓

$\mathbf{F}\ (\neg q)$ ✗ $q\ \mathbf{U}\ r$ ✓ $q\ \mathbf{U}\ \neg q$ ✗ $q\ \mathbf{W}\ \neg q$ ✓



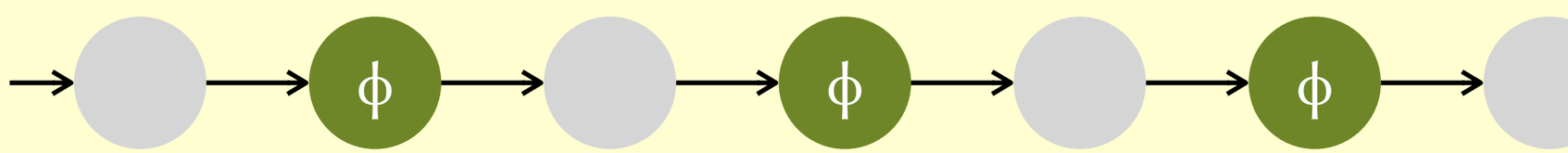






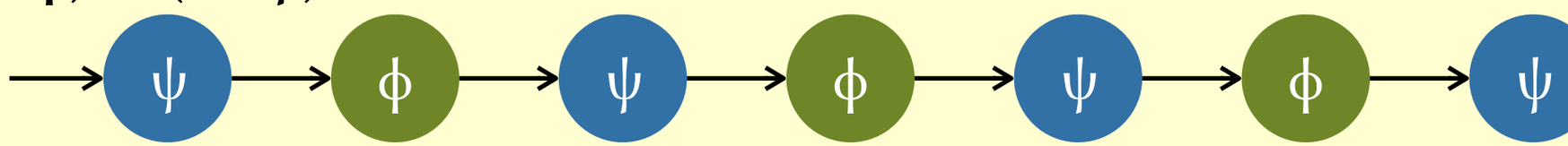
Some identities

- $\mathbf{G}\,\phi \overset{\checkmark}{\equiv} \neg\,\mathbf{F}\,(\neg\,\phi)$
- $\phi\,\mathbf{U}\,\psi \overset{\checkmark}{\equiv} (\phi\,\mathbf{W}\,\psi) \wedge (\mathbf{F}\,\psi)$
- $\phi\,\mathbf{W}\,\psi \overset{\checkmark}{\equiv} (\phi\,\mathbf{U}\,\psi) \vee (\mathbf{G}\,\phi)$
- $\mathbf{F}\,\mathbf{F}\,\phi \overset{\checkmark}{\equiv} \mathbf{F}\,\phi$
- $\mathbf{G}\,\mathbf{G}\,\phi \overset{\checkmark}{\equiv} \mathbf{G}\,\phi$
- $\mathbf{G}\,\mathbf{F}\,\phi \overset{\times}{\equiv} \mathbf{F}\,\mathbf{G}\,\phi$

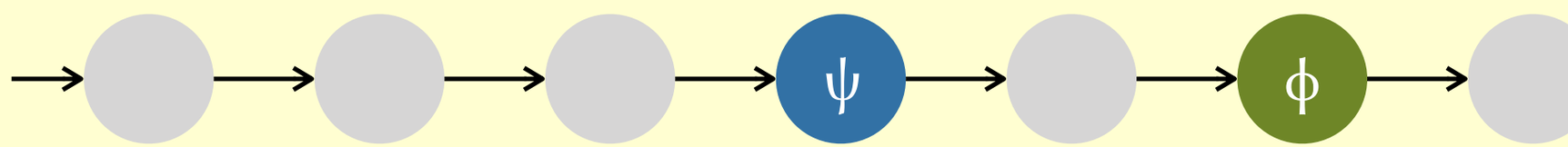


Some more identities

- $\mathbf{F}\,(\phi \vee \psi) \overset{\checkmark}{\equiv} (\mathbf{F}\,\phi) \vee (\mathbf{F}\,\psi)$
- $\mathbf{G}\,(\phi \vee \psi) \overset{\times}{\equiv} (\mathbf{G}\,\phi) \vee (\mathbf{G}\,\psi)$



- $\mathbf{F}\,(\phi \wedge \psi) \overset{\times}{\equiv} (\mathbf{F}\,\phi) \wedge (\mathbf{F}\,\psi)$



- $\mathbf{G}\,(\phi \wedge \psi) \overset{\checkmark}{\equiv} (\mathbf{G}\,\phi) \wedge (\mathbf{G}\,\psi)$

Temporal logic

- SymbiYosys can verify *immediate* assertions like:

```
assert (count > 0);
```

$\mathbf{G}\ (\text{count} > 0)$

- It can also handle *concurrent* assertions like:

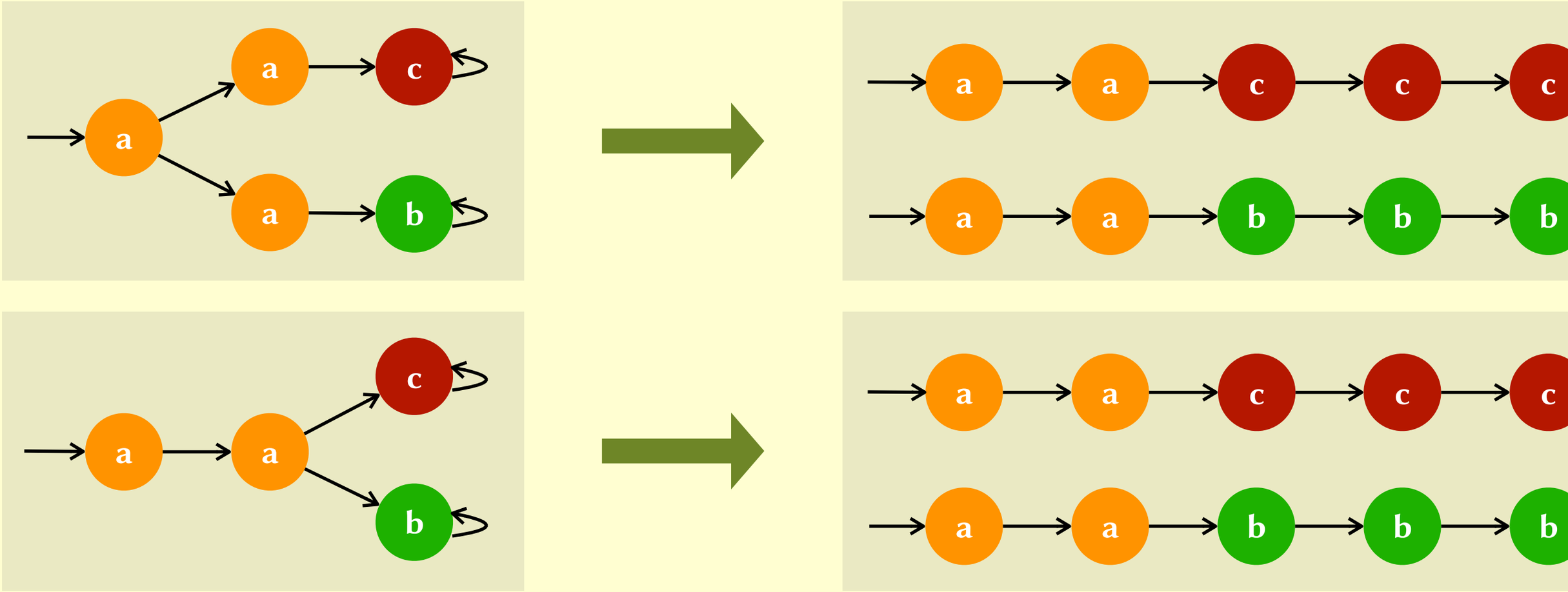
```
assert (wen > 0 |=> waddr > 0);
```

$\mathbf{G}\ (\text{wen} > 0 \longrightarrow \mathbf{X}\ (\text{waddr} > 0))$

- These can be generalised to formulas in *temporal logic*.

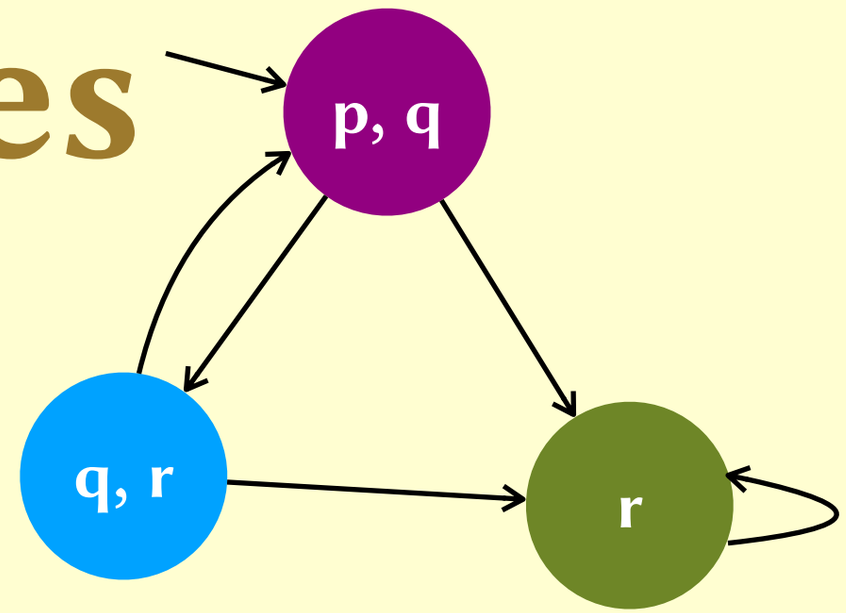
LTL's downfall...

- "From any a state, we can reach a c state."

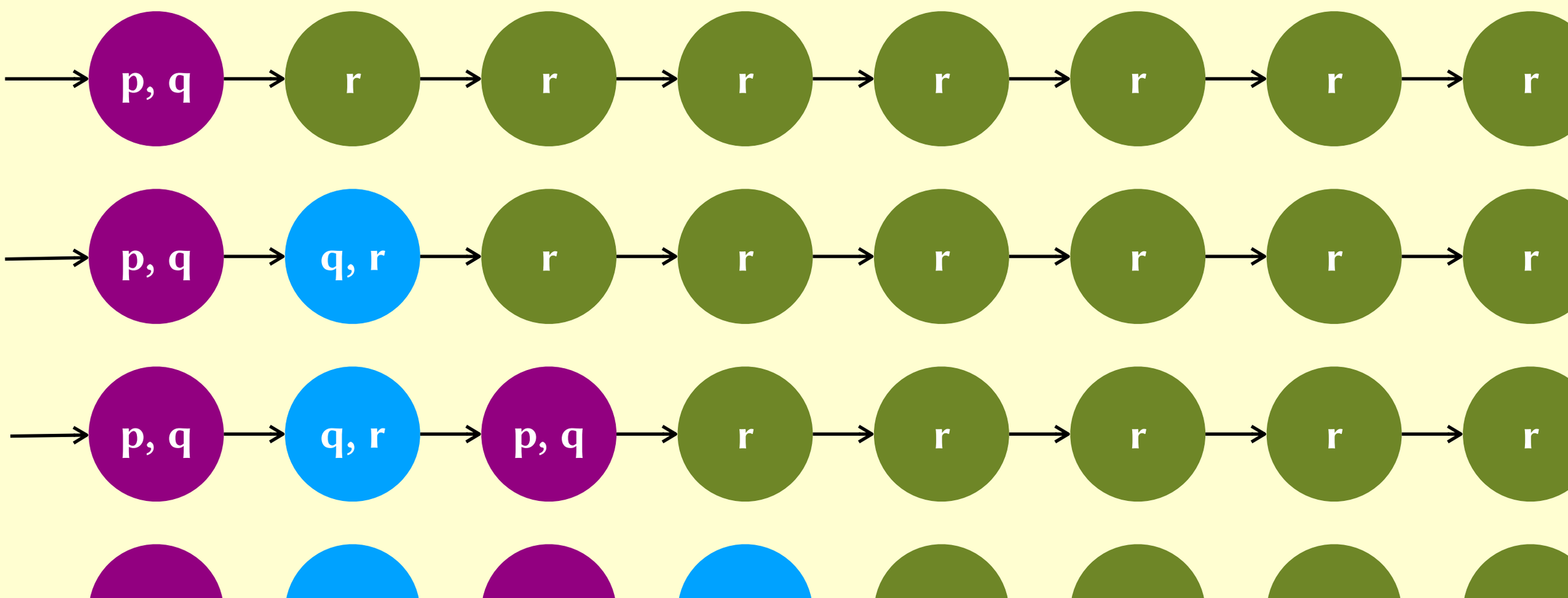


- Not expressible in LTL.

