

# Static Inference of Regular Grammars for Ad Hoc Parsers

**Michael Schröder** and Jürgen Cito

TU Wien, Vienna, Austria

OOPSLA 2025  
Singapore

**FWF** Österreichischer  
Wissenschaftsfonds

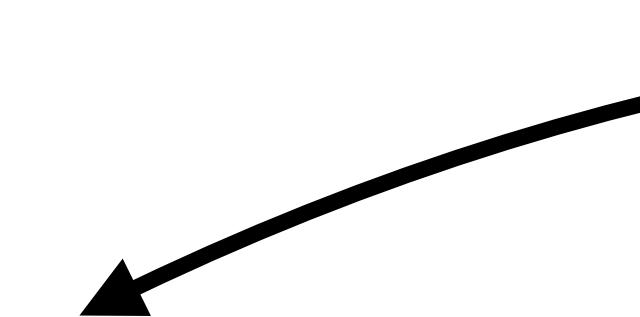
This research was funded in whole or in part by the  
Austrian Science Fund (FWF) [[10.55776/PIN3275223](https://doi.org/10.55776/PIN3275223)].



Informatics

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

What is the type of s?



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

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

```
>>> parser("a")
```

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

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

```
>>> parser("a")
>>> █
```

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

```
>>> parser("a")
>>> parser("b")
```

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

```
>>> parser("a")
>>> parser("b")
Traceback (most recent call last):
  File "<python-input-2>", line 1, in <module>
    parser("b")
    ~~~~~^~~~^
  File "<python-input-0>", line 5, in parser
    assert s[1] == "b"
    ~^~^
IndexError: string index out of range
>>> █
```

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

```
>>> parser("a")
>>> parser("b")
Traceback (most recent call last):
  File "<python-input-2>", line 1, in <module>
    parser("b")
    ~~~~~^~~~^
  File "<python-input-0>", line 5, in parser
    assert s[1] == "b"
    ~^~^
IndexError: string index out of range
>>> parser("ab")
```

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

```
>>> parser("a")
>>> parser("b")
Traceback (most recent call last):
  File "<python-input-2>", line 1, in <module>
    parser("b")
    ~~~~~^~~~~~
  File "<python-input-0>", line 5, in parser
    assert s[1] == "b"
    ~^~~
IndexError: string index out of range
>>> parser("ab")
Traceback (most recent call last):
  File "<python-input-3>", line 1, in <module>
    parser("ab")
    ~~~~~^~~~~~
  File "<python-input-0>", line 3, in parser
    assert len(s) == 1
    ~~~~~^~~~~~
```

