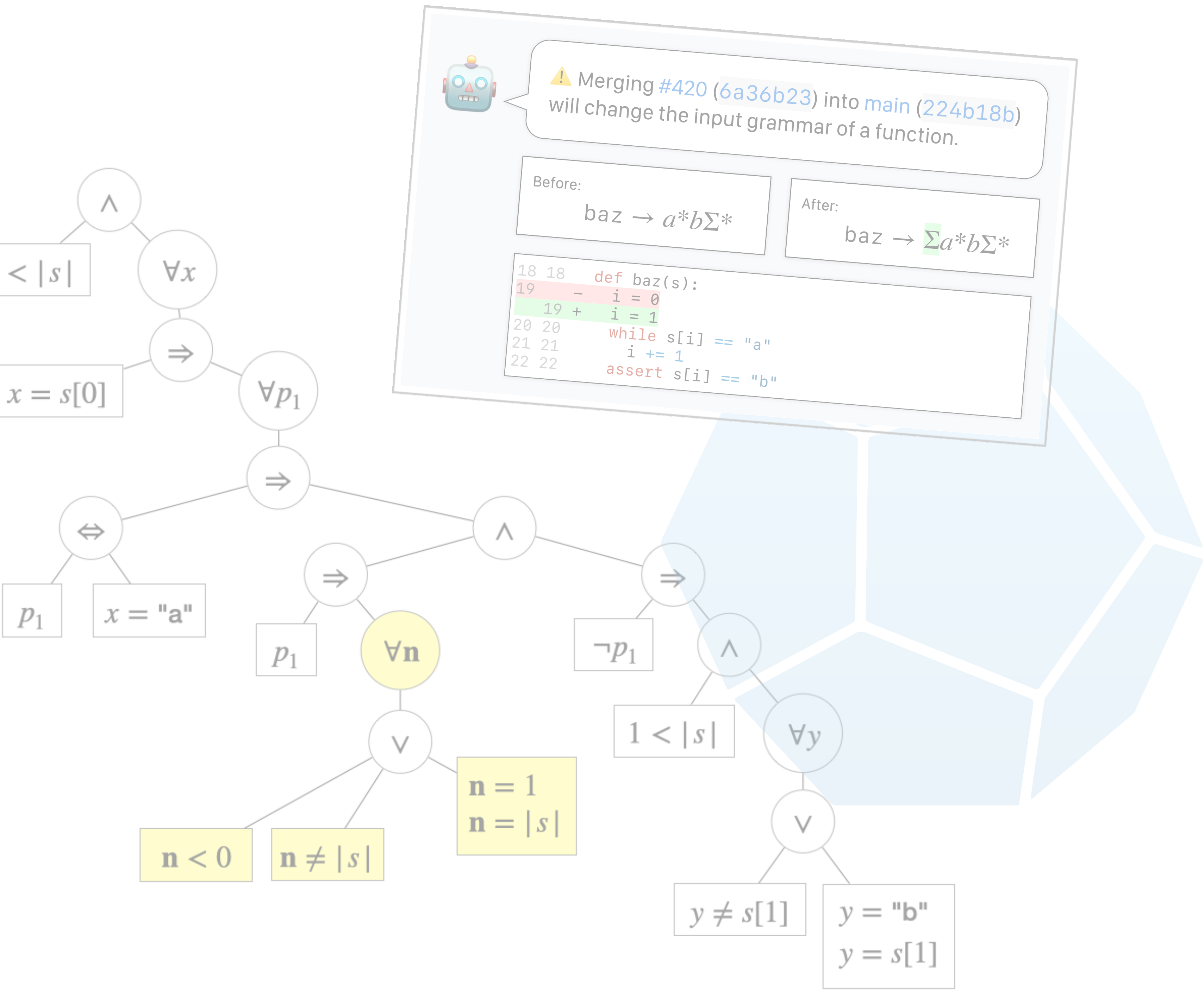
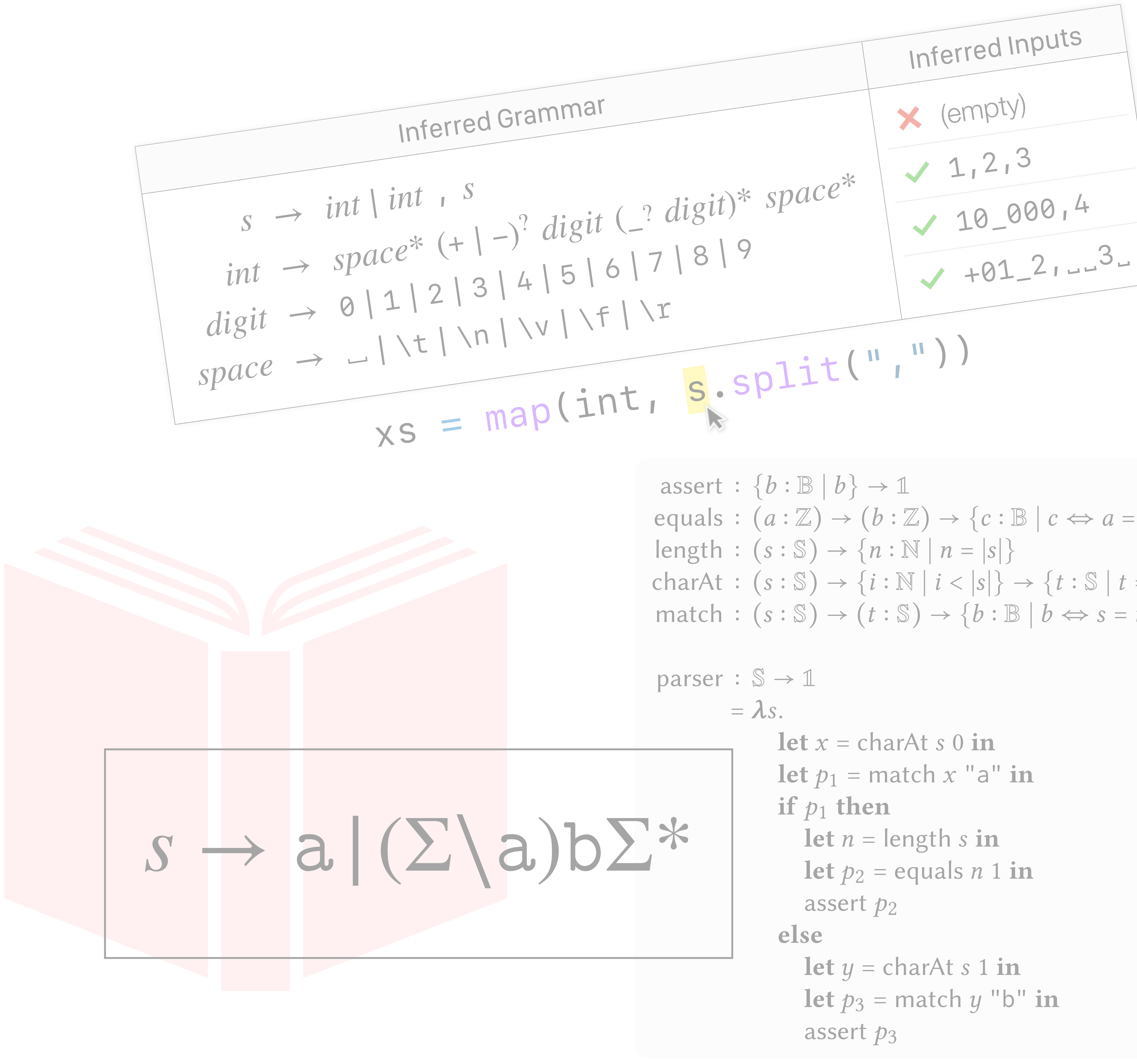
# Grammar Inference for Ad Hoc Parsers

https://mcschroeder.github.io/#splash2022





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


**Doctoral Symposium, SPLASH 2022**

**Tāmaki Makaurau, Aotearoa**
**Auckland, New Zealand**

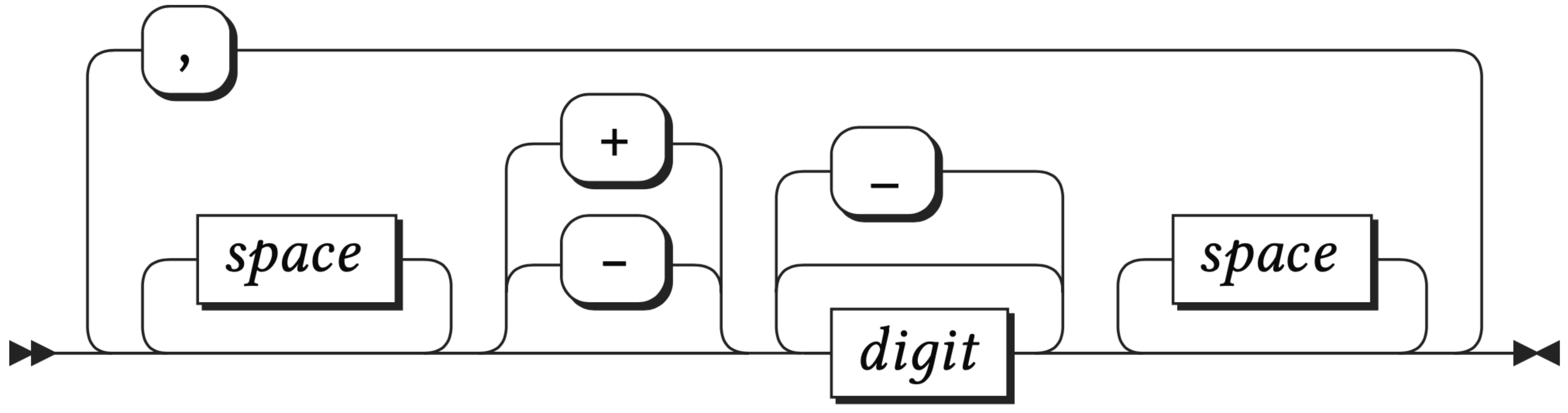
TU WIEN Informatics

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

"1,2,3"

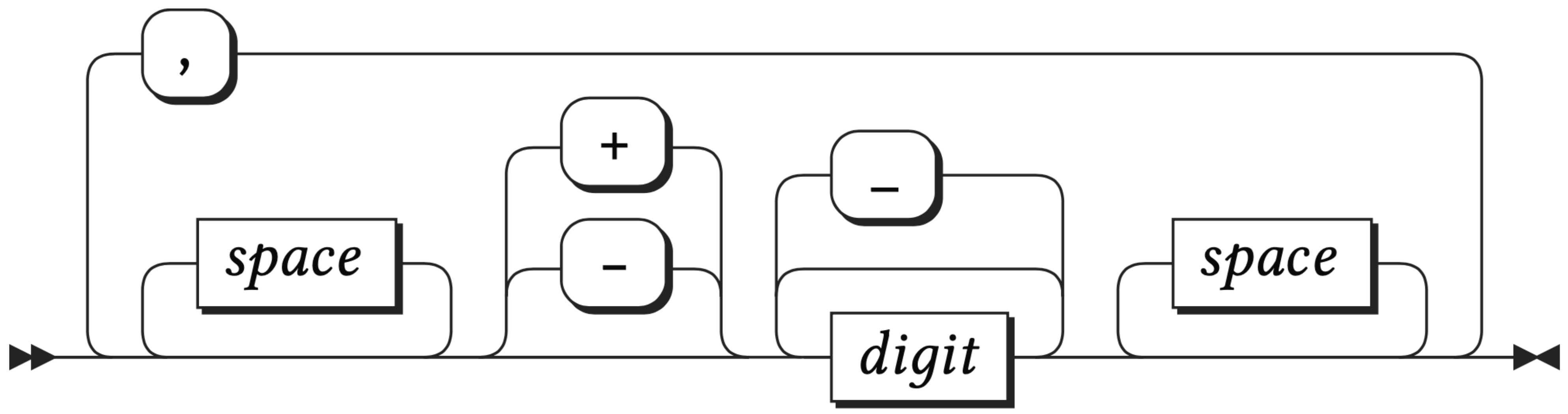
```
xs = map(int, s.split(","))
```

[1,2,3]




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

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




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

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

# Parser : Grammar ≈ Function : Type

# Type Inference

String

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

[Int]

# Type Inference
# + Grammar Inference

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

String{•}

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

[Int]

# Interactive Documentation

NEW

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

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

# Semantic Change Tracking

NEW

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

Before:

baz → $a^*b\Sigma^*$

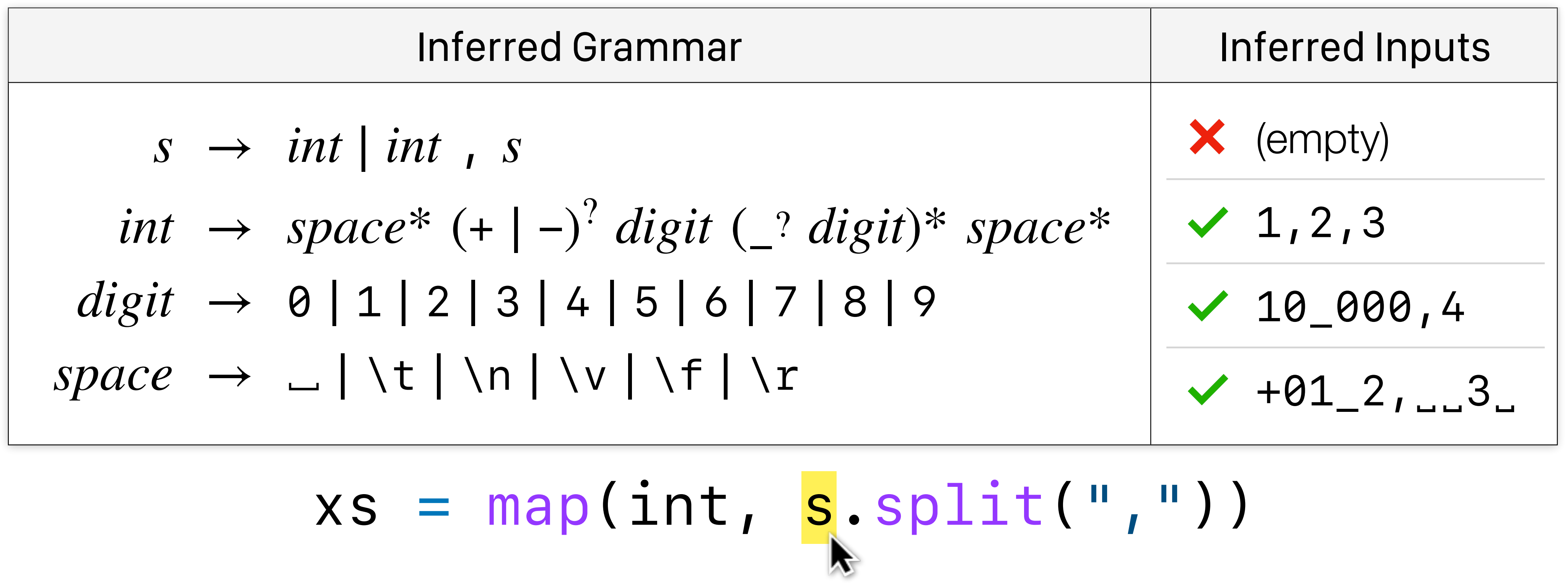
After:

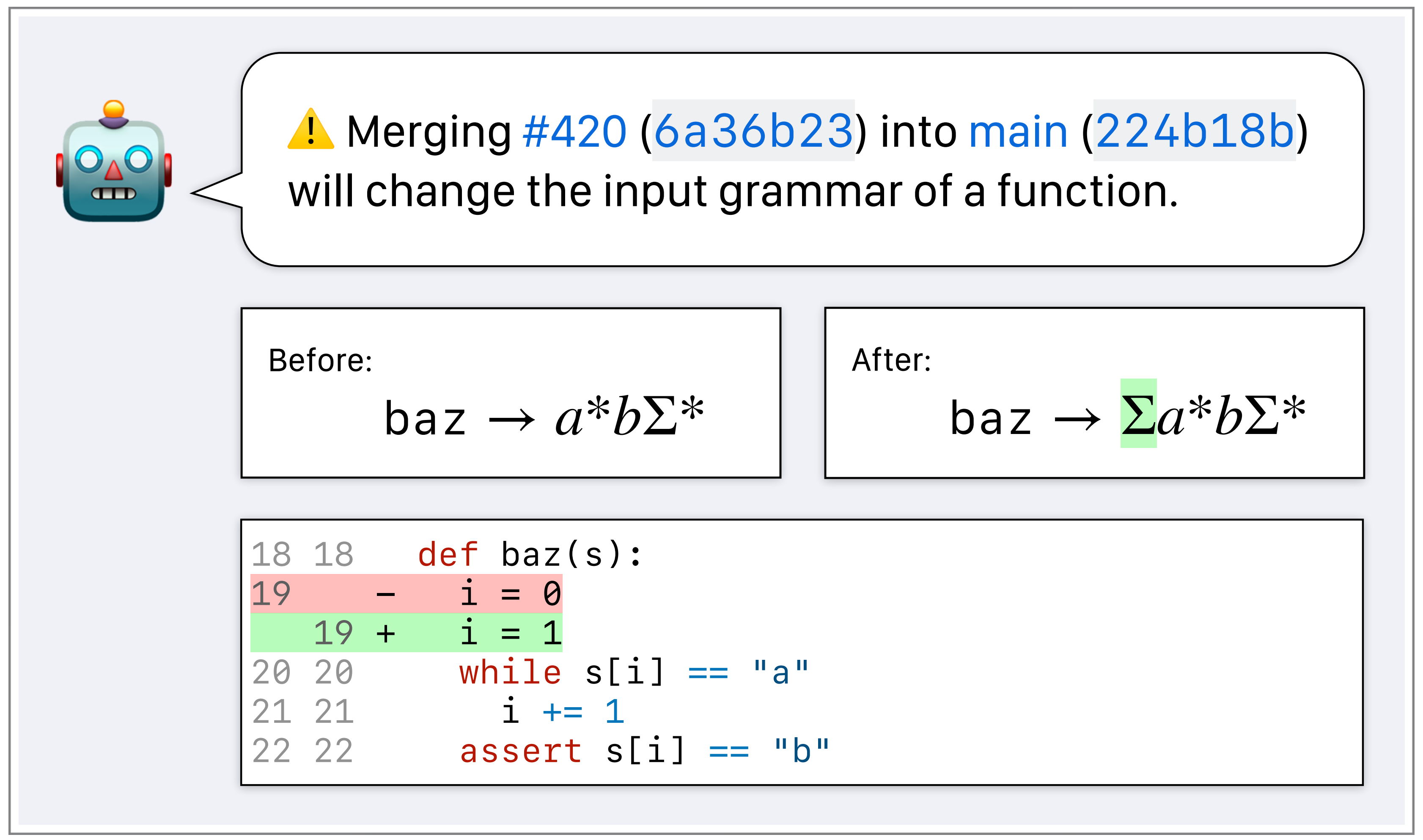
baz → $\Sigma a^*b\Sigma^*$

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

# Applications

- interactive documentation
- semantic change tracking
- grammar-aware refactoring
- parser sketching
- searching for parsers using their grammar
- detecting parser code clones
- grammar-based fuzzing
- …



# ✨ Automatic Grammar Inference ✨

ad hoc parser source

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

?

→

grammar

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

# PANINI

- simple $\lambda$-calculus in A-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

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

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

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 Types 101

## Refined Base Types

$$\{n : \mathsf{int} \mid n \geq 0\}$$

base type

refinement predicate
in a decidable logic
(e.g., QF_UFLIA)

# Refinement Types 101

## Dependent Function Types

$$\text{length} : (s : \text{string}) \rightarrow$$
$$\{n : \text{int} \mid n \geq 0 \land n = |\, s \,|\}$$

output types can
refer to input types

# Refinement Types 101

## Verification Conditions

| | |
|---|---|
| term | $\textbf{let}\ n = \text{length}\ s\ \textbf{in}\ \ldots$ |
| type | $\{n : \text{int} \mid n \geq 0 \wedge n = \lvert s \rvert\}$ |
| verification condition | $\forall n\,.\ n \geq 0 \wedge n = \lvert s \rvert \Rightarrow \ldots$ |

# Refinement Inference

- $\kappa$ variables represent unknown refinements
- most can be solved precisely (e.g., using FUSION)
- existing approaches struggle with “grammar variables”

PANINI program

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

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

verification template

“grammar variable” (constraint over input string) → $\kappa_0(s)$

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


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)

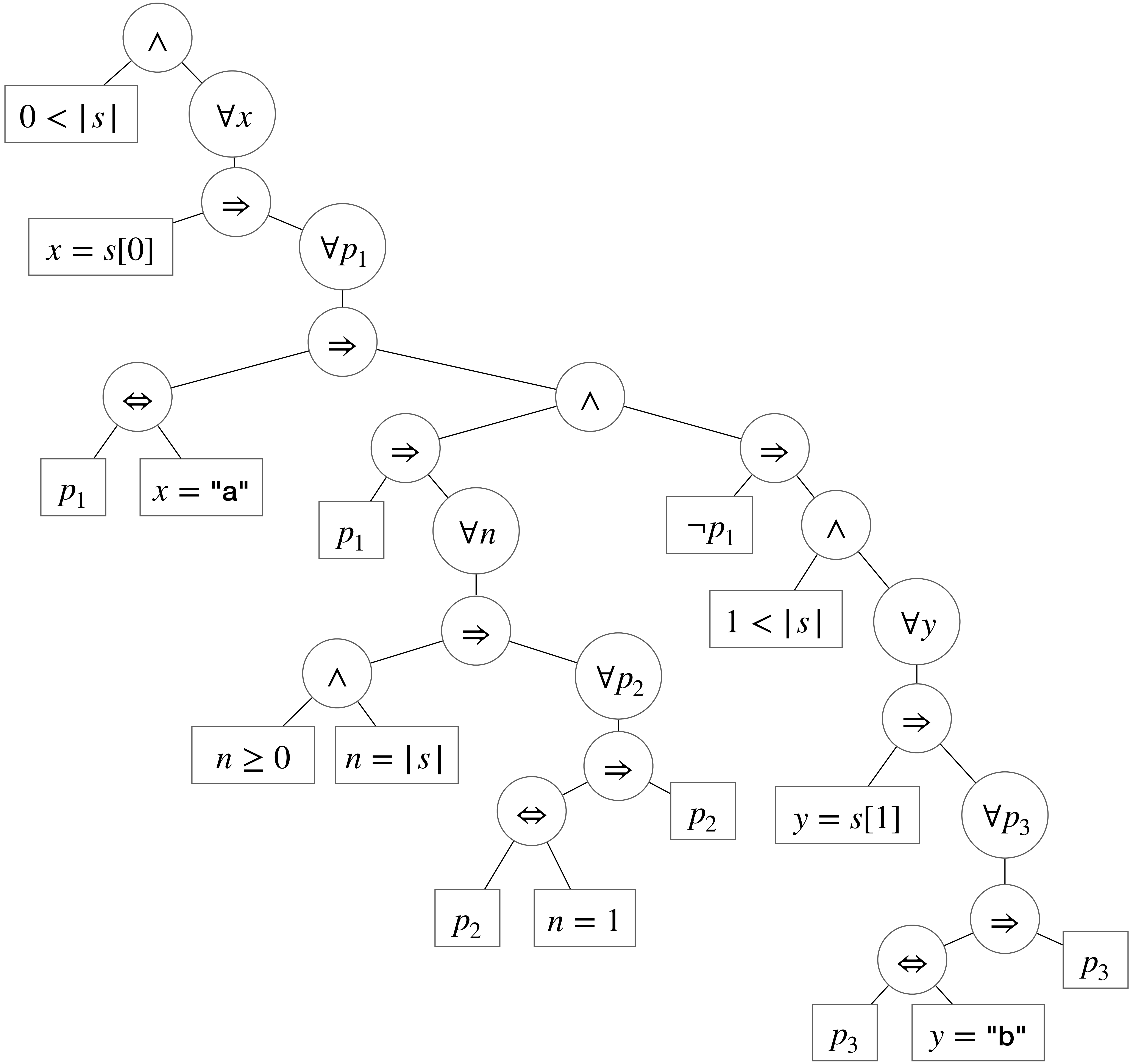
$$
\begin{aligned}
&\forall s.\ \kappa_0(s) \Rightarrow \\
&\quad 0 < |s| \wedge \forall x.\ x = s[0] \Rightarrow \\
&\qquad \forall p_1.\ p_1 \Leftrightarrow x = \texttt{"a"} \Rightarrow \\
&\qquad\quad (p_1 \Rightarrow \\
&\qquad\qquad \forall n.\ n \geq 0 \wedge n = |s| \Rightarrow \\
&\qquad\qquad\quad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

"grammar consequent"