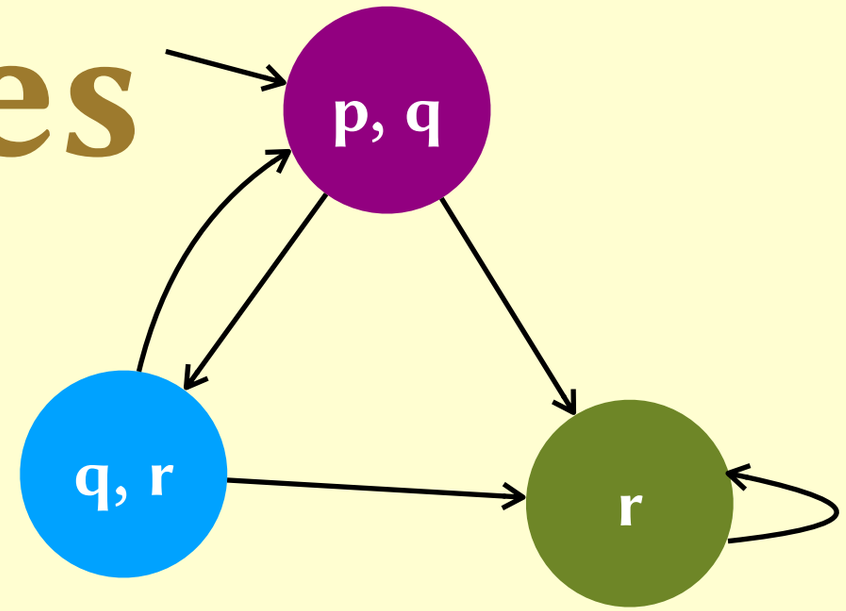
From paths to trees



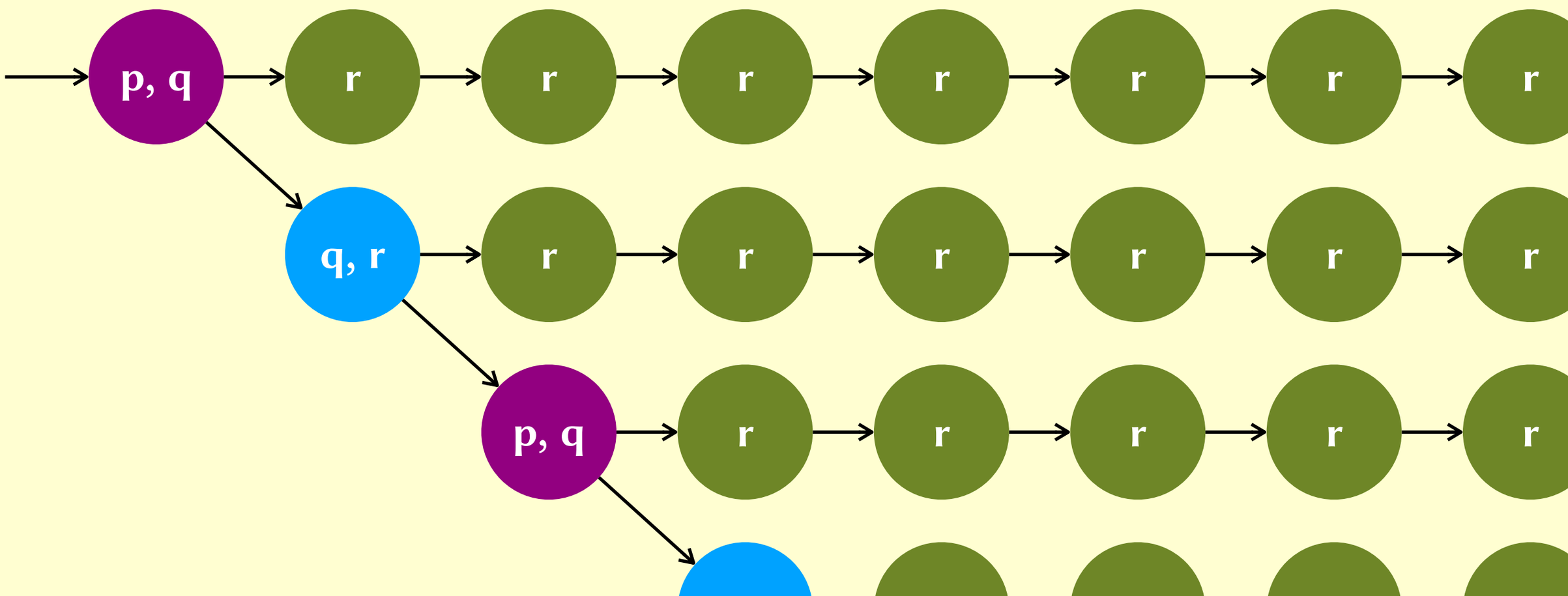
- Rather than unfolding the transition system into a set of paths...



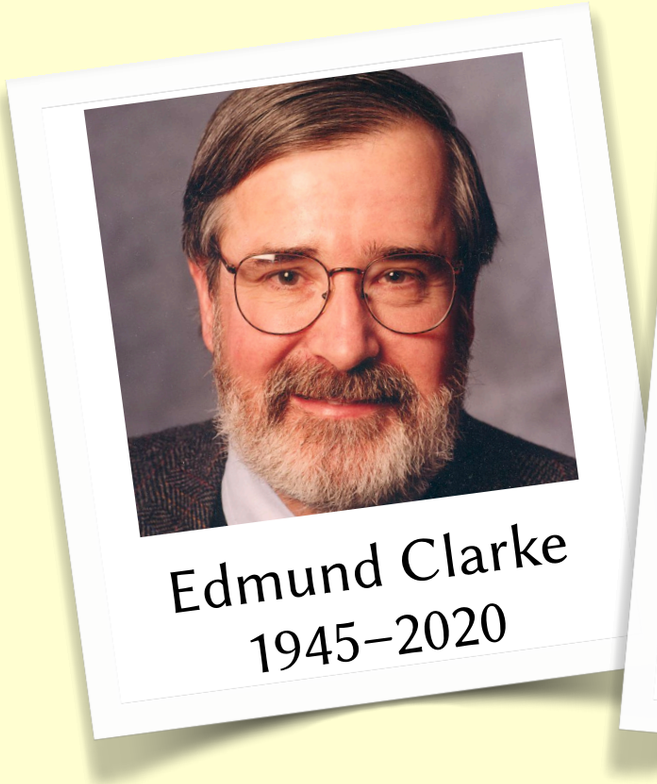
From paths to trees



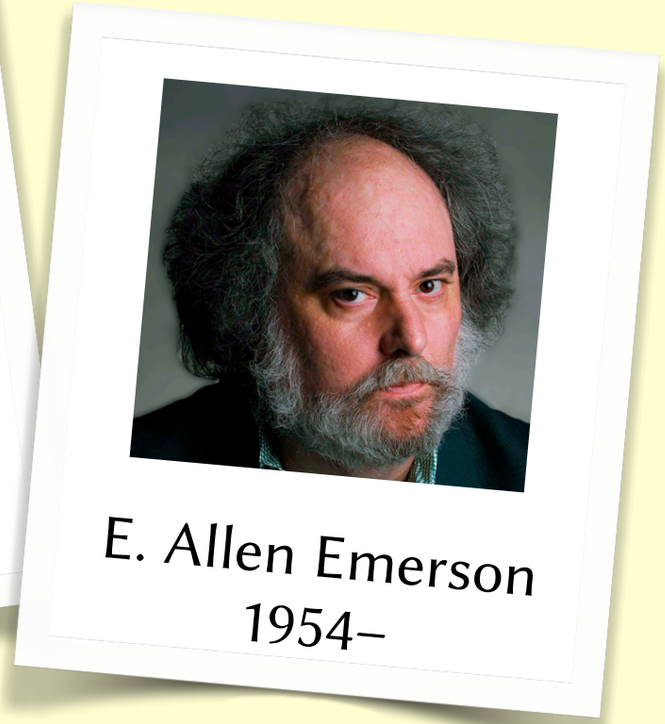
- ...unfold it into a single tree.
- Now formulas talk about this tree, rather than individual paths.



Computational Tree Logic



Edmund Clarke
1945–2020

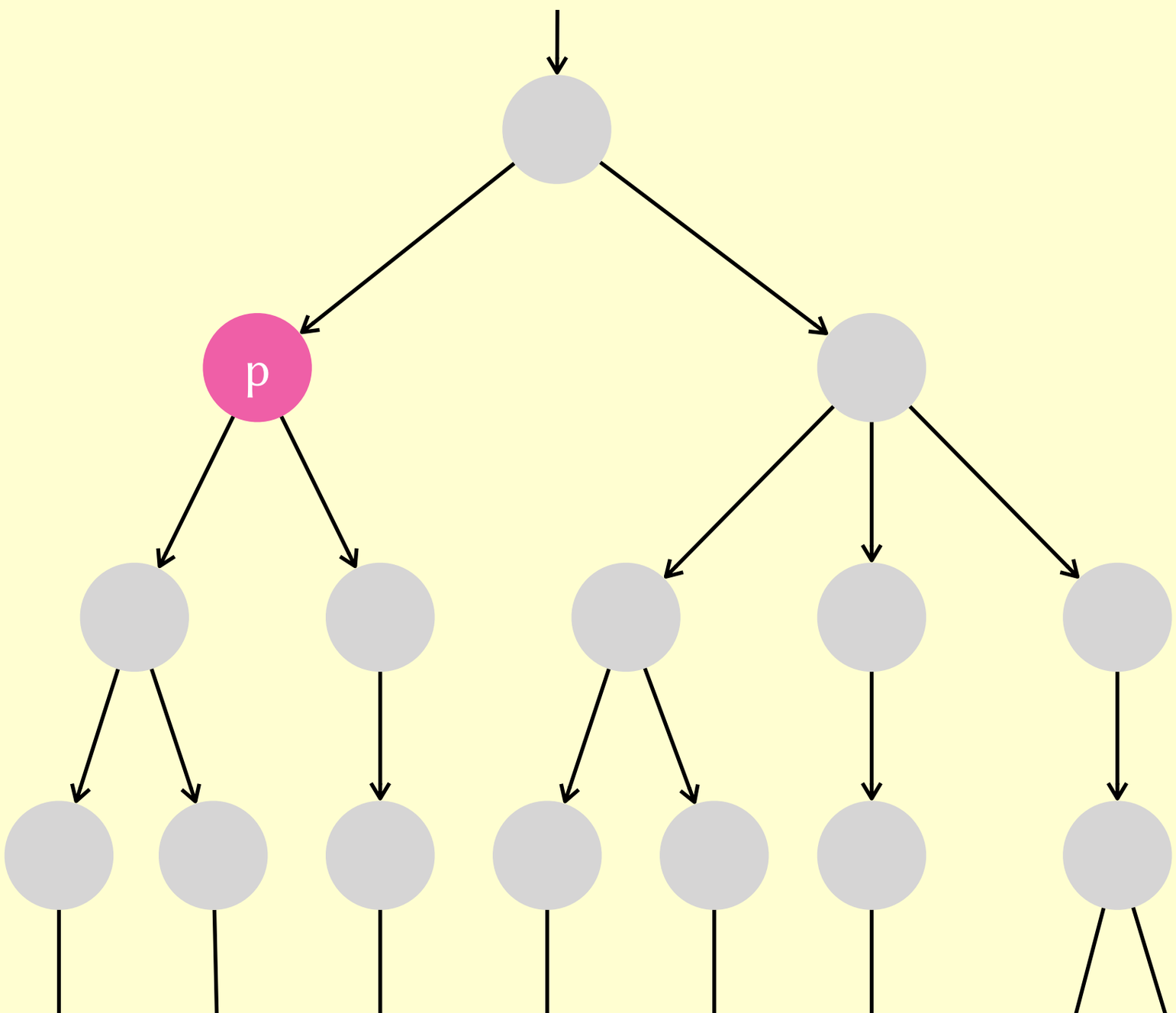


E. Allen Emerson
1954–

ϕ ::= true
| false
| p // atom
| $\neg\,\phi$ // negation
| $\phi \wedge \phi$ // conjunction
| $\phi \vee \phi$ // disjunction
| $\phi \rightarrow \phi$ // implication
| **AX** ϕ | **EX** ϕ // ne**X**t state (**A**ll paths / there **E**xists a path)
| **AF** ϕ | **EF** ϕ // some **F**uture state
| **AG** ϕ | **EG** ϕ // all future states (**G**lobally)
| **A**[ϕ **U** ϕ] | **E**[ϕ **U** ϕ] // **U**ntil

