Toward Grammar Inference via Refinement Types

https://mcschroeder.github.io/#tyde2022

Michael Schröder
TU Wien
Vienna, Austria
michael.schroeder@tuwien.ac.at

Jürgen Cito
TU Wien
Vienna, Austria
juergen.cito@tuwien.ac.at

Type-Driven Development (TyDe), ICFP 2022
Ljubljana, Slovenia

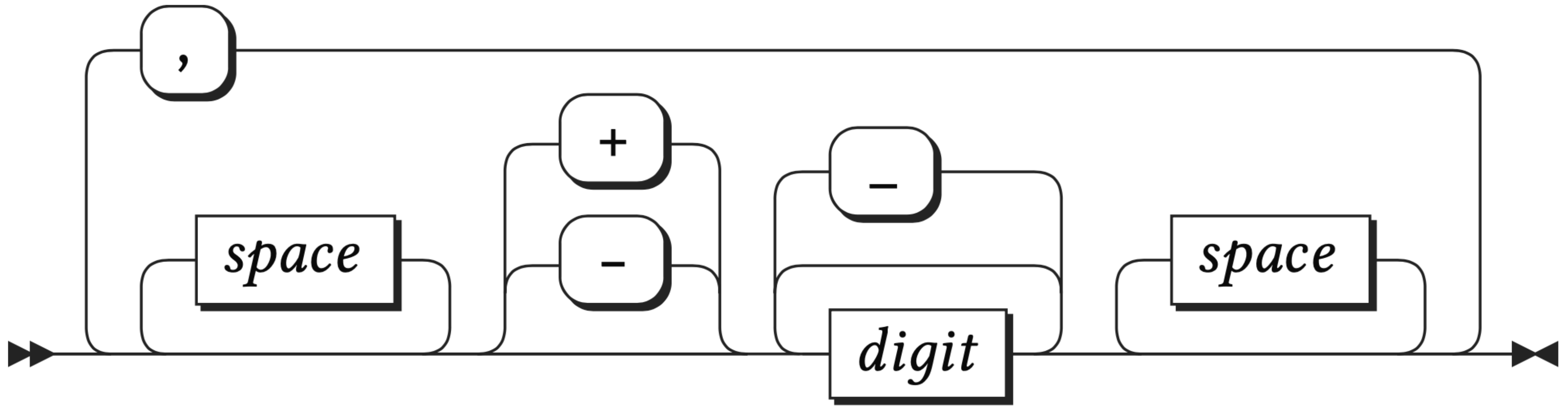
TU WIEN Informatics

```
xs = map(int, s.split(","))
```

"1,2,3"

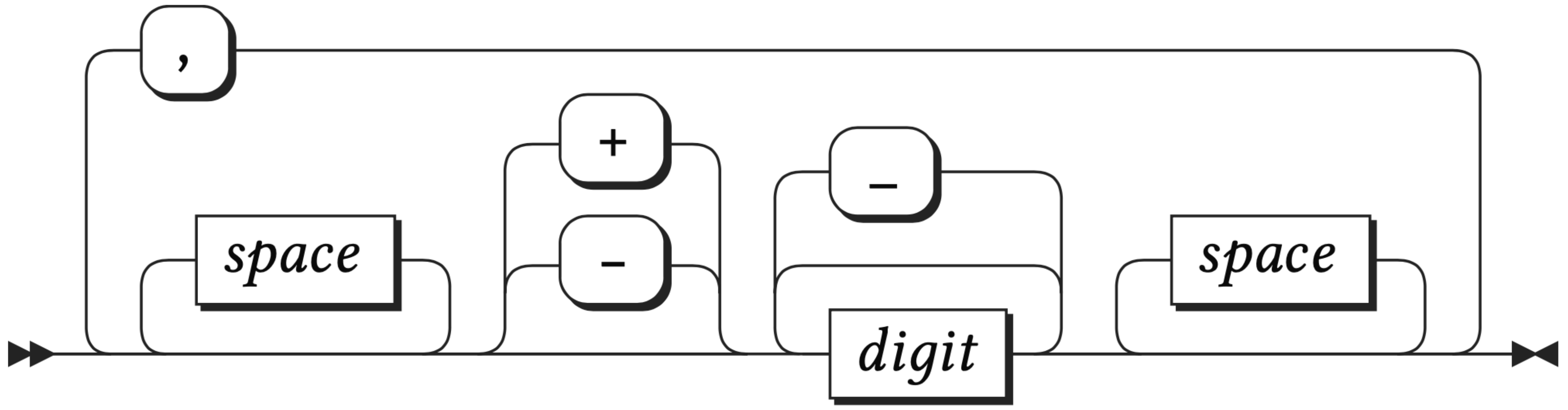
```
xs = map(int, s.split(","))
```

[1,2,3]



$$
\begin{array}{rcl}
s & \rightarrow & \mathit{int} \mid \mathit{int}\ \texttt{,}\ s \\
\mathit{int} & \rightarrow & \mathit{space}^{*}\ (\texttt{+} \mid \texttt{-})^{?}\ \mathit{digit}\ (\texttt{\_}^{?}\ \mathit{digit})^{*}\ \mathit{space}^{*} \\
\mathit{digit} & \rightarrow & \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
\mathit{space} & \rightarrow & \texttt{␣} \mid \texttt{\textbackslash t} \mid \texttt{\textbackslash n} \mid \texttt{\textbackslash v} \mid \texttt{\textbackslash f} \mid \texttt{\textbackslash r}
\end{array}
$$

```
xs = map(int, s.split(","))
```



$$
\begin{array}{rcl}
s & \rightarrow & \mathit{int} \mid \mathit{int}\ \texttt{,}\ s \\
\mathit{int} & \rightarrow & \mathit{space}^{*}\ (\texttt{+} \mid \texttt{-})^{?}\ \mathit{digit}\ (\texttt{\_}^{?}\ \mathit{digit})^{*}\ \mathit{space}^{*} \\
\mathit{digit} & \rightarrow & \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
\mathit{space} & \rightarrow & \text{␣} \mid \texttt{\textbackslash t} \mid \texttt{\textbackslash n} \mid \texttt{\textbackslash v} \mid \texttt{\textbackslash f} \mid \texttt{\textbackslash r}
\end{array}
$$

```
xs = map(int, s.split(","))
```

Parser : Grammar ≈ Function : Type

Type Inference

String

```
xs = map(int, s.split(","))
```

[Int]

✨ Grammar Inference ✨

$$
\begin{aligned}
s &\rightarrow int \mid int \texttt{ , } s \\
int &\rightarrow space^* \; (\texttt{+} \mid \texttt{-})^? \; digit \; (\texttt{\_}^? \; digit)^* \; space^* \\
digit &\rightarrow \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
space &\rightarrow \text{␣} \mid \texttt{\textbackslash t} \mid \texttt{\textbackslash n} \mid \texttt{\textbackslash v} \mid \texttt{\textbackslash f} \mid \texttt{\textbackslash r}
\end{aligned}
$$

String{•}

```
xs = map(int, s.split(","))
```

[Int]

Applications

semantic change tracking

⚠️ Merging #420 (6a36b23) into main (224b18b) will change the input grammar of a function.

Before:

baz $\rightarrow a^{*}b\Sigma^{*}$

After:

baz $\rightarrow \Sigma a^{*}b\Sigma^{*}$

```diff
   def baz(s):
-    i = 0
+    i = 1
     while s[i] == "a"
       i += 1
     assert s[i] == "b"
```

+ fuzz testing, program sketching, grammar-based refactoring, …

interactive documentation

Inferred Grammar	Inferred Inputs
$s \rightarrow int \mid int\ ,\ s$	✗ (empty)
$int \rightarrow space^{*}\ (+ \mid -)^{?}\ digit\ (\_^{?}\ digit)^{*}\ space^{*}$	✓ 1,2,3
$digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$	✓ 10_000,4
$space \rightarrow$ ␣ | \t | \n | \v | \f | \r	✓ +01_2,␣␣3␣

```python
xs = map(int, s.split(","))
```

searching for parsers via their grammars (cf. Hoogle)

🔍 s -> int | int , s

Viewer.py — 2 matches | Python

```python
    ranges = self.find_ranges()
    split = str.split
    point = map(int, split(self.text.index(CURRENT), ','))
    for start, end in ranges:
      startv = map(int, split(start, ','))
```

transformScriptTags.ts — 1 match | TypeScript

```typescript
      return null;
    }
    return rawValue.split(",").map(item => parseInt(item));
  }

```

- Schröder and Cito. 2022. Grammars for Free: Toward Grammar Inference for Ad Hoc Parsers. https://mcschroeder.github.io/#icse2022

Goal: Automatic Grammar Inference

ad hoc parser source

```
def parser(s):
  if s[0] == "a":
    assert len(s) == 1
  else:
    assert s[1] == "b"
```

?

grammar

$$s \to \mathrm{a} \mid (\Sigma \setminus \mathrm{a})\mathrm{b}\Sigma^*$$

Panini

- simple λ-calculus in Administrative Normal Form (ANF)
- refinement type system à la *Liquid Types*
- common string operations assumed as axioms

- idea: infer most precise refinement type for input string

ad hoc parser source

```python
def parser(s):
  if s[0] == "a":
    assert len(s) == 1
  else:
    assert s[1] == "b"
```

SSA/ANF transformation

Panini program

$$
\begin{aligned}
\text{assert} &: \{b : \mathbb{B} \mid b\} \to \mathbb{1} \\
\text{equals} &: (a : \mathbb{Z}) \to (b : \mathbb{Z}) \to \{c : \mathbb{B} \mid c \Leftrightarrow a = b\} \\
\text{length} &: (s : \mathbb{S}) \to \{n : \mathbb{N} \mid n = |s|\} \\
\text{charAt} &: (s : \mathbb{S}) \to \{i : \mathbb{N} \mid i < |s|\} \to \{t : \mathbb{S} \mid t = s[i]\} \\
\text{match} &: (s : \mathbb{S}) \to (t : \mathbb{S}) \to \{b : \mathbb{B} \mid b \Leftrightarrow s = t\}
\end{aligned}
$$

$$
\begin{aligned}
\text{parser} &: \mathbb{S} \to \mathbb{1} \\
&= \lambda s. \\
&\quad \textbf{let } x = \text{charAt } s\ 0 \textbf{ in} \\
&\quad \textbf{let } p_1 = \text{match } x\ \texttt{"a"} \textbf{ in} \\
&\quad \textbf{if } p_1 \textbf{ then} \\
&\quad\quad \textbf{let } n = \text{length } s \textbf{ in} \\
&\quad\quad \textbf{let } p_2 = \text{equals } n\ 1 \textbf{ in} \\
&\quad\quad \text{assert } p_2 \\
&\quad \textbf{else} \\
&\quad\quad \textbf{let } y = \text{charAt } s\ 1 \textbf{ in} \\
&\quad\quad \textbf{let } p_3 = \text{match } y\ \texttt{"b"} \textbf{ in} \\
&\quad\quad \text{assert } p_3
\end{aligned}
$$

- Braun et al. 2013. Simple and Efficient Construction of Static Single Assignment Form. https://doi.org/10.1007/978-3-642-37051-9_6
- Chakravarty et al. 2004. A functional perspective on SSA optimisation algorithms. https://doi.org/10.1016/S1571-0661(05)82596-4

Refinement Inference

- κ variables represent unknown refinements
- most can be solved precisely (e.g., using FUSION)
- existing approaches struggle with "grammar variables"

verification template

$$
\begin{array}{l}
\forall s.\ \boxed{\kappa_0(s)} \Rightarrow \\
\quad 0 < |s| \wedge \forall x.\ x = s[0] \Rightarrow \\
\quad\quad \forall p_1.\ p_1 \Leftrightarrow x = \texttt{"a"} \Rightarrow \\
\quad\quad\quad (p_1 \Rightarrow \\
\quad\quad\quad\quad \forall n.\ n \geq 0 \wedge n = |s| \Rightarrow \\
\quad\quad\quad\quad\quad \forall p_2.\ p_2 \Leftrightarrow n = 1 \Rightarrow \\
\quad\quad\quad\quad\quad\quad p_2) \\
\quad\quad \wedge\ (\neg p_1 \Rightarrow \\
\quad\quad\quad\quad 1 < |s| \wedge \forall y.\ y = s[1] \Rightarrow \\
\quad\quad\quad\quad\quad \forall p_3.\ p_3 \Leftrightarrow y = \texttt{"b"} \Rightarrow \\
\quad\quad\quad\quad\quad\quad p_3)
\end{array}
$$

"grammar variable"
(constraint over input string)

PANINI program

$$
\begin{array}{rcl}
\text{assert} & : & \{b : \mathbb{B} \mid b\} \to \mathbb{1} \\
\text{equals} & : & (a : \mathbb{Z}) \to (b : \mathbb{Z}) \to \{c : \mathbb{B} \mid c \Leftrightarrow a = b\} \\
\text{length} & : & (s : \mathbb{S}) \to \{n : \mathbb{N} \mid n = |s|\} \\
\text{charAt} & : & (s : \mathbb{S}) \to \{i : \mathbb{N} \mid i < |s|\} \to \{t : \mathbb{S} \mid t = s[i]\} \\
\text{match} & : & (s : \mathbb{S}) \to (t : \mathbb{S}) \to \{b : \mathbb{B} \mid b \Leftrightarrow s = t\}
\end{array}
$$

$$
\begin{array}{l}
\text{parser} : \{s : \mathbb{S} \mid \boxed{\kappa_0(s)}\} \to \mathbb{1} \\
\quad = \boldsymbol{\lambda} s. \\
\quad\quad \textbf{let } x = \text{charAt } s\ 0 \textbf{ in} \\
\quad\quad \textbf{let } p_1 = \text{match } x\ \texttt{"a"} \textbf{ in} \\
\quad\quad \textbf{if } p_1 \textbf{ then} \\
\quad\quad\quad \textbf{let } n = \text{length } s \textbf{ in} \\
\quad\quad\quad \textbf{let } p_2 = \text{equals } n\ 1 \textbf{ in} \\
\quad\quad\quad \text{assert } p_2 \\
\quad\quad \textbf{else} \\
\quad\quad\quad \textbf{let } y = \text{charAt } s\ 1 \textbf{ in} \\
\quad\quad\quad \textbf{let } p_3 = \text{match } y\ \texttt{"b"} \textbf{ in} \\
\quad\quad\quad \text{assert } p_3
\end{array}
$$

- Jhala and Vazou. 2020. Refinement Types: A Tutorial. https://arxiv.org/abs/2010.07763
- Cosman and Jhala. 2017. Local Refinement Typing. https://doi.org/10.1145/3110270
- Rondon et al. 2008. Liquid Types. https://doi.org/10.1145/1375581.1375602

Grammar Solving

- base solution on “grammar consequent”

verification template

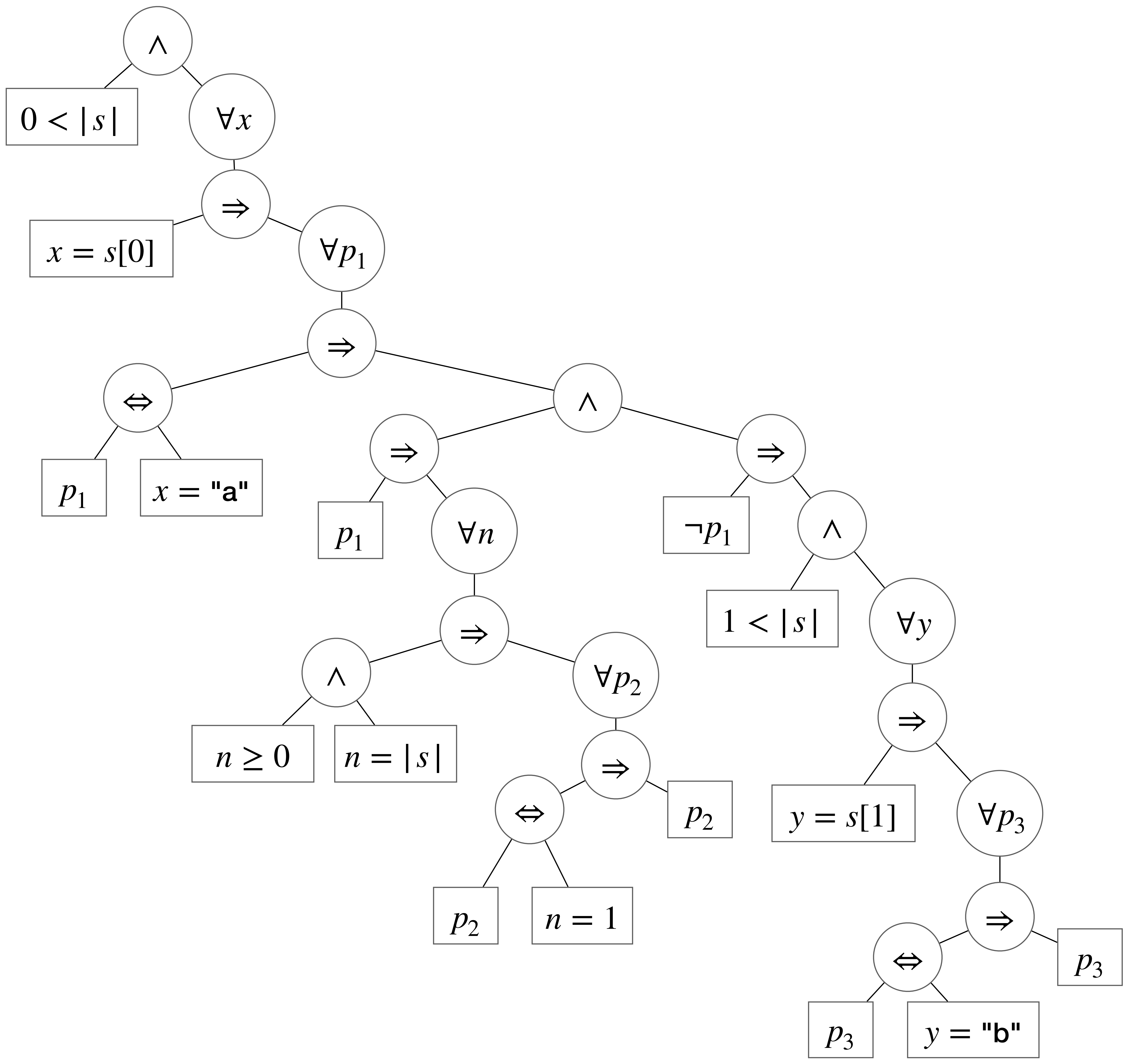
“grammar variable”
(constraint over input string)

$$
\begin{aligned}
&\forall s.\ \boxed{\kappa_0(s)} \Rightarrow \\
&\quad \left.\begin{aligned}
&0 < |s| \land \forall x.\ x = s[0] \Rightarrow \\
&\quad \forall p_1.\ p_1 \Leftrightarrow x = \text{"a"} \Rightarrow \\
&\qquad (p_1 \Rightarrow \\
&\qquad\quad \forall n.\ n \geq 0 \land n = |s| \Rightarrow \\
&\qquad\qquad \forall p_2.\ p_2 \Leftrightarrow n = 1 \Rightarrow \\
&\qquad\qquad\quad p_2) \\
&\quad\ \land (\neg p_1 \Rightarrow \\
&\qquad\quad 1 < |s| \land \forall y.\ y = s[1] \Rightarrow \\
&\qquad\qquad \forall p_3.\ p_3 \Leftrightarrow y = \text{"b"} \Rightarrow \\
&\qquad\qquad\quad p_3)
\end{aligned}\right\} \text{“grammar consequent”}
\end{aligned}
$$