# 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

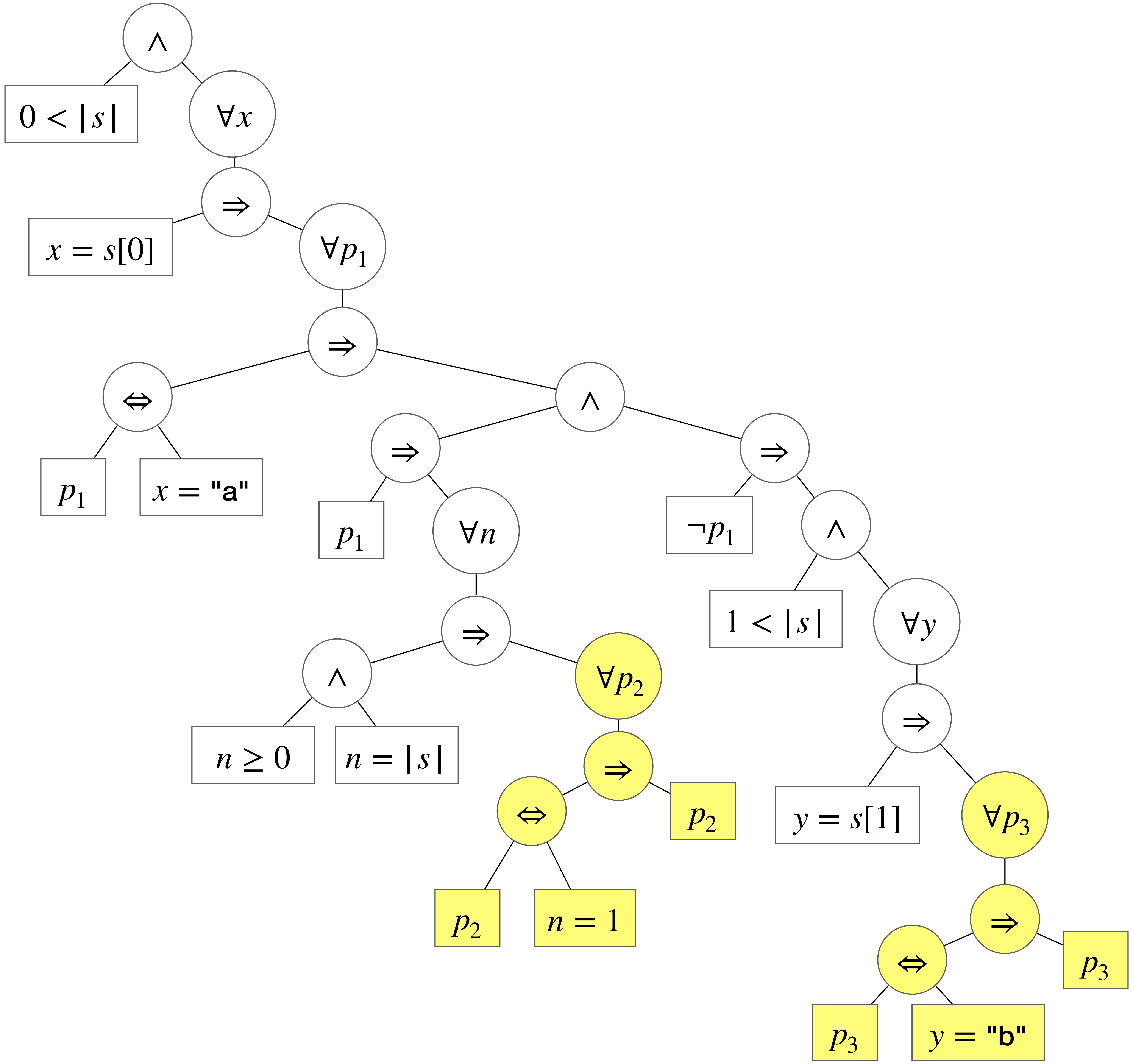


# Grammar Solving

- 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

$$\forall a\,.\,(a \Leftrightarrow b) \Rightarrow a \quad \to \quad b$$

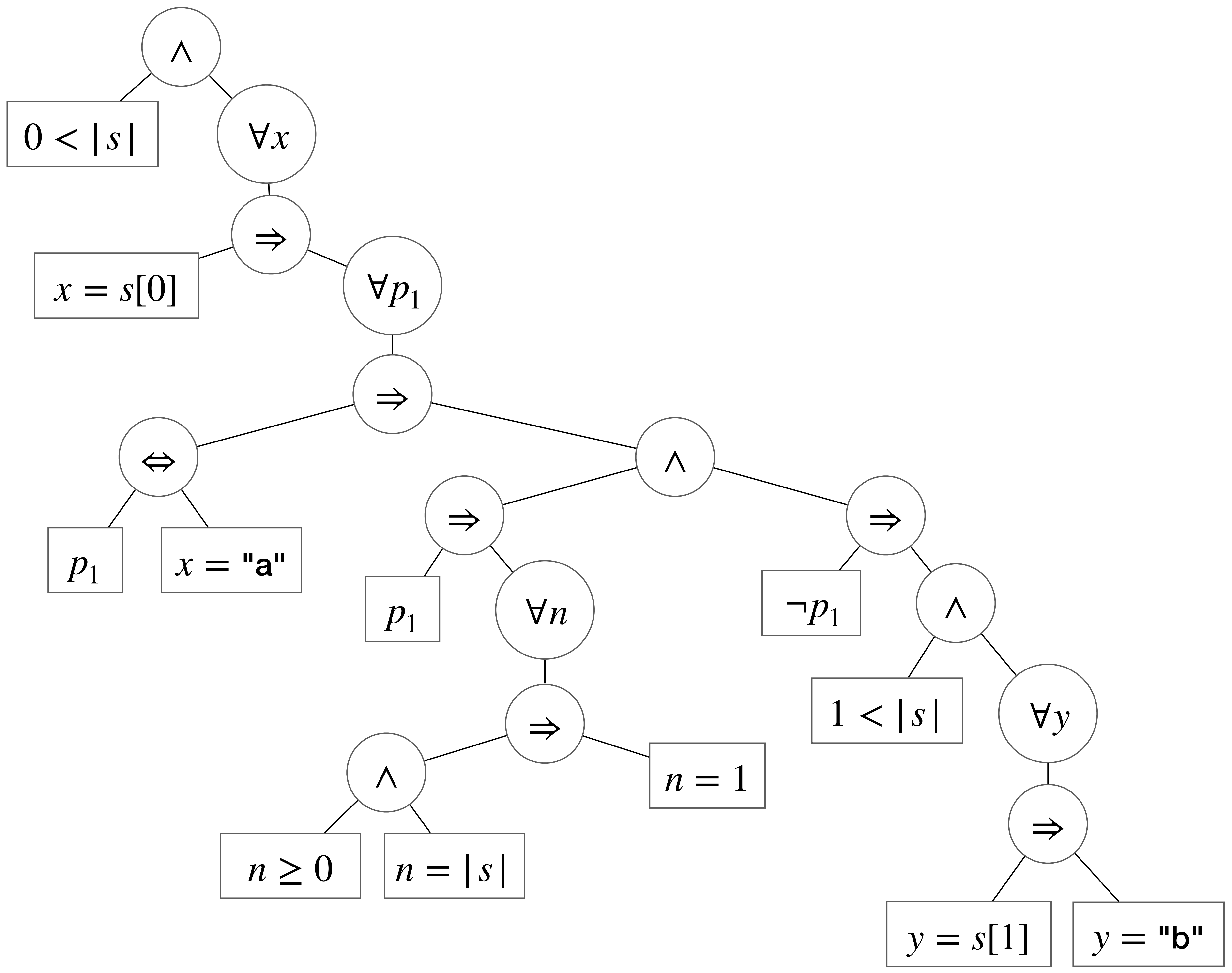


# Grammar Solving

- 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

$$\forall a\,.\,(a \Leftrightarrow b) \Rightarrow a \quad \rightarrow \quad b$$

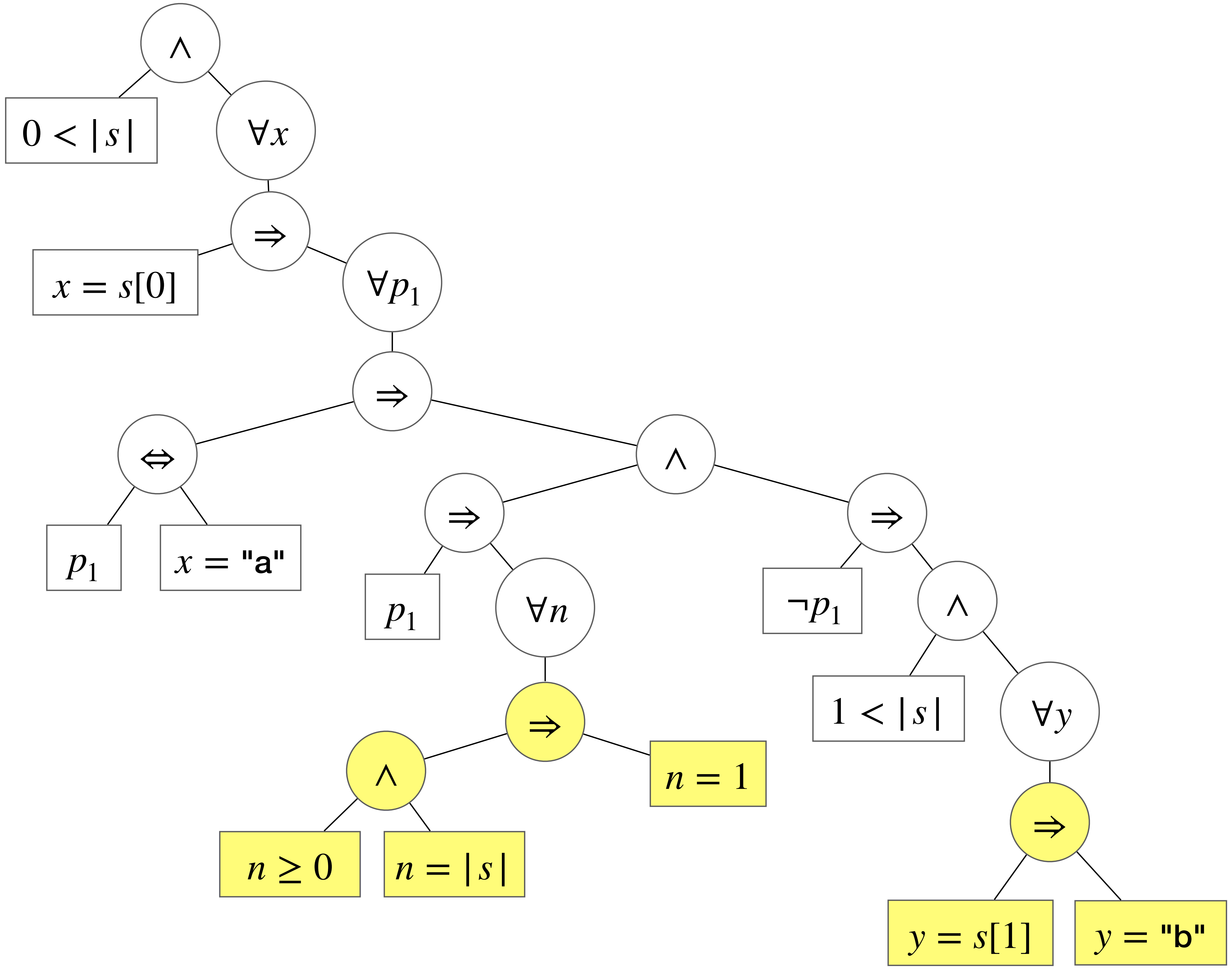


# Grammar Solving

- 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

$$a \Rightarrow b \quad \rightarrow \quad \neg a \vee (a \sqcap b)$$

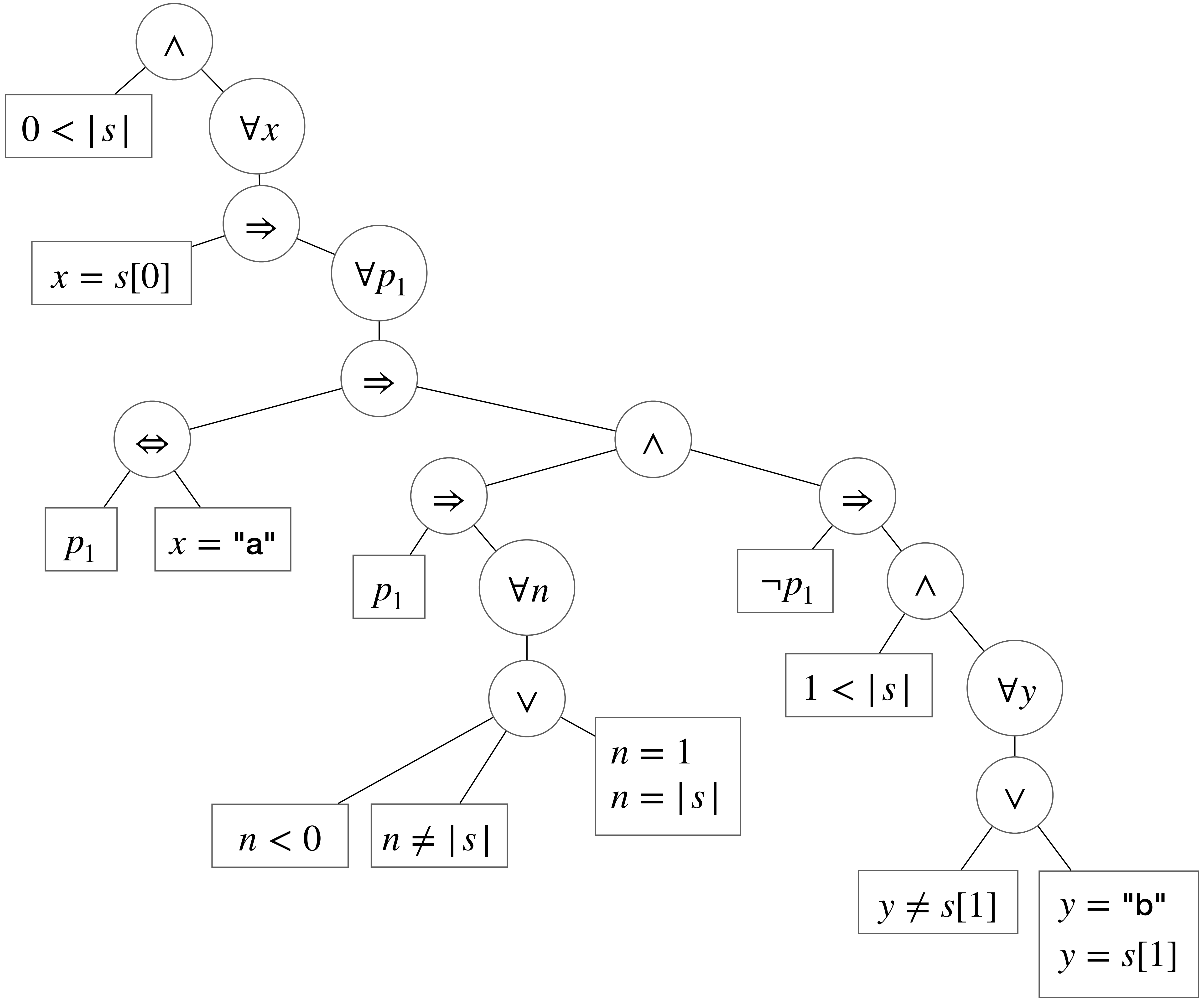


# Grammar Solving

- 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

$$a \Rightarrow b \quad \rightarrow \quad \neg a \vee (a \sqcap b)$$

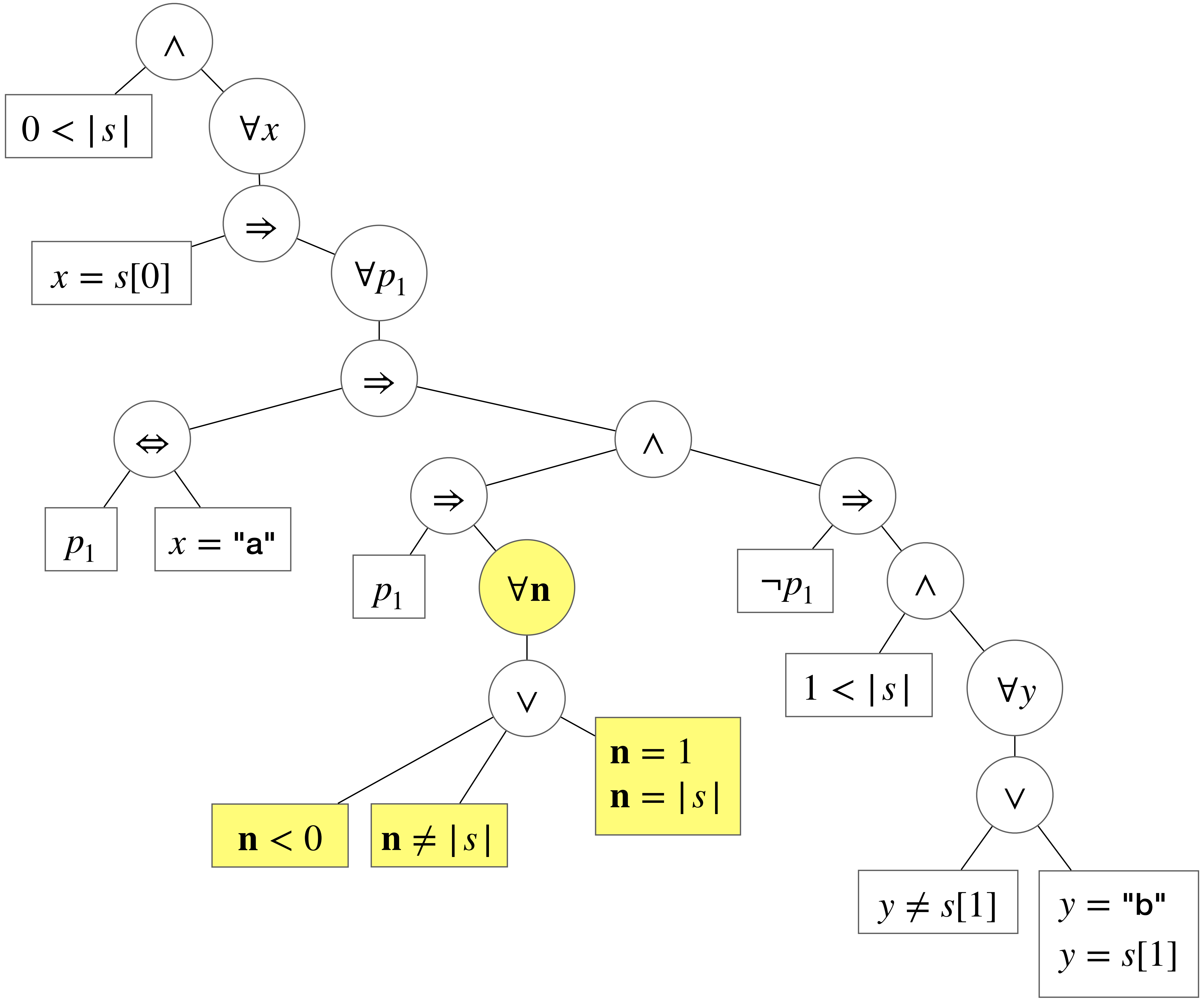


# Grammar Solving

- 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

$\forall x\,.\,\varphi \quad \rightarrow \quad$ resolve $x$ in $\varphi$

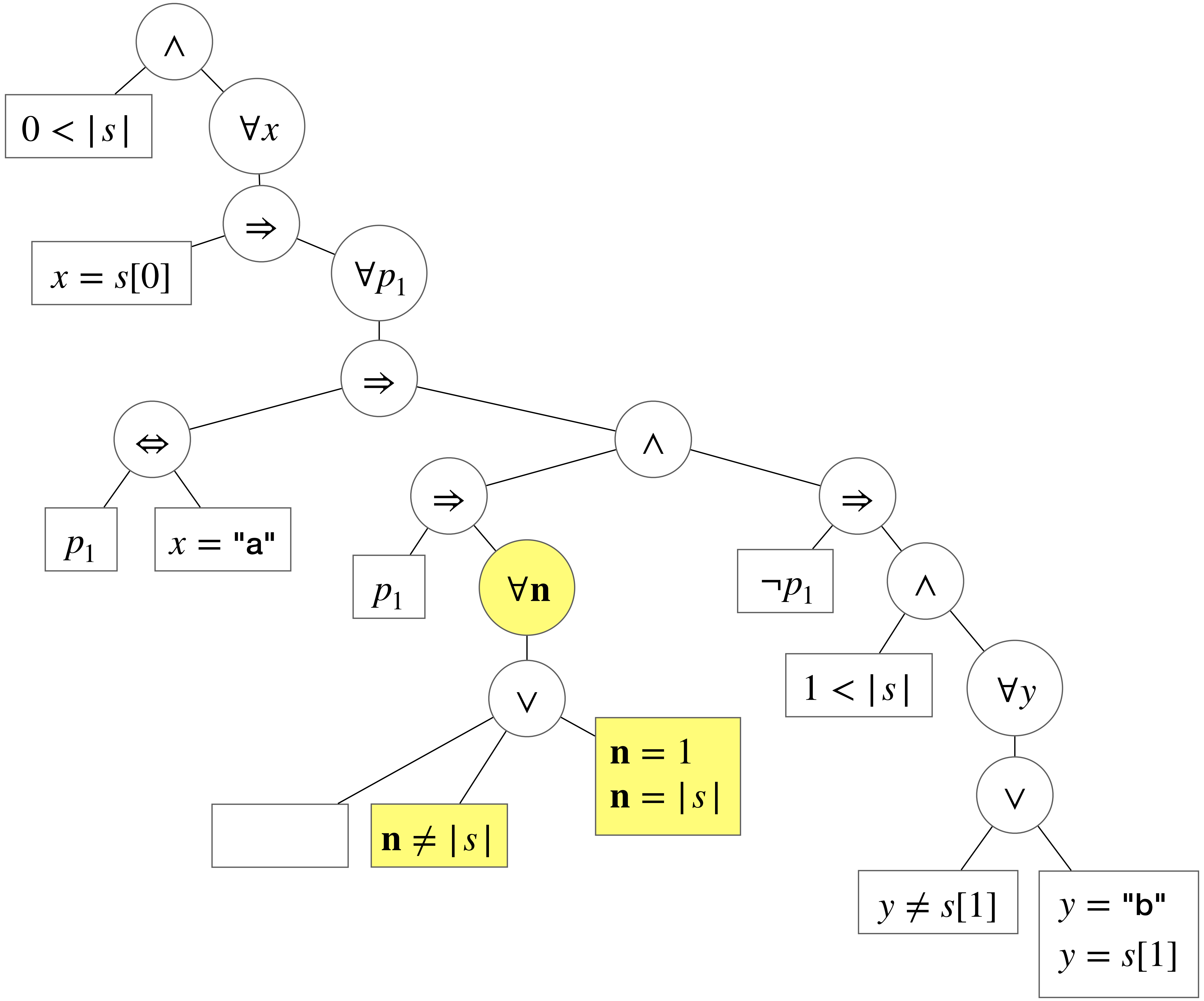


# Grammar Solving

- 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

$\forall x\,.\,\varphi \quad \rightarrow \quad$ resolve $x$ in $\varphi$