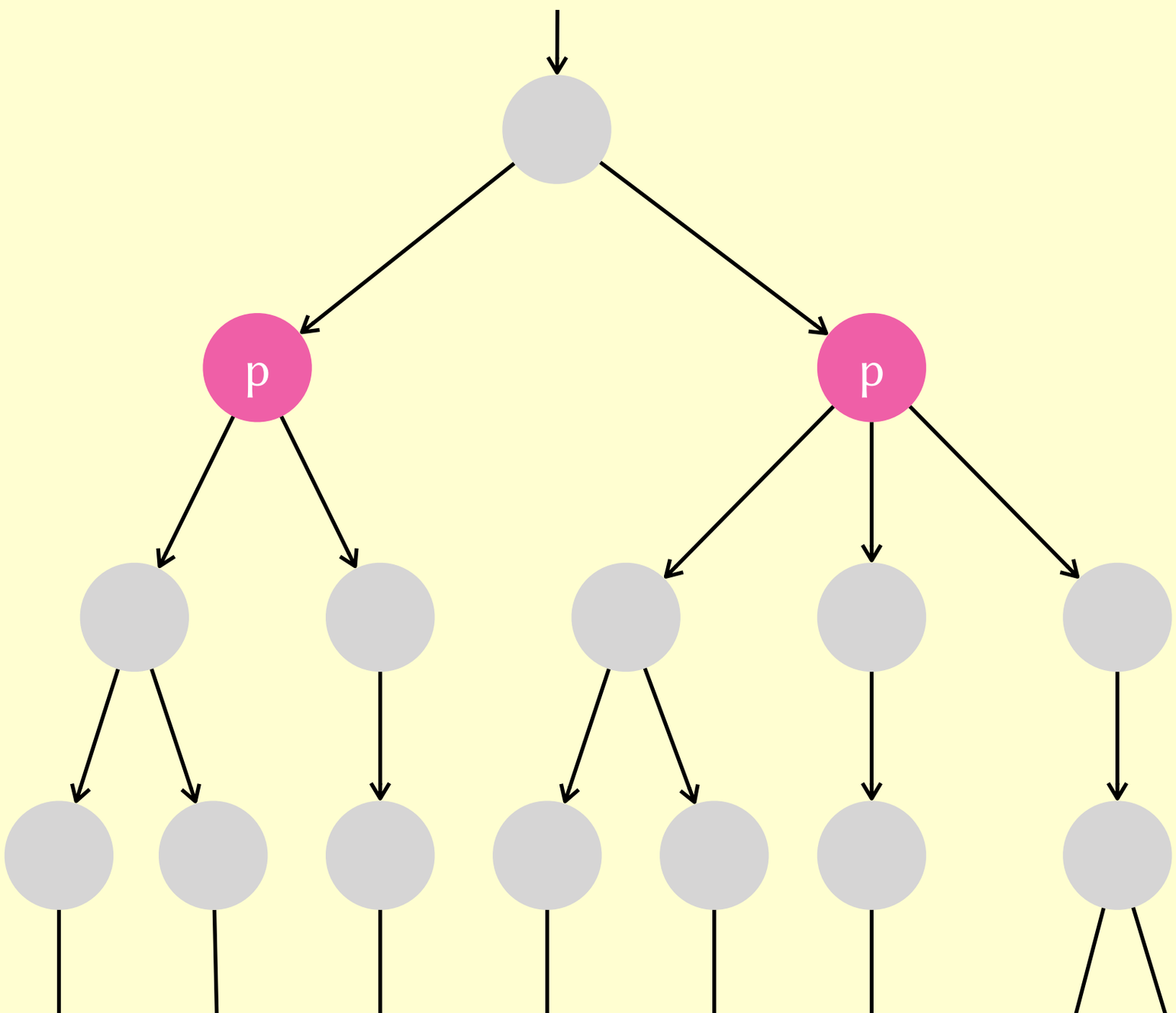
Computational Tree Logic

EX p

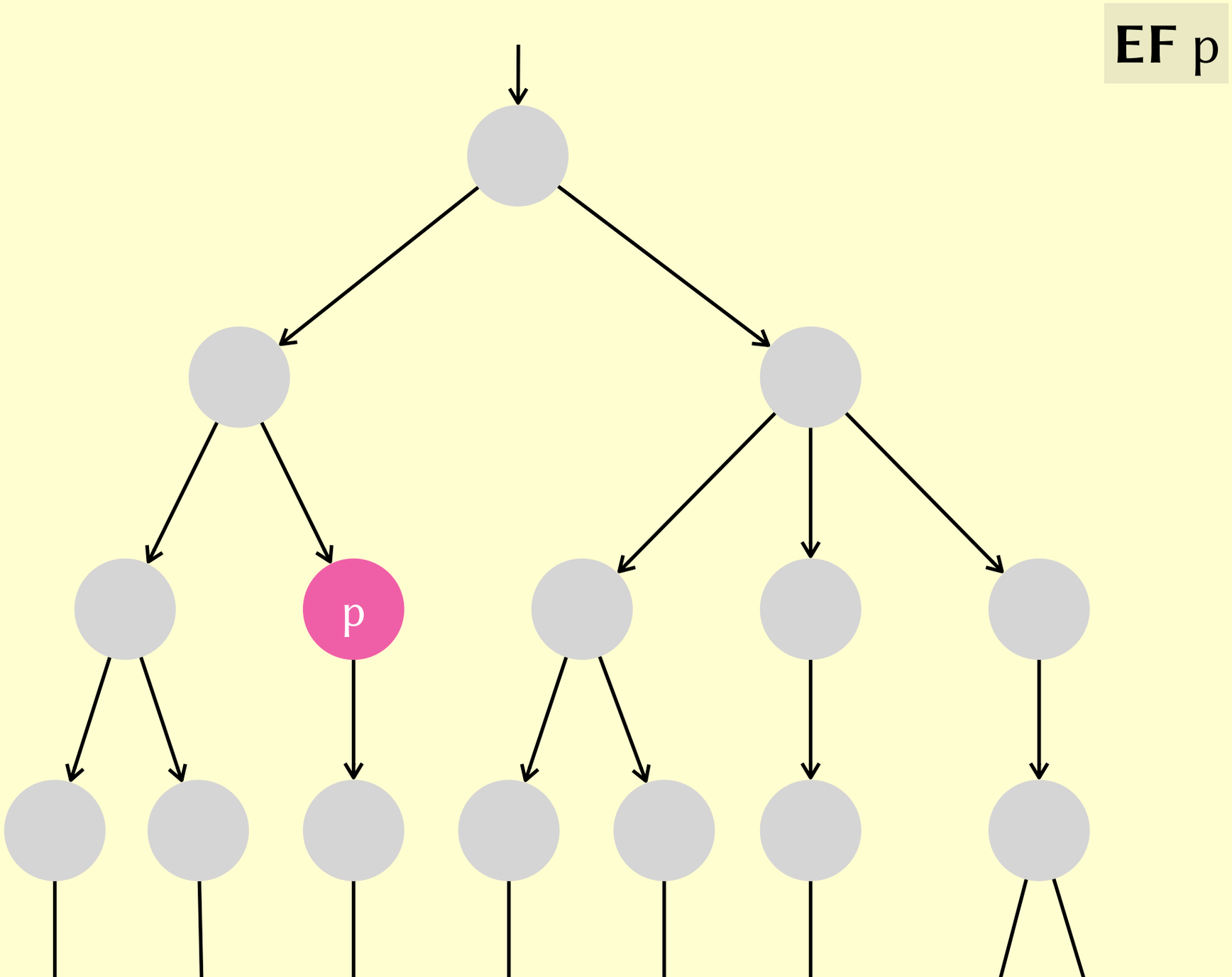


Computational Tree Logic

AX p

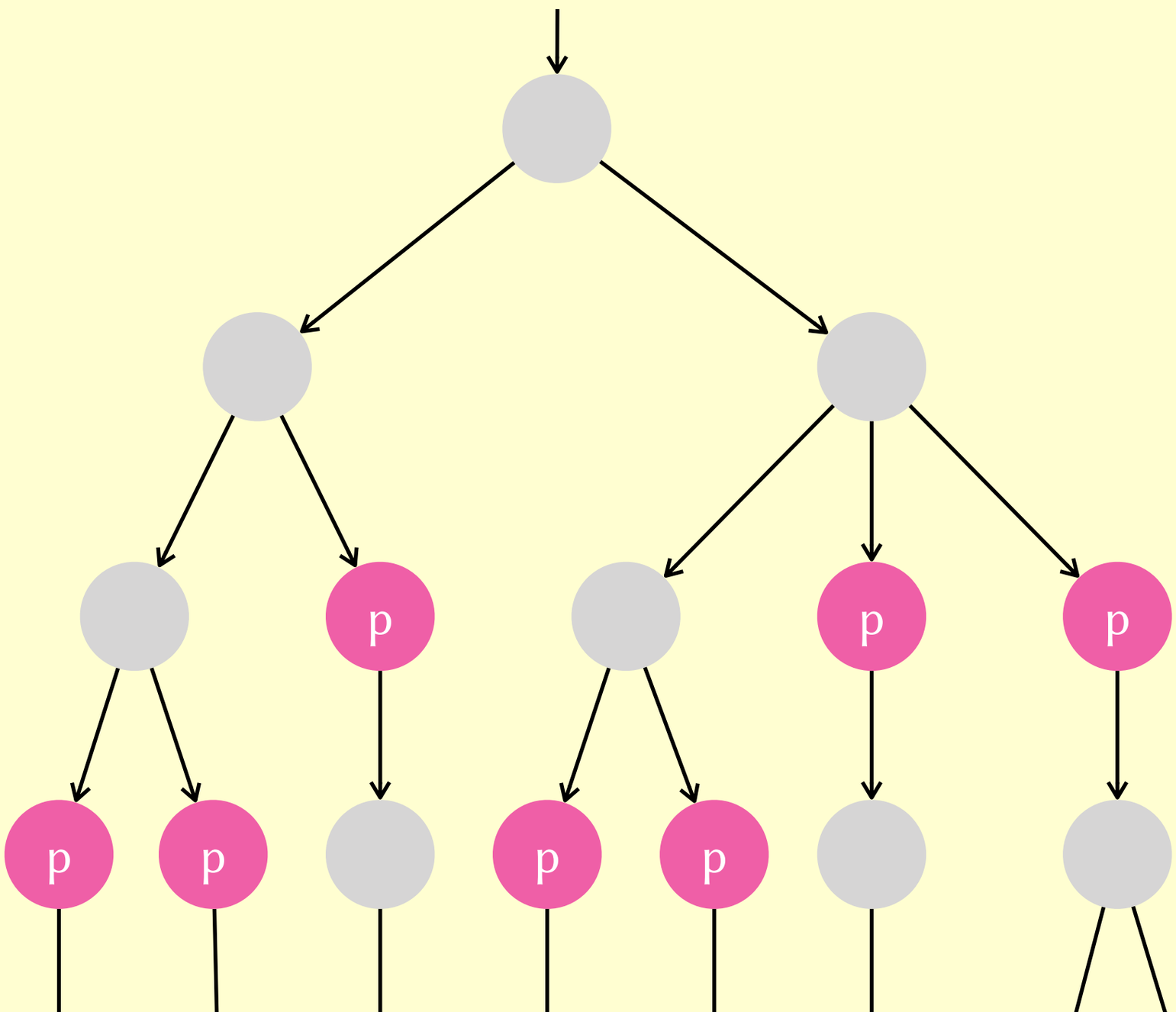


Computational Tree Logic



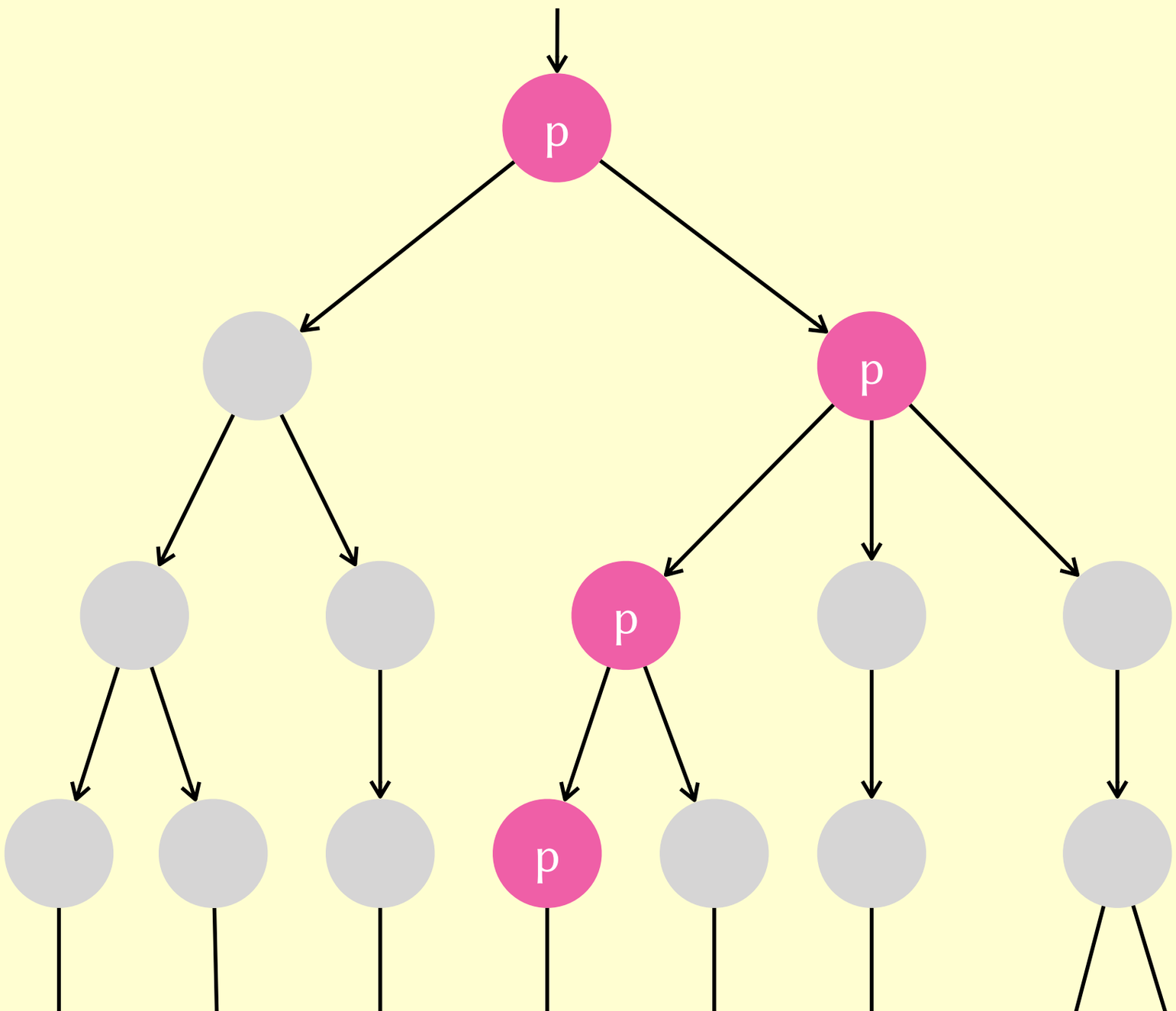