Grammar Solving

- base solution on “grammar consequent”

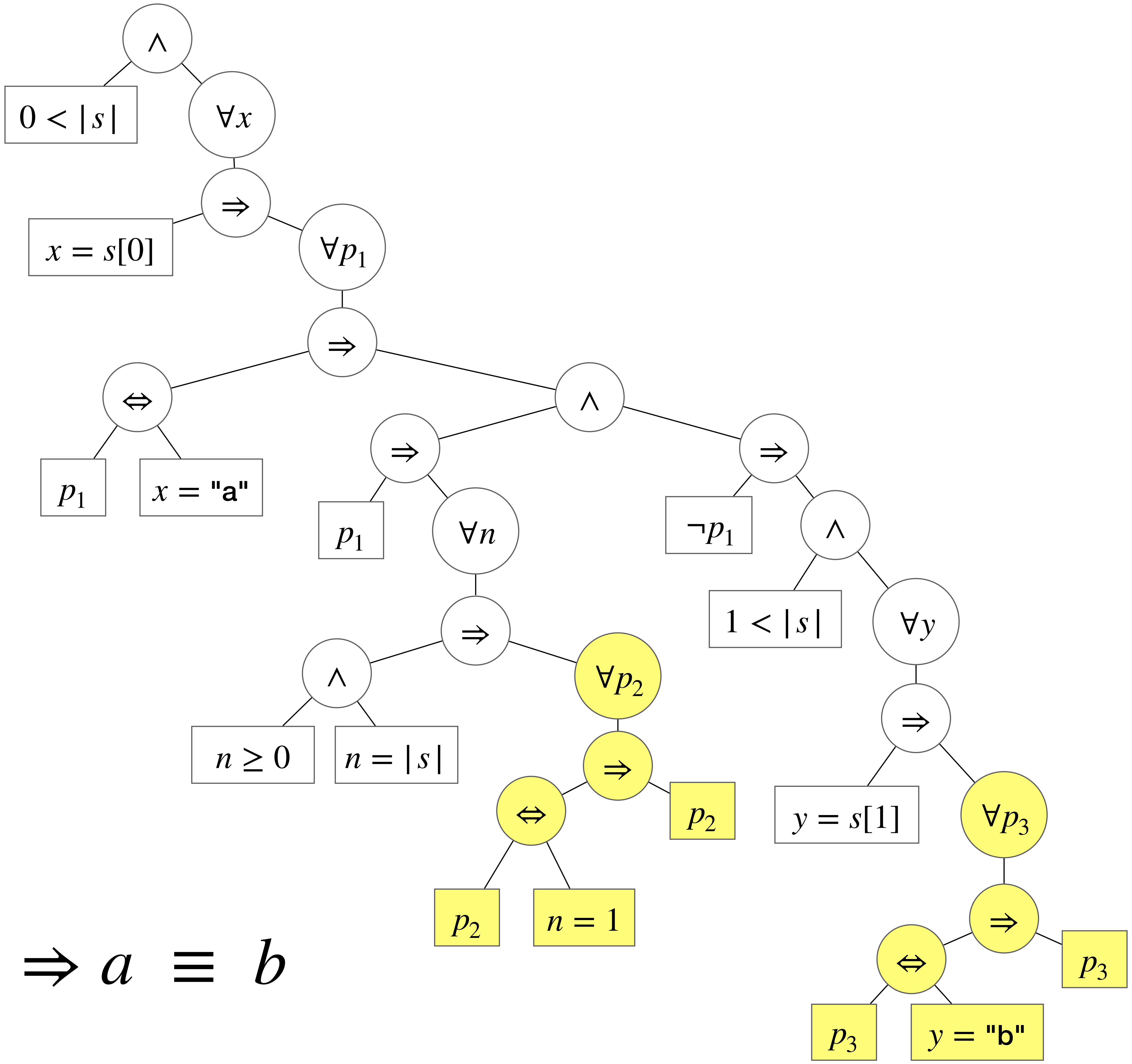
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Grammar Solving

- base solution on “grammar consequent”

- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations

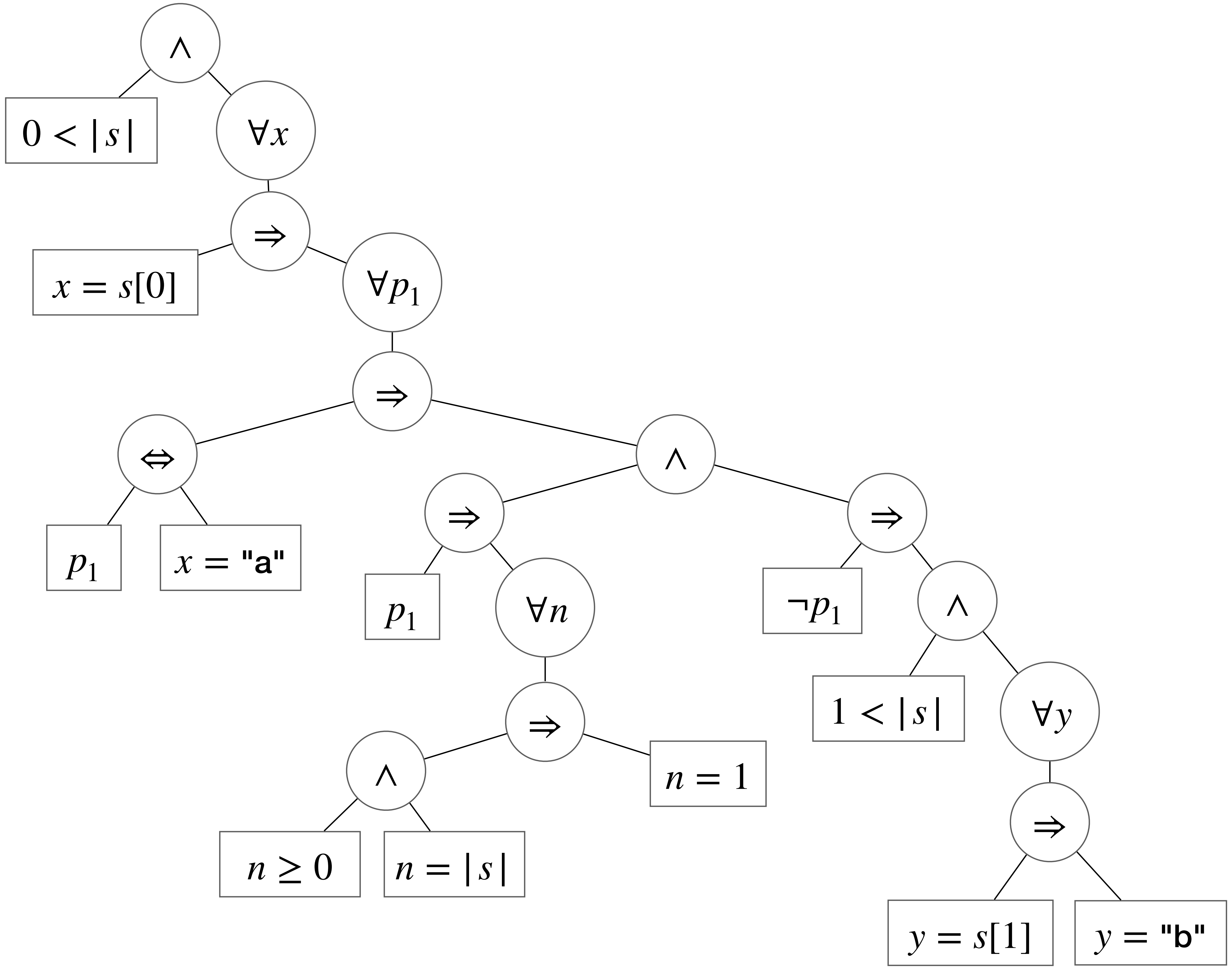


$$\forall a \,.\, (a \Leftrightarrow b) \Rightarrow a \;\equiv\; b$$

Grammar Solving

- base solution on "grammar consequent"

- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations

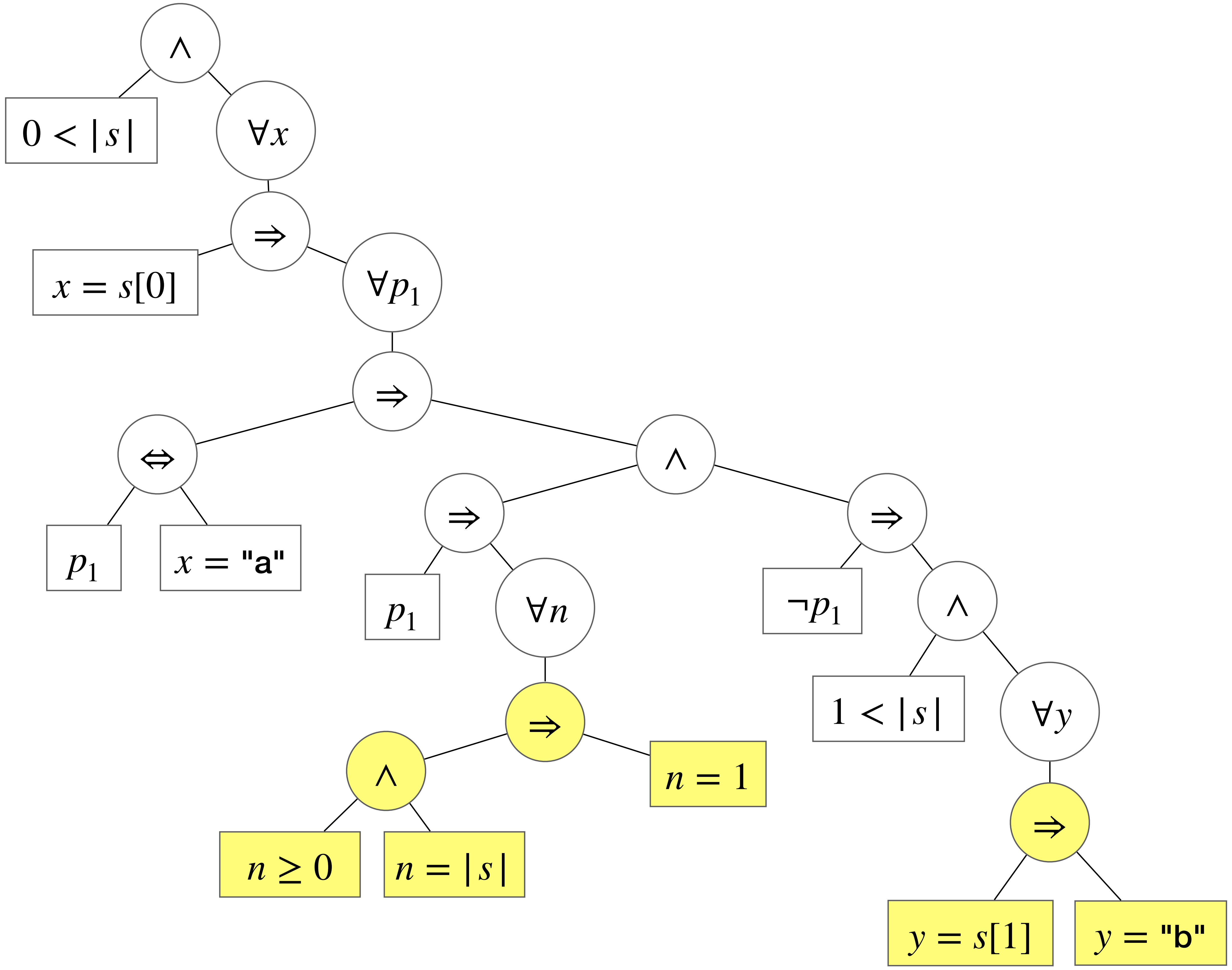


$$\forall a\,.\,(a \Leftrightarrow b) \Rightarrow a \;\equiv\; b$$

Grammar Solving

- base solution on “grammar consequent”

- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations

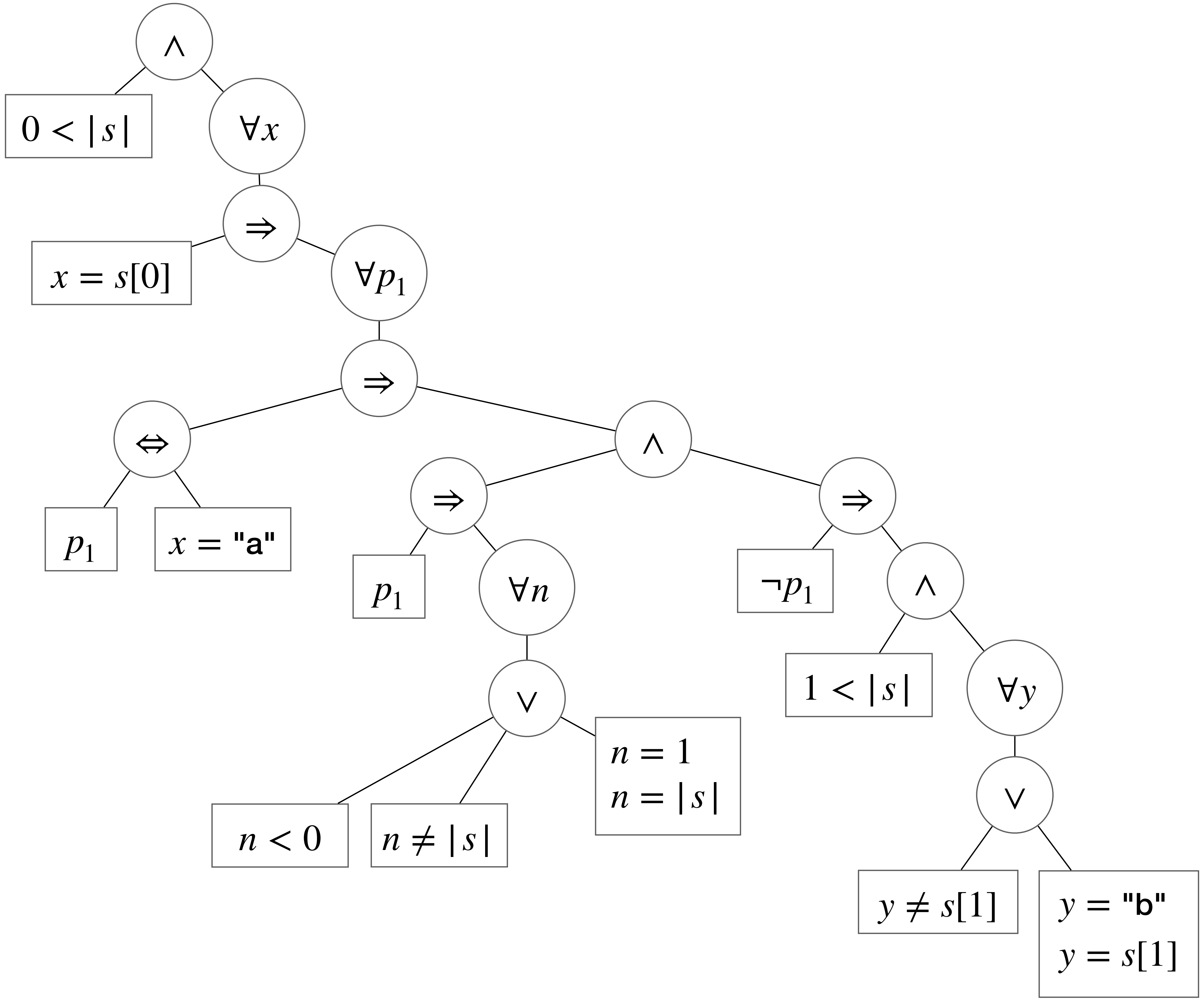


$$a \Rightarrow b \equiv \neg a \vee (a \sqcap b)$$

Grammar Solving

- base solution on “grammar consequent”

- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations

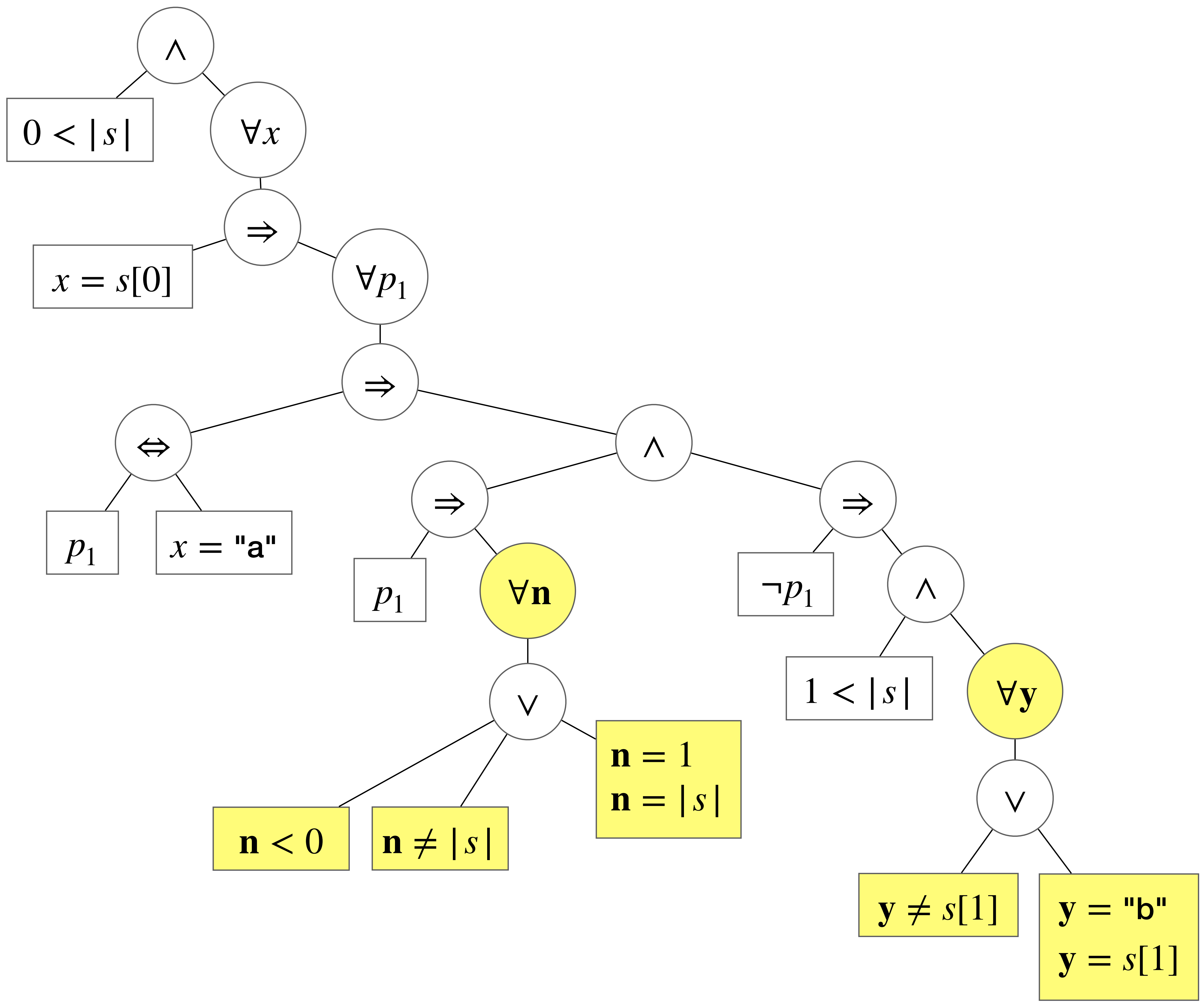


$$a \Rightarrow b \equiv \neg a \vee (a \sqcap b)$$

Grammar Solving

- base solution on “grammar consequent”

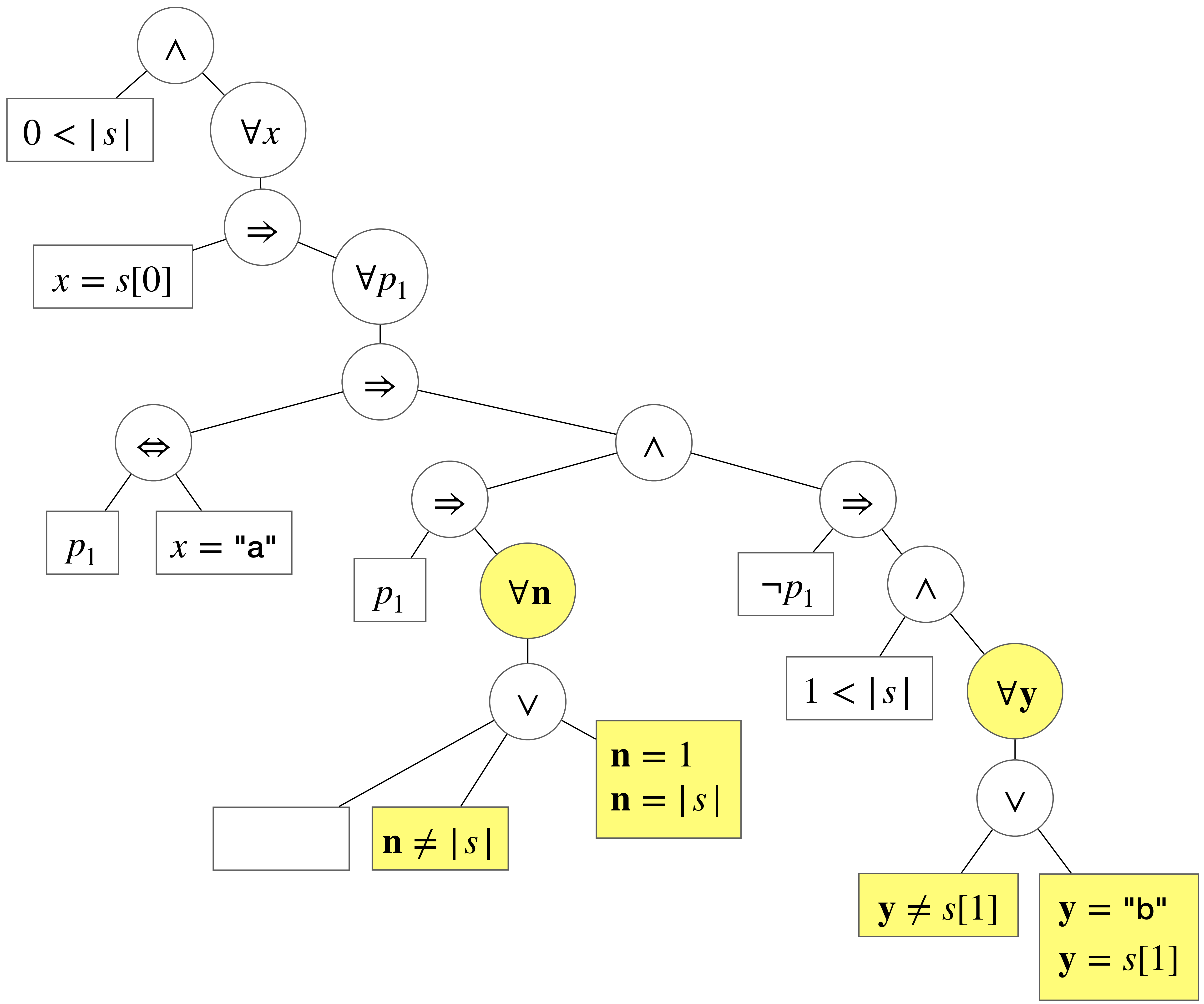
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Grammar Solving

- base solution on “grammar consequent”

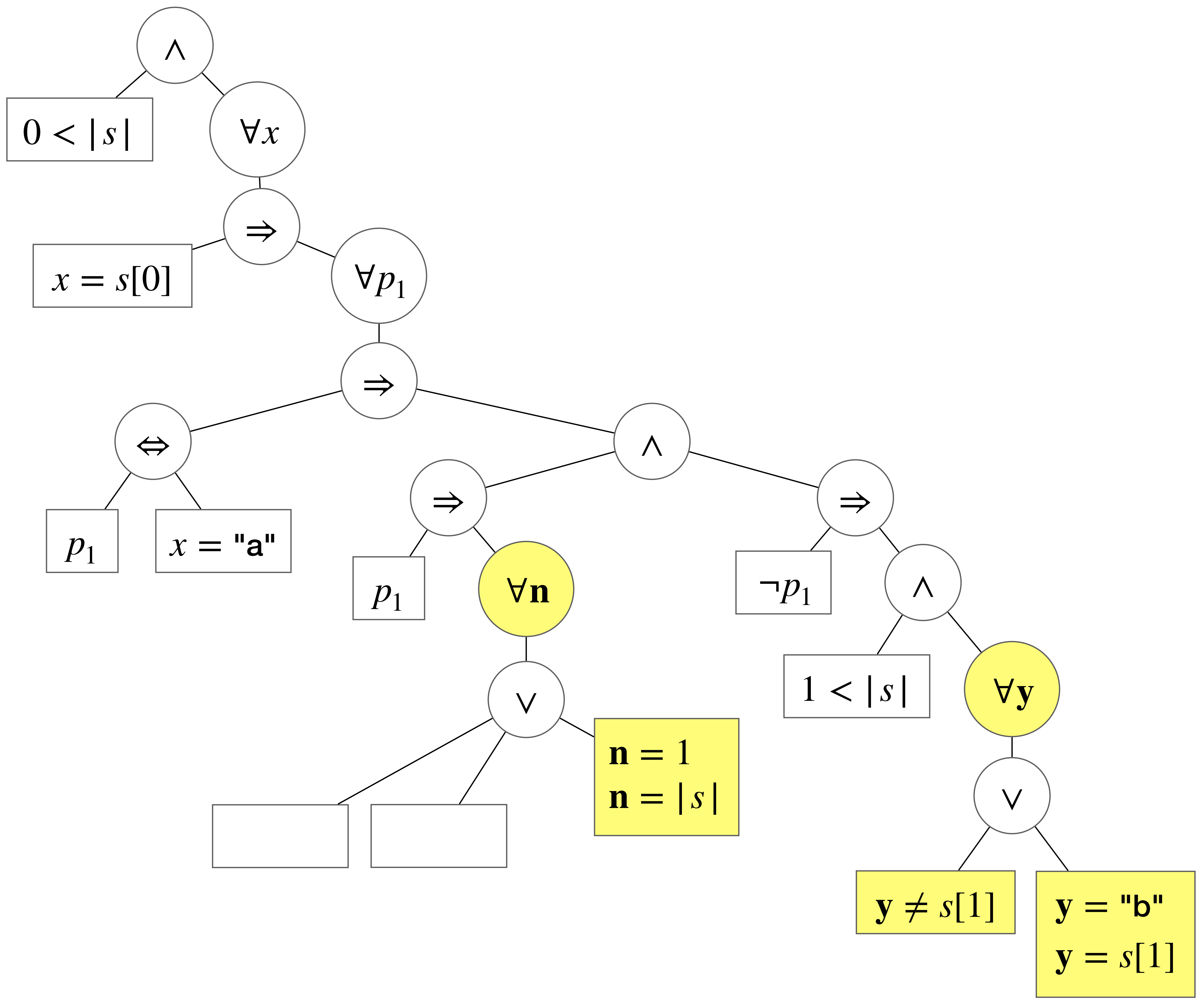
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Grammar Solving

- base solution on “grammar consequent”

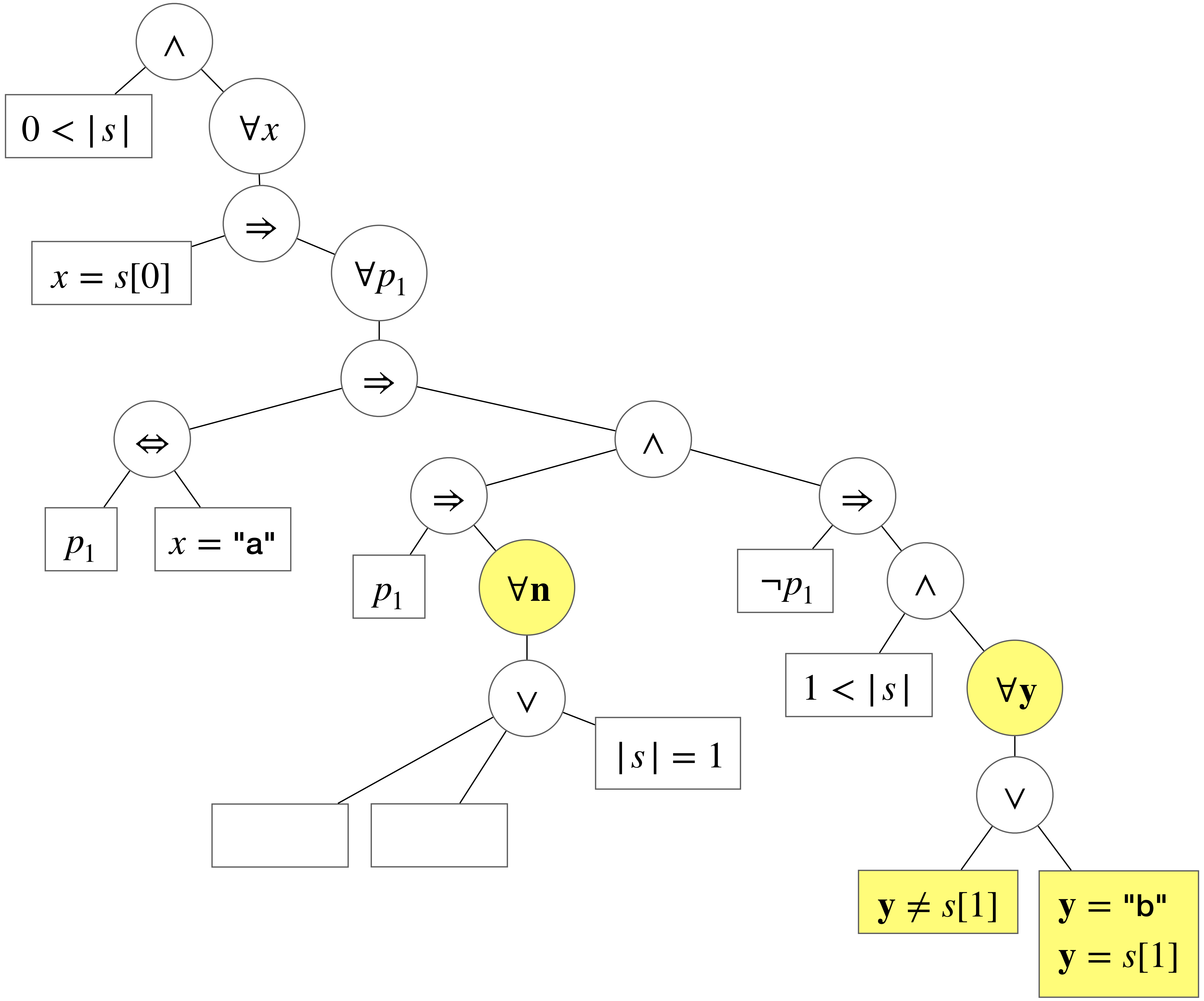
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Grammar Solving

- base solution on "grammar consequent"

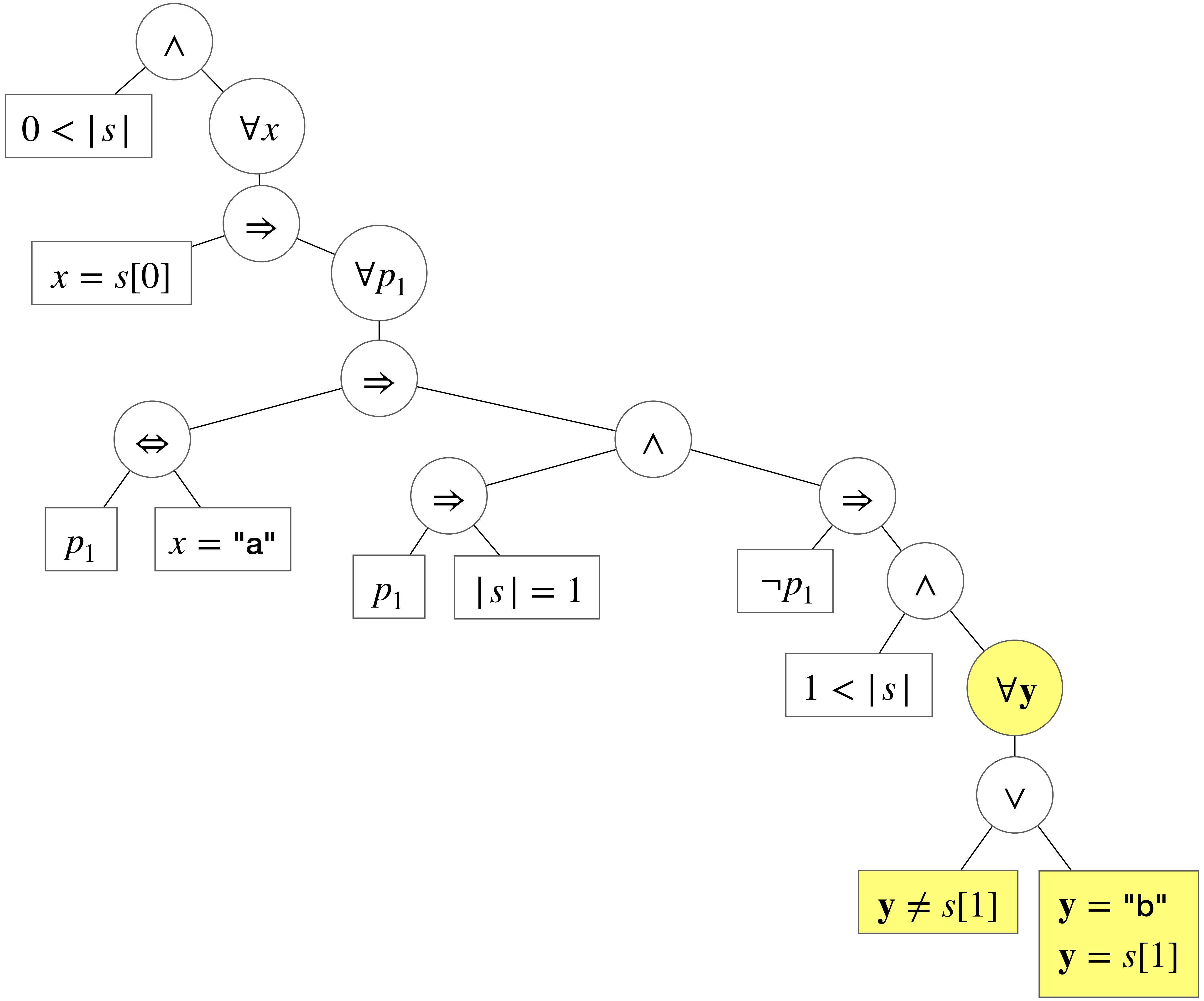
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Grammar Solving

- base solution on “grammar consequent”