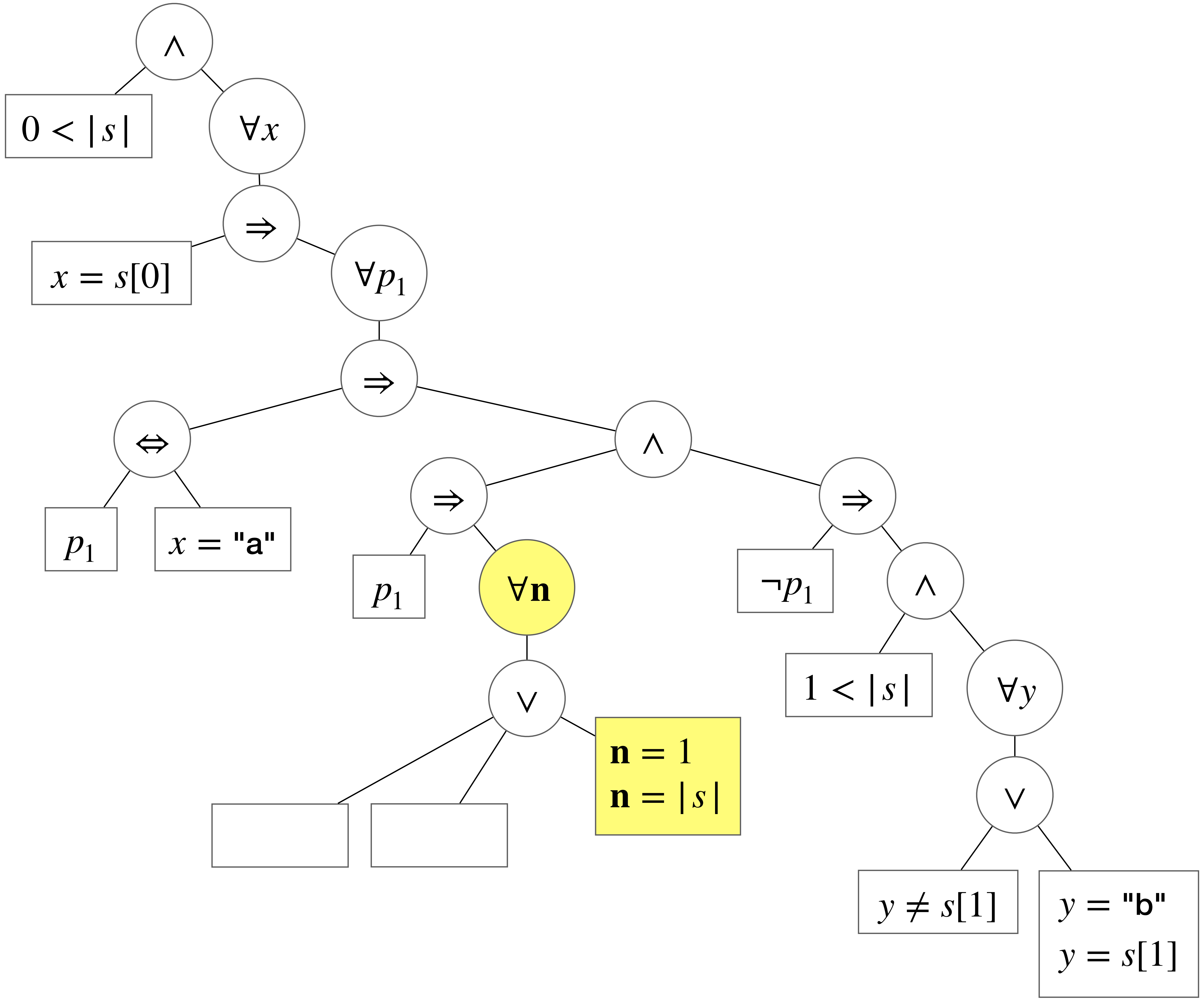


# Grammar Solving

- 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

$\forall x \,.\, \varphi \quad \rightarrow \quad$ resolve $x$ in $\varphi$

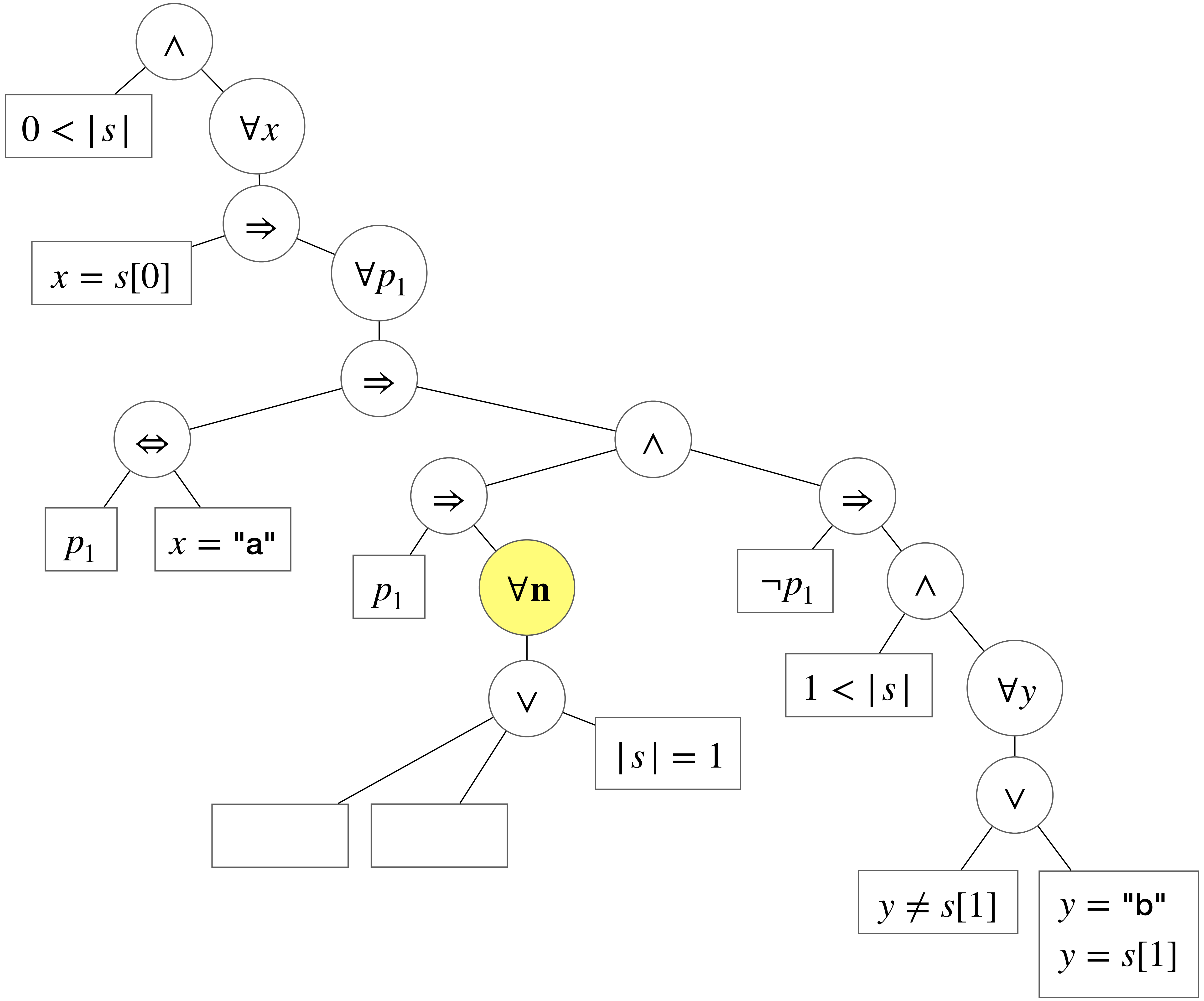


# Grammar Solving

- 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

$\forall x \,.\, \varphi \quad \rightarrow \quad$ resolve $x$ in $\varphi$

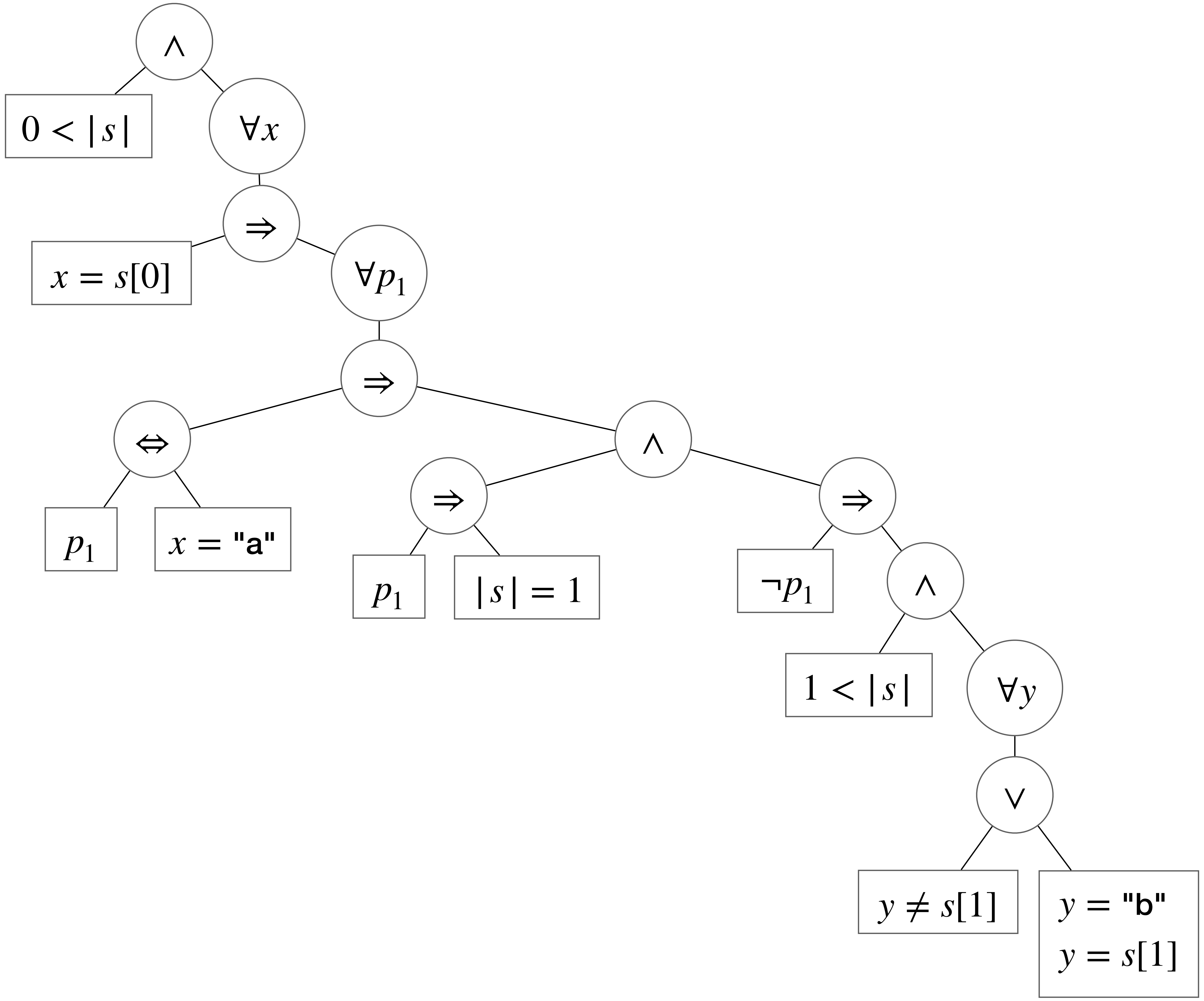


# Grammar Solving

- 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

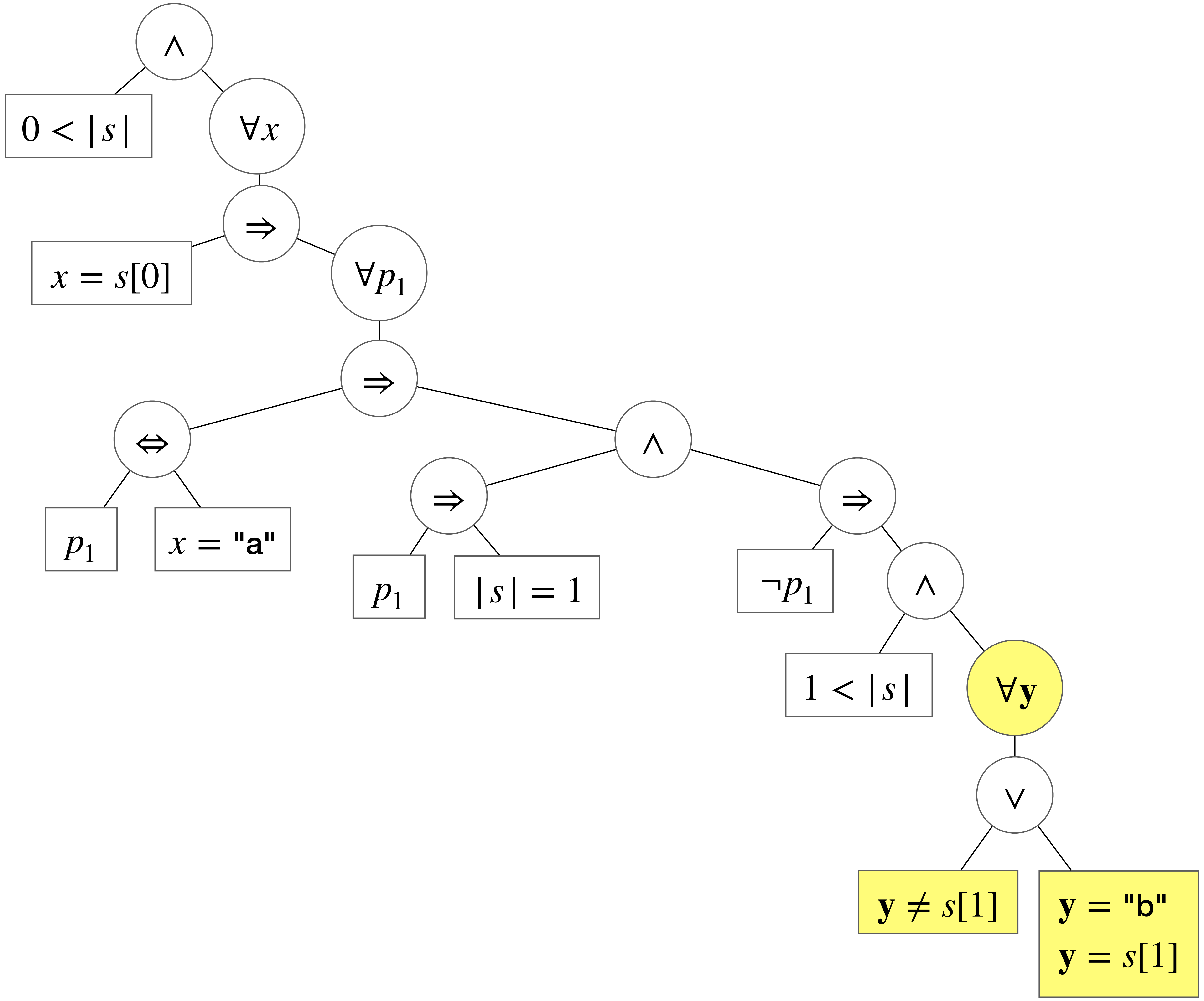
$\forall x \,.\, \varphi \quad \rightarrow \quad$ resolve $x$ in $\varphi$



# 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



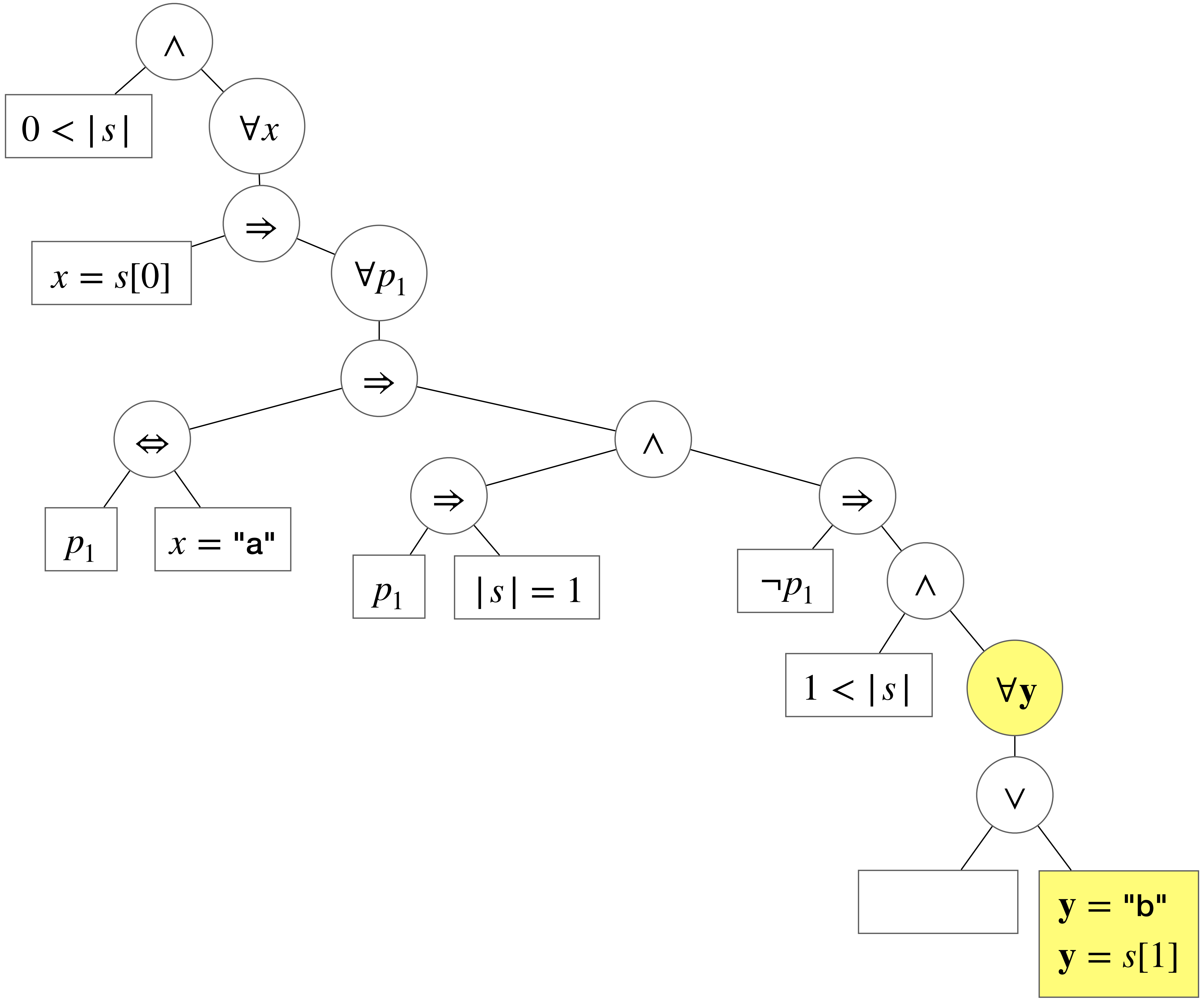

$\forall x . \varphi \quad \rightarrow \quad$ resolve $x$ in $\varphi$

# 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

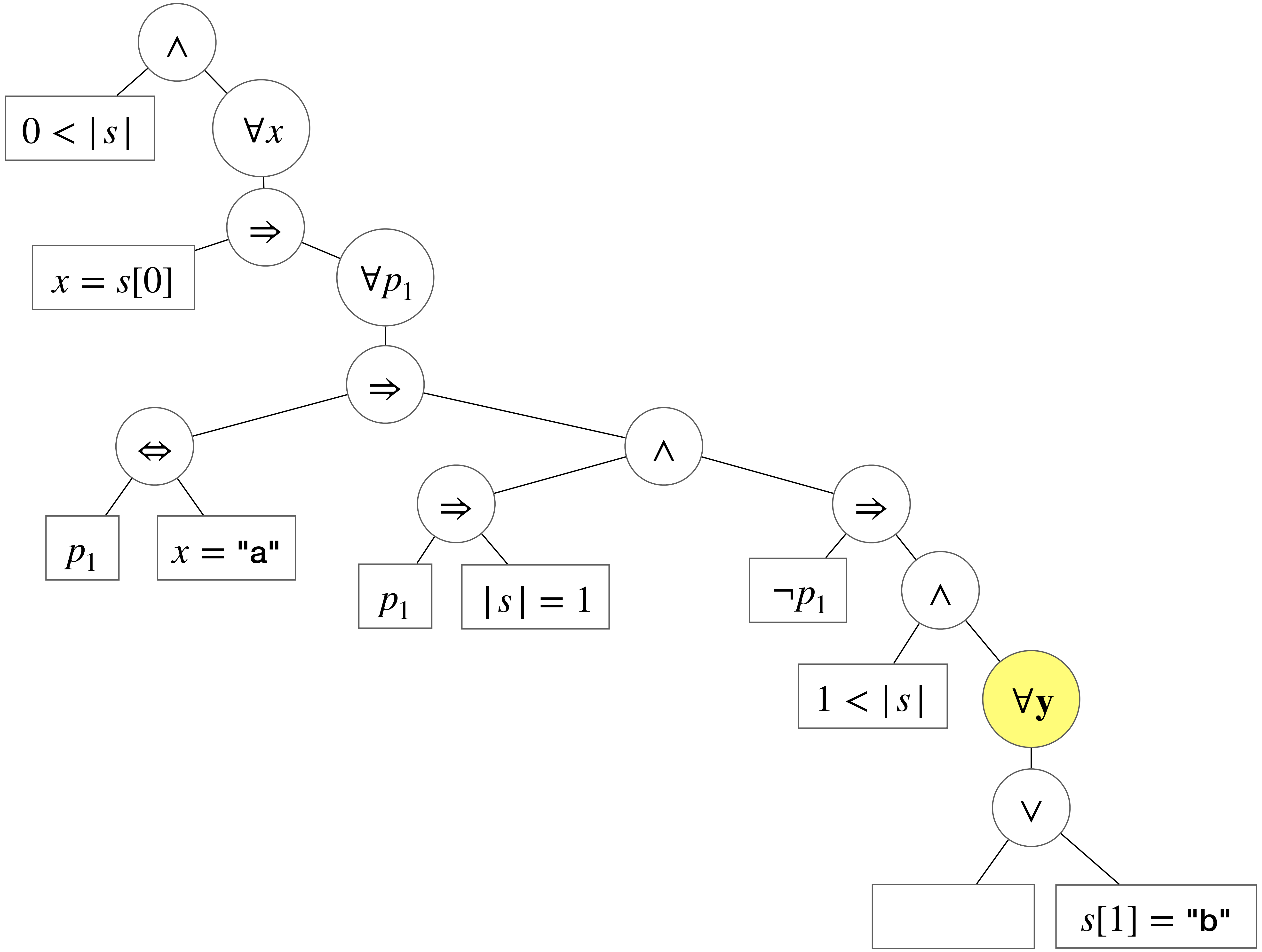
$$\forall x\,.\,\varphi \quad \rightarrow \quad \text{resolve } x \text{ in } \varphi$$



# 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

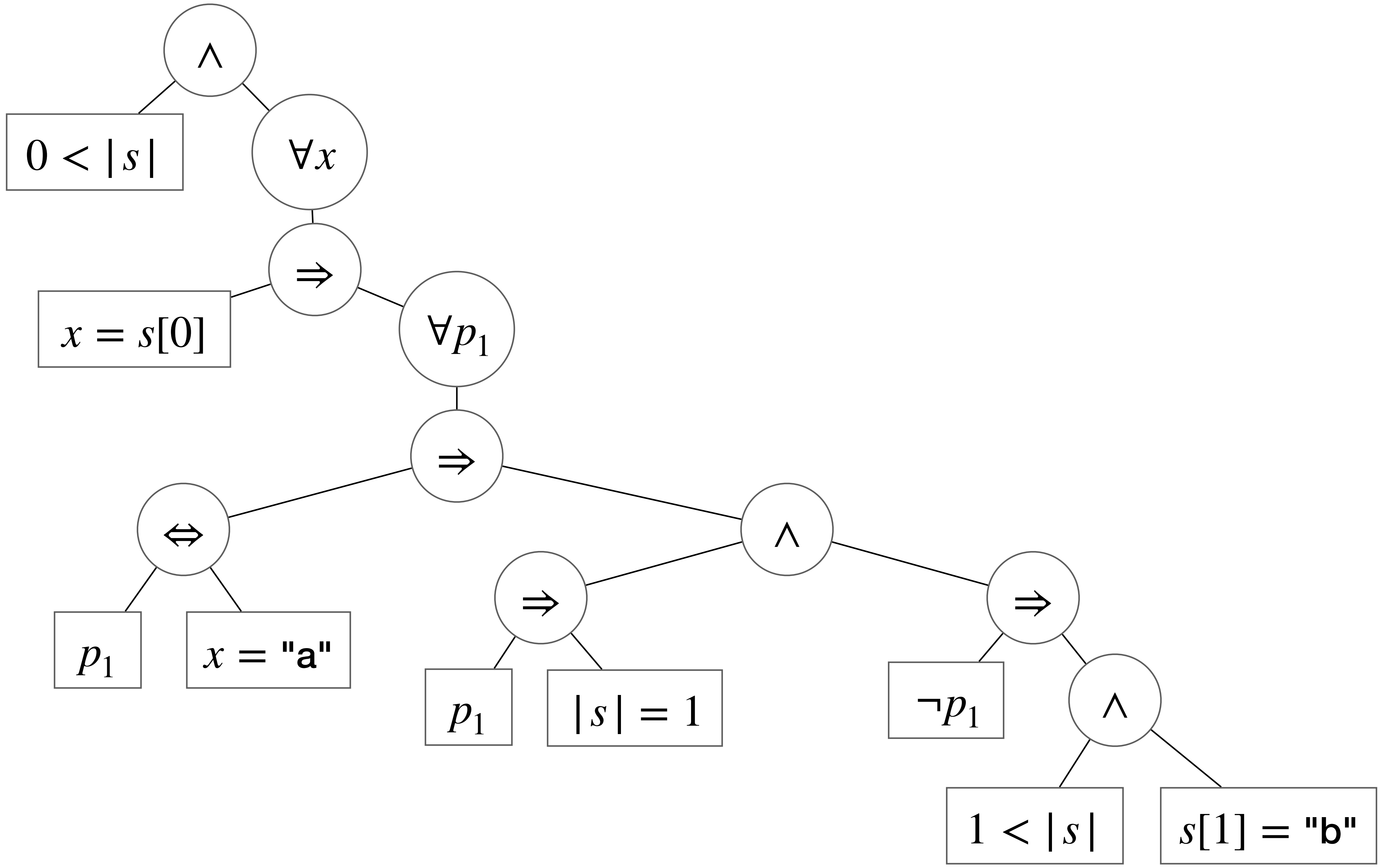



$$\forall x \,.\, \varphi \quad \rightarrow \quad \text{resolve } x \text{ in } \varphi$$