Computational Tree Logic

AF p



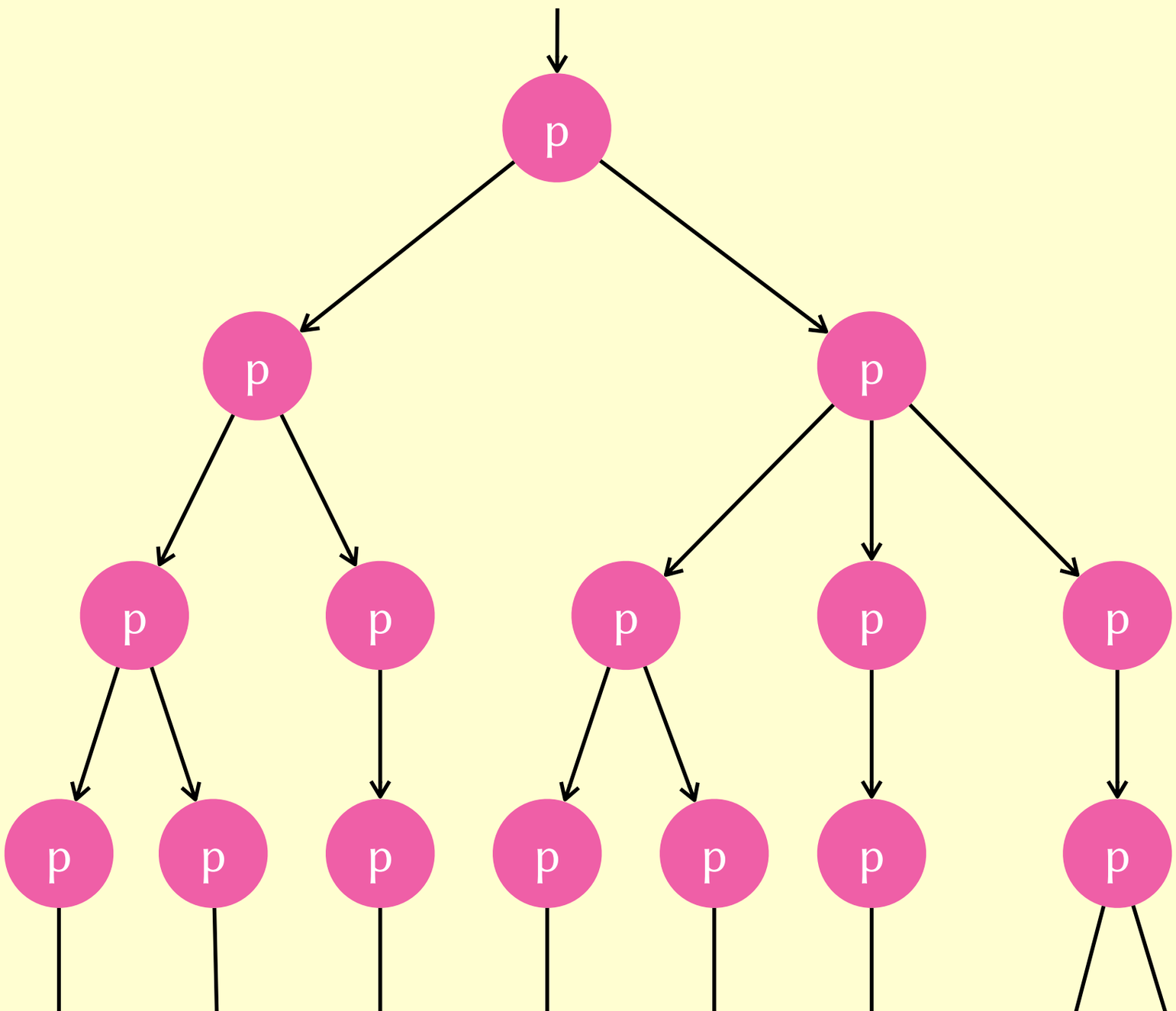
Computational Tree Logic

EG p



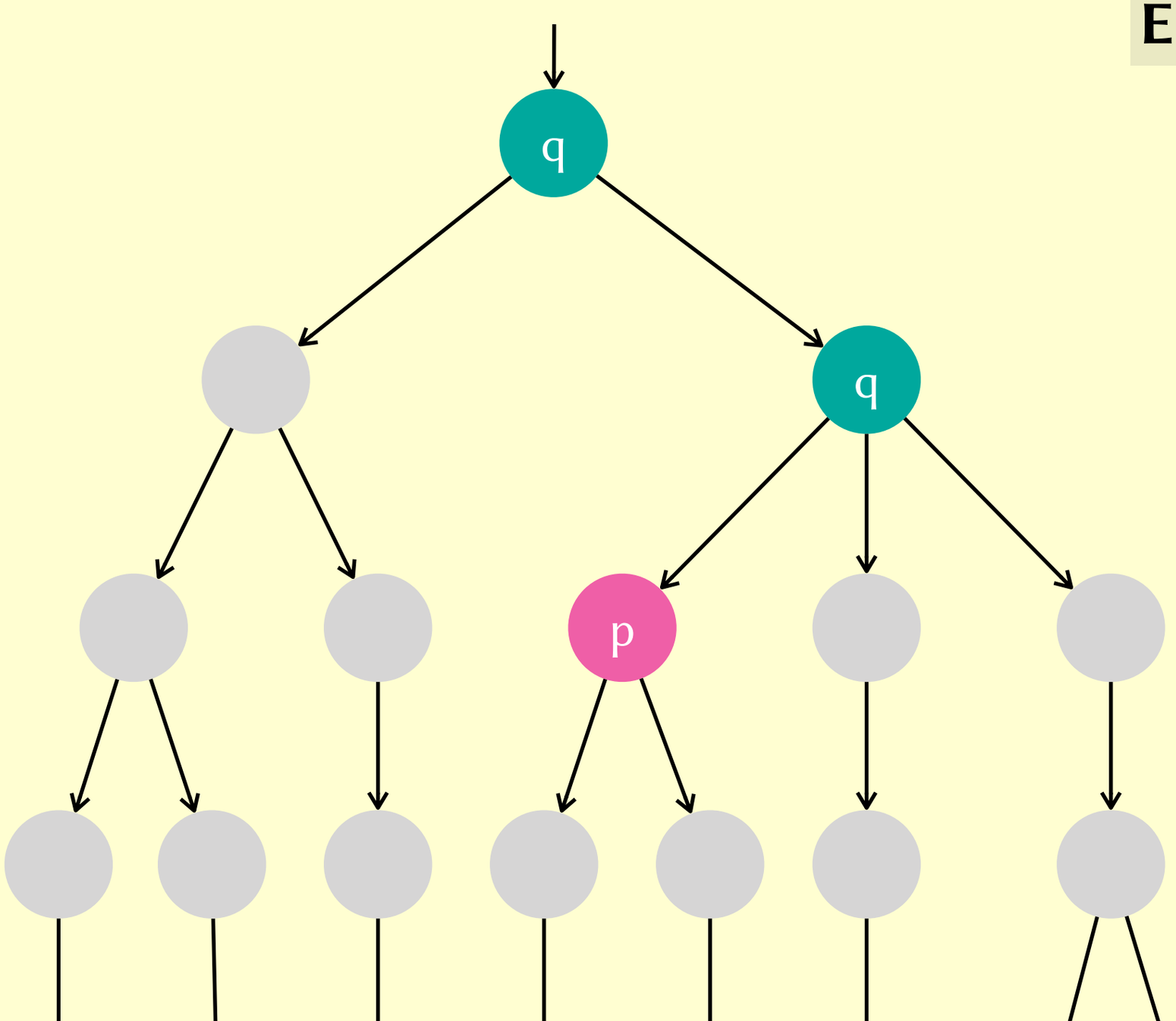
Computational Tree Logic

AG p



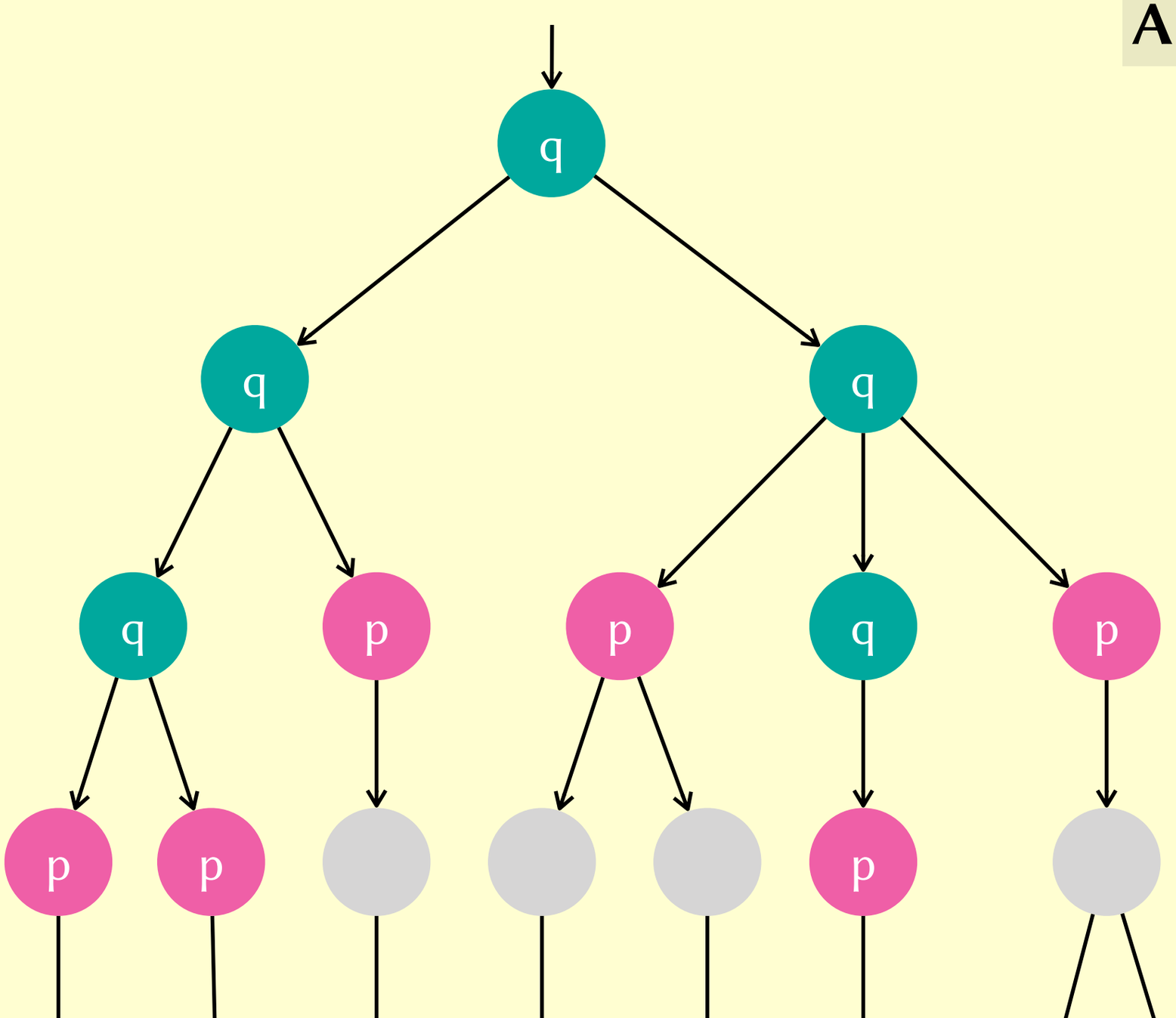
Computational Tree Logic

E[q **U** p]