**AssertionError**

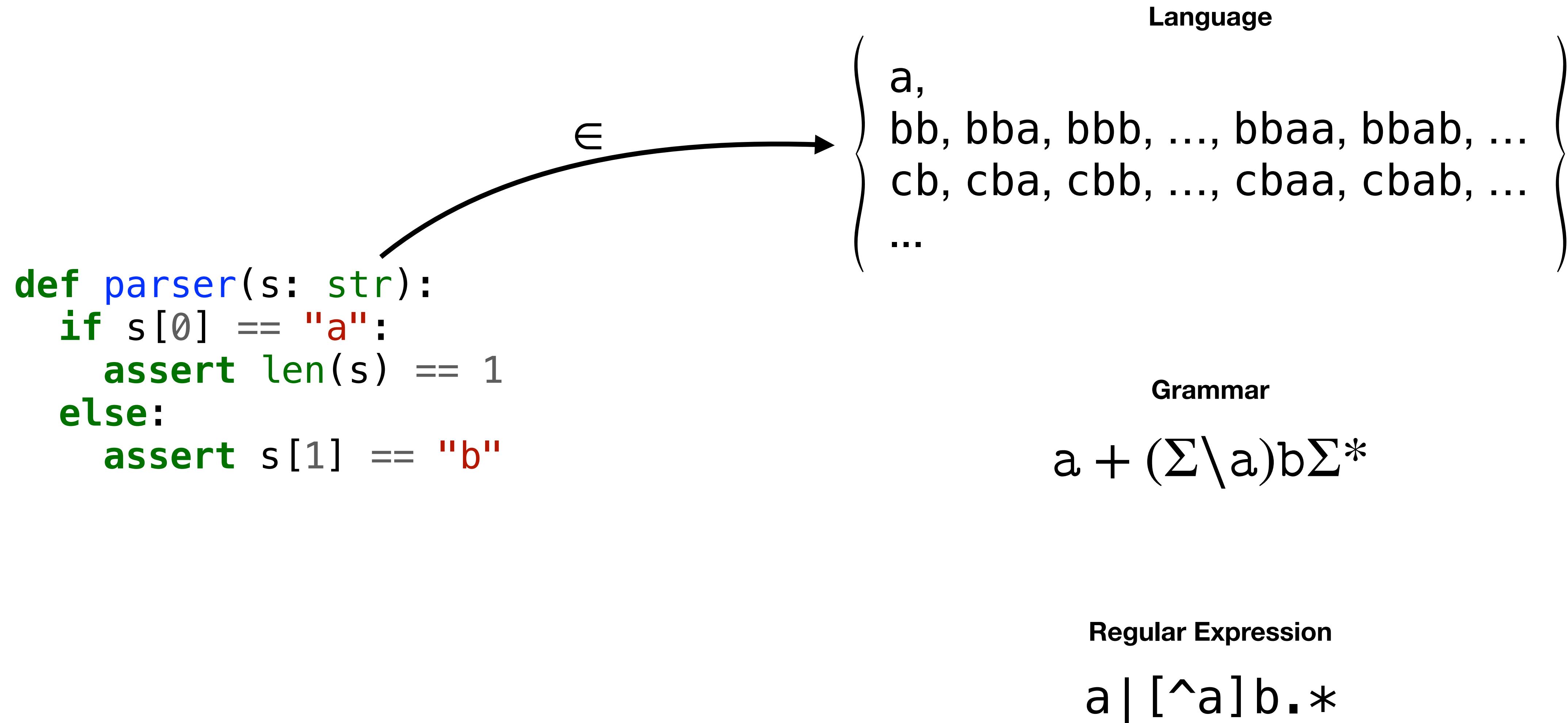
```
>>> |
```

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

```
>>> parser("a")
>>> parser("b")
Traceback (most recent call last):
  File "<python-input-2>", line 1, in <module>
    parser("b")
    ~~~~~^~~~~~
  File "<python-input-0>", line 5, in parser
    assert s[1] == "b"
    ~^~~
IndexError: string index out of range
>>> parser("ab")
Traceback (most recent call last):
  File "<python-input-3>", line 1, in <module>
    parser("ab")
    ~~~~~^~~~~~
  File "<python-input-0>", line 3, in parser
    assert len(s) == 1
    ~~~~~^~~~~~
AssertionError
>>> parser("bb")|
```

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

```
>>> parser("a")
>>> parser("b")
Traceback (most recent call last):
  File "<python-input-2>", line 1, in <module>
    parser("b")
    ~~~~~^~~~~~
  File "<python-input-0>", line 5, in parser
    assert s[1] == "b"
    ~^~~
IndexError: string index out of range
>>> parser("ab")
Traceback (most recent call last):
  File "<python-input-3>", line 1, in <module>
    parser("ab")
    ~~~~~^~~~~~
  File "<python-input-0>", line 3, in parser
    assert len(s) == 1
    ~~~~~^~~~~~
AssertionError
>>> parser("bb")
>>> |
```



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

What is the type of *s*?  
What is the grammar of *str*?