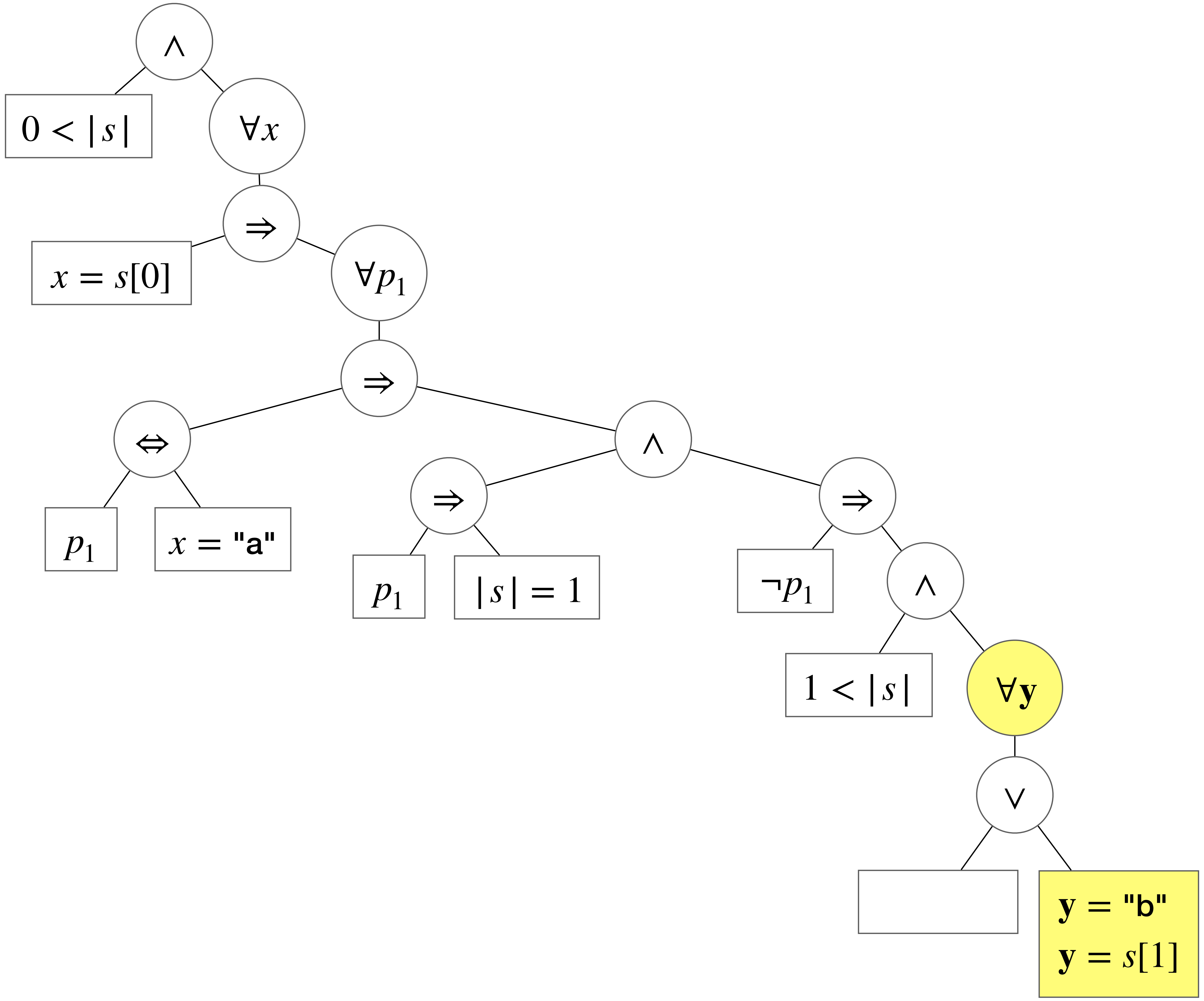
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Grammar Solving

- base solution on "grammar consequent"

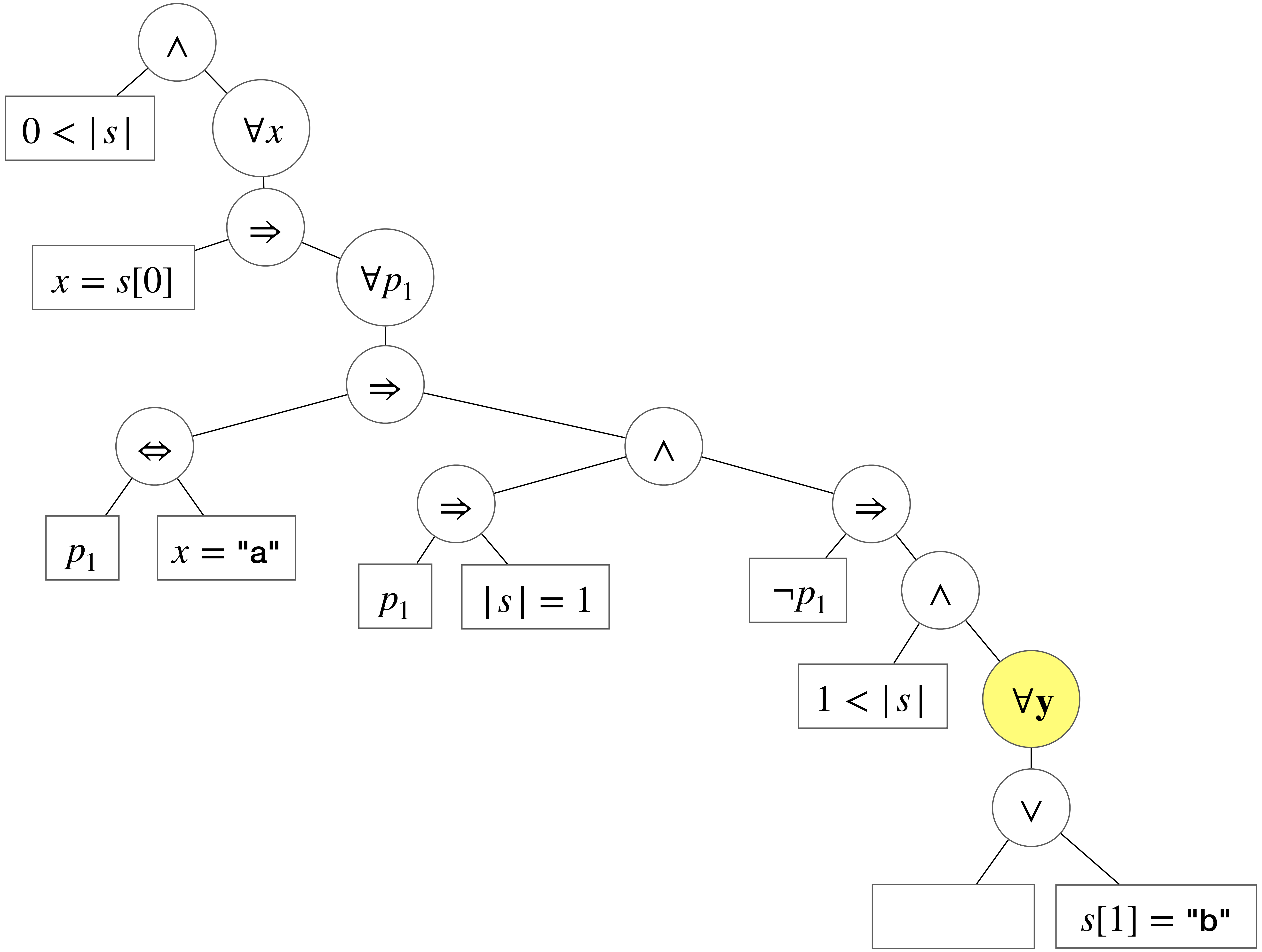
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Grammar Solving

- base solution on "grammar consequent"

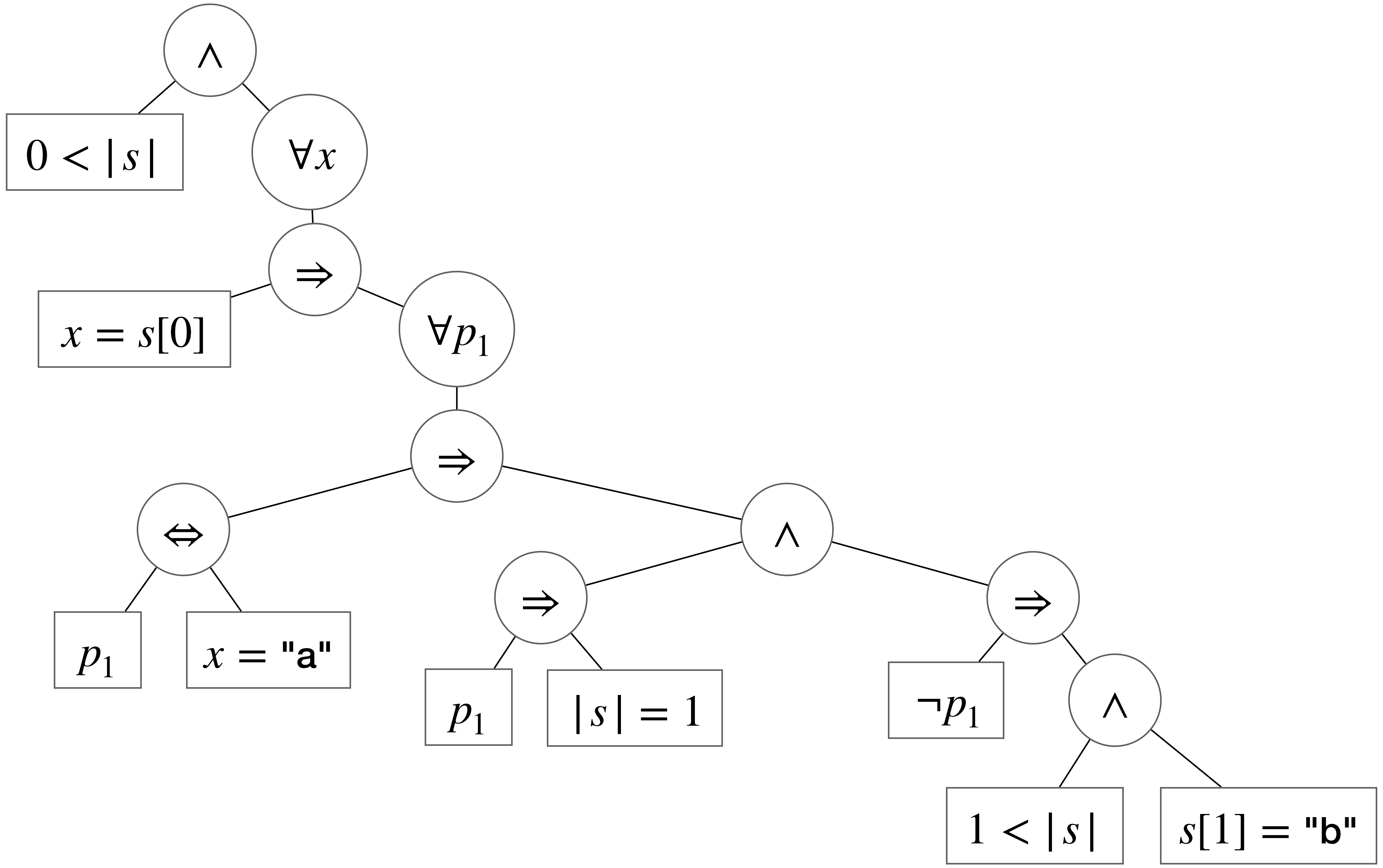
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Grammar Solving

- base solution on “grammar consequent”

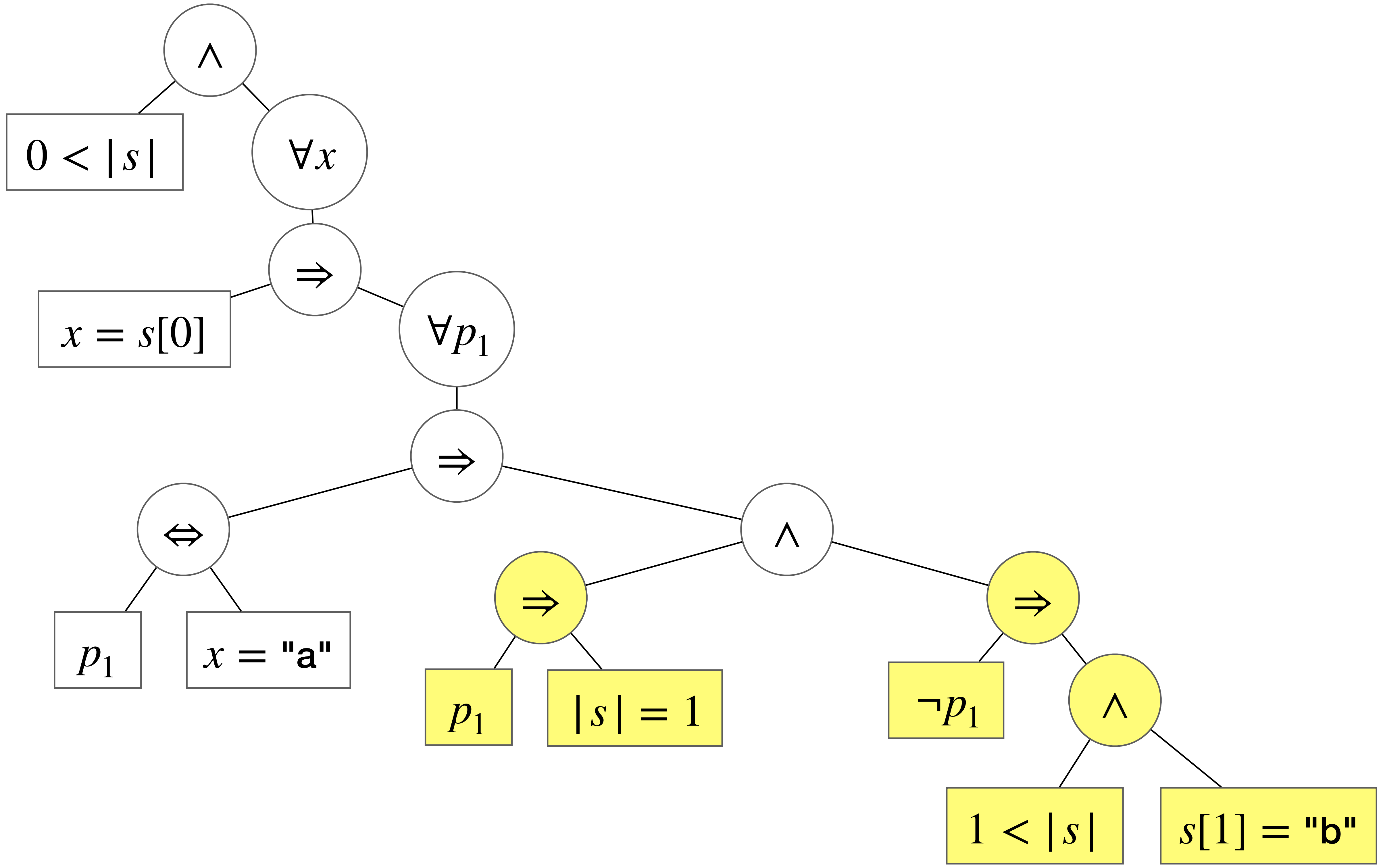
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Grammar Solving

- base solution on "grammar consequent"

- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations

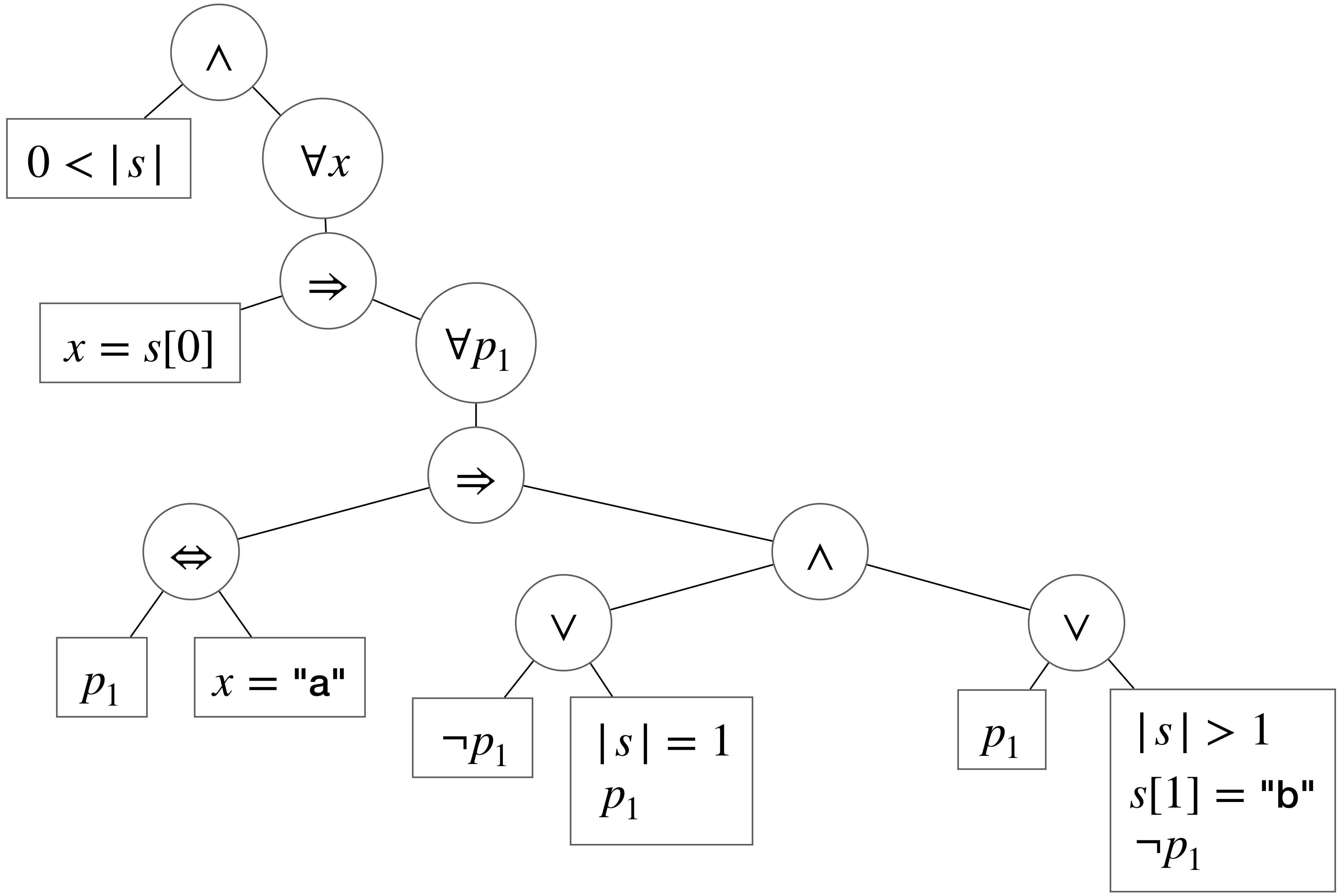


$$a \Rightarrow b \equiv \neg a \vee (a \sqcap b)$$

Grammar Solving

- base solution on “grammar consequent”

- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations

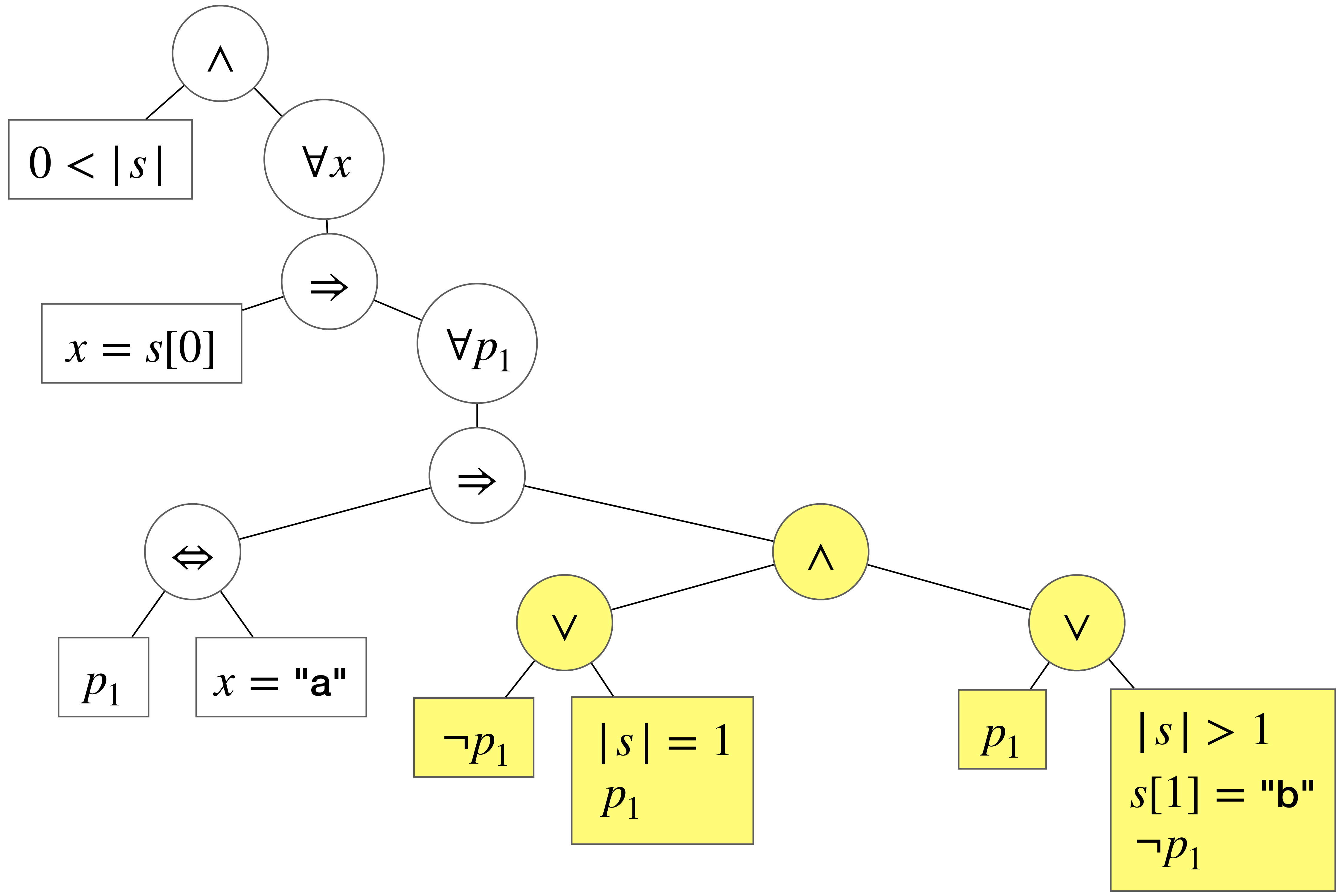


$$a \Rightarrow b \;\equiv\; \neg a \vee (a \sqcap b)$$

Grammar Solving

- base solution on “grammar consequent”

- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations

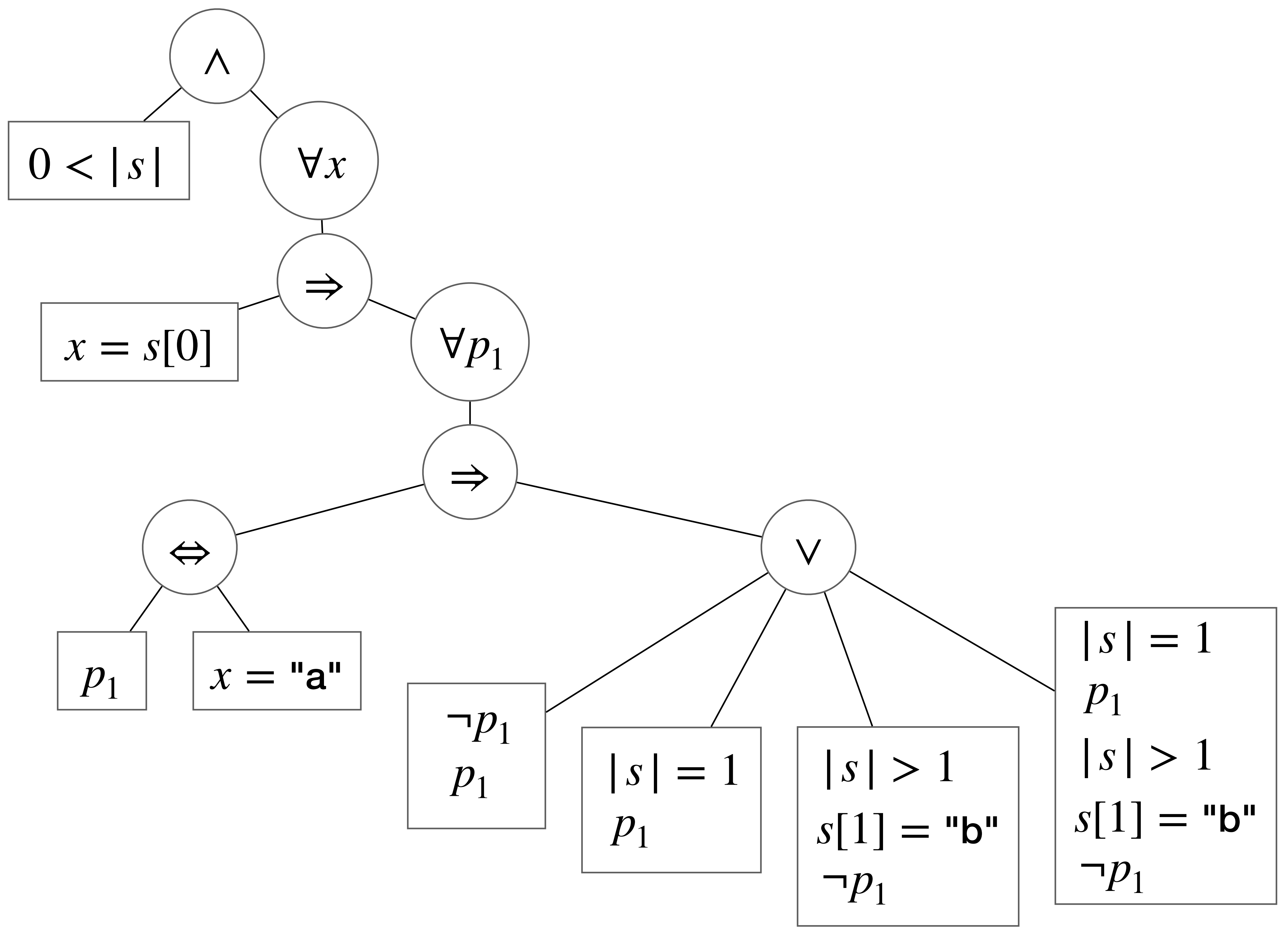


$$(a \lor b) \land (\neg a \lor c) \;\equiv\; b \lor c$$

Grammar Solving

- base solution on "grammar consequent"

- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations

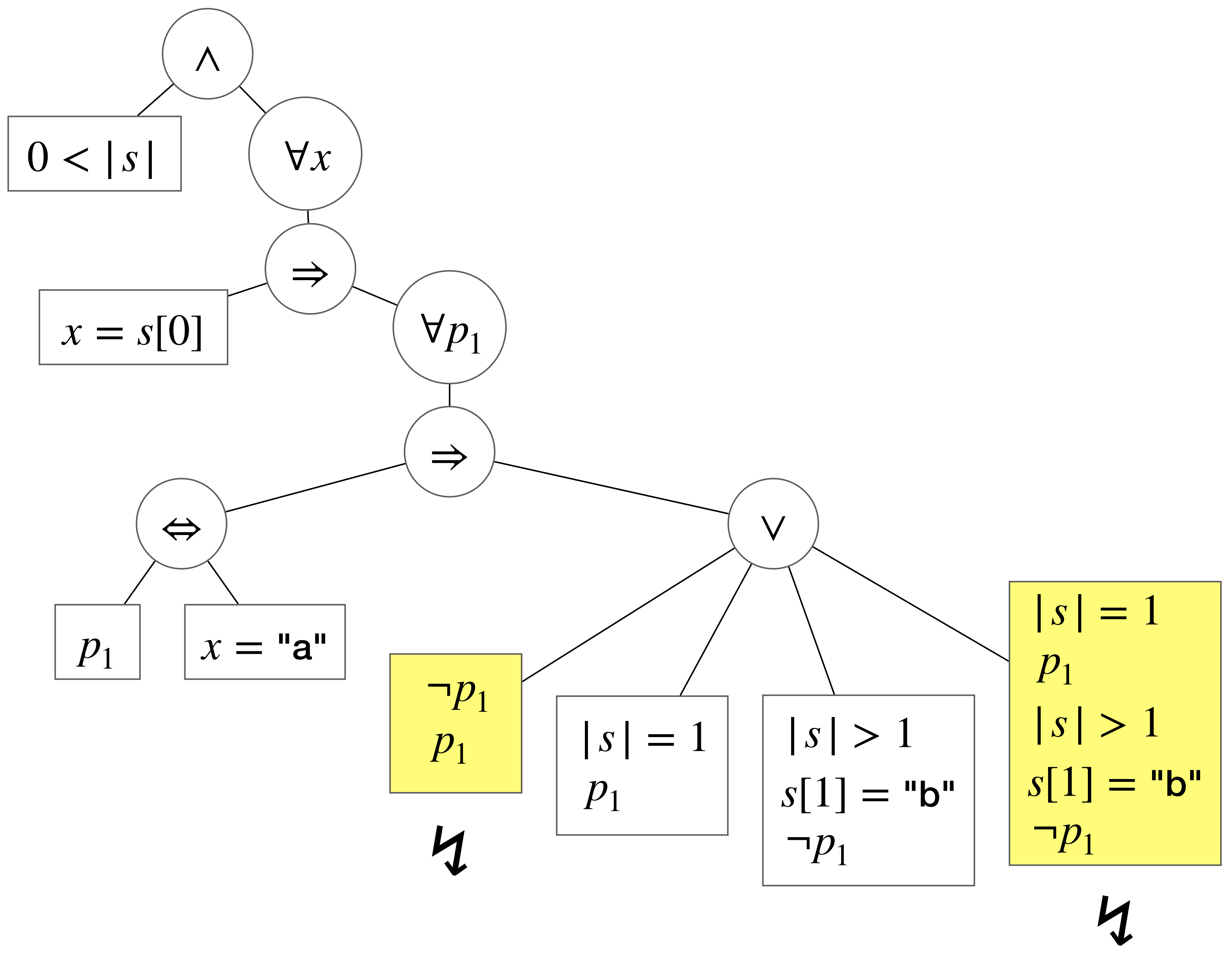


$$(a \vee b) \wedge (\neg a \vee c) \;\equiv\; b \vee c$$

Grammar Solving

- base solution on "grammar consequent"

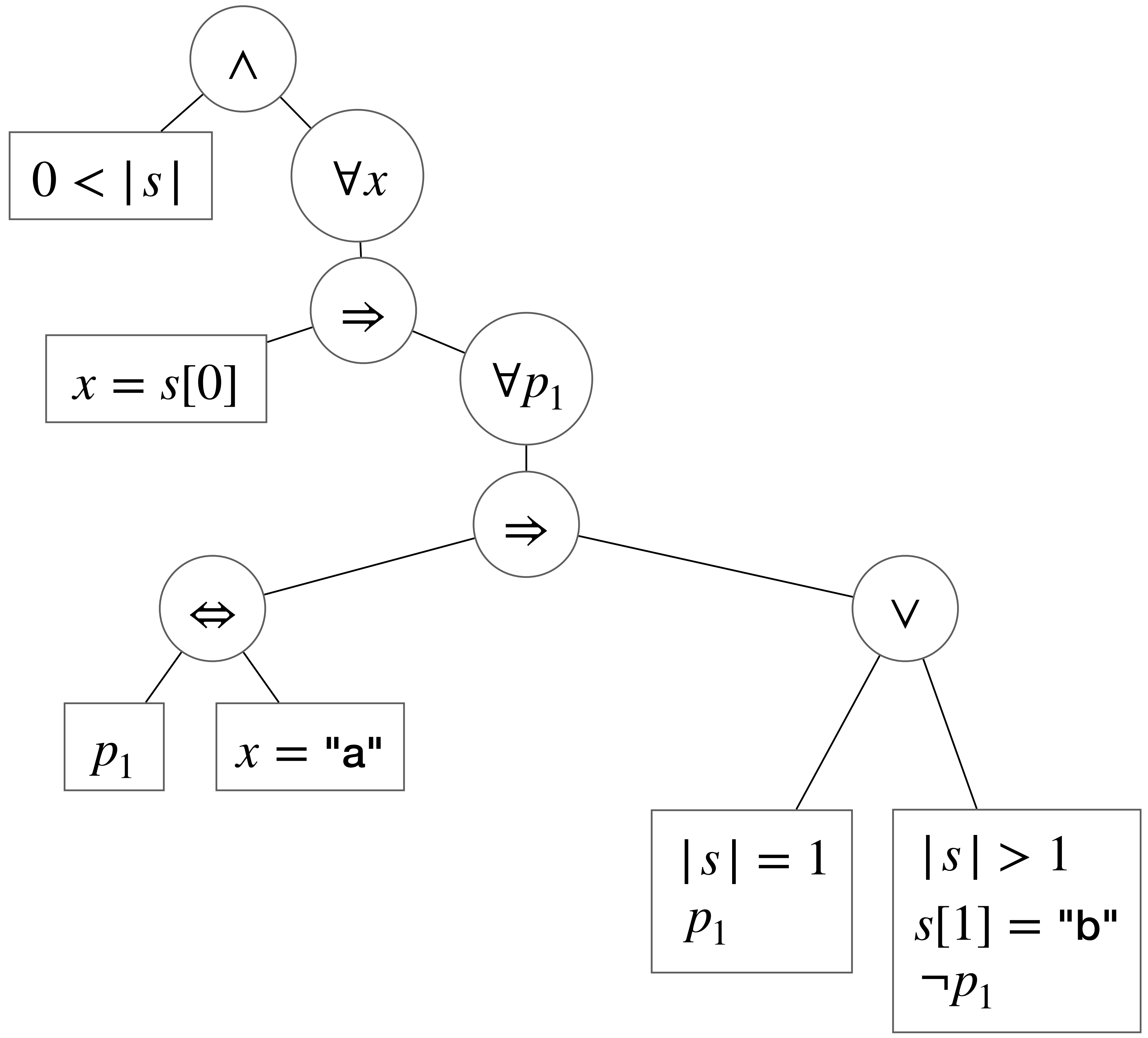
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Grammar Solving

- base solution on “grammar consequent”

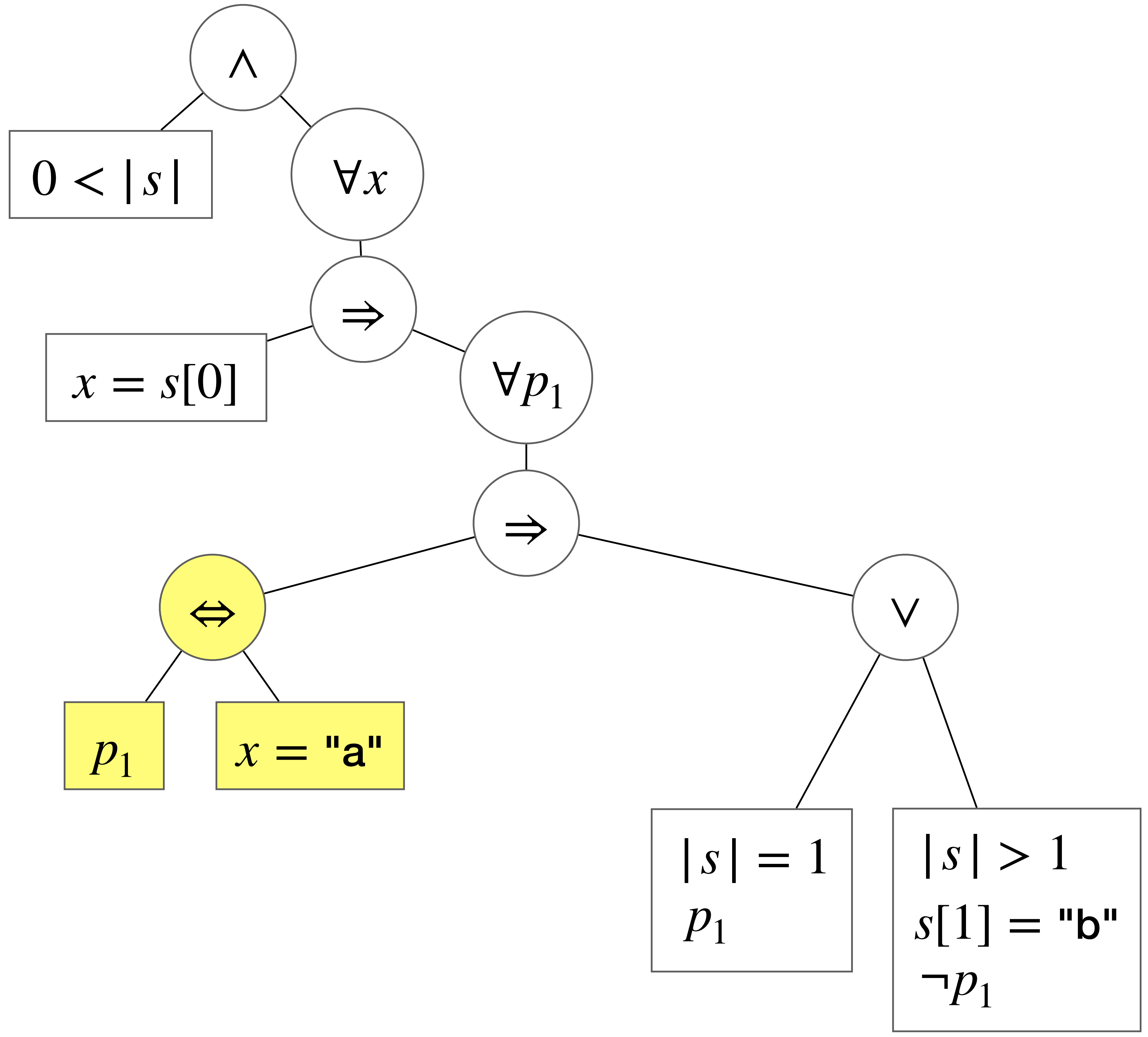
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Grammar Solving