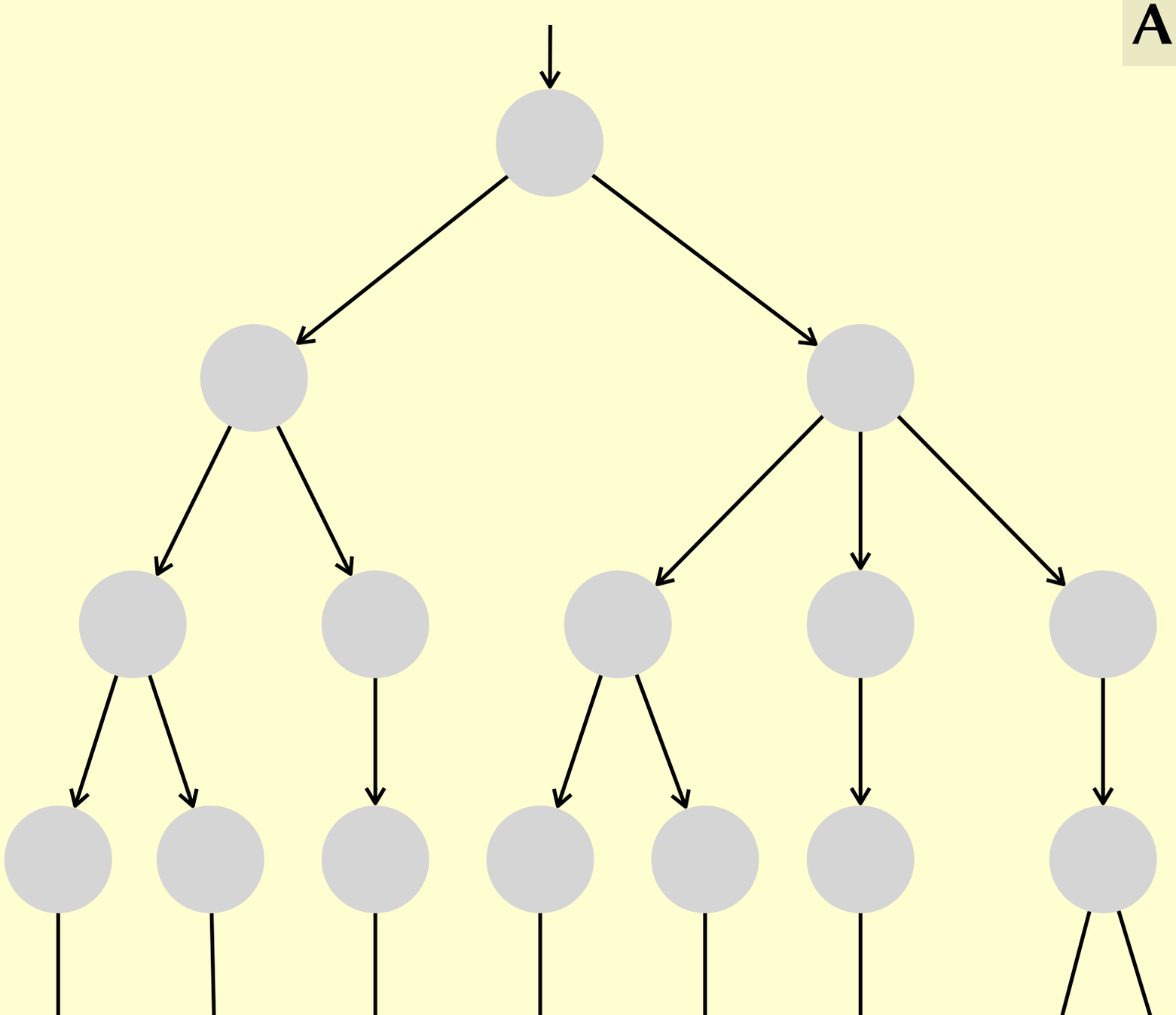
Computational Tree Logic

A[q **U** p]