Grammar

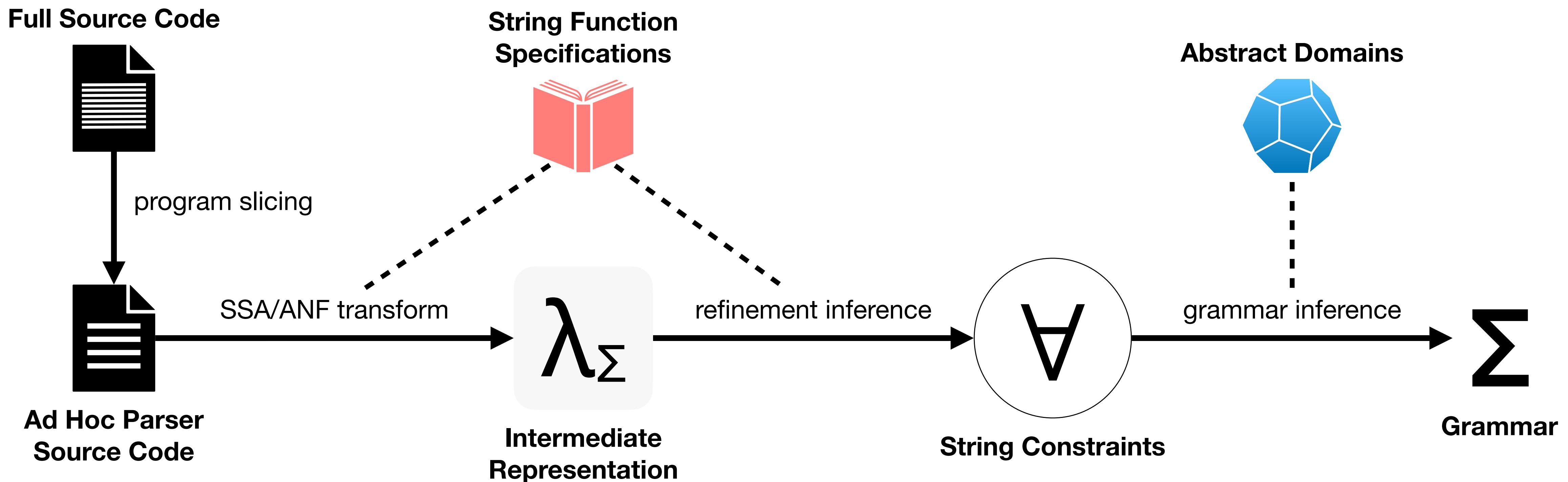
$$a + (\Sigma \setminus a)b\Sigma^*$$

Regular Expression

$$a | [{}^a]b.*$$

# PANINI

*“grammars for free”*



# Intermediate Representation $\lambda_{\Sigma}$

- simple  $\lambda$ -calculus in ANF

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

SSA/ANF transformation

```
parser = λs:S.
  let x = charAt s 0 in
  let p1 = eqChar x 'a' in
  if p1 then
    let n = length s in
    let p2 = eq n 1 in
    assert p2
  else
    let y = charAt s 1 in
    let p3 = eqChar y 'b' in
    assert p3
```

# Intermediate Representation $\lambda_\Sigma$

- simple  $\lambda$ -calculus in ANF
- common string operations as axiomatic specifications

Python	$\lambda_\Sigma$ specification
<pre> <b>def parser(s: str):</b>     <b>if</b> s[0] == "a":         <b>assert</b> len(s) == 1     <b>else:</b>         <b>assert</b> s[1] == "b" </pre>	<p style="text-align: center;">SSA/ANF transformation</p>

---

<b>assert</b> b	$\text{assert} : \{b : \mathbb{B} \mid b\} \rightarrow \mathbb{1}$
a == b	$\text{eq} : (a : \mathbb{Z}) \rightarrow (b : \mathbb{Z}) \rightarrow \{c : \mathbb{B} \mid c \Leftrightarrow a = b\}$
<b>len</b> (s)	$\text{length} : (s : \mathbb{S}) \rightarrow \{n : \mathbb{N} \mid n =  s \}$
s[i]	$\text{charAt} : (s : \mathbb{S}) \rightarrow \{i : \mathbb{N} \mid i <  s \} \rightarrow \{c : \mathbb{C}\text{h} \mid c = s[i]\}$
s == t	$\text{eqChar} : (s : \mathbb{C}\text{h}) \rightarrow (t : \mathbb{C}\text{h}) \rightarrow \{b : \mathbb{B} \mid b \Leftrightarrow s = t\}$

```

parser =  $\lambda s : \mathbb{S}.$ 
    let x = charAt s 0 in
    let p1 = eqChar x 'a' in
    if p1 then
        let n = length s in
        let p2 = eq n 1 in
        assert p2
    else
        let y = charAt s 1 in
        let p3 = eqChar y 'b' in
        assert p3

```

# Intermediate Representation $\lambda_\Sigma$

- simple  $\lambda$ -calculus in ANF
- common string operations as axiomatic specifications
- refinement type system à la *Liquid Types*

Python	$\lambda_\Sigma$ specification
<pre> <b>def</b> parser(s: str):     <b>if</b> s[0] == "a":         <b>assert</b> len(s) == 1     <b>else</b>:         <b>assert</b> s[1] == "b" </pre>	<p style="text-align: center;">SSA/ANF transformation</p>