- base solution on “grammar consequent”

- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations

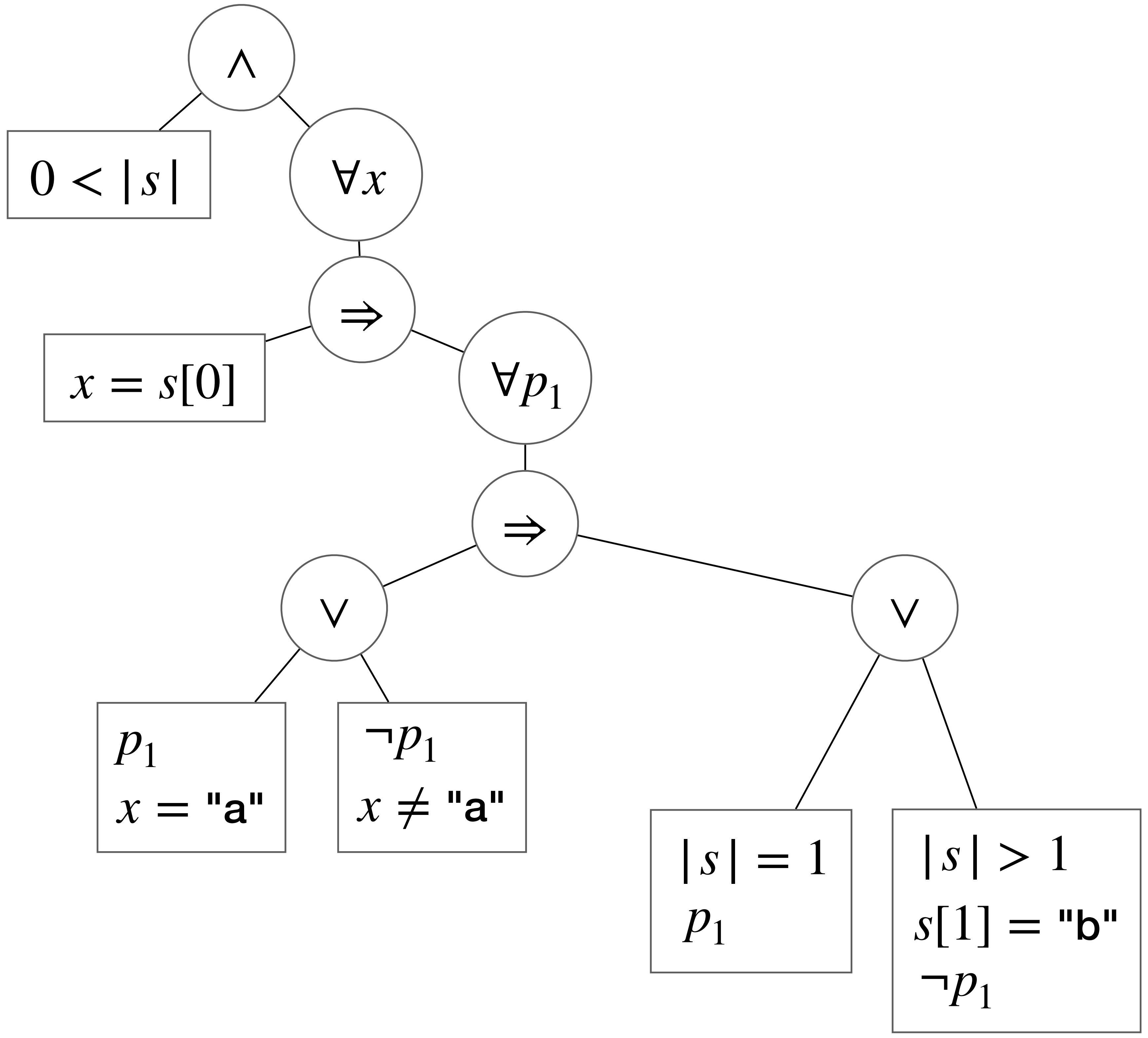


$$a \Leftrightarrow b \;\equiv\; (\neg a \wedge \neg b) \vee (a \wedge b)$$

Grammar Solving

- base solution on “grammar consequent”

- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations

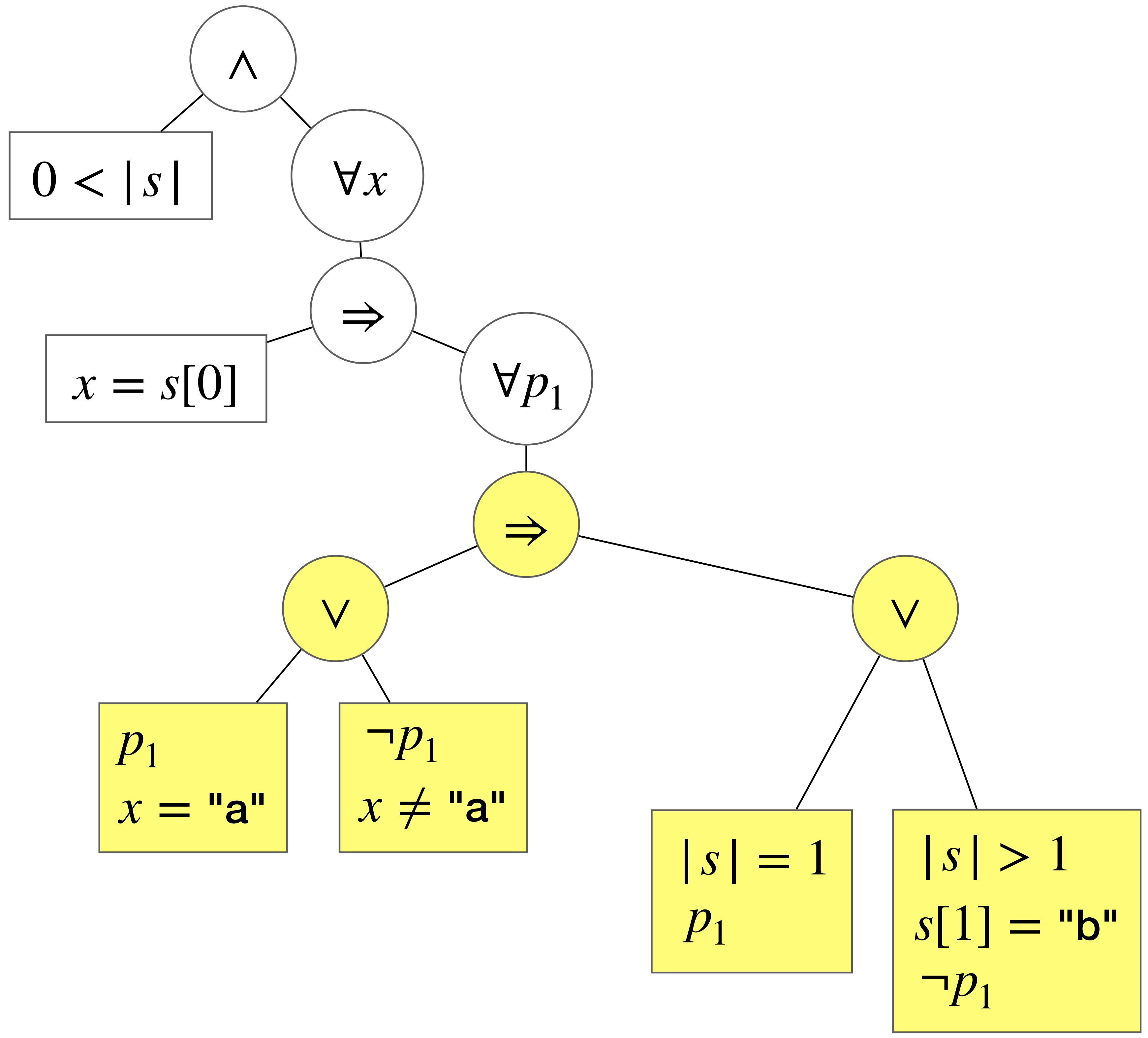


$$a \Leftrightarrow b \;\equiv\; (\neg a \wedge \neg b) \vee (a \wedge b)$$

Grammar Solving

- base solution on “grammar consequent”

- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



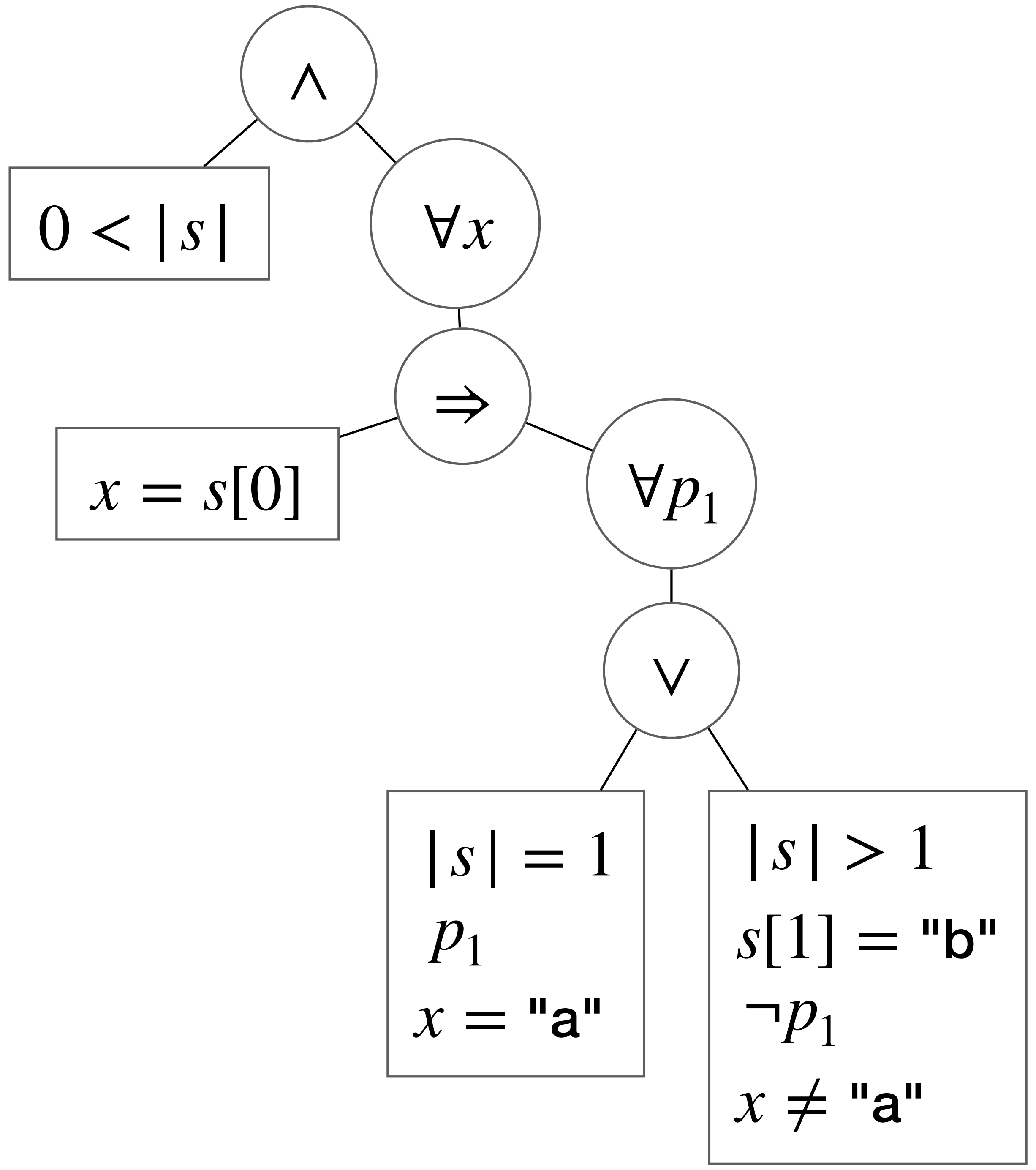
$$(a \vee b) \Rightarrow c \;\equiv\; (a \Rightarrow c) \vee (b \Rightarrow c)$$

$$a \Rightarrow b \;\equiv\; \neg a \vee (a \sqcap b)$$

Grammar Solving

- base solution on "grammar consequent"

- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



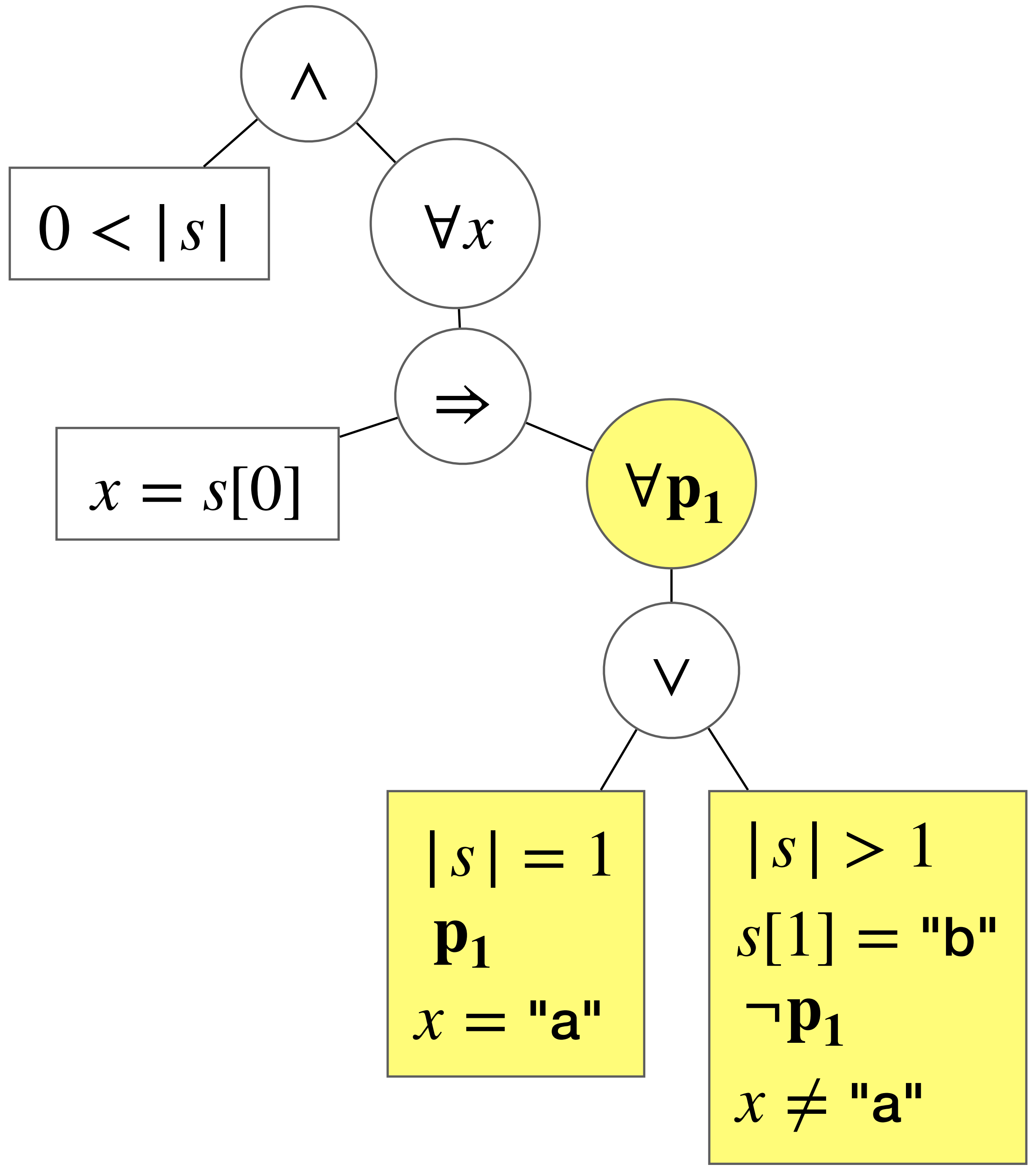
$$(a \vee b) \Rightarrow c \;\equiv\; (a \Rightarrow c) \vee (b \Rightarrow c)$$

$$a \Rightarrow b \;\equiv\; \neg a \vee (a \sqcap b)$$

Grammar Solving

- base solution on “grammar consequent”

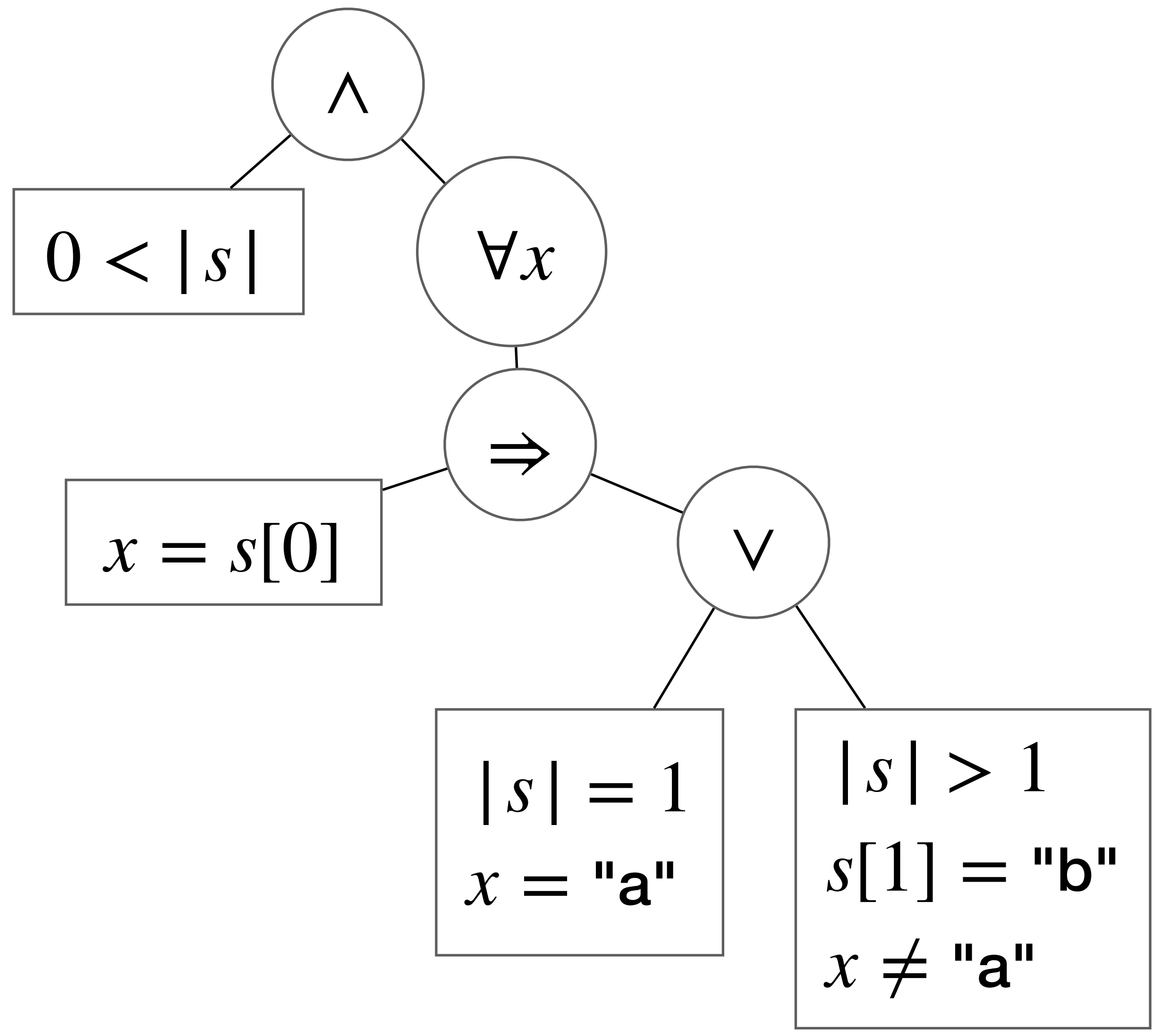
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Grammar Solving