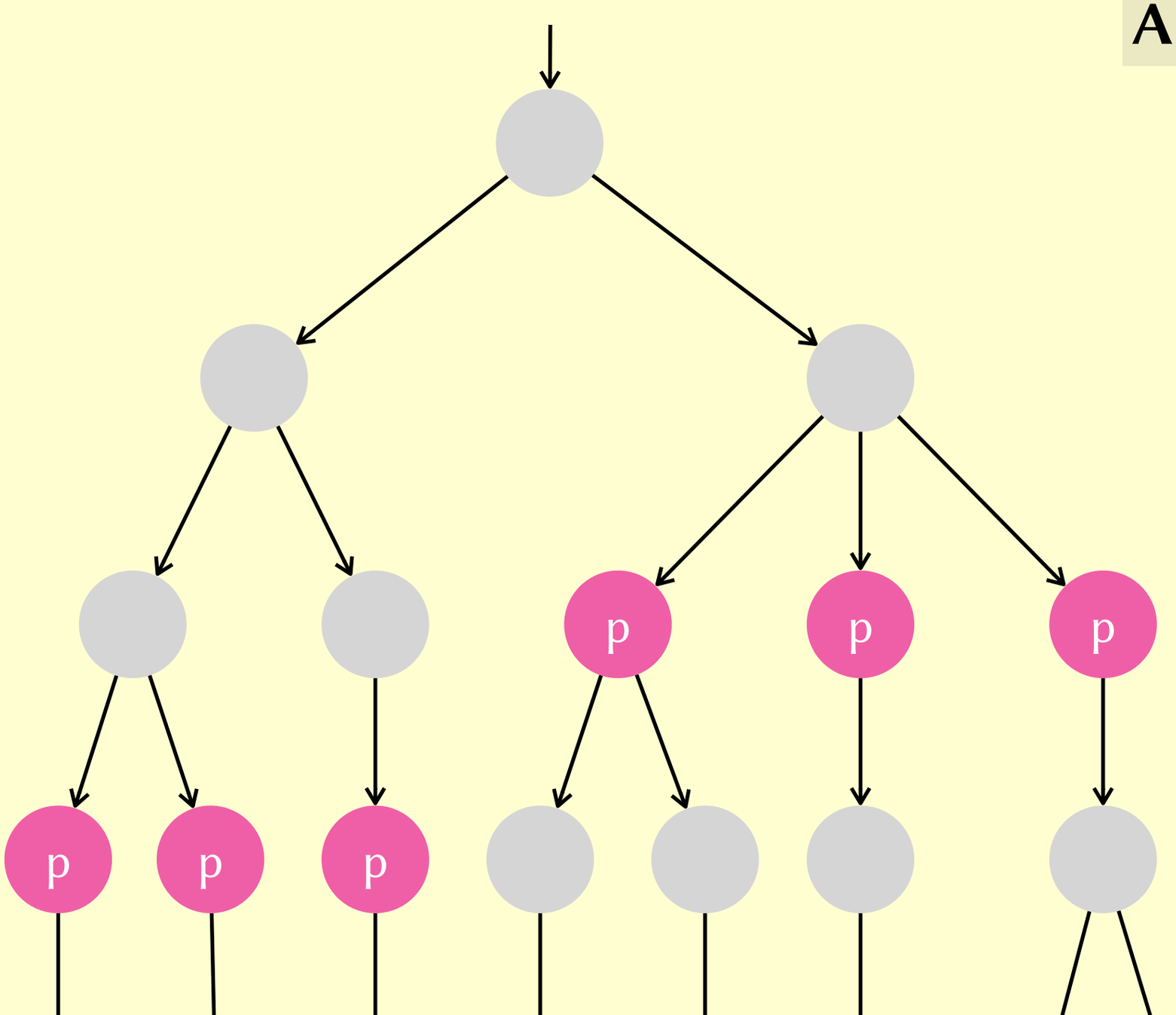
Computational Tree Logic

AF AX p



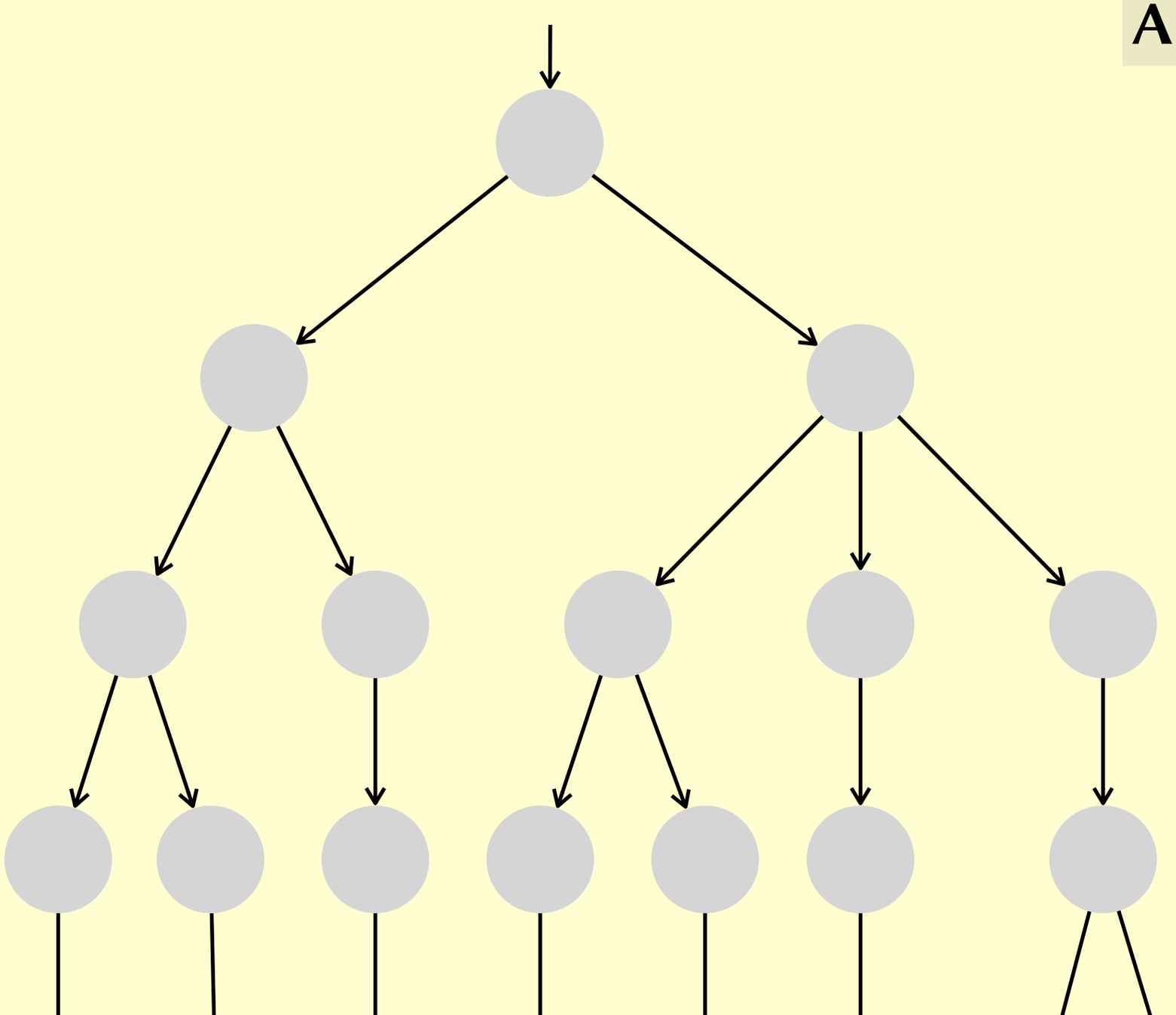
Computational Tree Logic

AF AX p



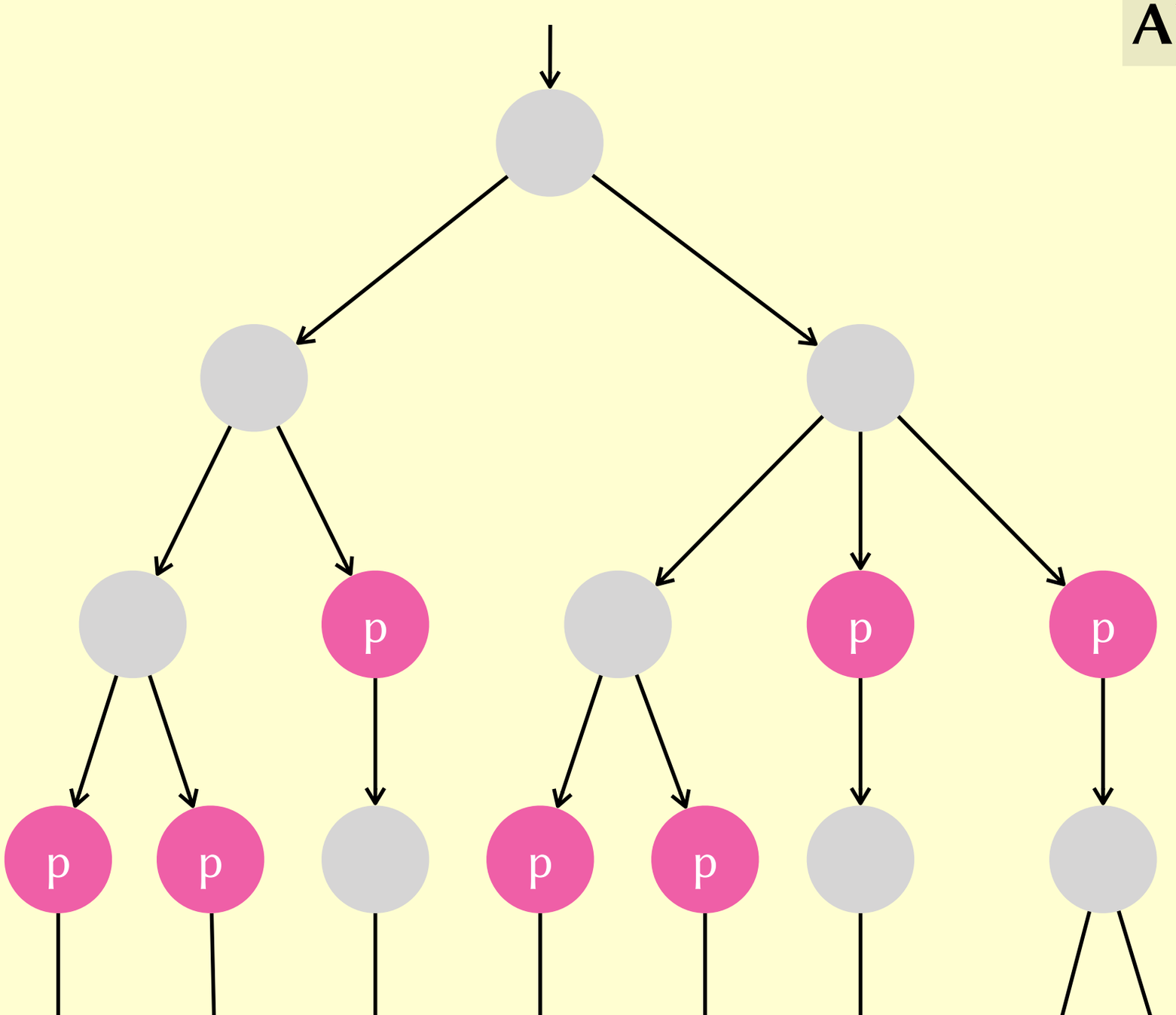
Computational Tree Logic

AX AF p



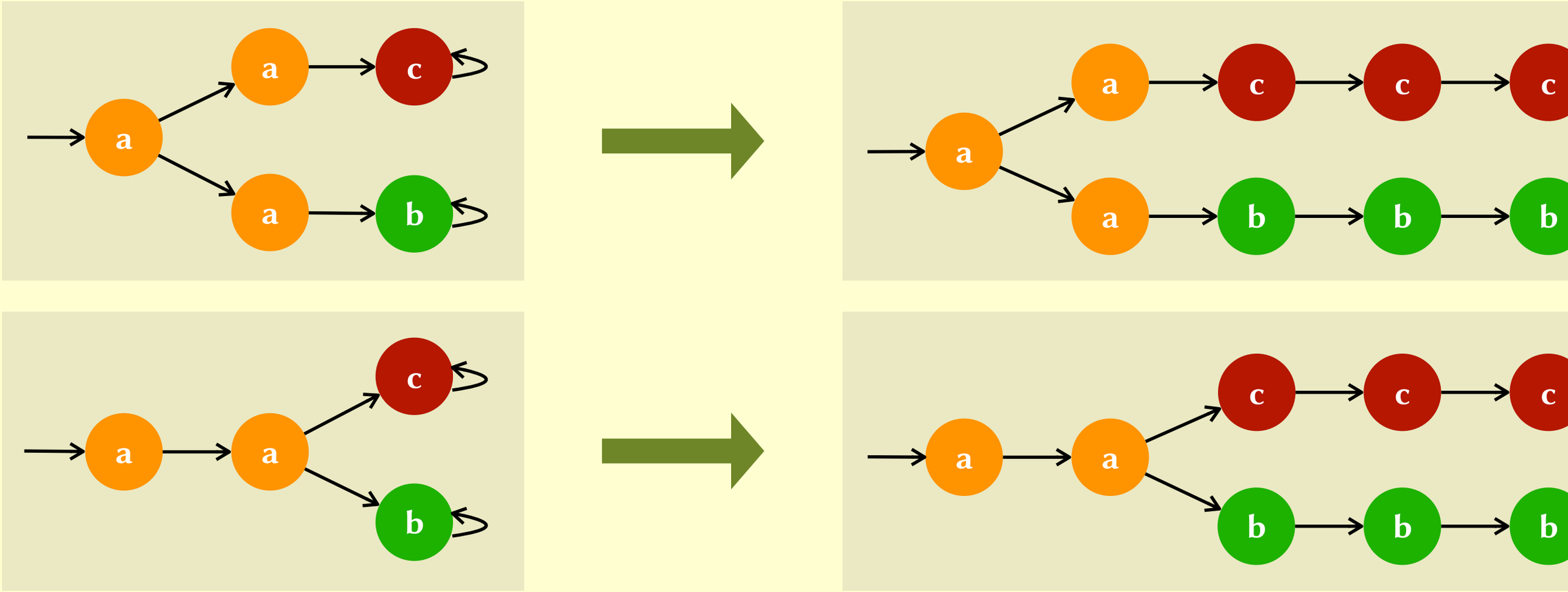
Computational Tree Logic

AX AF p



CTL to the rescue...

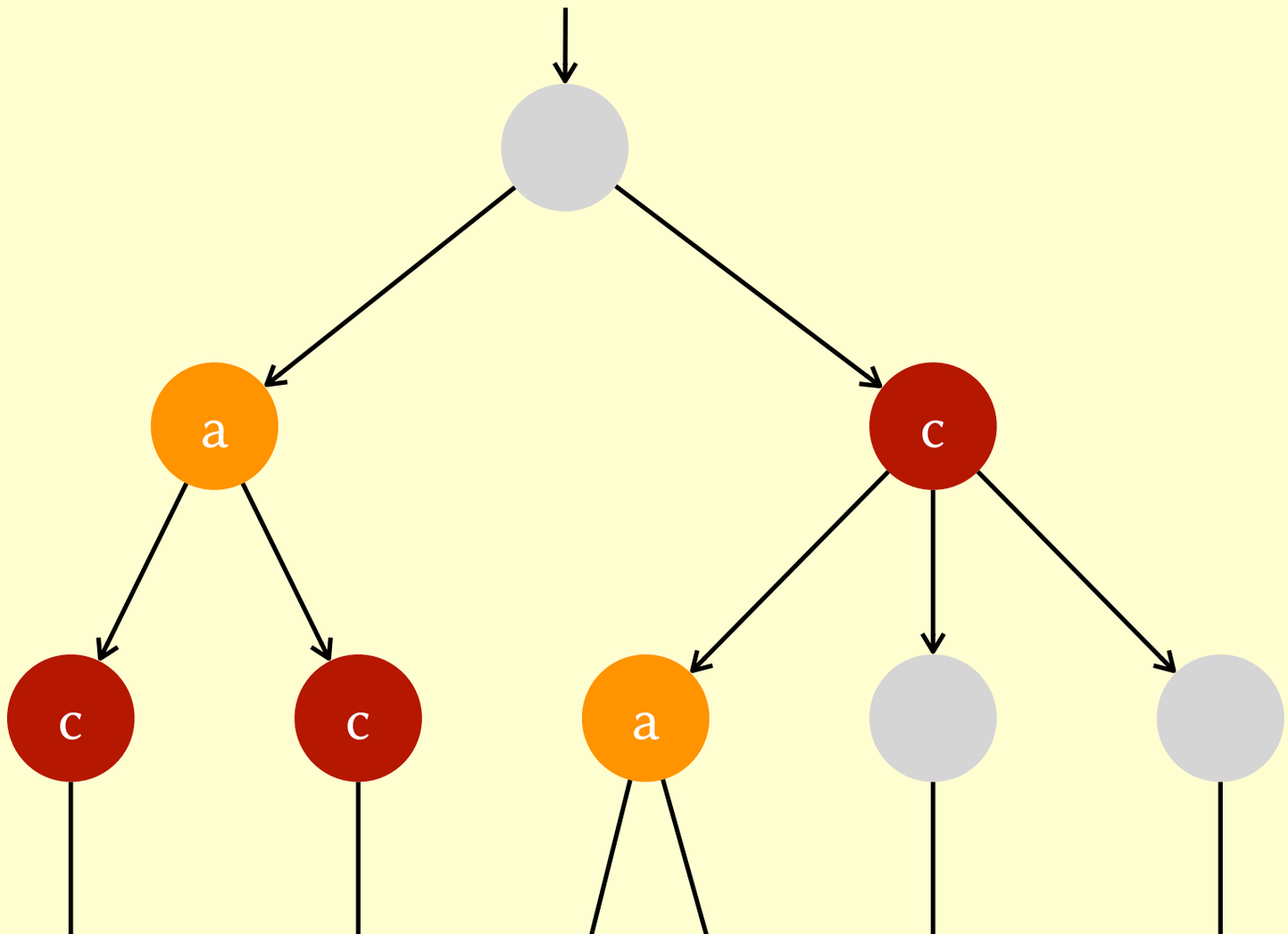
- "From any a state, we can reach a c state."



- **AG**(a $\longrightarrow$ **EF** c)

CTL's downfall...

- "All paths that contain an a also contain a c."
- (**AF** a) ⟶ (**AF** c) ✘
- **AF** (a ⟶ **EF** c) ✘
- **F** a ⟶ **F** c ✔



LTL vs CTL

- Some properties can be expressed in CTL but not LTL, e.g. "from every state, it is possible to reach an accepting state".
- Some properties can be expressed in LTL but not CTL, e.g. "every path that has a foo also has a baz".
- LTL formulas may be simpler to understand.
- CTL formulas may be more efficient to model-check.

CTL*

$$
\begin{aligned}
\phi ::= {} & \text{true} \\
\mid {} & \text{false} \\
\mid {} & p \\
\mid {} & \neg\, \phi \\
\mid {} & \phi \wedge \phi \\
\mid {} & \phi \vee \phi \\
\mid {} & \phi \rightarrow \phi \\
\mid {} & \mathbf{A}\ \alpha \\
\mid {} & \mathbf{E}\ \alpha
\end{aligned}
\qquad
\begin{aligned}
\alpha ::= {} & \phi \\
\mid {} & \neg\, \alpha \\
\mid {} & \alpha \wedge \alpha \\
\mid {} & \alpha \vee \alpha \\
\mid {} & \alpha \rightarrow \alpha \\
\mid {} & \mathbf{X}\ \alpha \\
\mid {} & \mathbf{F}\ \alpha \\
\mid {} & \mathbf{G}\ \alpha \\
\mid {} & \alpha\ \mathbf{U}\ \alpha
\end{aligned}
$$

EGF p

"There exists a path with infinitely many ps."

- Not actually used; just nice to know that LTL and CTL are both special cases of a more general temporal logic.

This lecture

- Temporal logic
- Binary decision diagrams

Boolean functions

- Could represent Boolean functions via their truth tables.
- The time taken to check f≡g is exponential in the number of variables.

A	B	C	f
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**0**
1	1	1	**0**

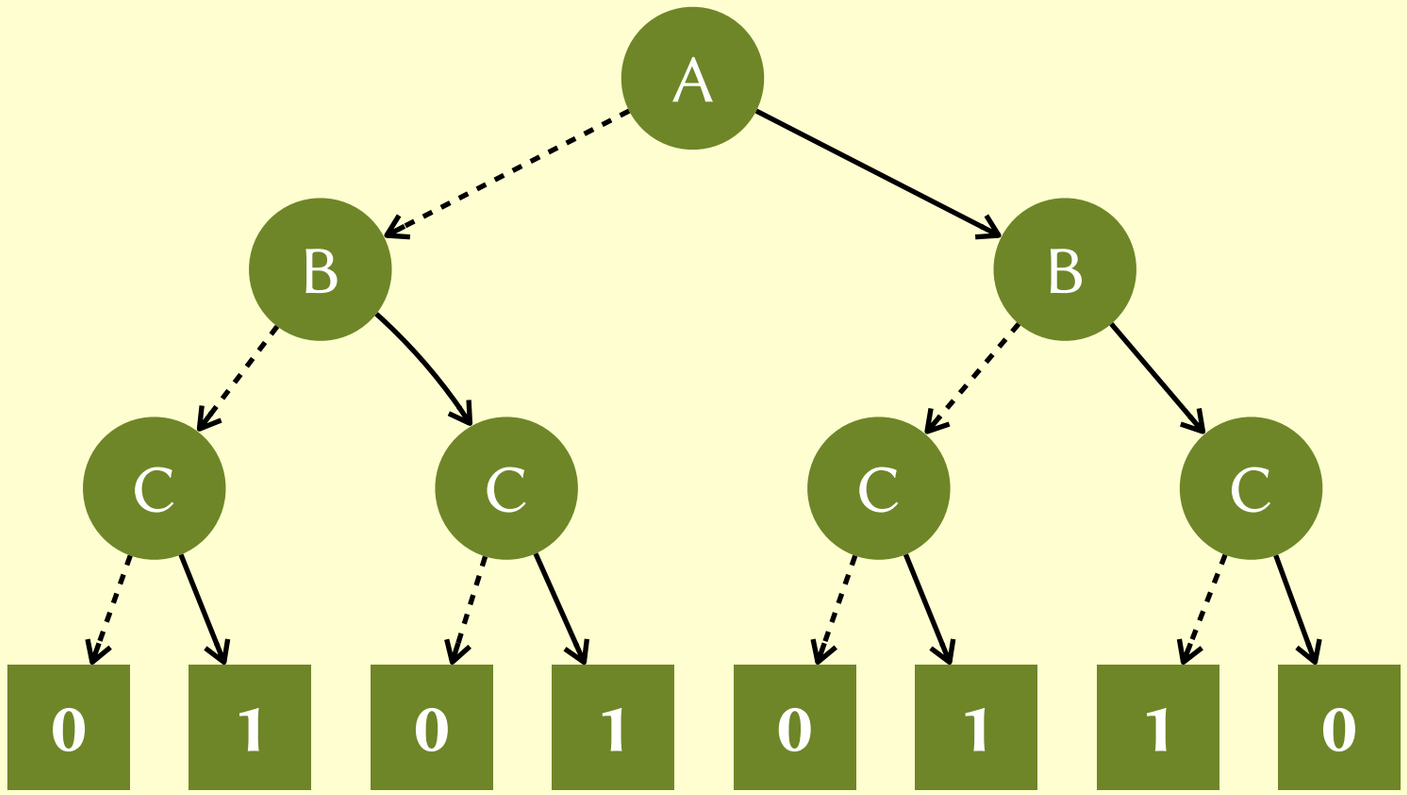
A	B	C	g
0	0	0	**0**
0	0	1	**1**
0	1	0	**0**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

Boolean functions

- Could represent Boolean functions as syntactic formulas.
- E.g. $f = (A \wedge \neg B \wedge \neg C) \vee (\neg A \wedge (B \oplus C))$
  $g = (\neg A \wedge C) \vee (A \wedge (B \oplus C))$
- The formulas may be compact, but the time taken to check $f \equiv g$ is still exponential in the number of variables.