# 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



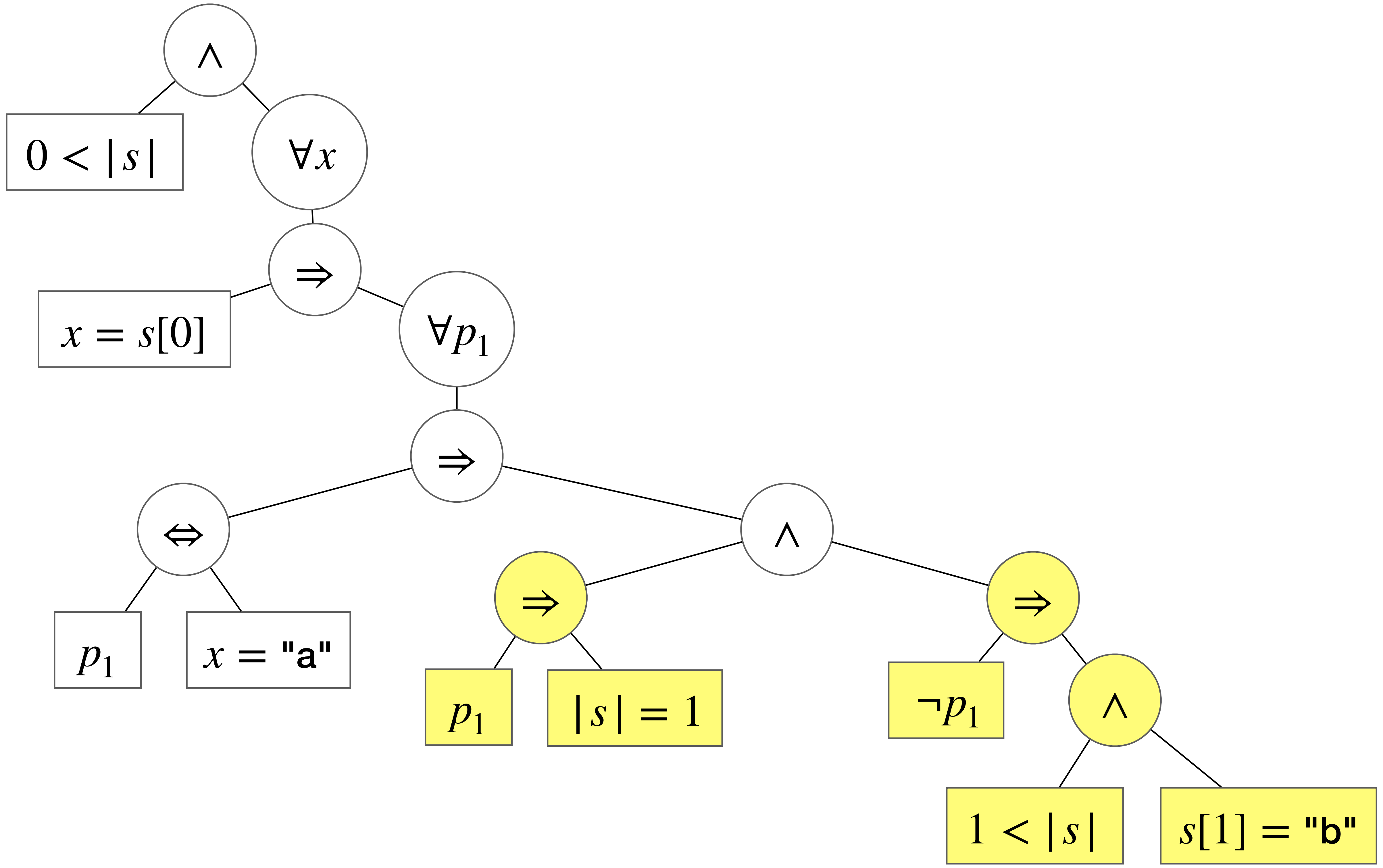

$\forall x \,.\, \varphi \quad \rightarrow \quad$ resolve $x$ in $\varphi$

Schröder and Cito. 2022. Toward Grammar Inference via Refinement Types (Extended Abstract). https://mcschroeder.github.io/#tyde2022

# 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

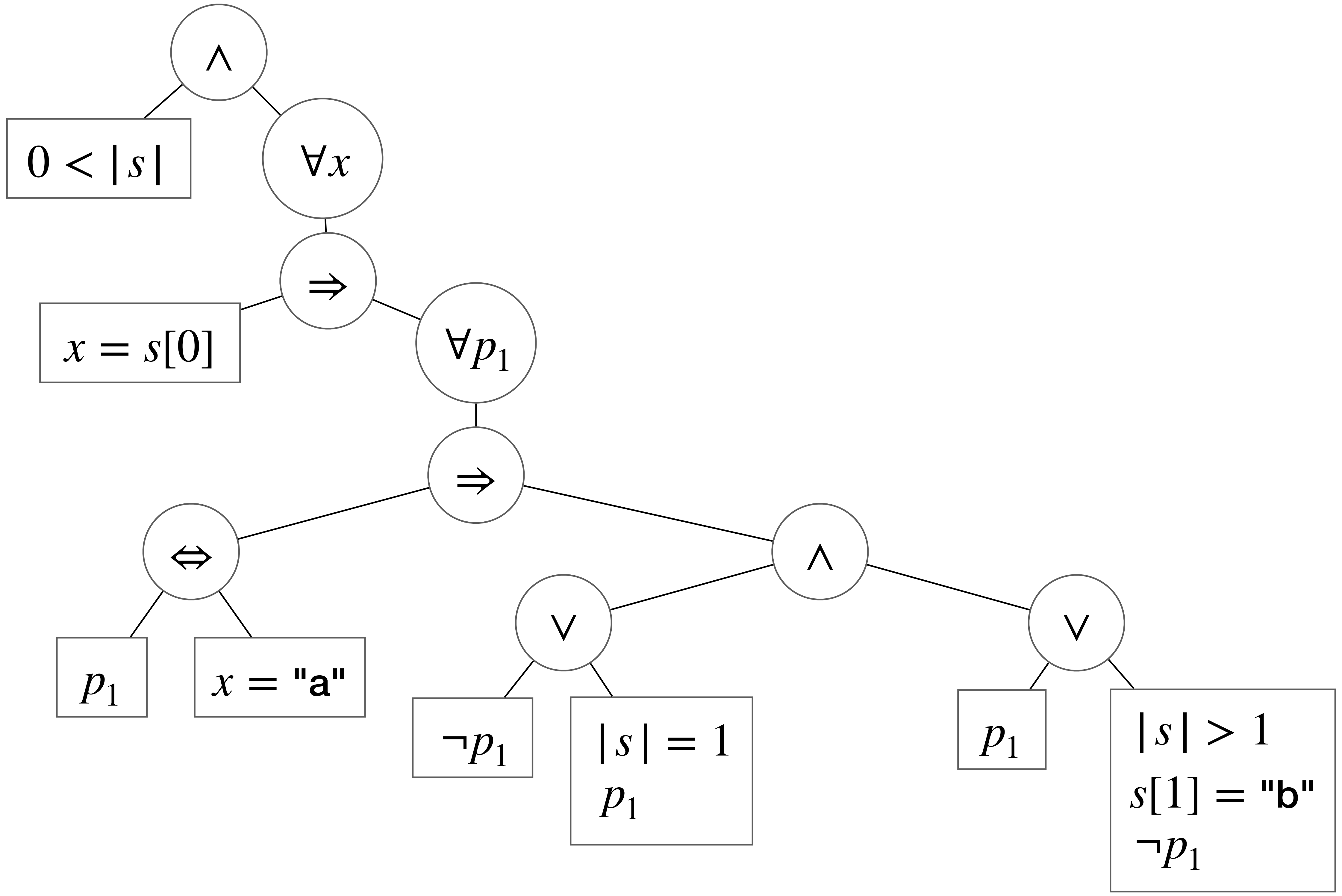



$$a \Rightarrow b \quad \to \quad \neg a \vee (a \sqcap b)$$

# 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

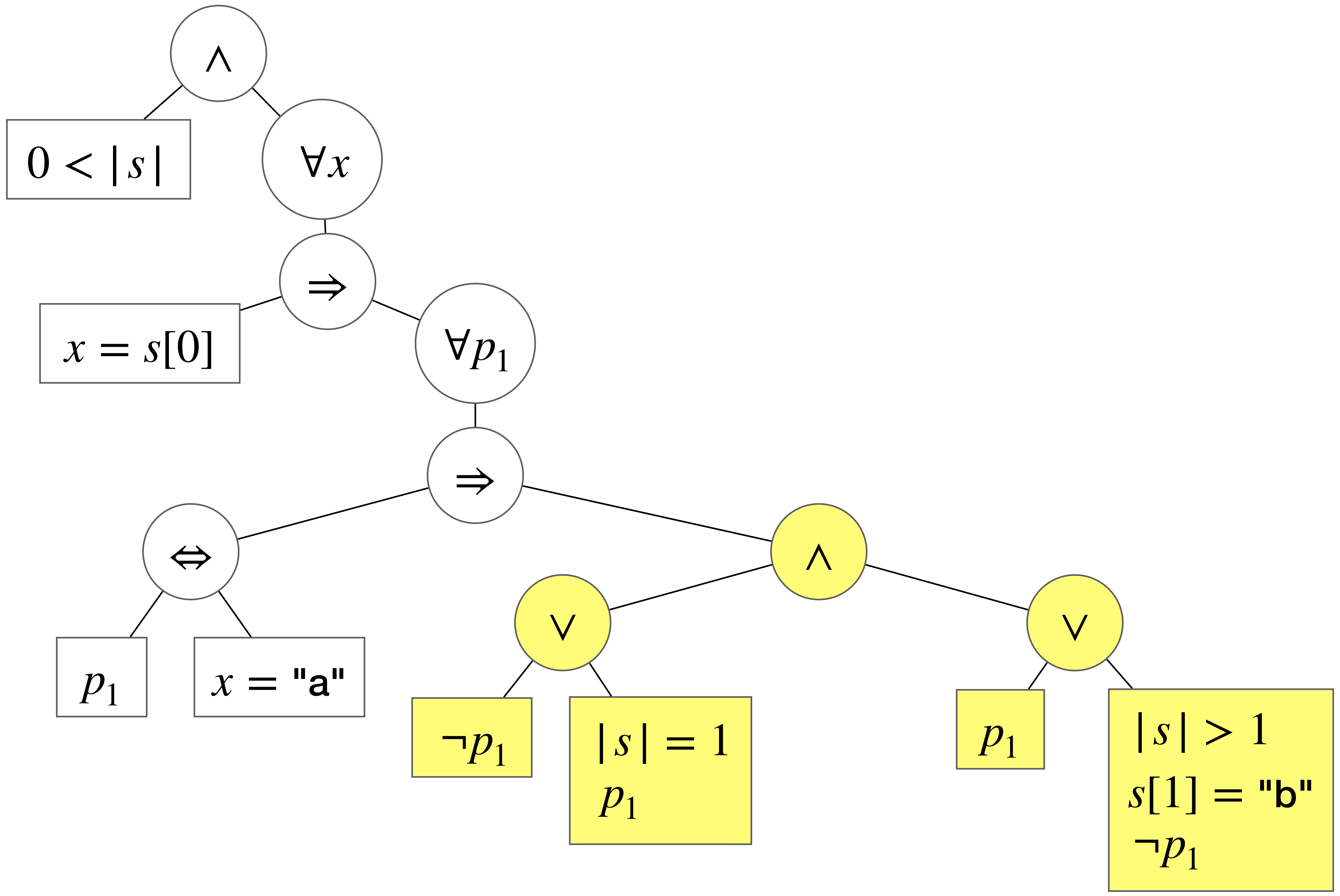



$$a \Rightarrow b \quad \rightarrow \quad \neg a \vee (a \sqcap b)$$

# 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

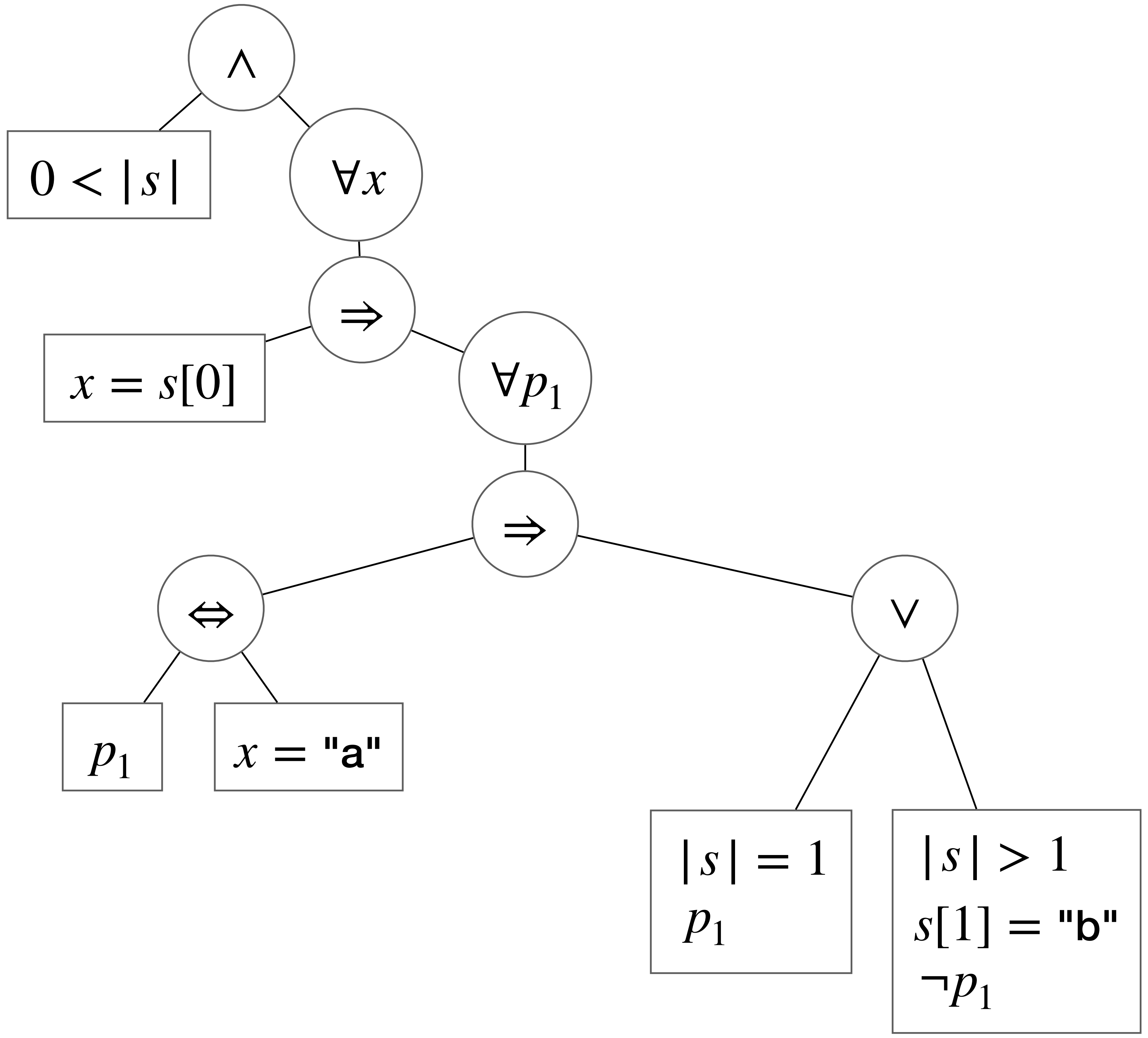



$$(a \lor b) \land (\neg a \lor c) \quad \rightarrow \quad b \lor c$$

# 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

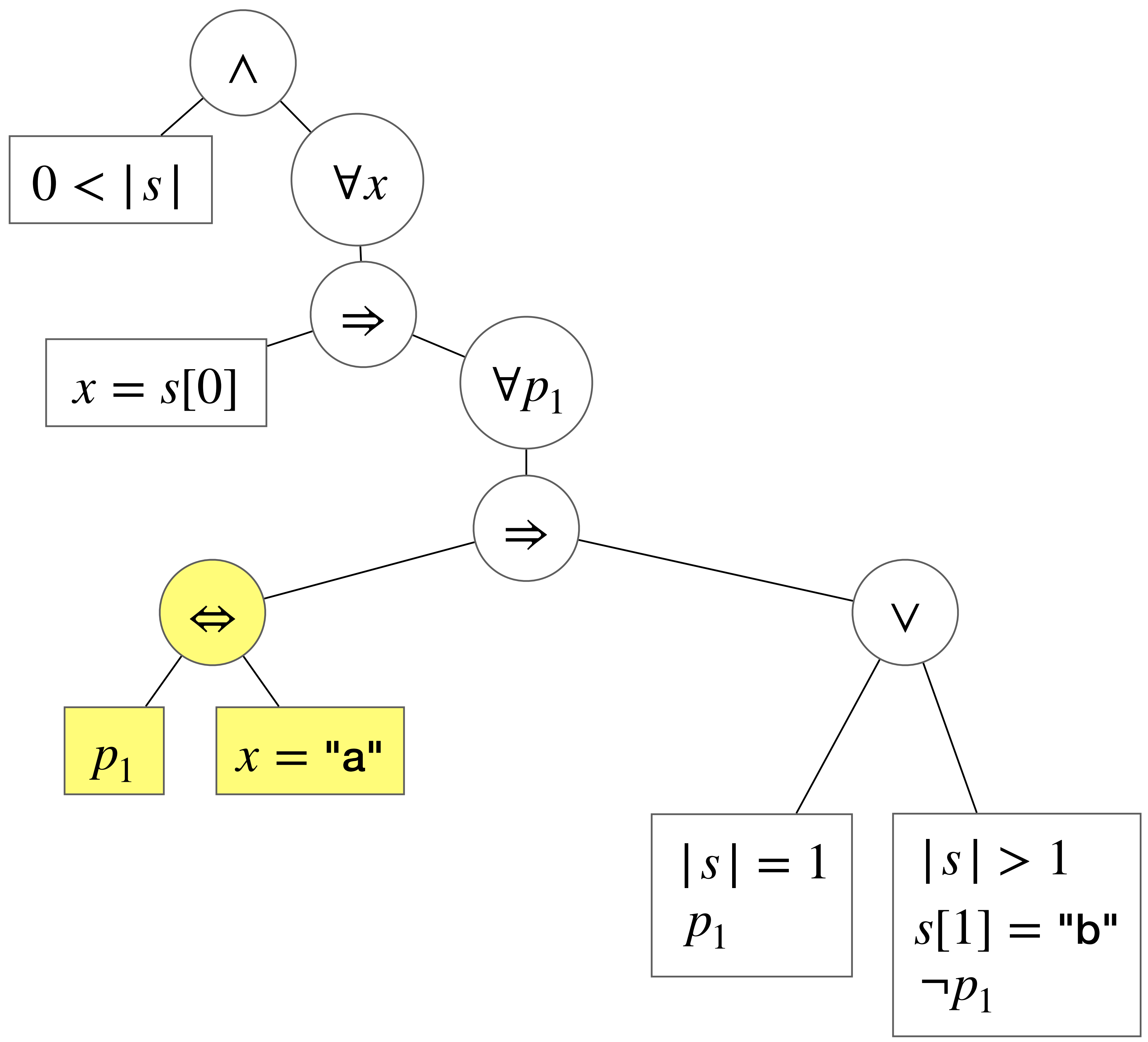



$$(a \lor b) \land (\neg a \lor c) \quad \rightarrow \quad b \lor c$$

# 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

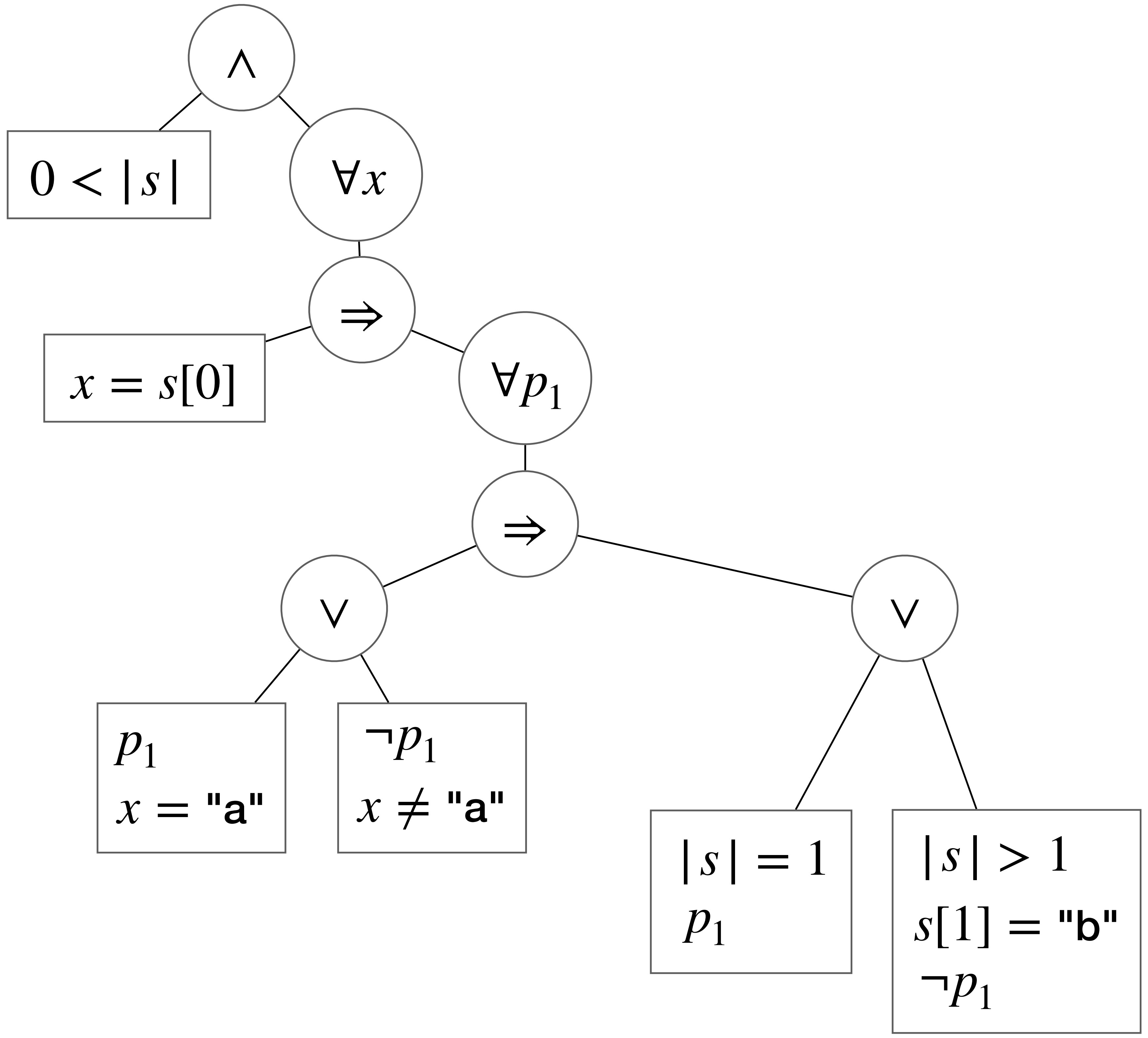



$$a \Leftrightarrow b \quad \rightarrow \quad (\neg a \land \neg b) \lor (a \land b)$$