- base solution on “grammar consequent”

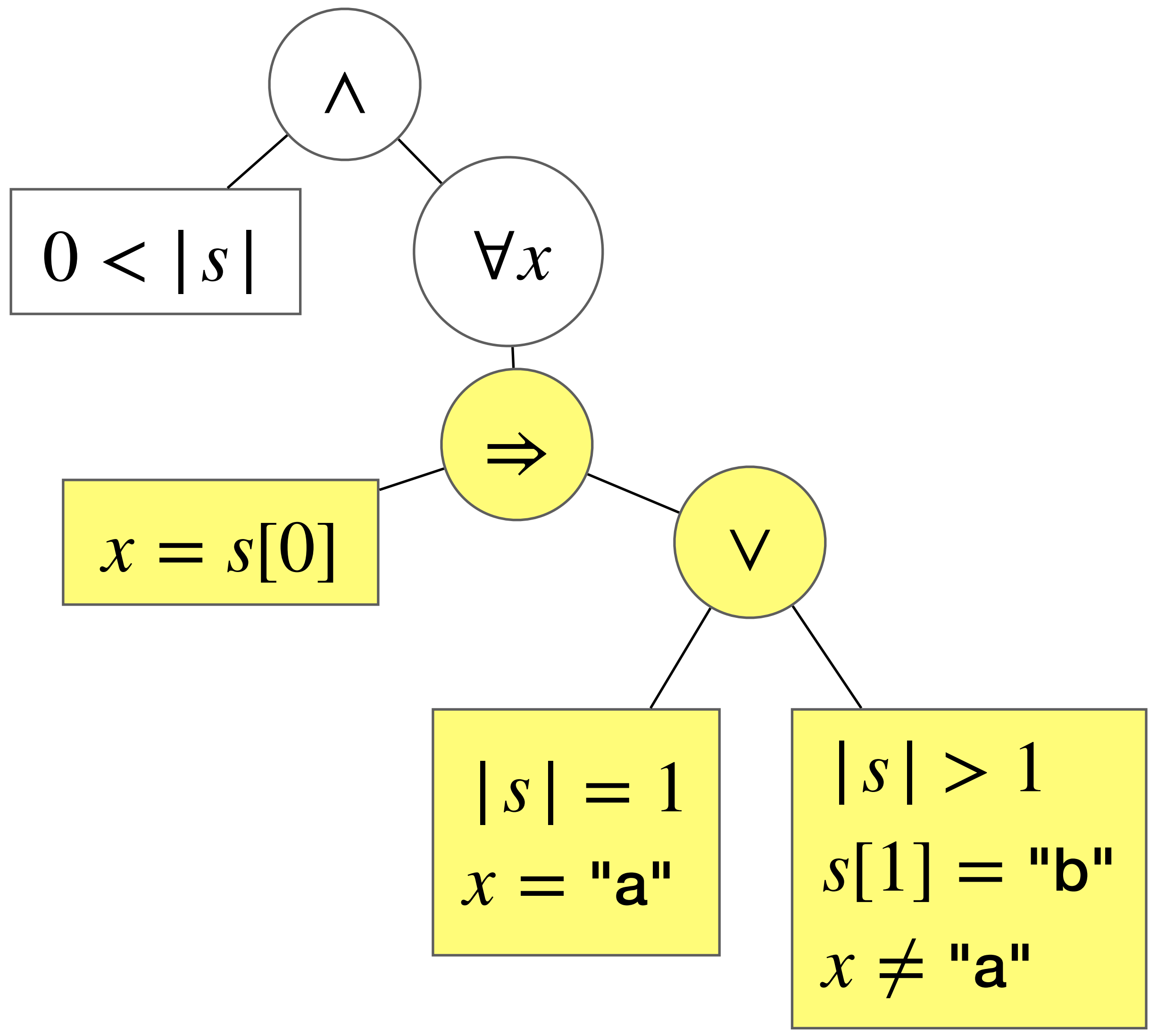
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Grammar Solving

- base solution on "grammar consequent"

- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations

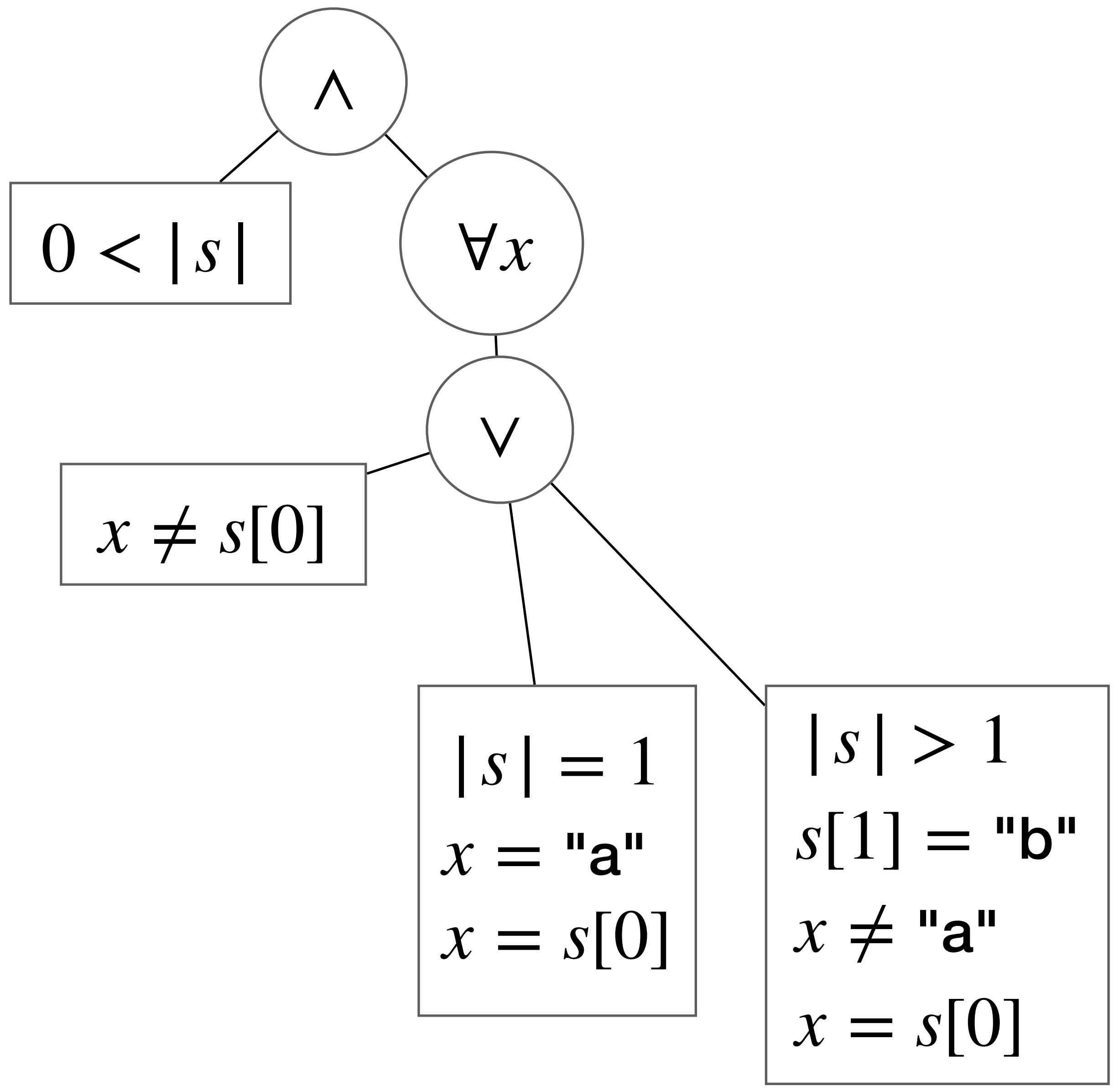


$$a \Rightarrow b \;\equiv\; \neg a \vee (a \sqcap b)$$

Grammar Solving

- base solution on “grammar consequent”

- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations

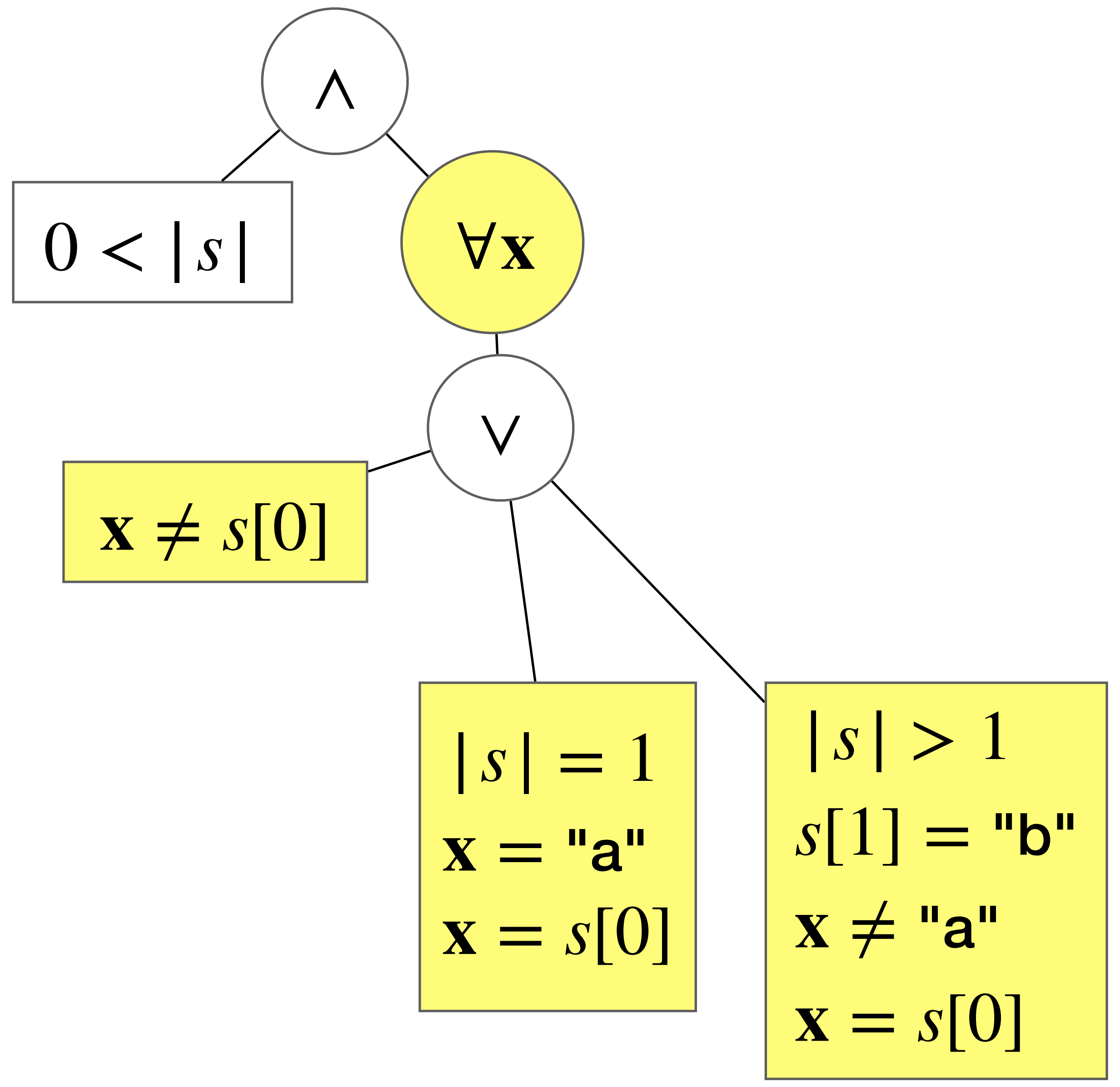


$$a \Rightarrow b \;\equiv\; \neg a \vee (a \sqcap b)$$

Grammar Solving

- base solution on “grammar consequent”

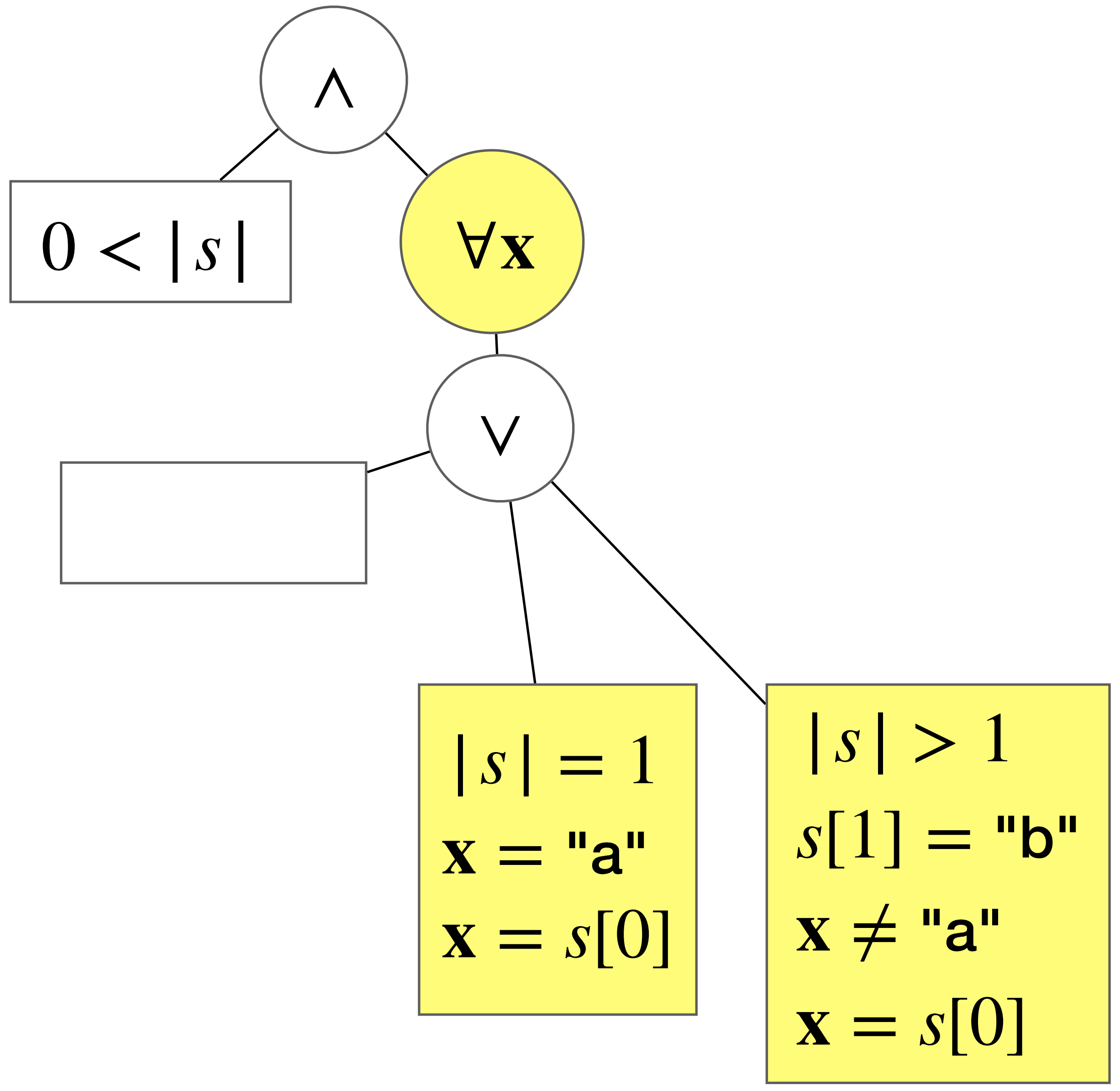
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Grammar Solving

- base solution on “grammar consequent”