Boolean functions

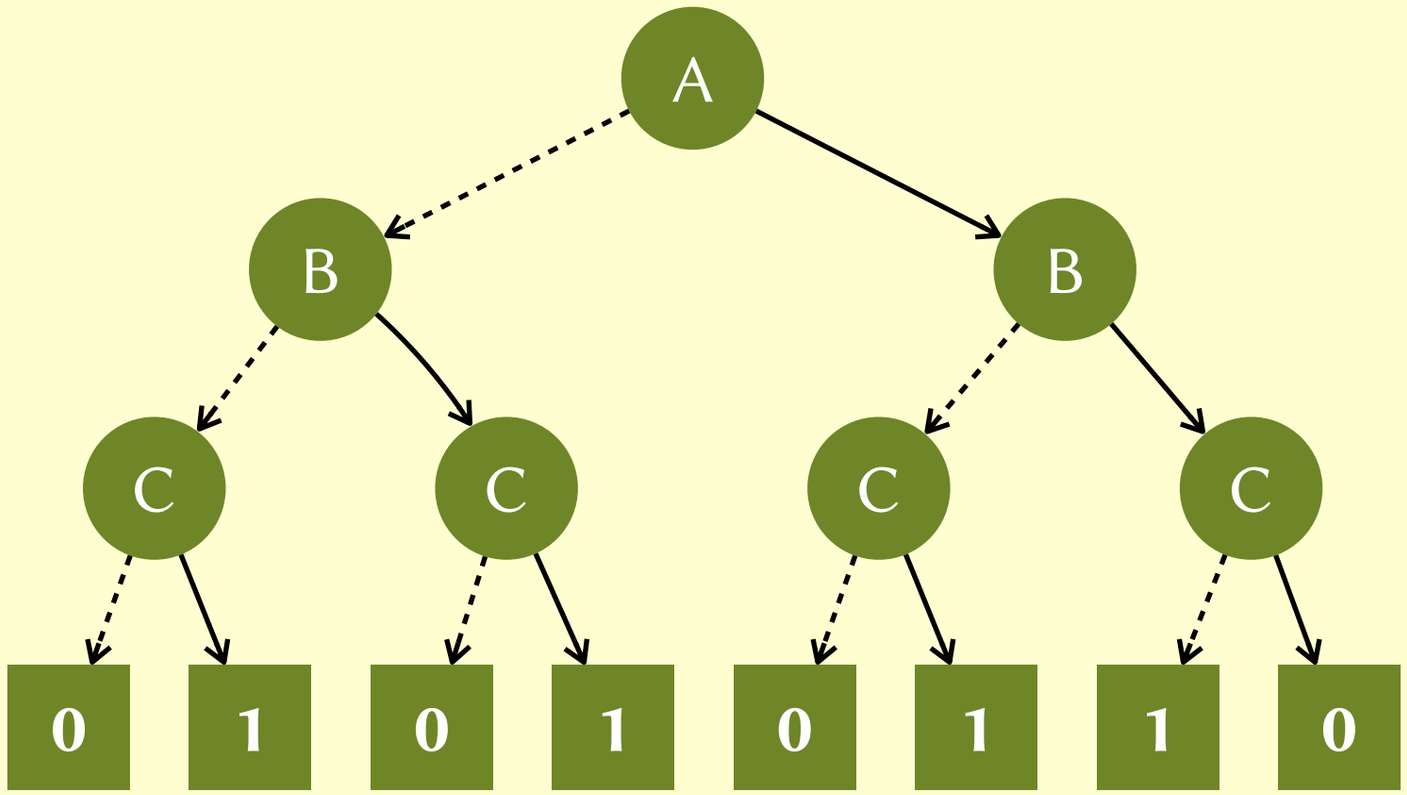
- Could represent Boolean functions as a binary decision tree.



- This is no more space efficient than the truth table, but…

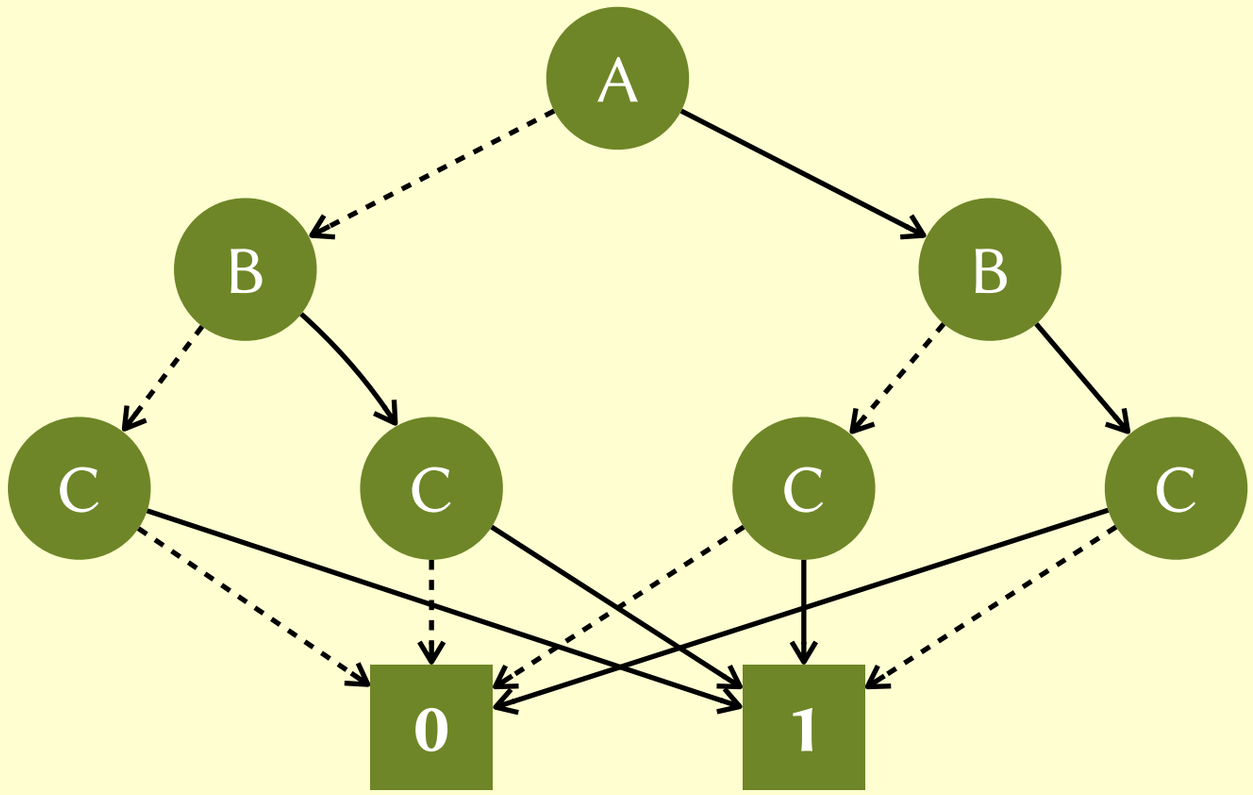
Boolean functions

1. Share all the 0s and 1s.



Boolean functions

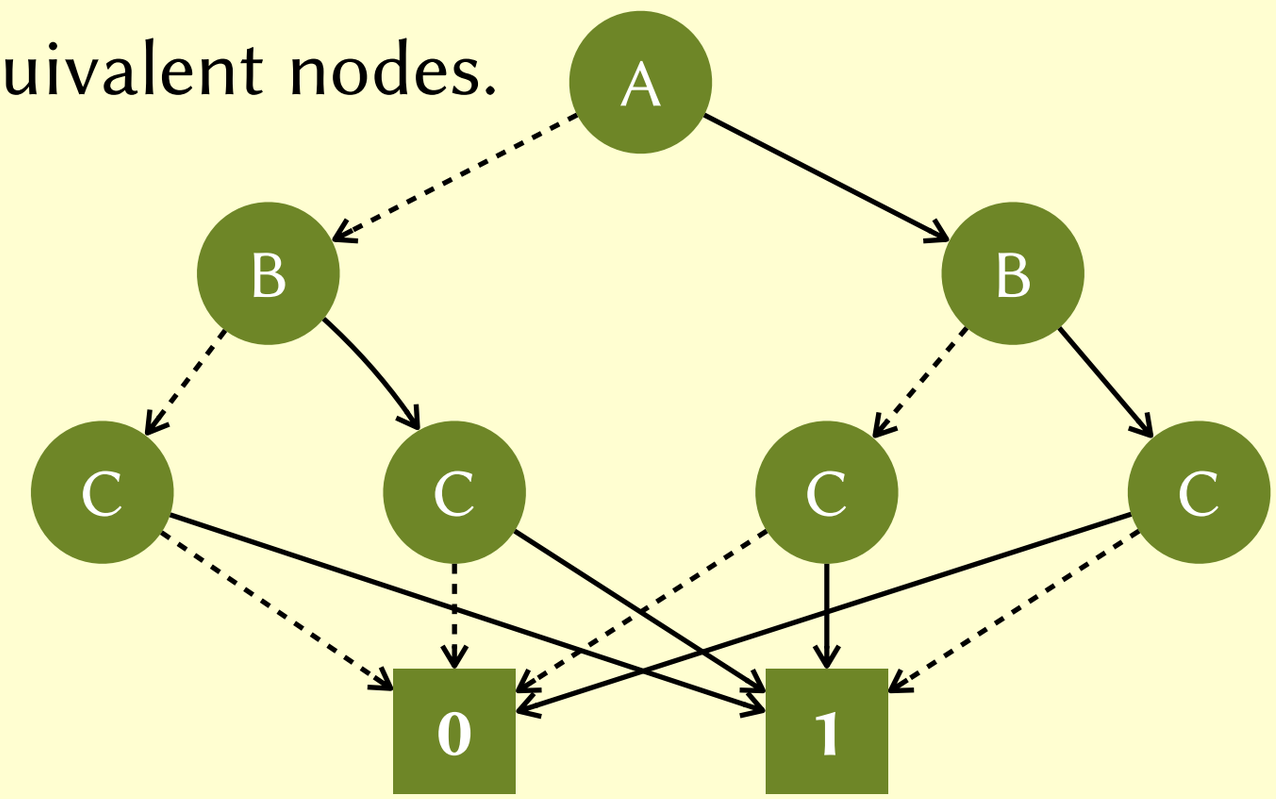
1. Share all the 0s and 1s.



Boolean functions

1. Share all the 0s and 1s.

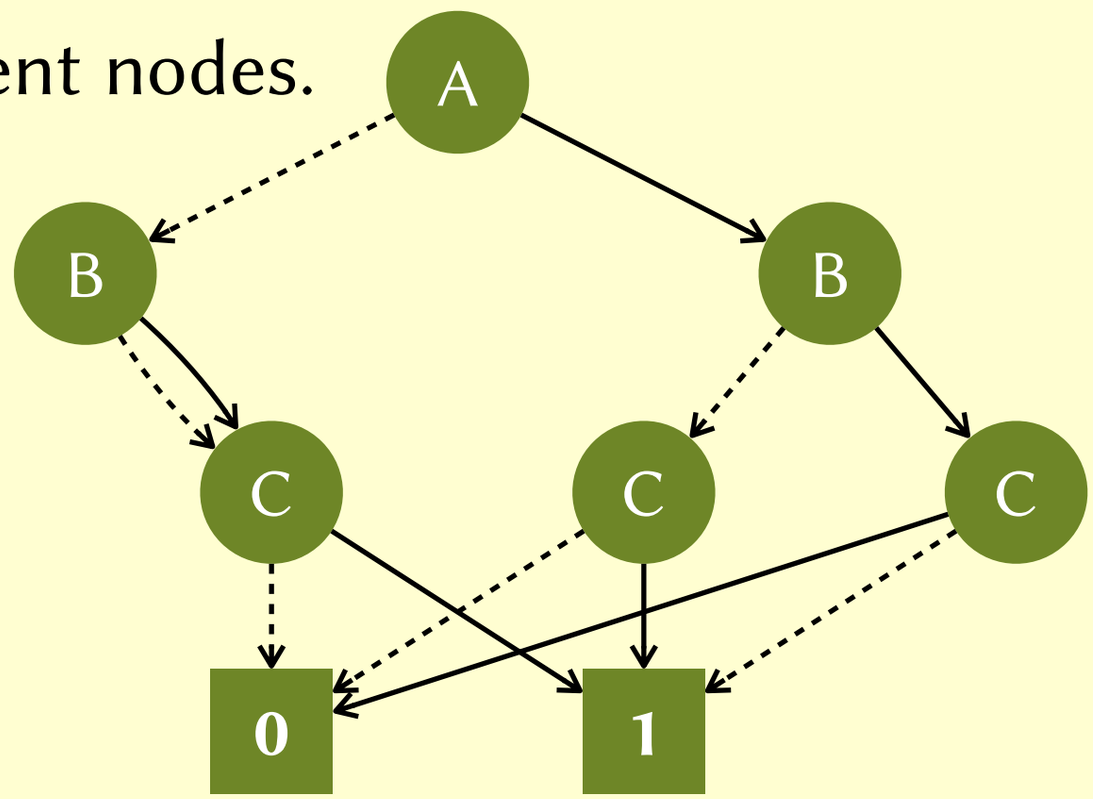
2. Merge equivalent nodes.