# 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



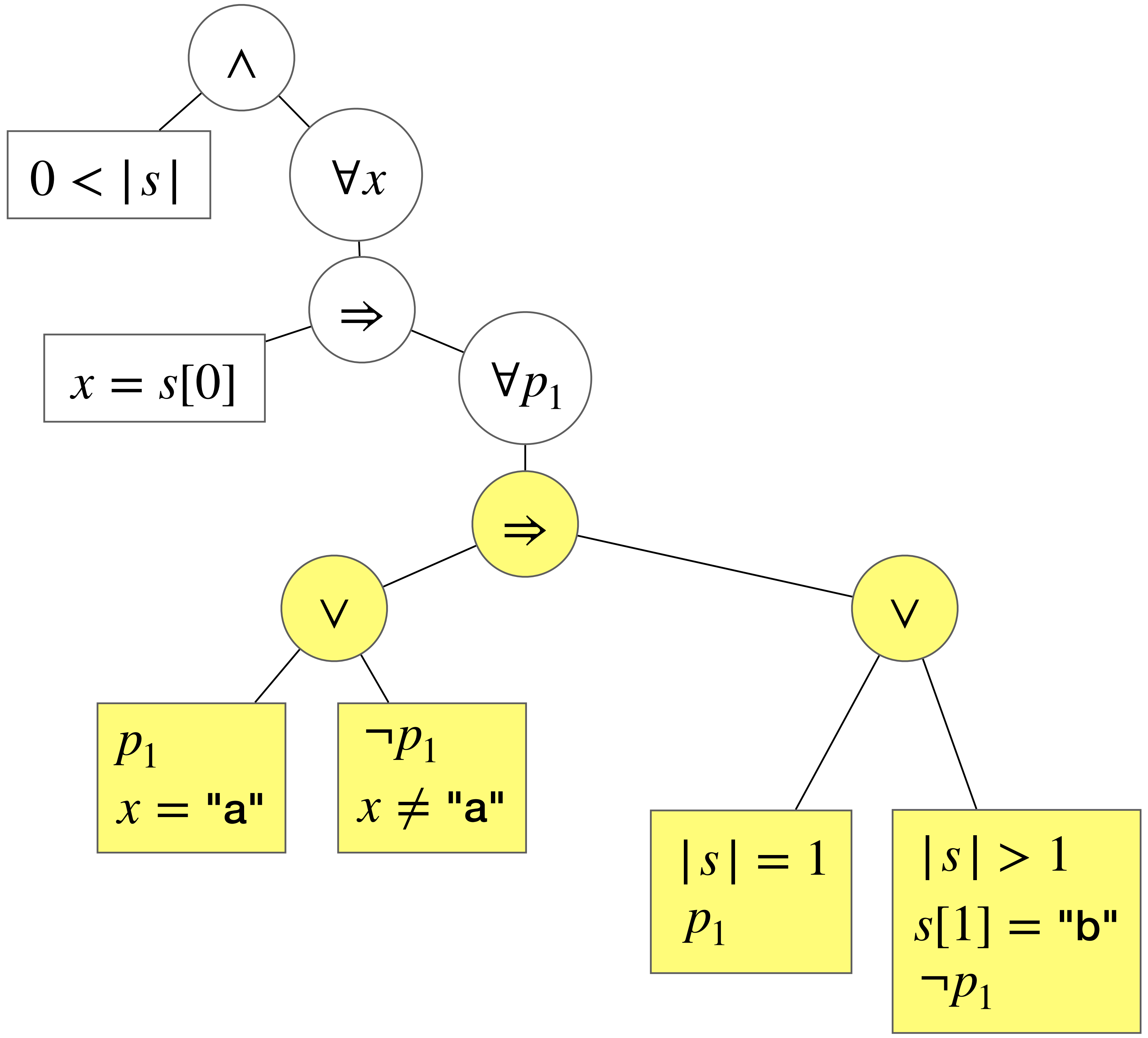

$$a \Leftrightarrow b \quad \rightarrow \quad (\neg a \wedge \neg b) \vee (a \wedge b)$$

Schröder and Cito. 2022. Toward Grammar Inference via Refinement Types (Extended Abstract). https://mcschroeder.github.io/#tyde2022

# 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



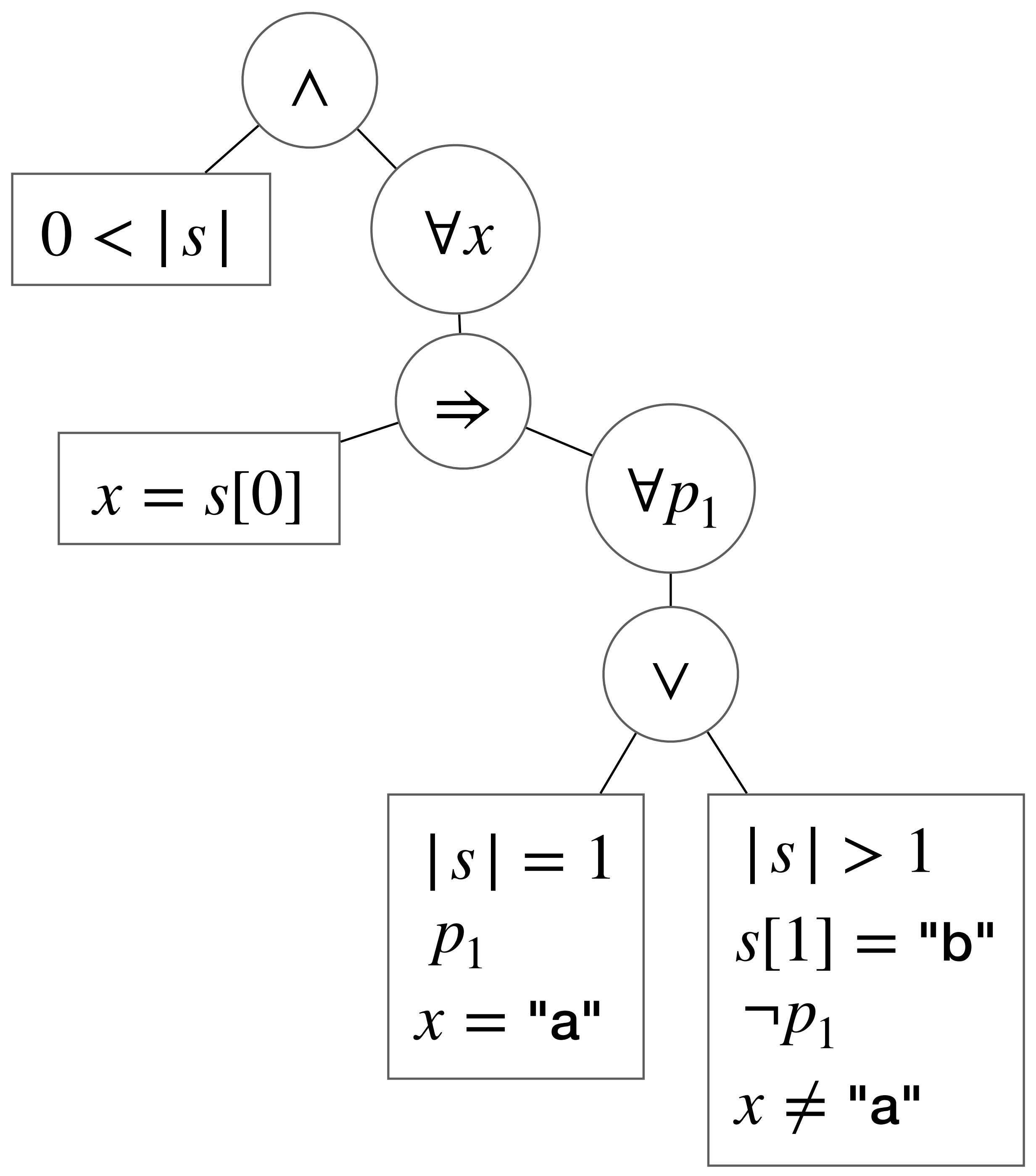

$$(a \vee b) \Rightarrow c \quad \rightarrow \quad (a \Rightarrow c) \vee (b \Rightarrow c)$$

$$a \Rightarrow b \quad \rightarrow \quad \neg a \vee (a \sqcap b)$$

# 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




$$(a \vee b) \Rightarrow c \quad \rightarrow \quad (a \Rightarrow c) \vee (b \Rightarrow c)$$

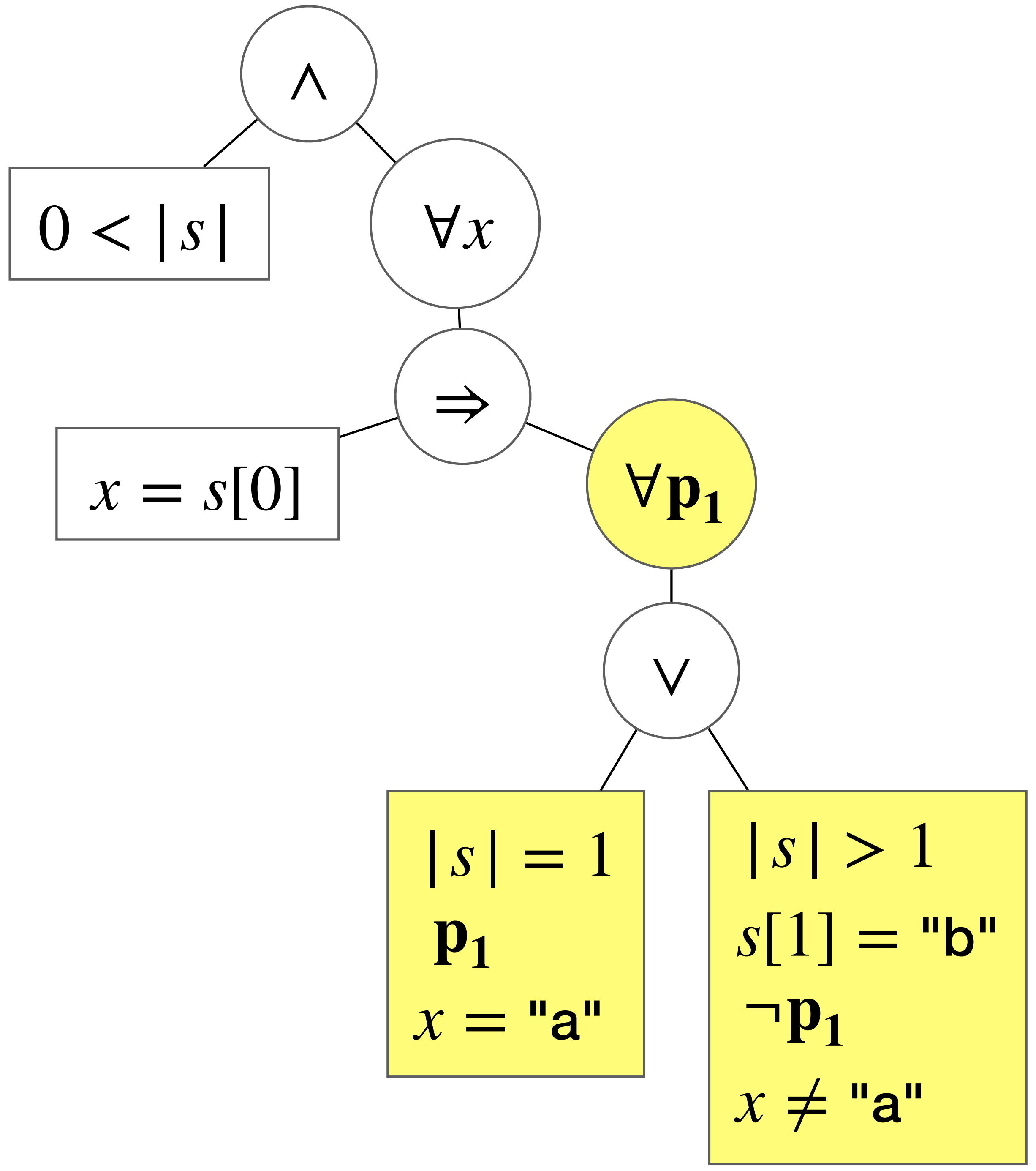
$$a \Rightarrow b \quad \rightarrow \quad \neg a \vee (a \sqcap b)$$

Schröder and Cito. 2022. Toward Grammar Inference via Refinement Types (Extended Abstract). https://mcschroeder.github.io/#tyde2022

# Grammar Solving

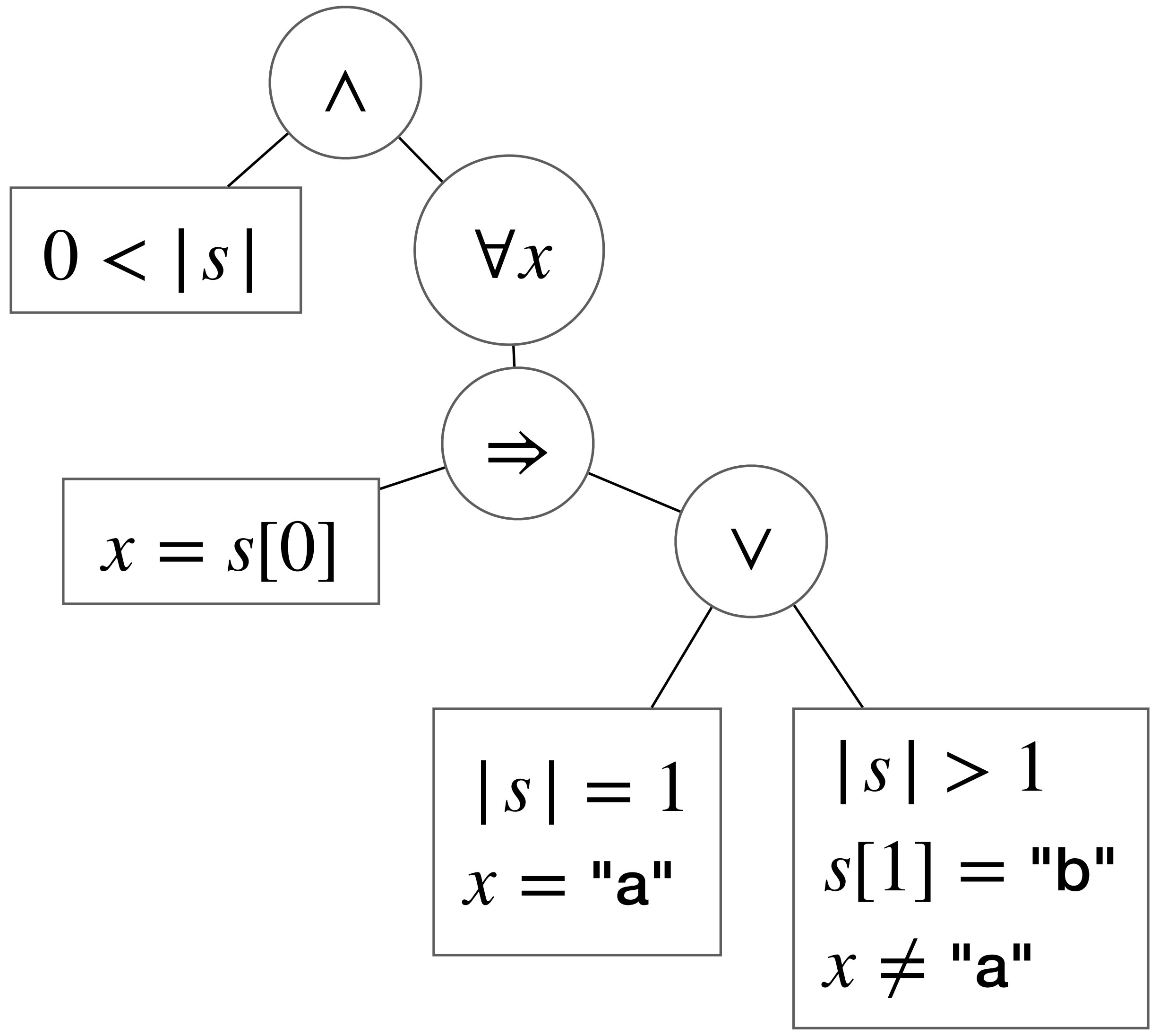
- base solution on “grammar consequent”

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




$\forall x \, . \, \varphi \quad \rightarrow \quad$ resolve $x$ in $\varphi$

# Grammar Solving




- base solution on "grammar consequent"

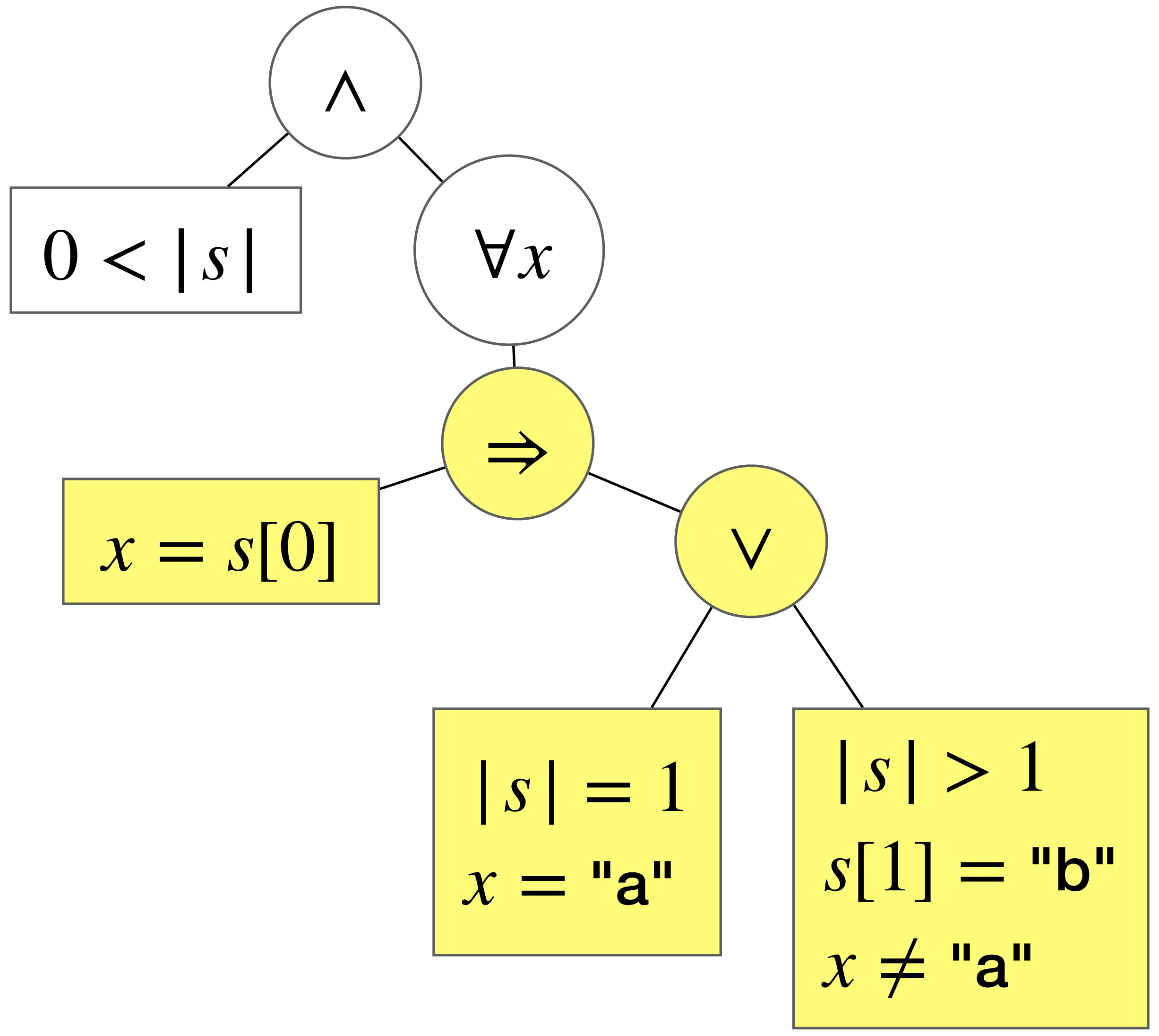
- minimize via bottom-up tree rewriting
- apply Boolean equivalences to reach DNF
- eliminate quantifiers by resolving equations
- use (precise) abstract value representations

$\forall x \,.\, \varphi \quad \rightarrow \quad$ resolve $x$ in $\varphi$

# 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

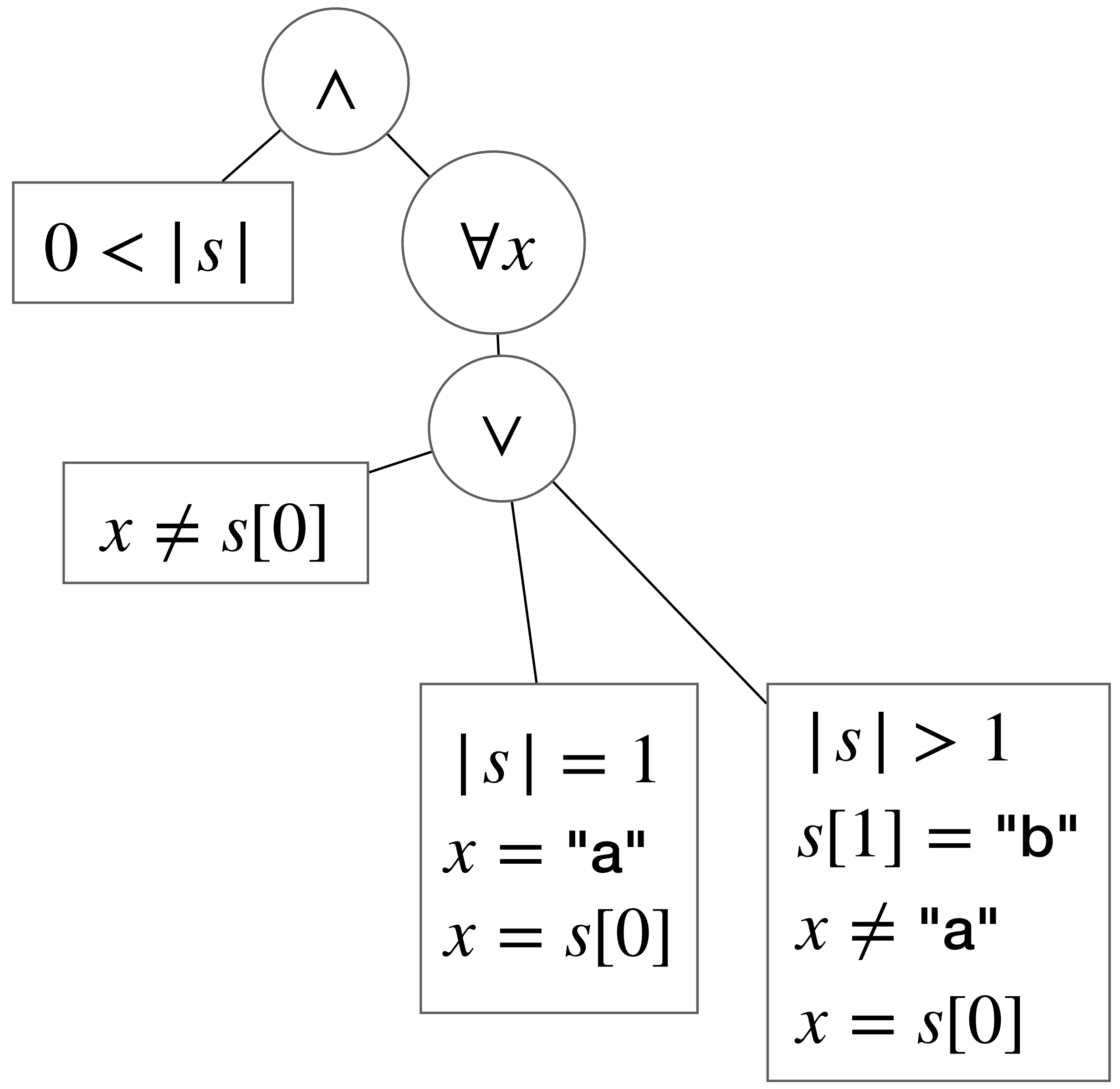



$$a \Rightarrow b \quad \rightarrow \quad \neg a \vee (a \sqcap b)$$

# 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



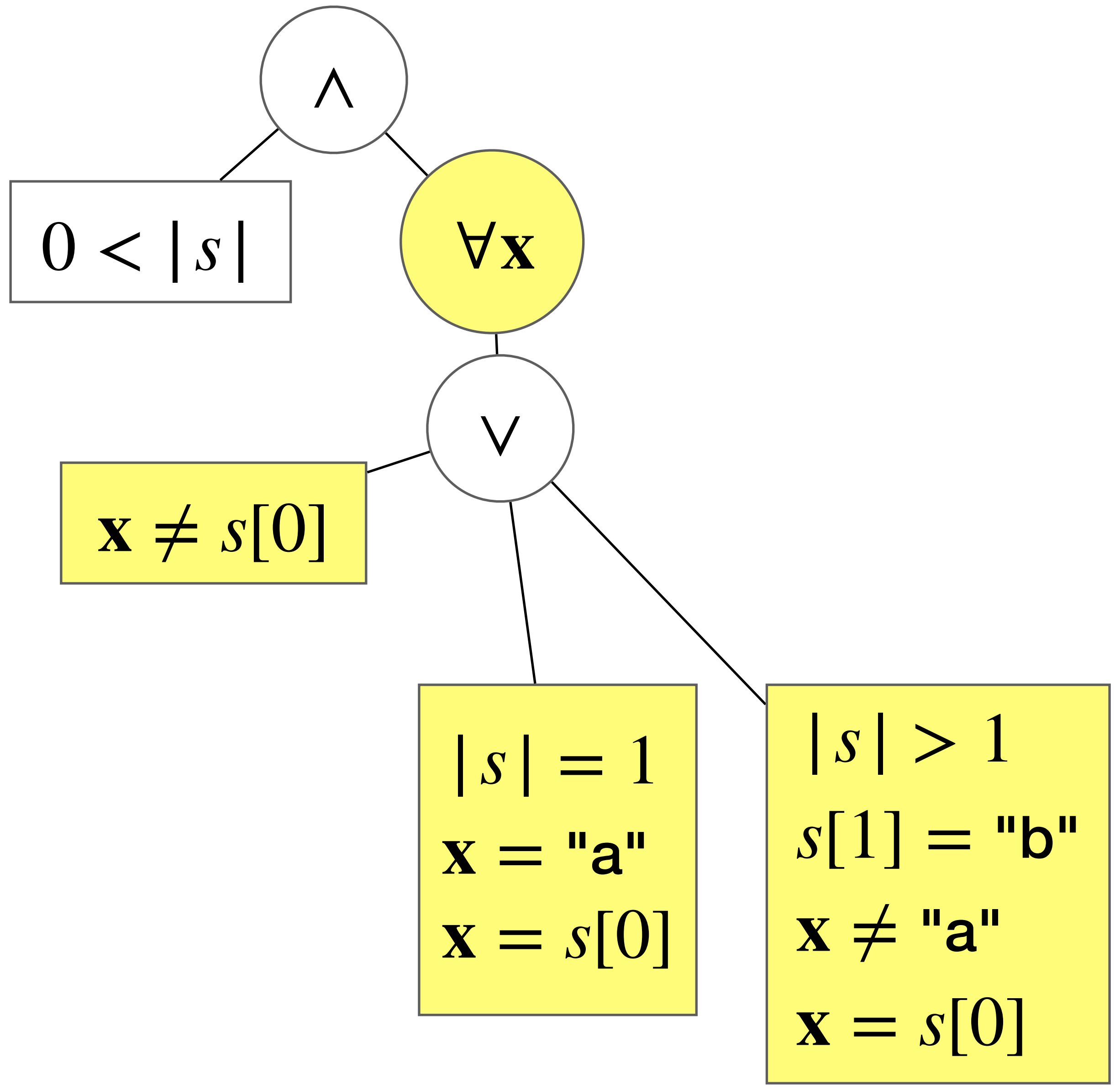

$$a \Rightarrow b \quad \rightarrow \quad \neg a \vee (a \sqcap b)$$

Schröder and Cito. 2022. Toward Grammar Inference via Refinement Types (Extended Abstract). https://mcschroeder.github.io/#tyde2022