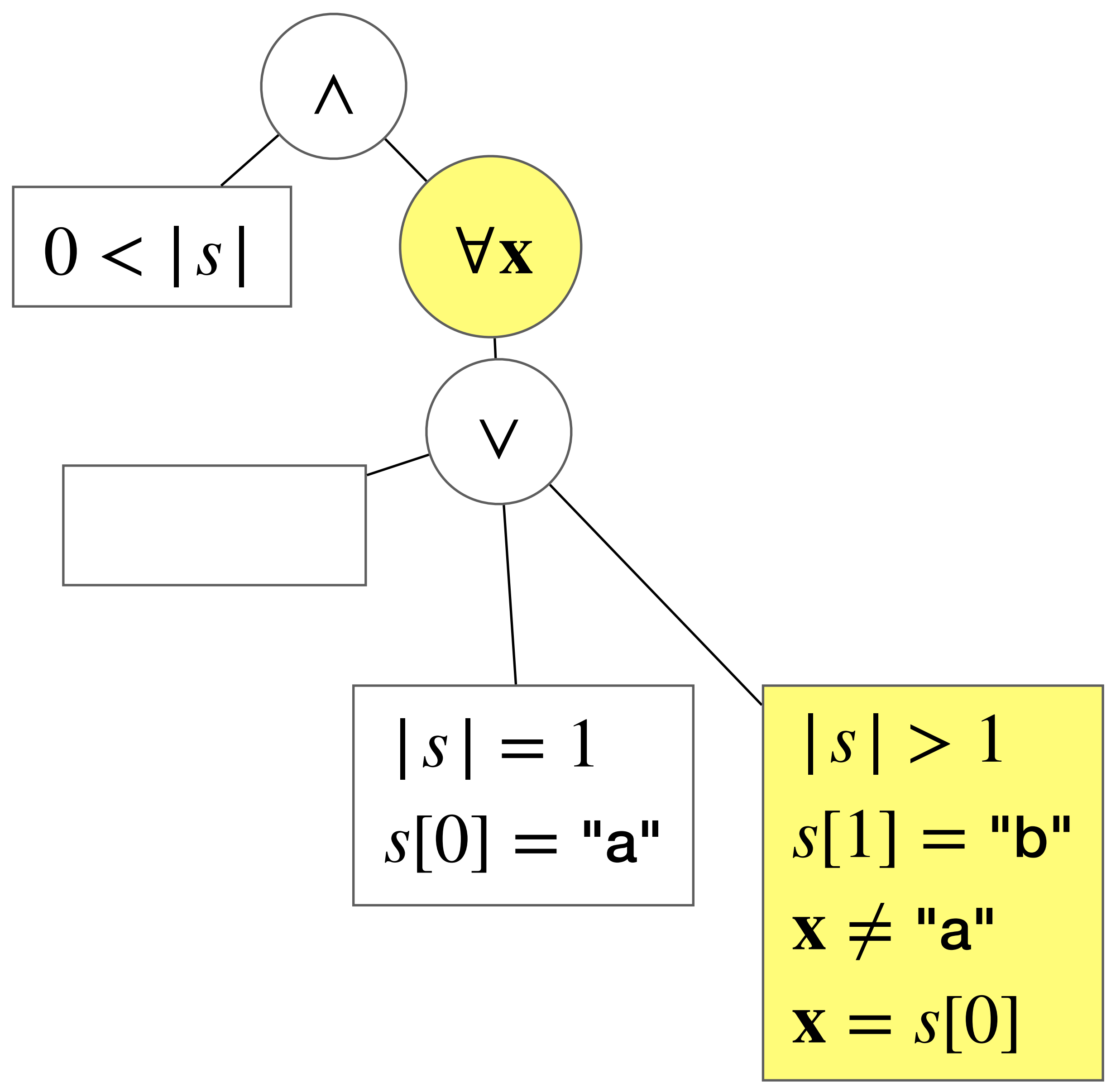
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Grammar Solving

- base solution on “grammar consequent”

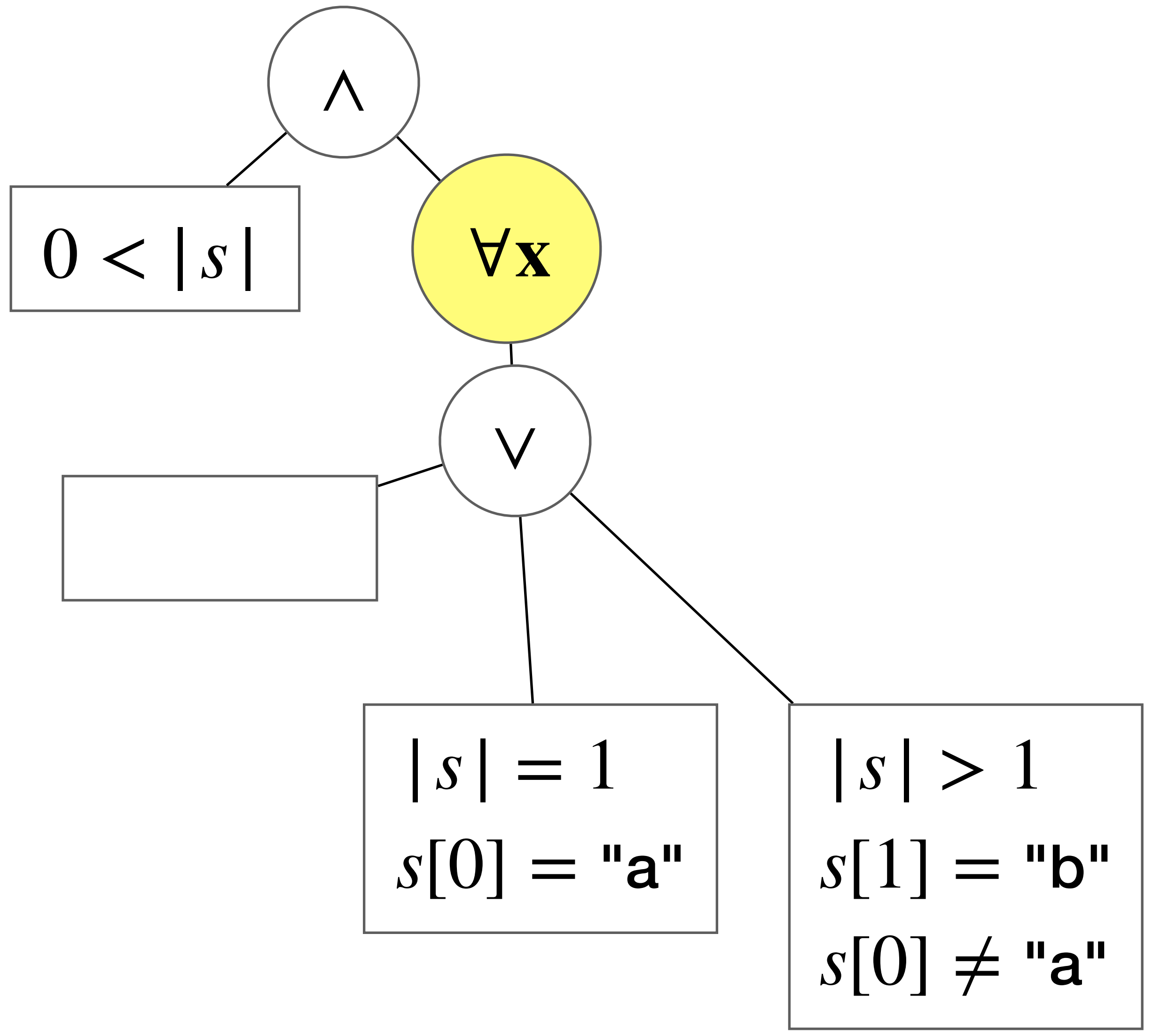
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Grammar Solving

- base solution on “grammar consequent”

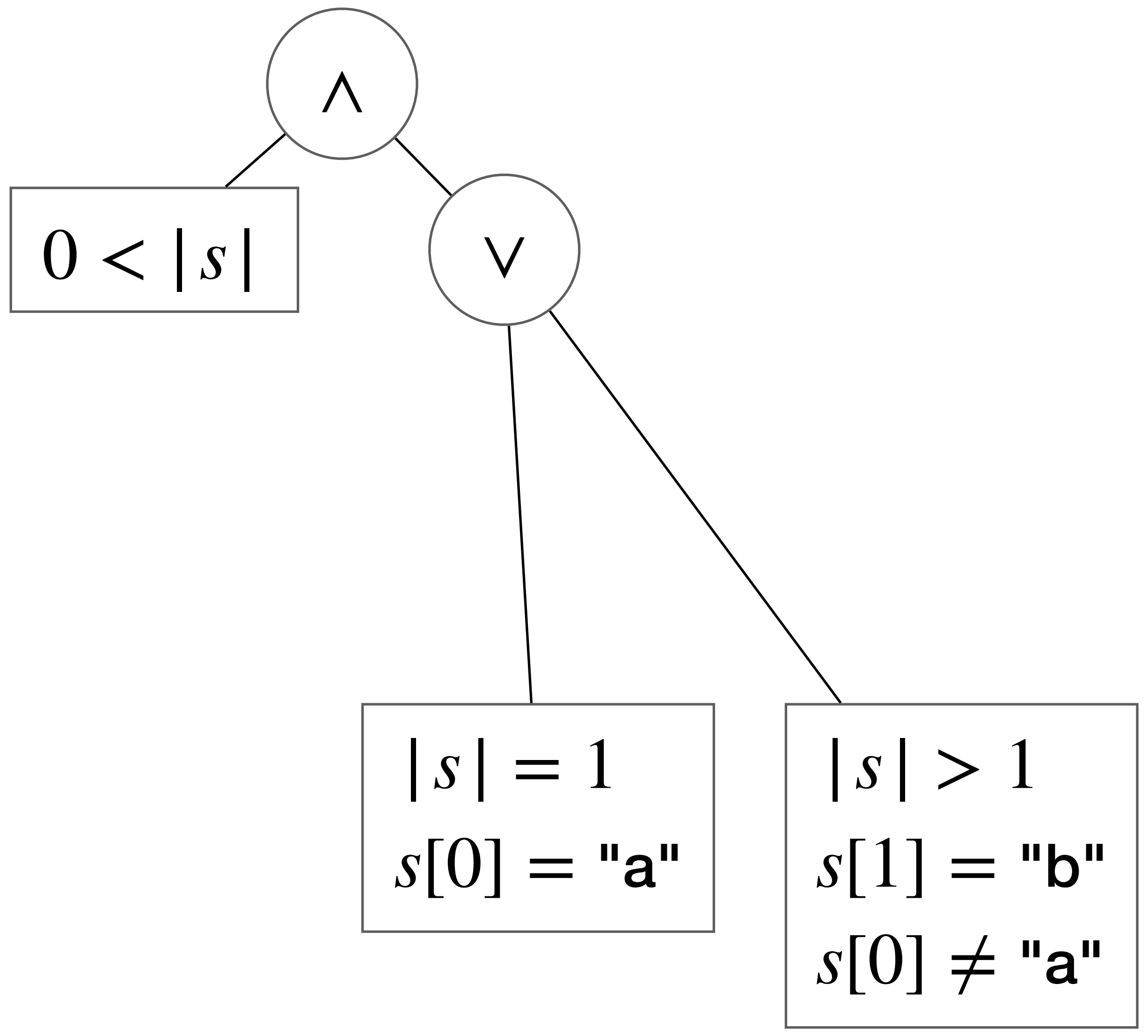
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Grammar Solving

- base solution on “grammar consequent”

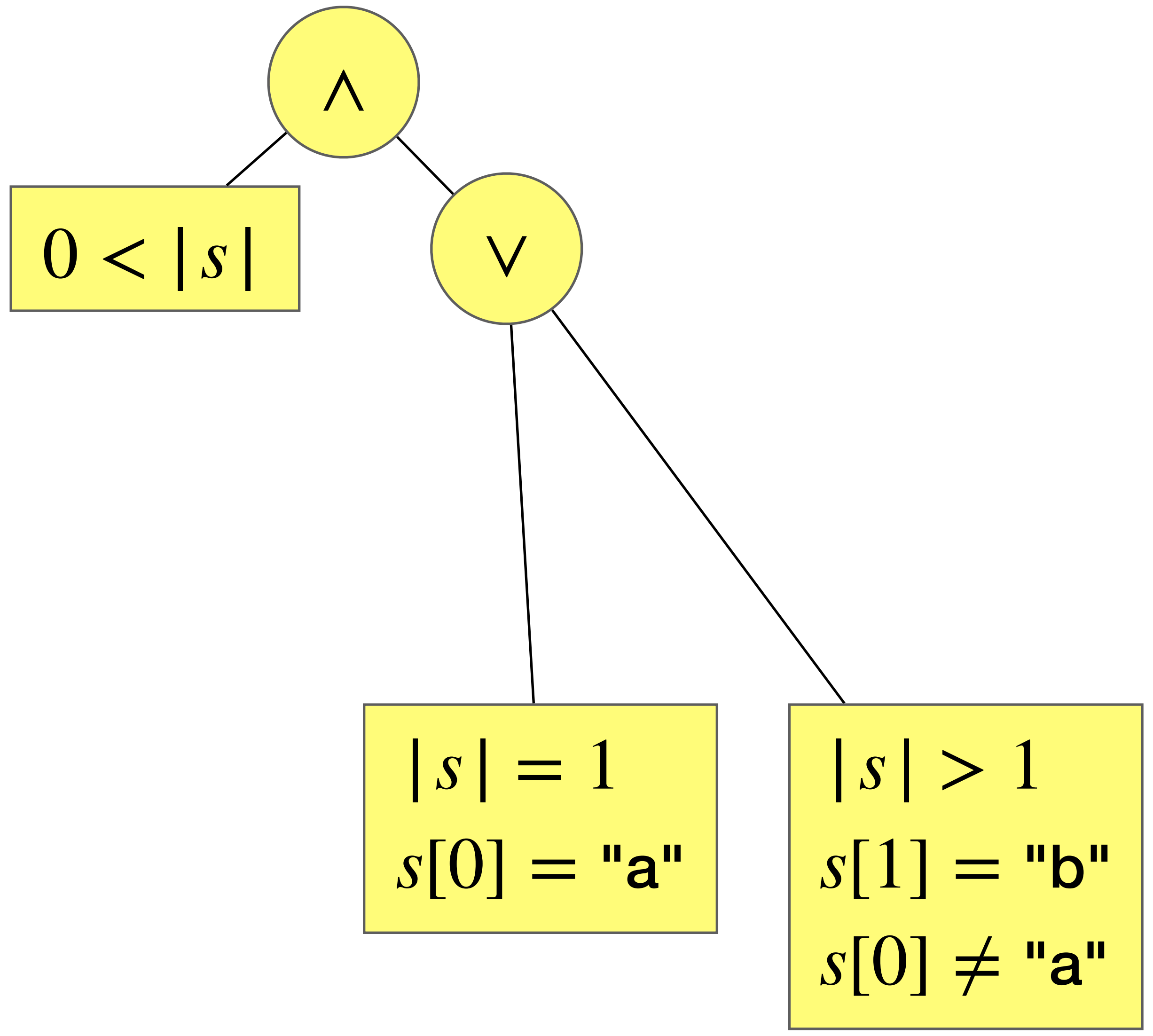
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Grammar Solving

- base solution on “grammar consequent”

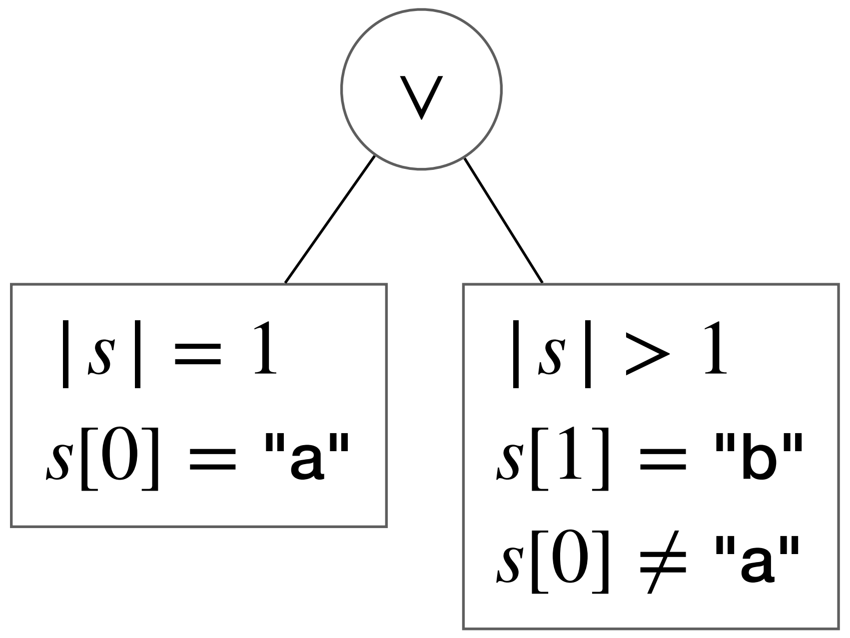
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Grammar Solving

- base solution on “grammar consequent”

- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations



Enjoy your grammar!

- apply string predicate in verification template to continue type checking
- present grammar to user or applications

minimal DNF string predicate

$\vee$

$\lvert s \rvert = 1$	$\lvert s \rvert > 1$
$s[0] = $ **"a"**	$s[1] = $ **"b"**
	$s[0] \neq$ **"a"**

verification condition

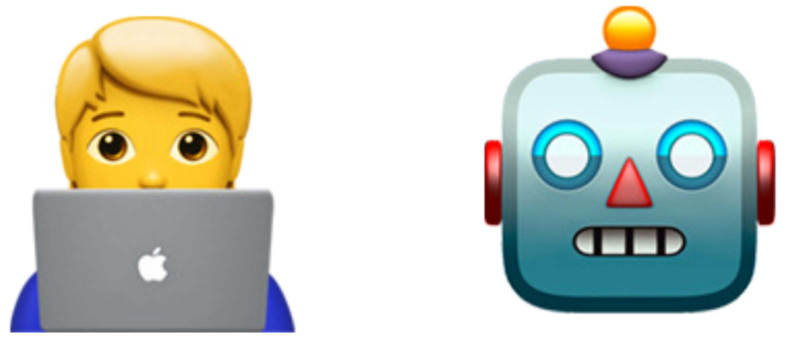
$$\forall s.\ (s = \text{"a"}) \vee (s[0] \neq \text{"a"} \wedge s[1] = \text{"b"}) \Rightarrow$$
$$0 < |s| \wedge \forall x.\ x = s[0] \Rightarrow$$
$$\forall p_1.\ p_1 \Leftrightarrow x = \text{"a"} \Rightarrow$$
$$(p_1 \Rightarrow$$

SMT solving

SAT

grammar

$$s \rightarrow \mathtt{a} \,|\, (\Sigma \backslash \mathtt{a})\mathtt{b}\Sigma^*$$



Status / Future Work

✔ PANINI proof-of-concept

🔜 formalization of grammar solving algorithm

⌛ end-to-end grammar inference system

- library of string operation specifications
- evaluation on corpus of curated ad hoc parser samples
- large-scale mining study of grammars in the wild
- application prototypes (build bot, IDE plugin,…)

⌛ user study on grammar comprehension

Toward Grammar Inference via Refinement Types

https://mcschroeder.github.io/#tyde2022

Michael Schröder
TU Wien
Vienna, Austria
michael.schroeder@tuwien.ac.at

Jürgen Cito
TU Wien
Vienna, Austria
juergen.cito@tuwien.ac.at

Type-Driven Development (TyDe), ICFP 2022
Ljubljana, Slovenia

TU WIEN Informatics