Boolean functions

1. Share all the 0s and 1s.

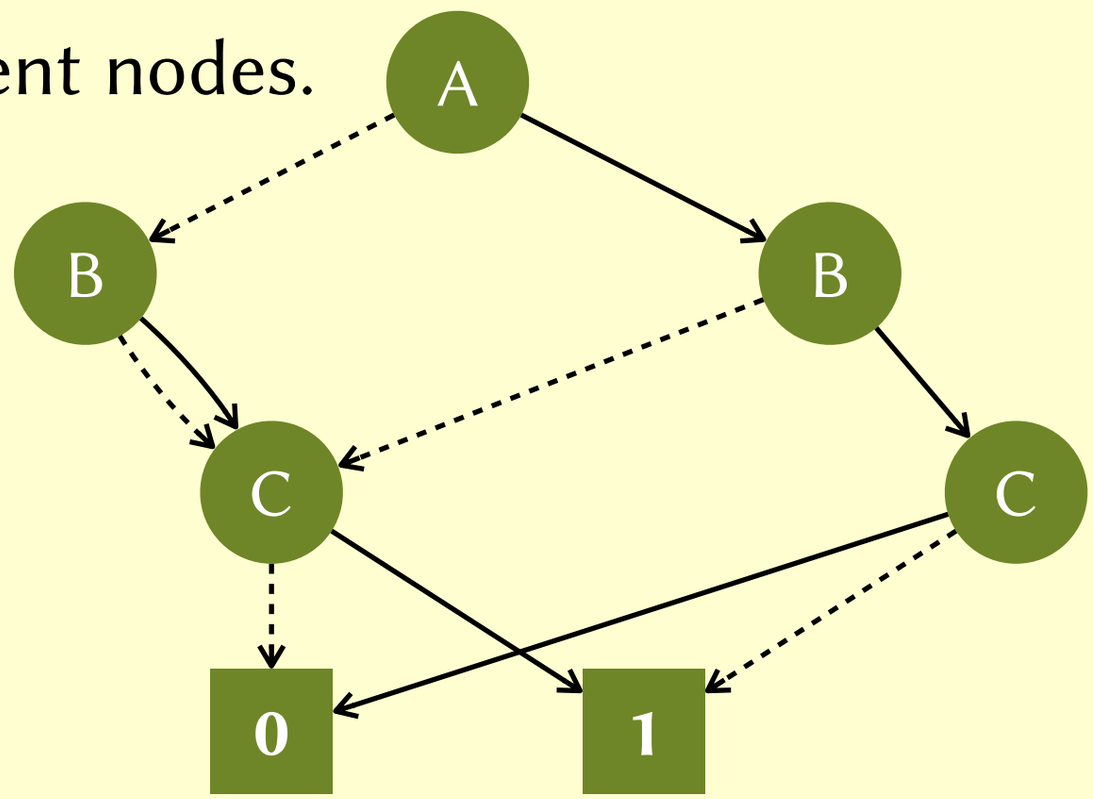
2. Merge equivalent nodes.



Boolean functions

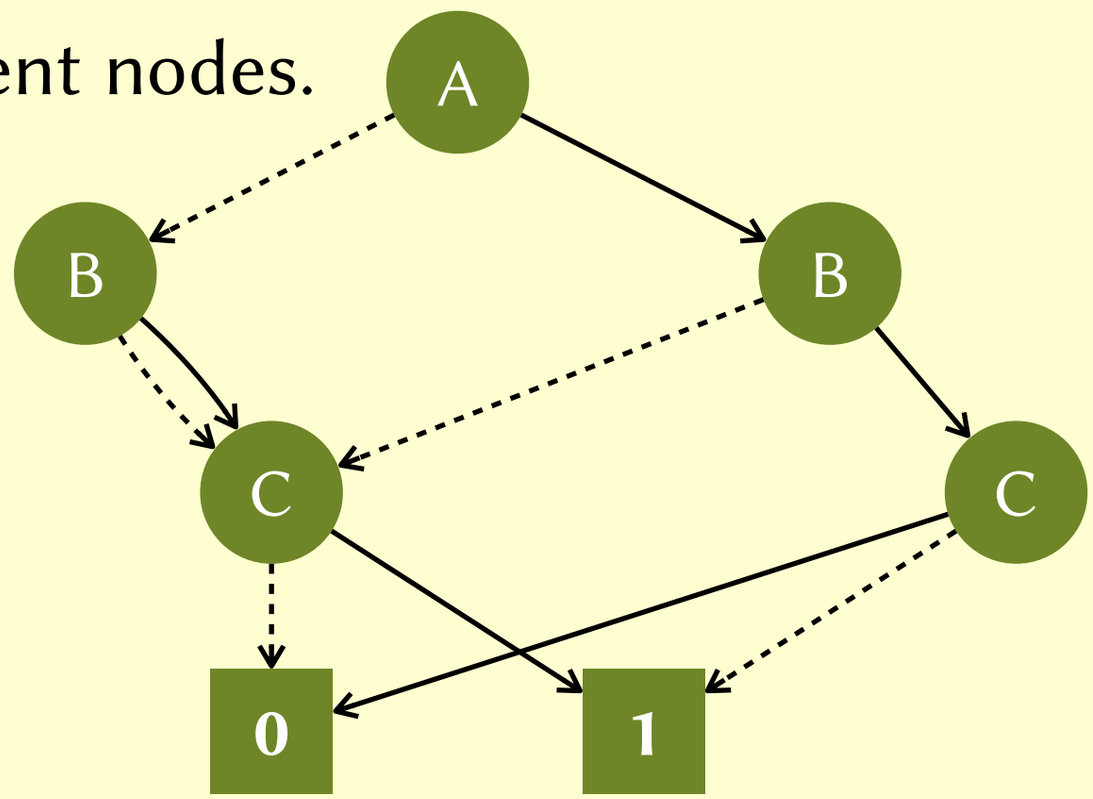
1. Share all the 0s and 1s.

2. Merge equivalent nodes.



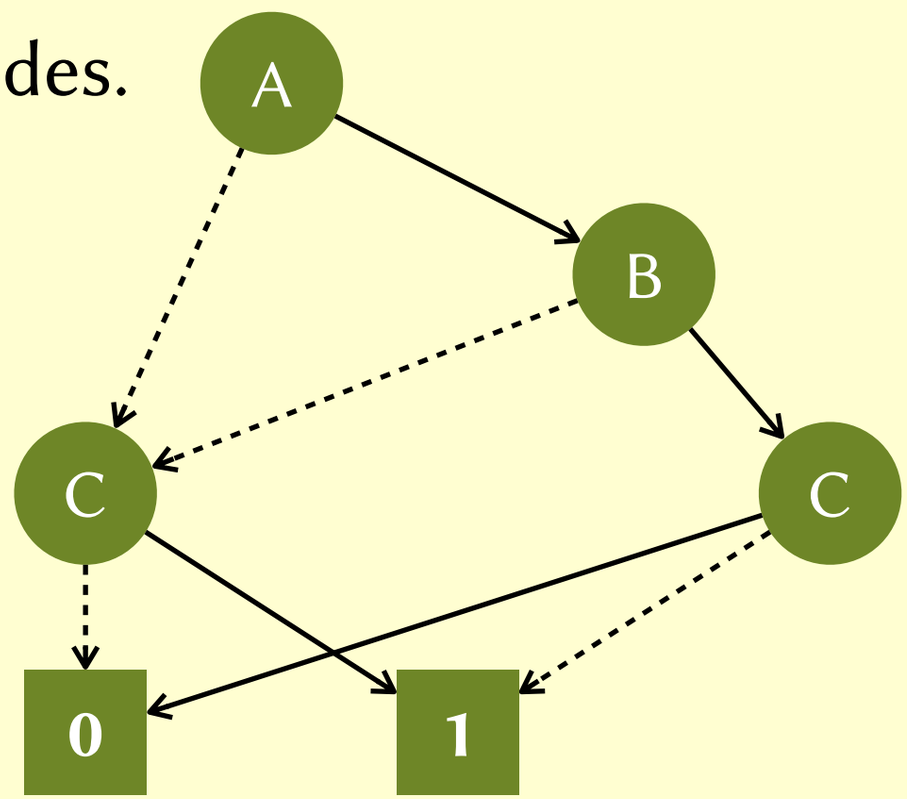
Boolean functions

1. Share all the 0s and 1s.
2. Merge equivalent nodes.
3. If both arrows from a node point to the same place, remove the node.



Boolean functions

1. Share all the 0s and 1s.
2. Merge equivalent nodes.
3. If both arrows from a node point to the same place, remove the node.



- We have produced a **binary decision diagram** (BDD).

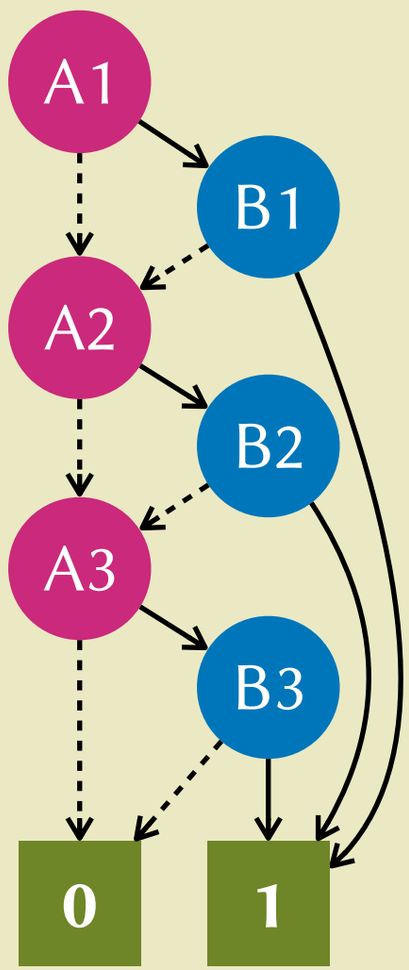
BDDs

- BDDs can be *compact*.
- BDDs are a *canonical representation*: if two formulas are equivalent, their BDDs will be the same. (Assuming BDDs are fully reduced and the order of variables is fixed.)
- Choosing a good order of variables is critical!

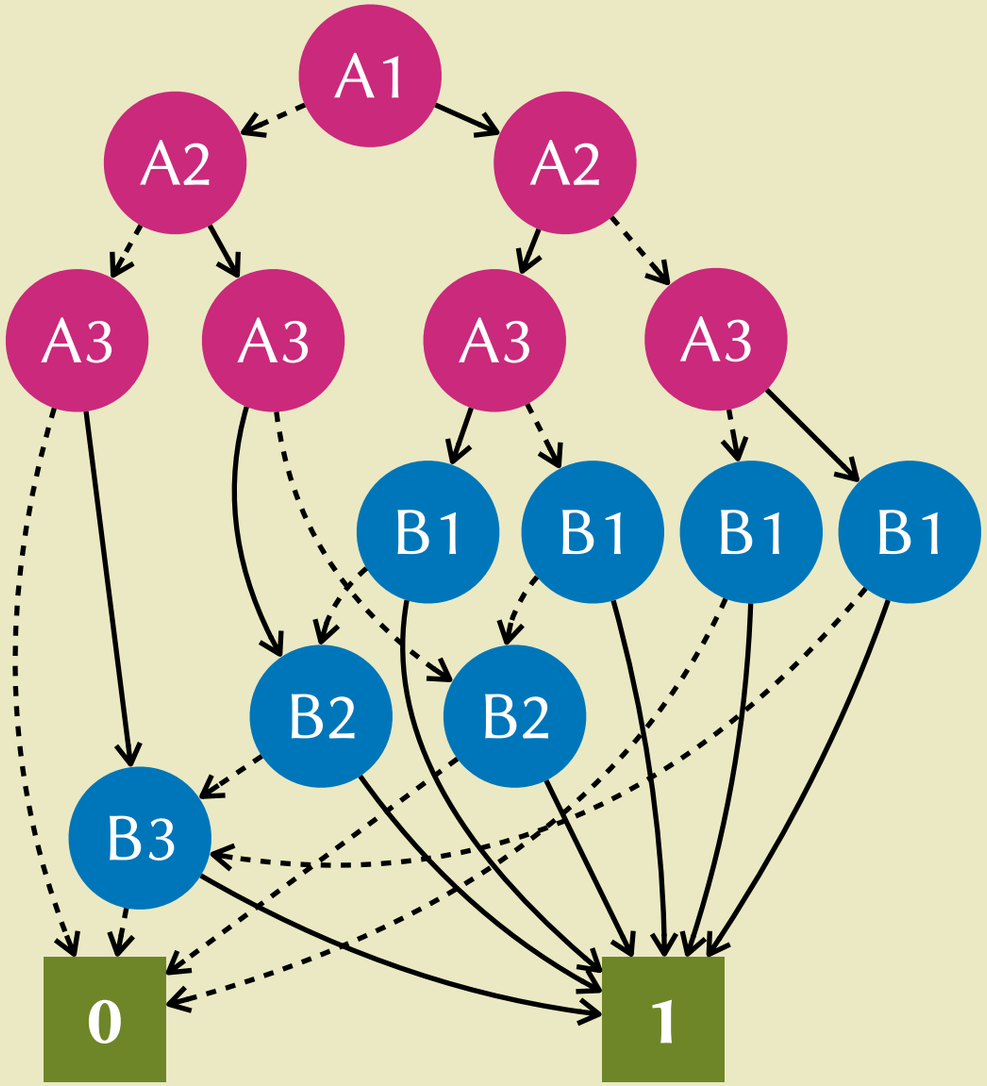
BDDs

- Suppose our formula is (A1 ∧ B1) ∨ (A2 ∧ B2) ∨ (A3 ∧ B3).

Order A1, B1, A2, B2, A3, B3:



Order A1, A2, A3, B1, B2, B3:



BDDs

- BDDs can be *compact*.
- BDDs are a *canonical representation*: if two formulas are equivalent, their BDDs will be the same. (Assuming BDDs are fully reduced and the order of variables is fixed.)
- Choosing a good order of variables is critical!
- BDDs aren't a silver bullet, but are often quite efficient.

Thank you!

- Please fill out the MEQ feedback form!