# 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



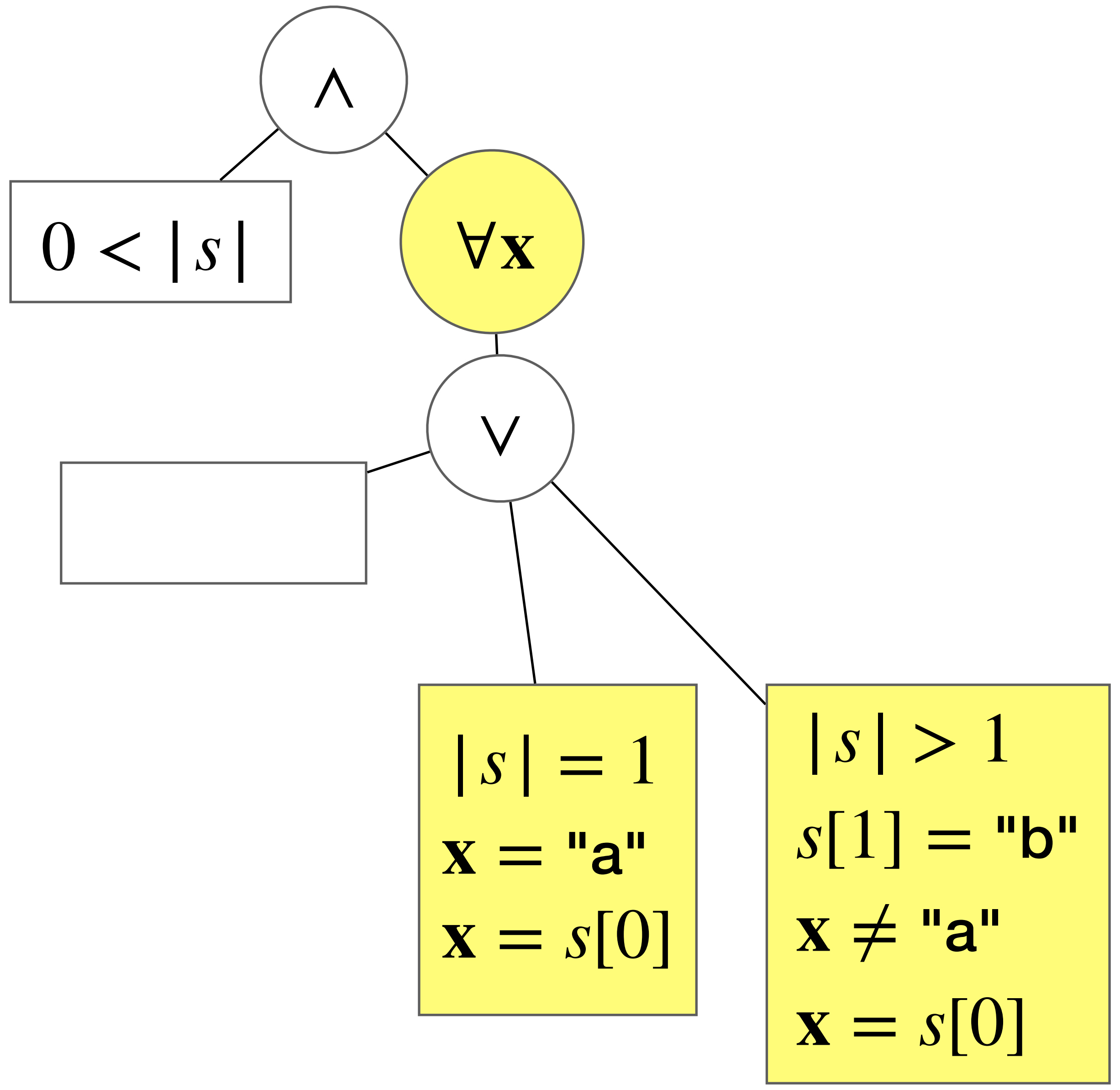

$\forall x \,.\, \varphi \quad \rightarrow \quad$ resolve $x$ in $\varphi$

Schröder and Cito. 2022. Toward Grammar Inference via Refinement Types (Extended Abstract). https://mcschroeder.github.io/#tyde2022

# 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

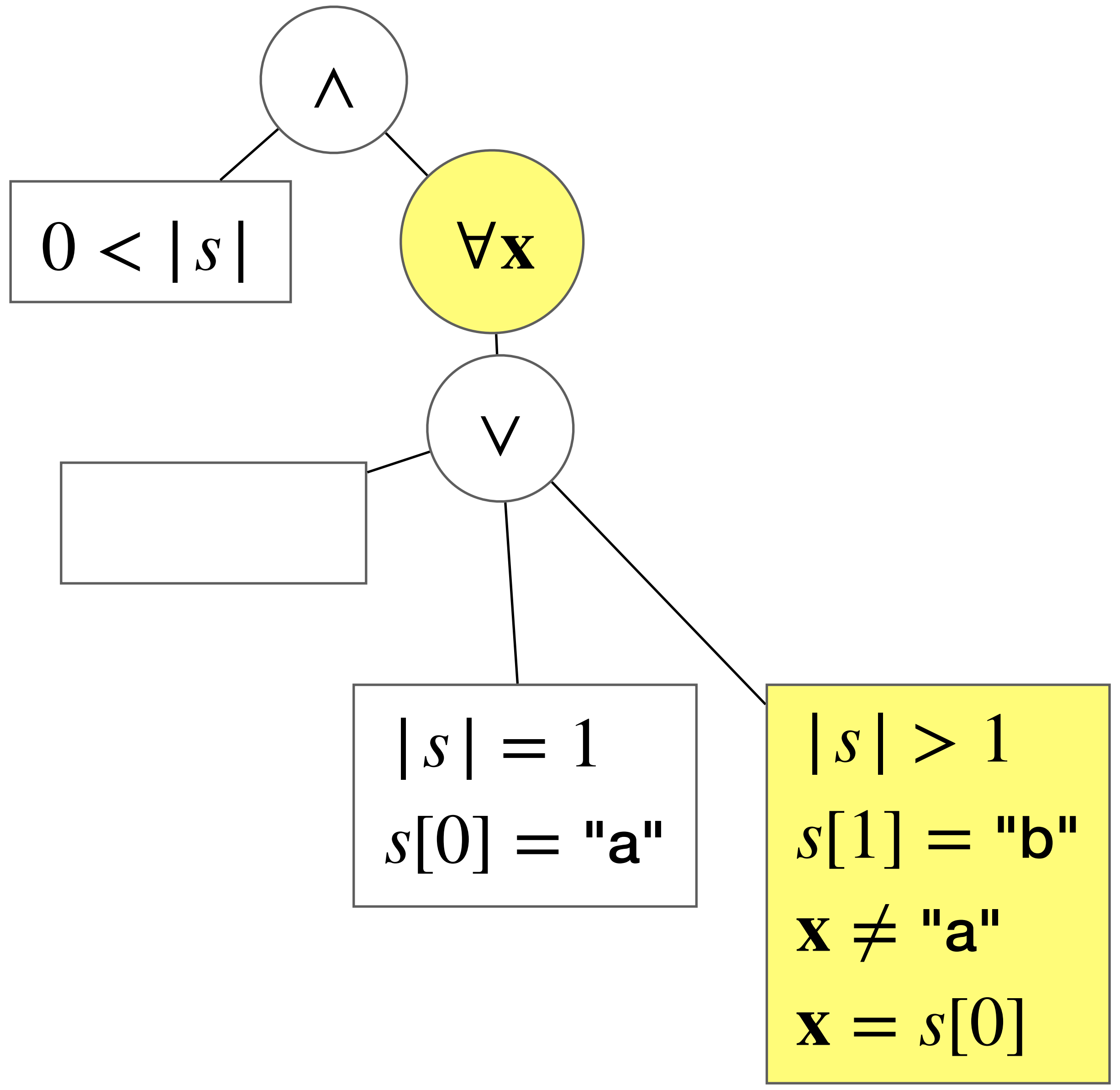



$\forall x \,.\, \varphi \quad \rightarrow \quad$ resolve $x$ in $\varphi$

# 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

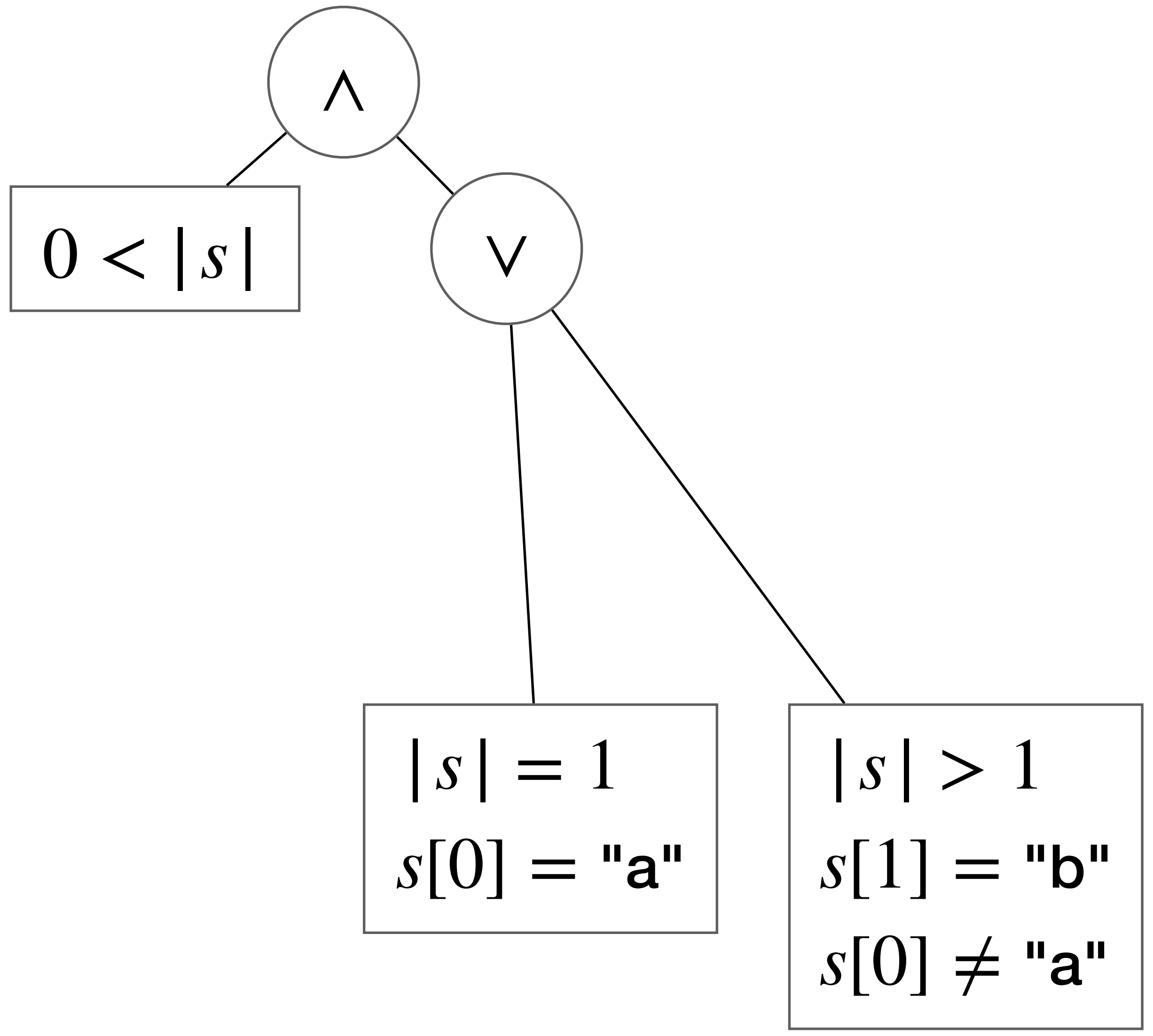



$\forall x \,.\, \varphi \quad \rightarrow \quad$ resolve $x$ in $\varphi$

# 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

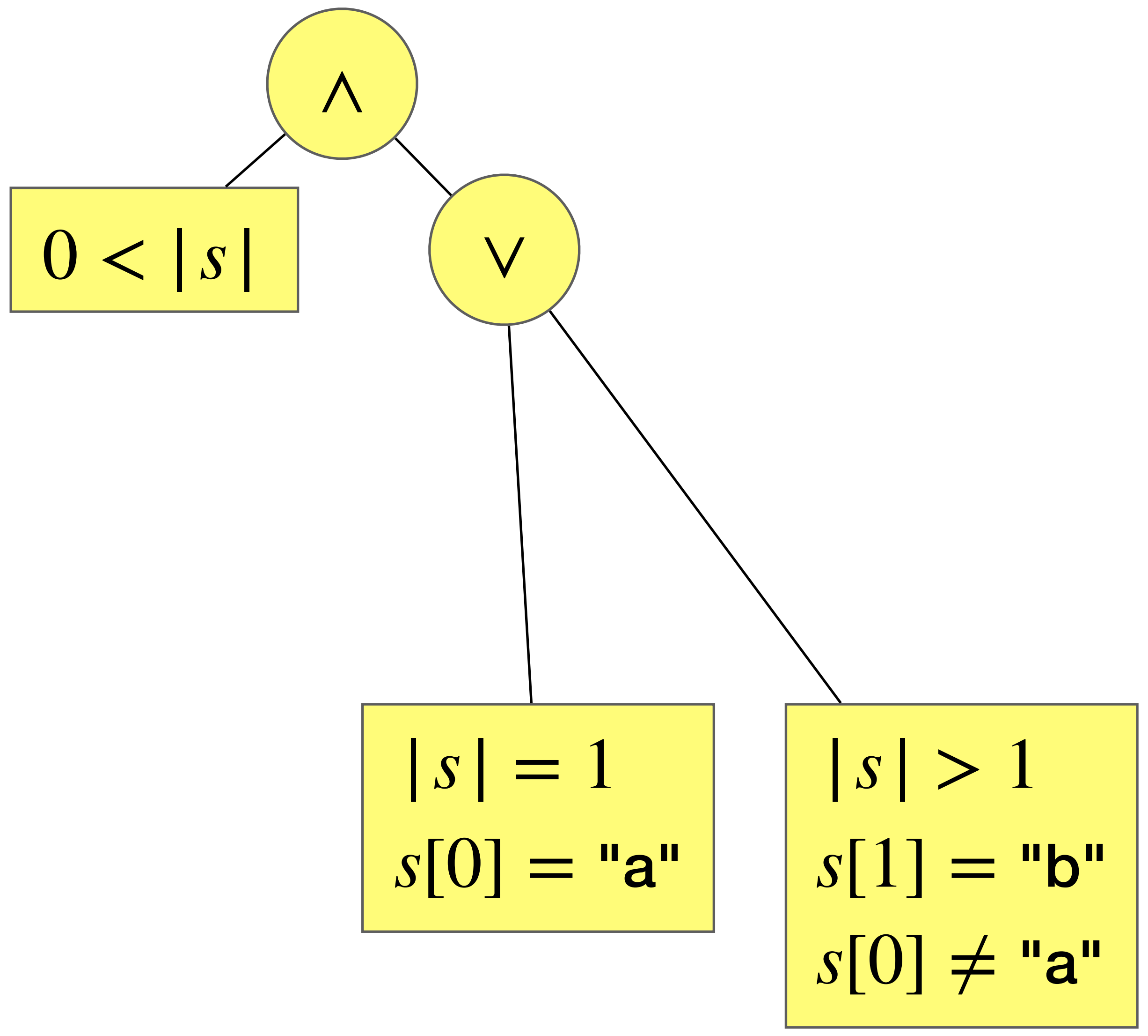



$$\forall x \,.\, \varphi \quad \rightarrow \quad \text{resolve } x \text{ in } \varphi$$

# 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

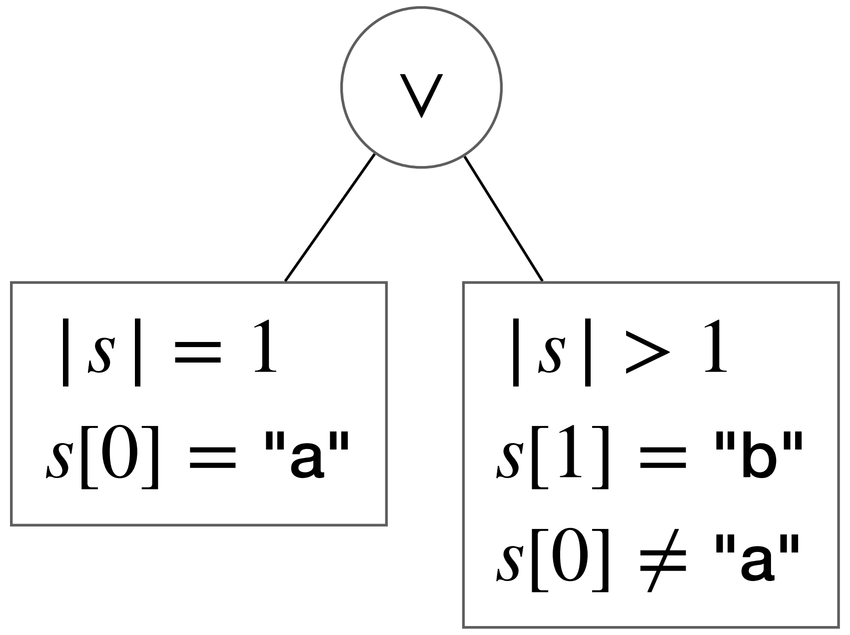



$$a \wedge (b \vee c) \quad \rightarrow \quad (a \wedge b) \vee (a \wedge c)$$

# 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

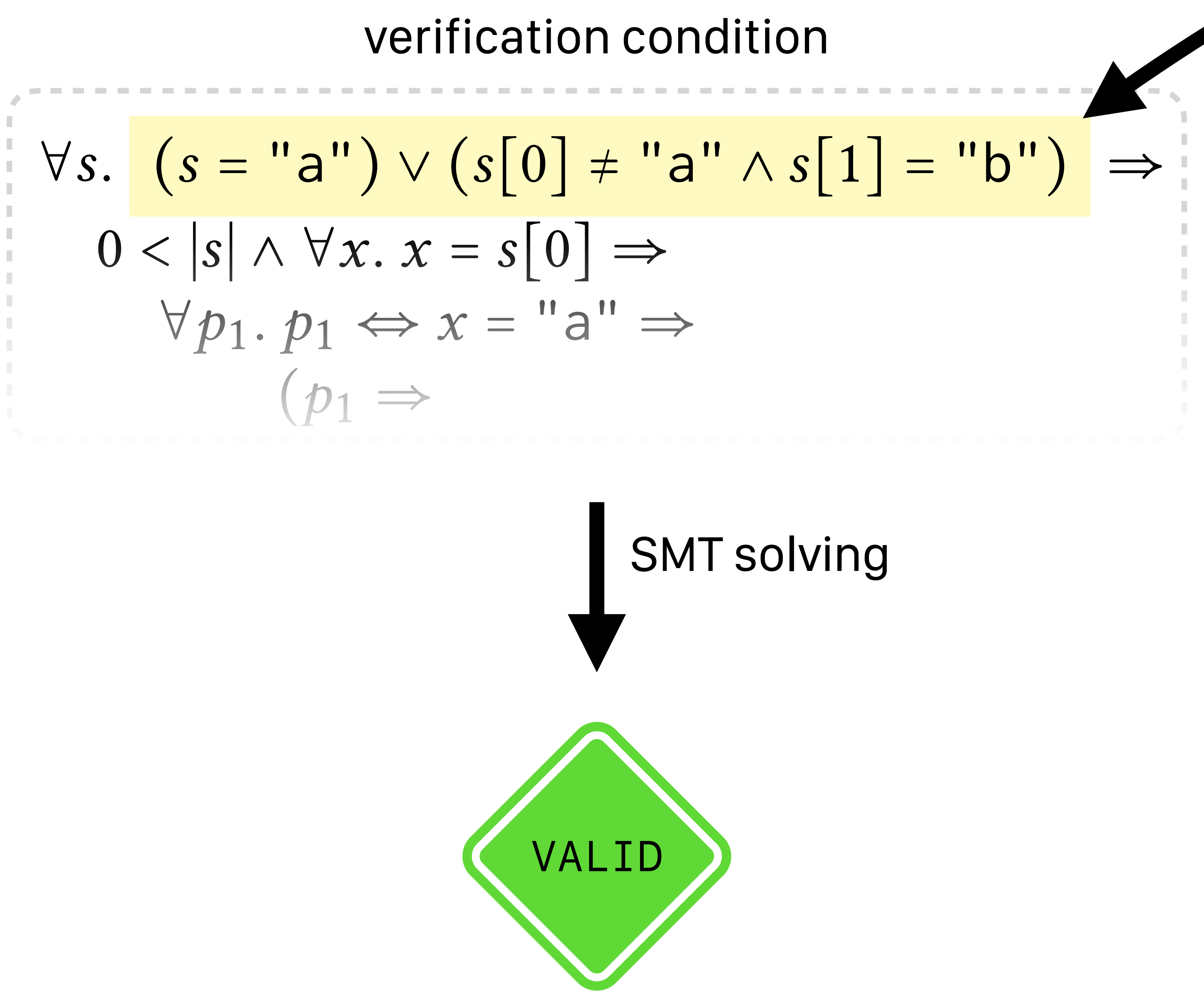


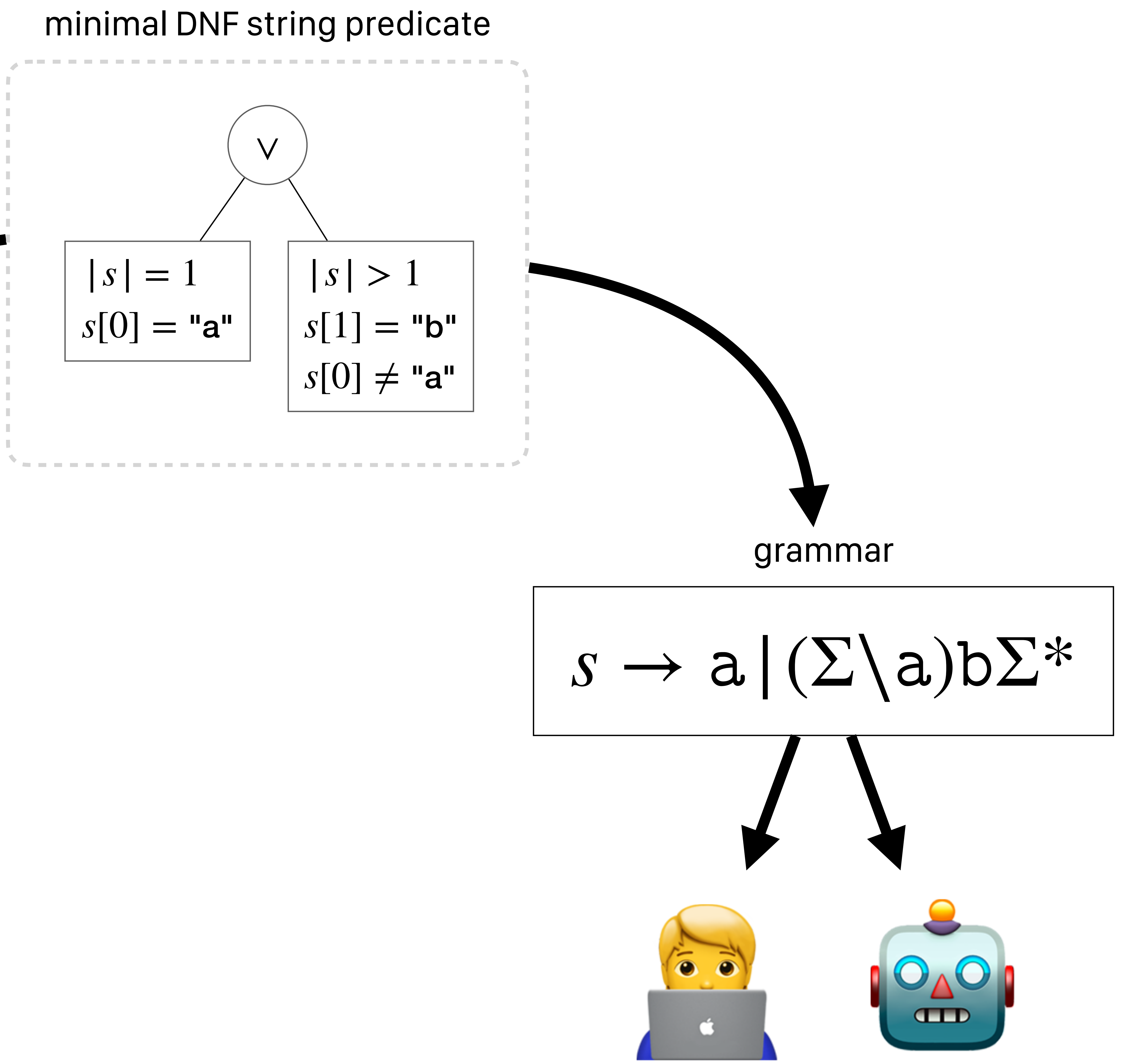

$$a \wedge (b \vee c) \quad \rightarrow \quad (a \wedge b) \vee (a \wedge c)$$

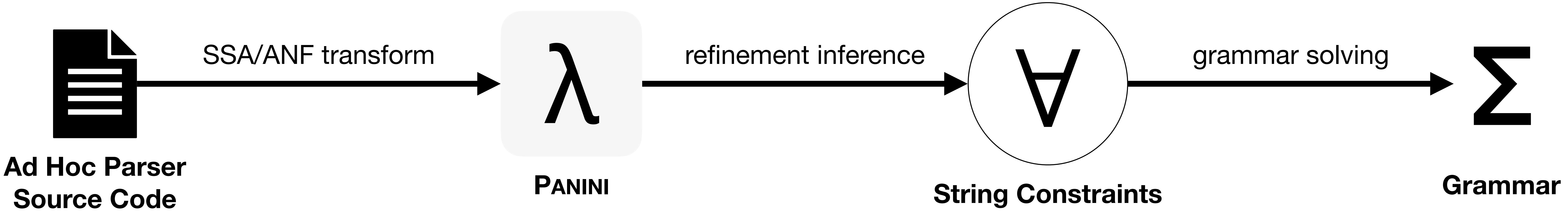
Schröder and Cito. 2022. Toward Grammar Inference via Refinement Types (Extended Abstract). https://mcschroeder.github.io/#tyde2022

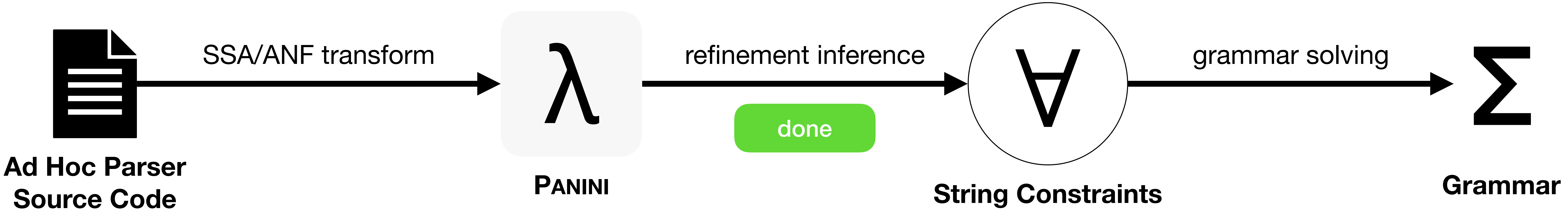
# Enjoy your grammar!

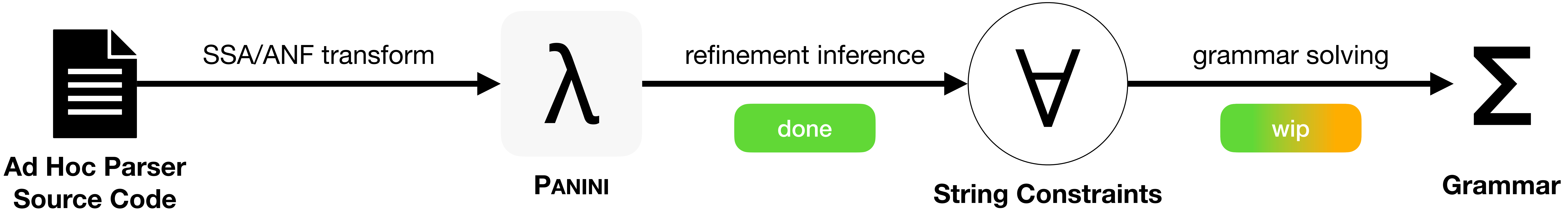
- apply string predicate in verification template to continue type checking
- present grammar to user or applications

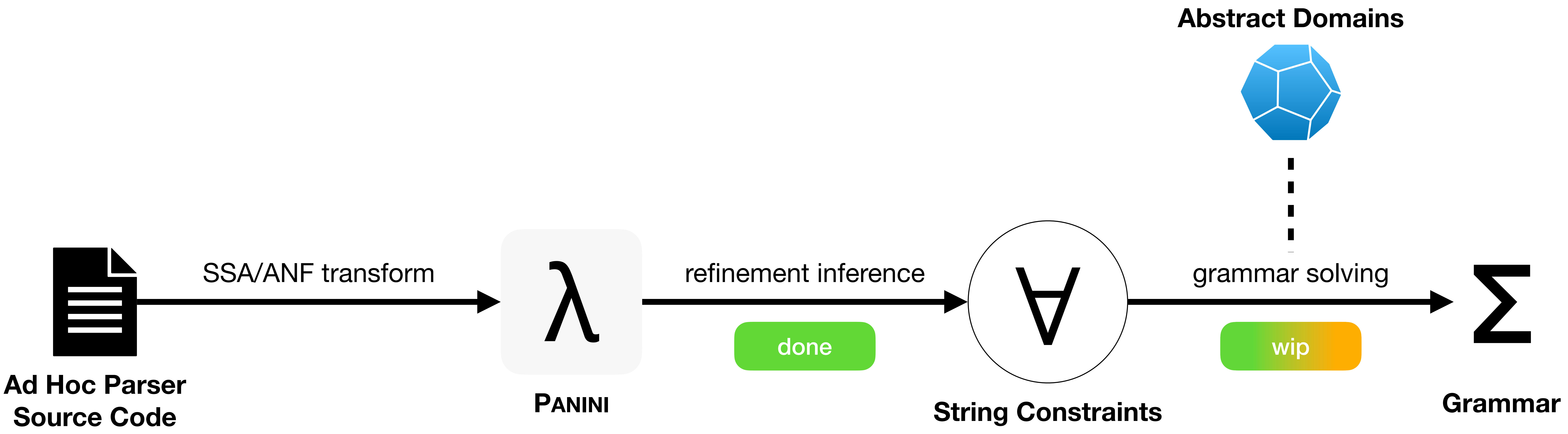


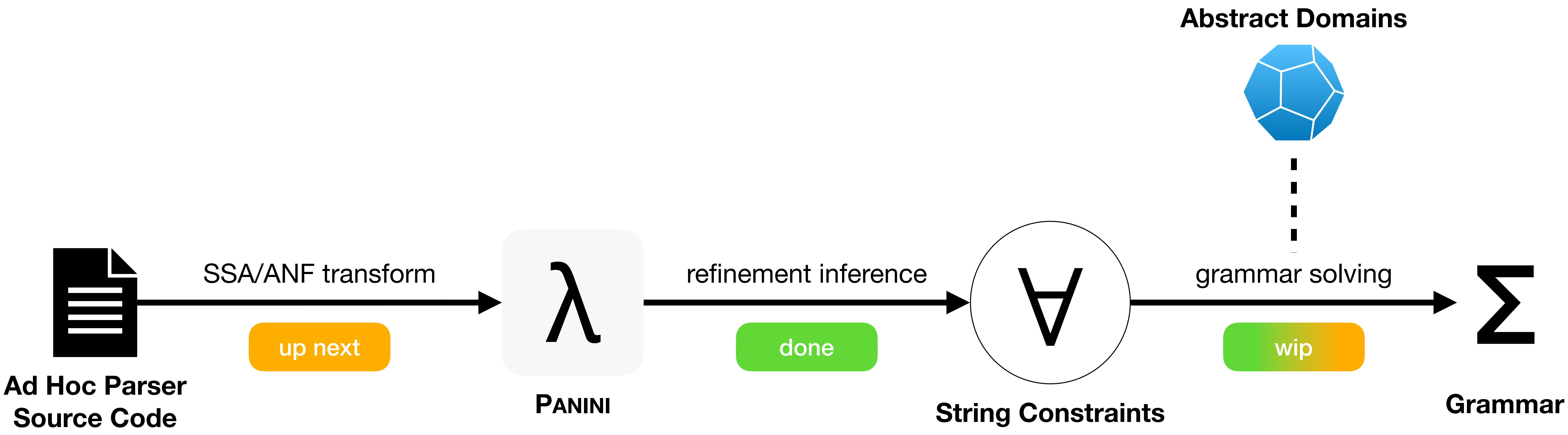

SSA/ANF transform
refinement inference
grammar solving
λ
∀
Σ
Ad Hoc Parser Source Code
PANINI
String Constraints
Grammar

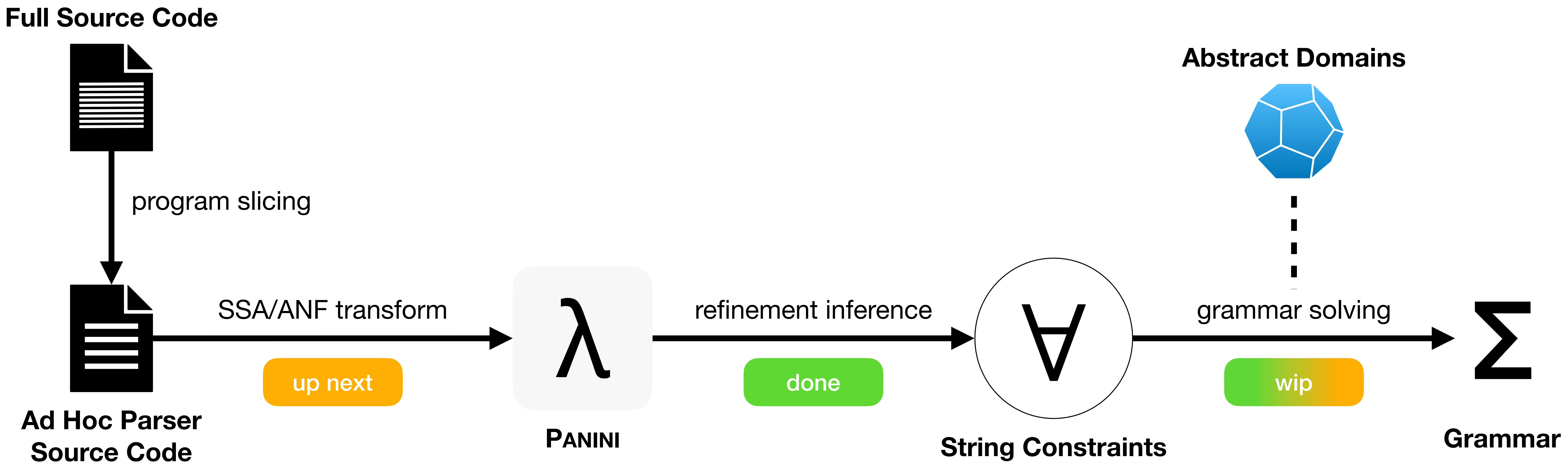

SSA/ANF transform
refinement inference
done
grammar solving
Ad Hoc Parser Source Code
PANINI
String Constraints
Grammar


SSA/ANF transform
Ad Hoc Parser
Source Code
λ
PANINI
refinement inference
done
∀
String Constraints
grammar solving
wip
Σ
Grammar

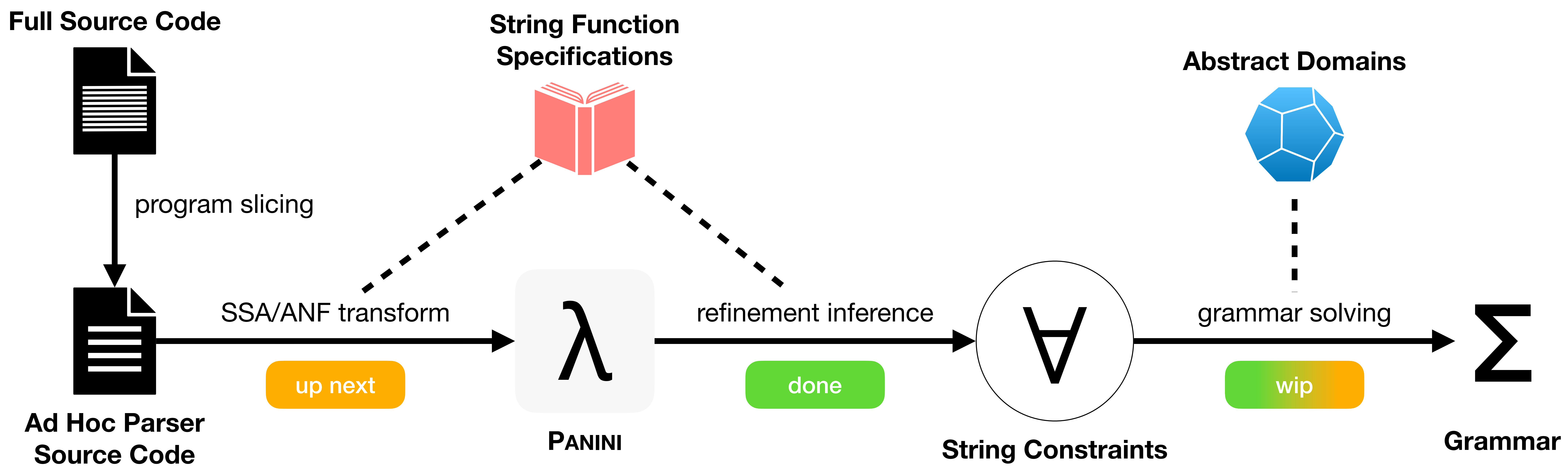


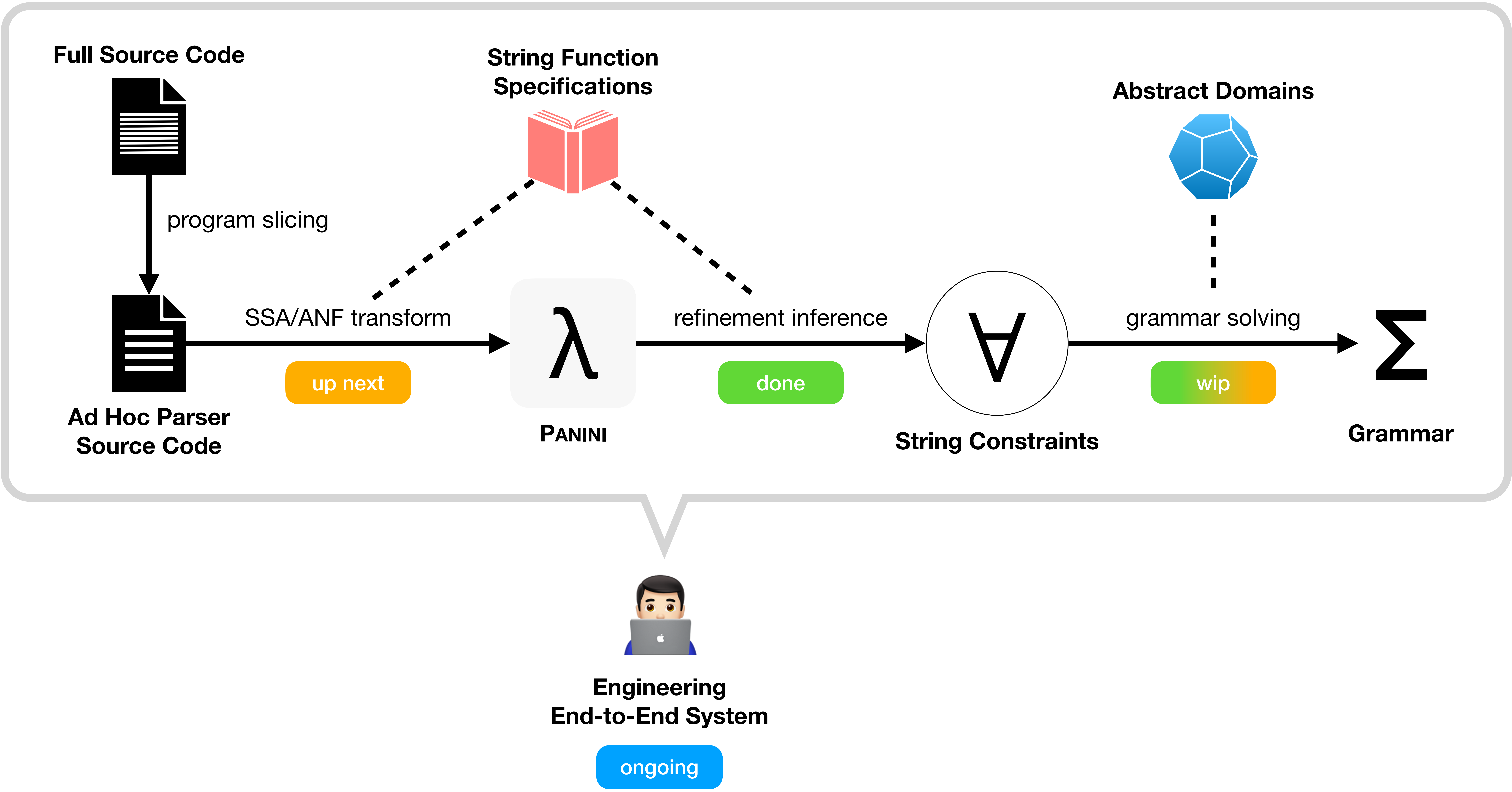
Abstract Domains
SSA/ANF transform
refinement inference
grammar solving
λ
∀
Σ
done
wip
Ad Hoc Parser
Source Code
PANINI
String Constraints
Grammar

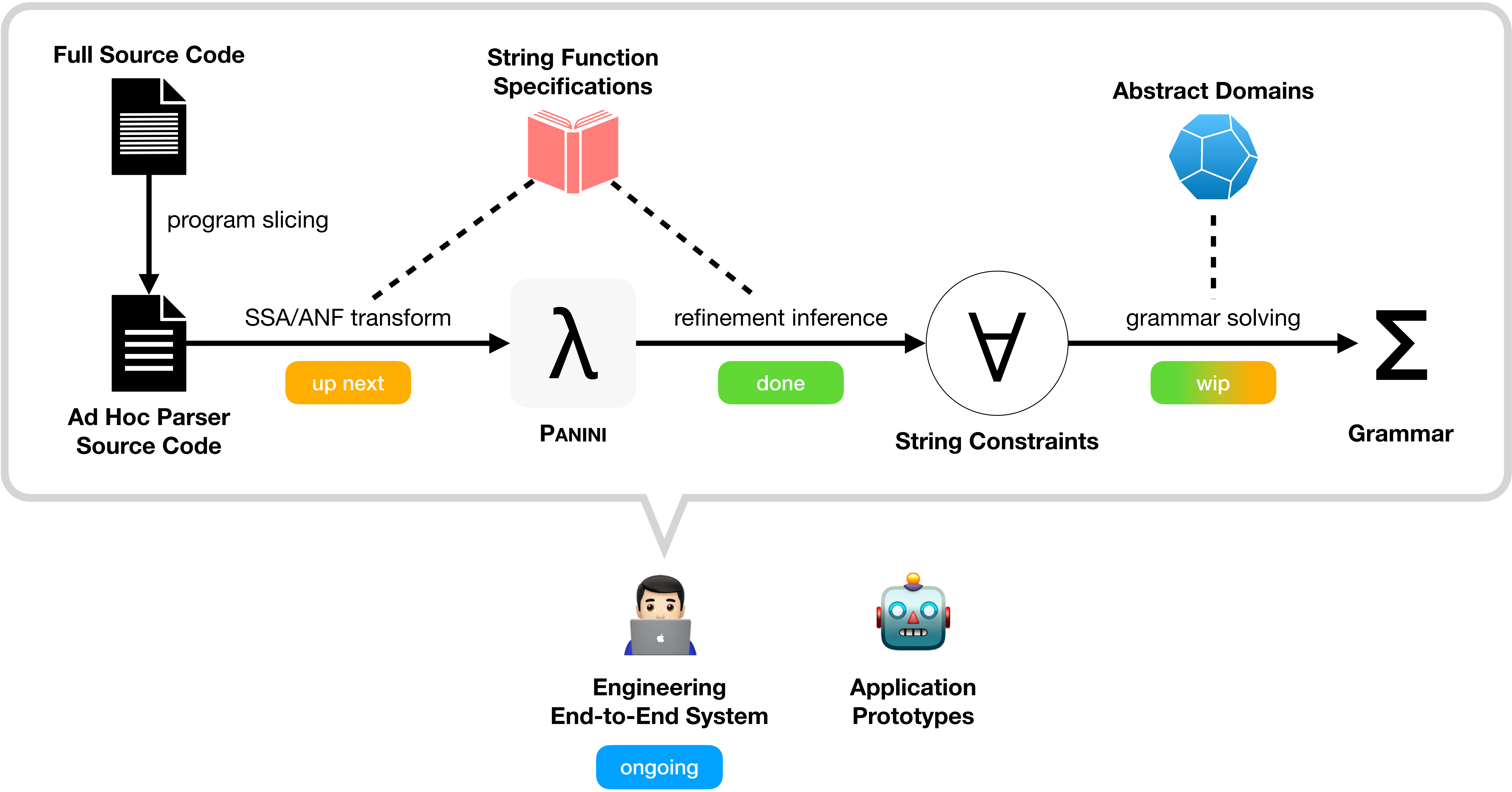

Abstract Domains
SSA/ANF transform
up next
refinement inference
done
grammar solving
wip
Ad Hoc Parser
Source Code
PANINI
String Constraints
Grammar



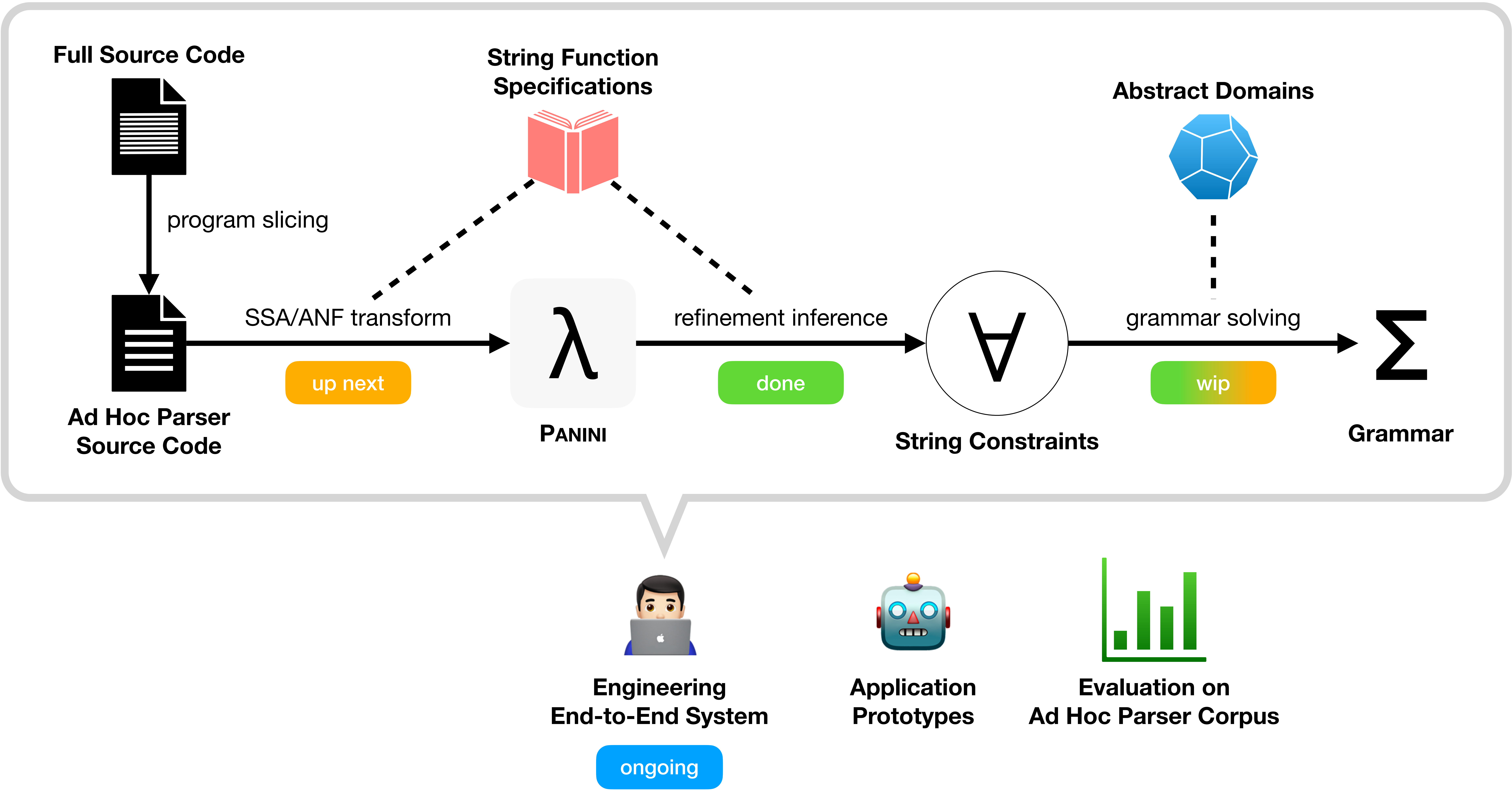
Full Source Code
program slicing
Ad Hoc Parser
Source Code
SSA/ANF transform
up next
λ
PANINI
refinement inference
done
∀
String Constraints
Abstract Domains
grammar solving
wip
Σ
Grammar

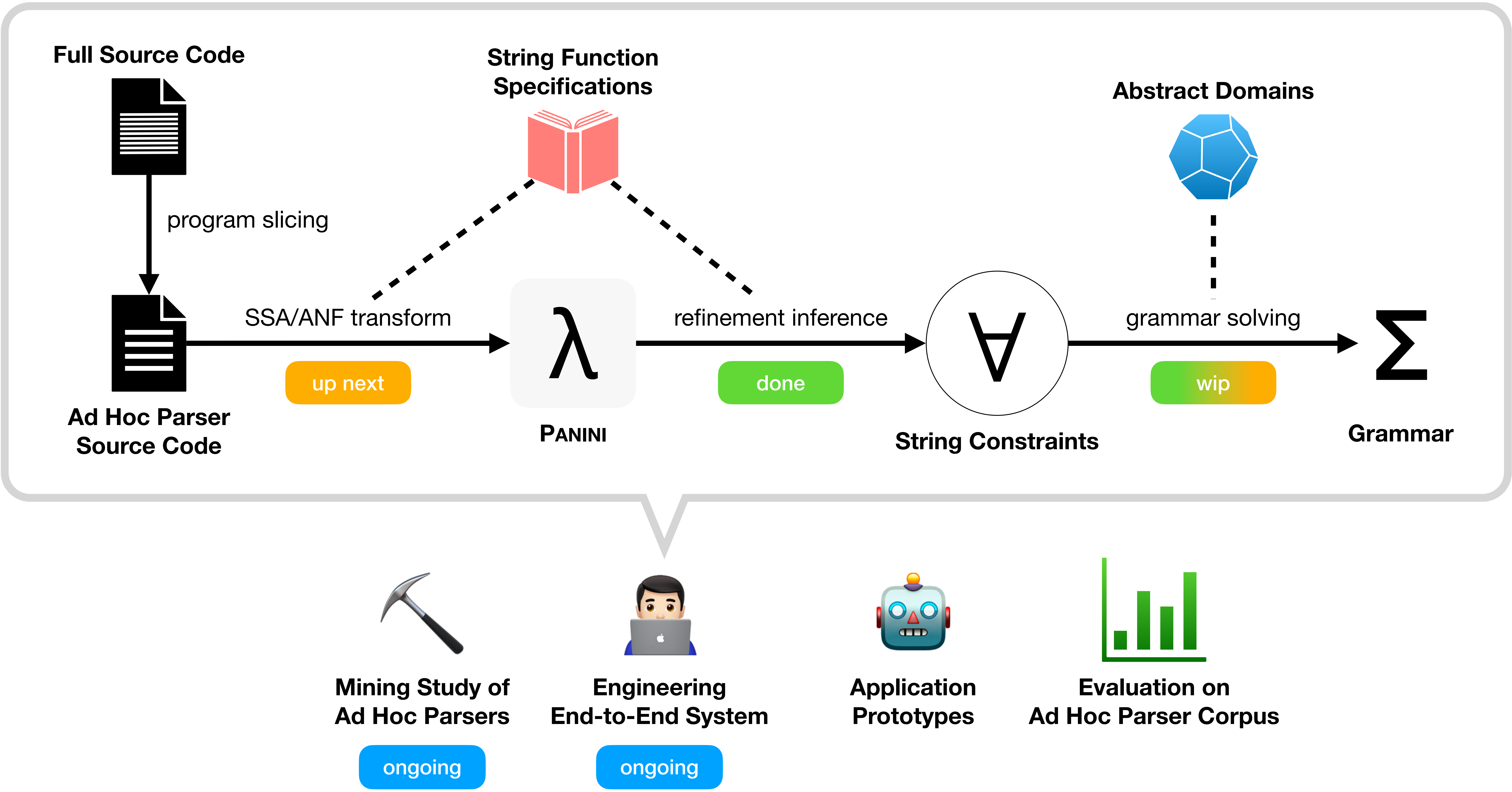

Full Source Code
program slicing
Ad Hoc Parser
Source Code
String Function
Specifications
SSA/ANF transform
up next
λ
PANINI
refinement inference
done
∀
String Constraints
Abstract Domains
grammar solving
wip
Σ
Grammar


Full Source Code
String Function Specifications
Abstract Domains
program slicing
SSA/ANF transform
up next
λ
refinement inference
done
∀
grammar solving
wip
Σ
Ad Hoc Parser Source Code
PANINI
String Constraints
Grammar
Engineering End-to-End System
ongoing

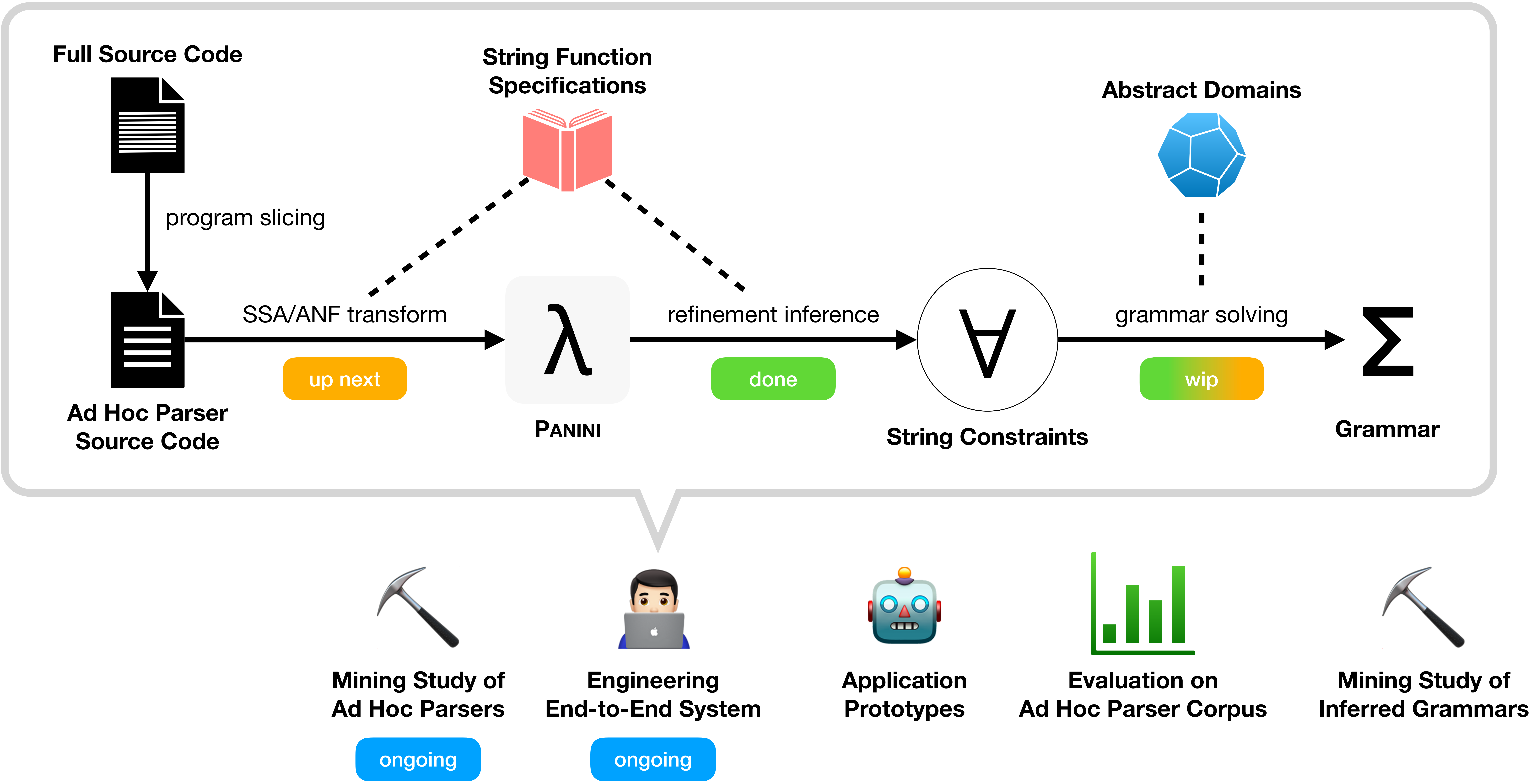


Full Source Code
String Function Specifications
Abstract Domains
program slicing
SSA/ANF transform
up next
λ
PANINI
refinement inference
done
∀
String Constraints
grammar solving
wip
Σ
Grammar
Ad Hoc Parser Source Code
Engineering End-to-End System
ongoing
Application Prototypes

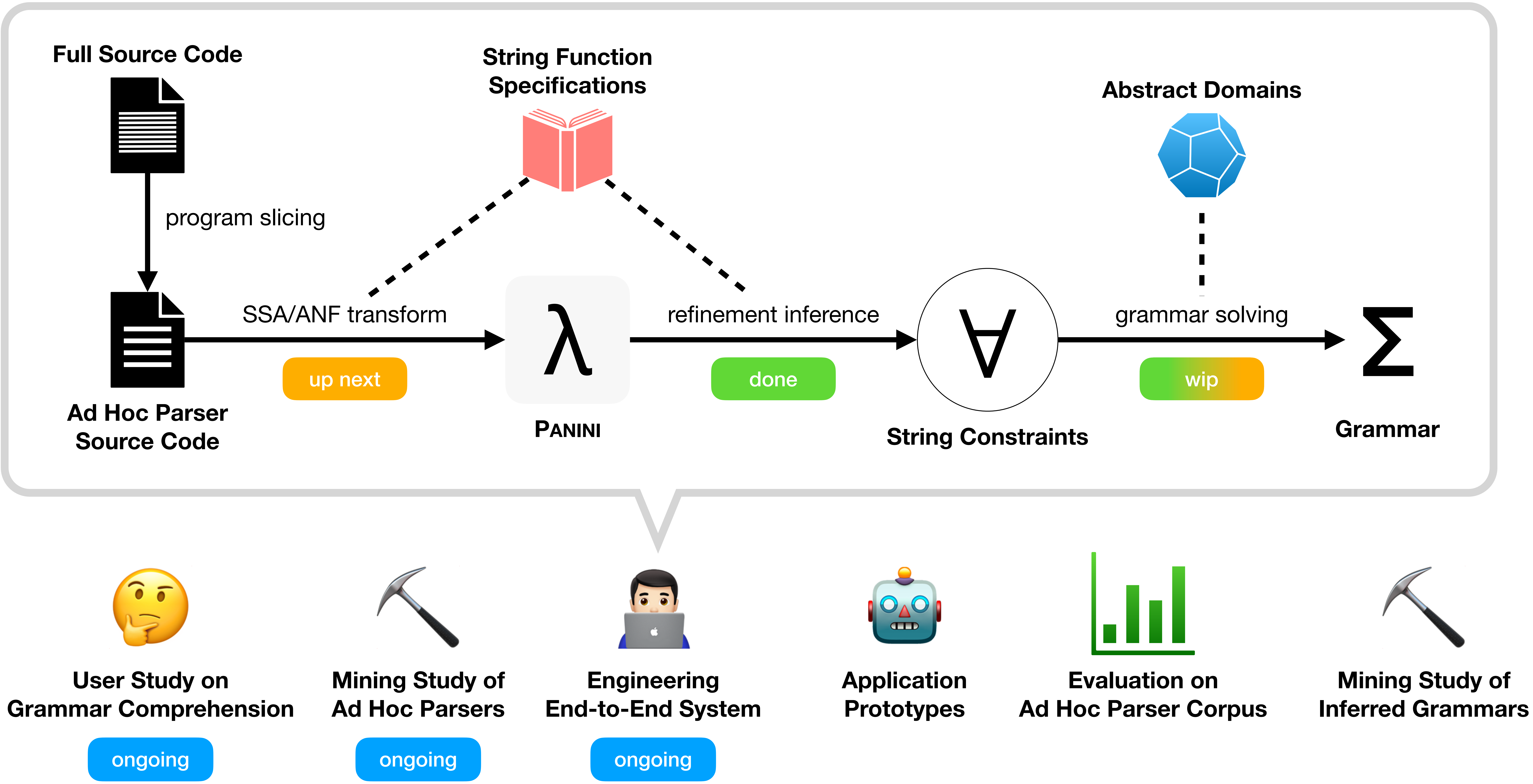

Full Source Code
program slicing
Ad Hoc Parser
Source Code
SSA/ANF transform
up next
String Function
Specifications
λ
PANINI
refinement inference
done
∀
String Constraints
Abstract Domains
grammar solving
wip
Σ
Grammar
Engineering
End-to-End System
ongoing
Application
Prototypes
Evaluation on
Ad Hoc Parser Corpus



Full Source Code
program slicing
Ad Hoc Parser
Source Code
SSA/ANF transform
up next
String Function
Specifications
λ
PANINI
refinement inference
done
∀
String Constraints
Abstract Domains
grammar solving
wip
Σ
Grammar
Mining Study of
Ad Hoc Parsers
ongoing
Engineering
End-to-End System
ongoing
Application
Prototypes
Evaluation on
Ad Hoc Parser Corpus

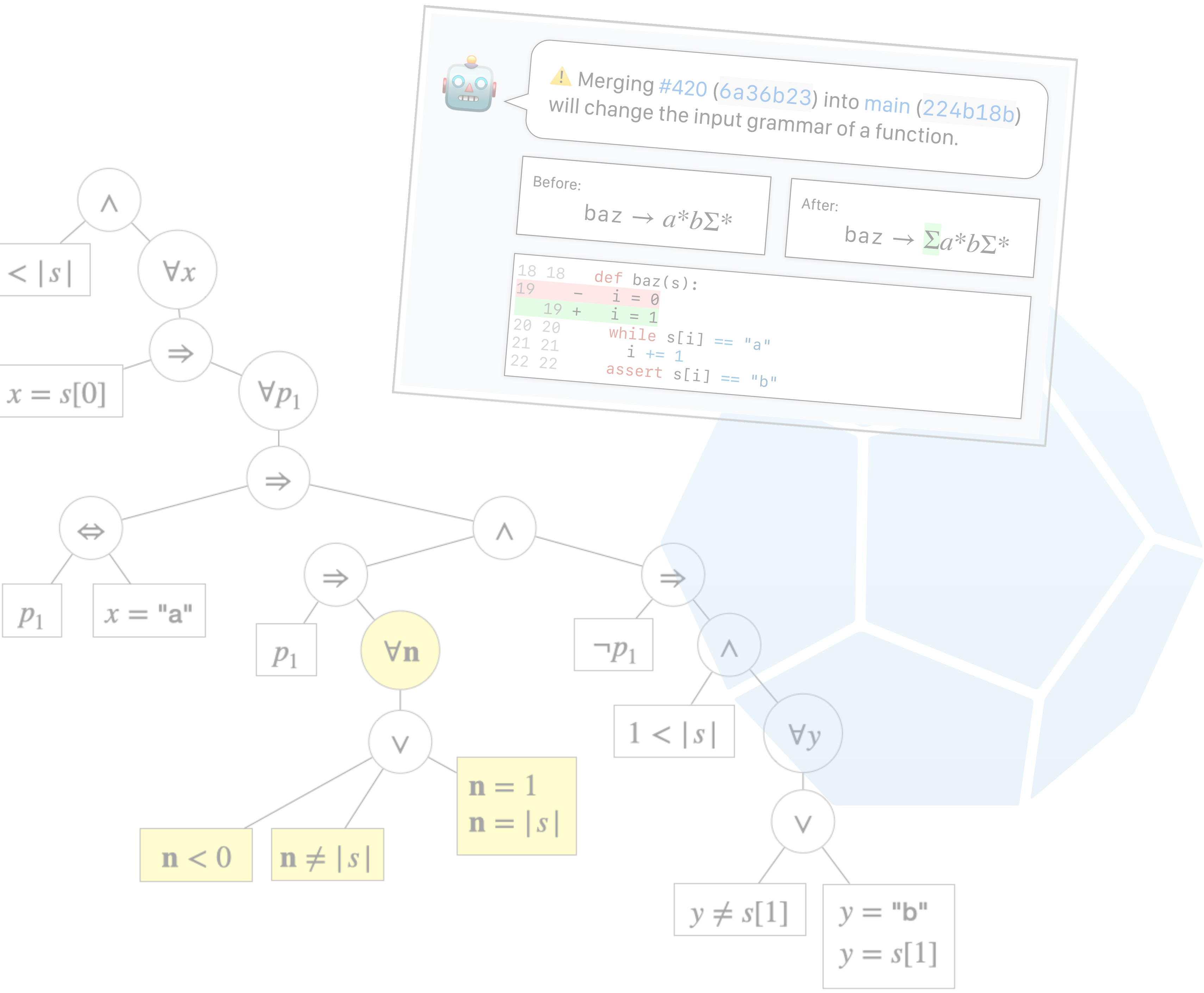


Full Source Code
program slicing
Ad Hoc Parser
Source Code
String Function
Specifications
SSA/ANF transform
up next
λ
PANINI
refinement inference
done
∀
String Constraints
Abstract Domains
grammar solving
wip
Σ
Grammar
Mining Study of
Ad Hoc Parsers
ongoing
Engineering
End-to-End System
ongoing
Application
Prototypes
Evaluation on
Ad Hoc Parser Corpus
Mining Study of
Inferred Grammars

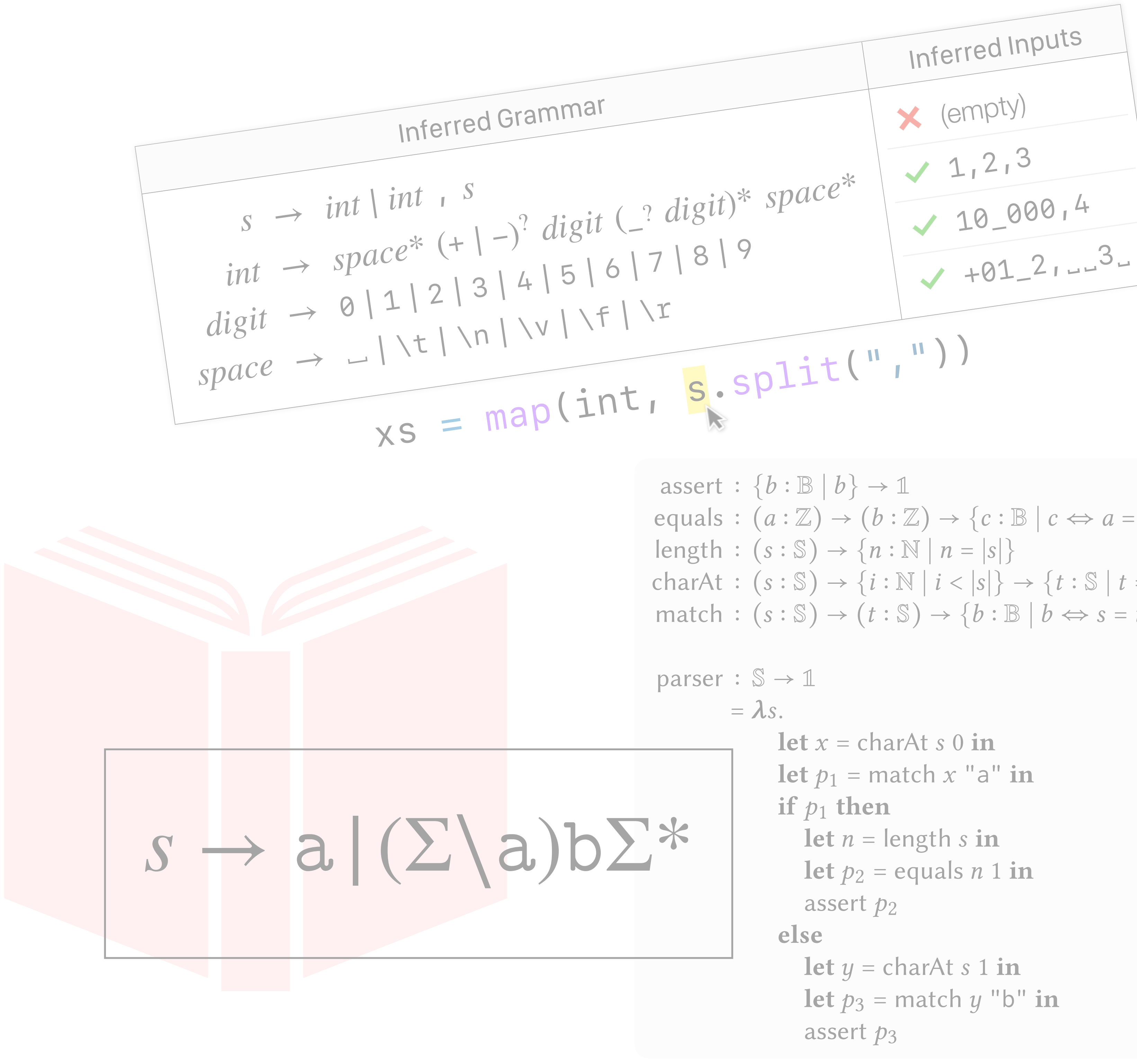

Full Source Code
program slicing
Ad Hoc Parser
Source Code
String Function
Specifications
SSA/ANF transform
up next
λ
PANINI
refinement inference
done
∀
String Constraints
Abstract Domains
grammar solving
wip
Σ
Grammar
User Study on
Grammar Comprehension
ongoing
Mining Study of
Ad Hoc Parsers
ongoing
Engineering
End-to-End System
ongoing
Application
Prototypes
Evaluation on
Ad Hoc Parser Corpus
Mining Study of
Inferred Grammars

# Grammar Inference for Ad Hoc Parsers

https://mcschroeder.github.io/#splash2022




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


**Doctoral Symposium, SPLASH 2022**

**Tāmaki Makaurau, Aotearoa**
**Auckland, New Zealand**

TU WIEN Informatics