---

<b>assert</b> b	$\text{assert} : \{b : \mathbb{B} \mid b\} \rightarrow \mathbb{1}$
a == b	$\text{eq} : (a : \mathbb{Z}) \rightarrow (b : \mathbb{Z}) \rightarrow \{c : \mathbb{B} \mid c \Leftrightarrow a = b\}$
<b>len</b> (s)	$\text{length} : (s : \mathbb{S}) \rightarrow \{n : \mathbb{N} \mid n =  s \}$
s[i]	$\text{charAt} : (s : \mathbb{S}) \rightarrow \{i : \mathbb{N} \mid i <  s \} \rightarrow \{c : \mathbb{C}\mathbb{h} \mid c = s[i]\}$
s == t	$\text{eqChar} : (s : \mathbb{C}\mathbb{h}) \rightarrow (t : \mathbb{C}\mathbb{h}) \rightarrow \{b : \mathbb{B} \mid b \Leftrightarrow s = t\}$

```

parser : {s : $ | ?} → 1
parser = λs:$.
  let x = charAt s 0 in
  let p1 = eqChar x 'a' in
  if p1 then
    let n = length s in
    let p2 = eq n 1 in
    assert p2
  else
    let y = charAt s 1 in
    let p3 = eqChar y 'b' in
    assert p3

```

# Refinement Inference

- type system generates *verification conditions* (VCs)
- validity of VC entails correctness of type

verification condition

$$\begin{aligned} \forall s. \kappa(s) \Rightarrow & \\ (\forall v_1. v_1 = 0 \Rightarrow v_1 \geq 0 \wedge v_1 < |s|) \wedge & \\ (\forall x. x = s[0] \Rightarrow & \\ (\forall p_1. p_1 = \text{true} \Leftrightarrow x = 'a' \Rightarrow & \\ (p_1 = \text{true} \Rightarrow & \\ (\forall n. n \geq 0 \wedge n = |s| \Rightarrow n = 1)) \wedge & \\ (p_1 = \text{false} \Rightarrow & \\ (\forall v_2. v_2 = 1 \Rightarrow v_2 \geq 0 \wedge v_2 < |s|) \wedge & \\ (\forall y. y = s[1] \Rightarrow y = 'b'))))) & \end{aligned}$$

$\models$

```

parser : {s:S | κ(s)} → 1
parser = λs:S.
  let x = charAt s 0 in
  let p1 = eqChar x 'a' in
  if p1 then
    let n = length s in
    let p2 = eq n 1 in
    assert p2
  else
    let y = charAt s 1 in
    let p3 = eqChar y 'b' in
    assert p3
  
```

# Refinement Inference

- type system generates *verification conditions* (VCs)
- $\kappa$  variables represent unknown refinements that still need to be found

verification condition

$$\begin{aligned} \forall s. \kappa(s) \Rightarrow & \\ (\forall v_1. v_1 = 0 \Rightarrow v_1 \geq 0 \wedge v_1 < |s|) \wedge & \\ (\forall x. x = s[0] \Rightarrow & \\ (\forall p_1. p_1 = \text{true} \Leftrightarrow x = 'a' \Rightarrow & \\ (p_1 = \text{true} \Rightarrow & \\ (\forall n. n \geq 0 \wedge n = |s| \Rightarrow n = 1)) \wedge & \\ (p_1 = \text{false} \Rightarrow & \\ (\forall v_2. v_2 = 1 \Rightarrow v_2 \geq 0 \wedge v_2 < |s|) \wedge & \\ (\forall y. y = s[1] \Rightarrow y = 'b'))))) & \end{aligned}$$

$\models$

```

parser : {s:S |  $\kappa(s)$ } → 1
parser =  $\lambda s:S.$ 
        let x = charAt s 0 in
        let p1 = eqChar x 'a' in
        if p1 then
            let n = length s in
            let p2 = eq n 1 in
            assert p2
        else
            let y = charAt s 1 in
            let p3 = eqChar y 'b' in
            assert p3
    
```

# Grammar Inference

- key insight: the top-level VC of a parser is always  $\forall s. \kappa(s) \Rightarrow \varphi$
- the solution  $\kappa(s) \mapsto \varphi$  trivially validates the VC

verification condition

$$\begin{aligned} \forall s. \kappa(s) \Rightarrow & \\ & (\forall v_1. v_1 = 0 \Rightarrow v_1 \geq 0 \wedge v_1 < |s|) \wedge \\ & (\forall x. x = s[0] \Rightarrow \\ & (\forall p_1. p_1 = \text{true} \Leftrightarrow x = 'a' \Rightarrow \\ & (p_1 = \text{true} \Rightarrow \\ & (\forall n. n \geq 0 \wedge n = |s| \Rightarrow n = 1)) \wedge \\ & (p_1 = \text{false} \Rightarrow \\ & (\forall v_2. v_2 = 1 \Rightarrow v_2 \geq 0 \wedge v_2 < |s|) \wedge \\ & (\forall y. y = s[1] \Rightarrow y = 'b'))))) \end{aligned}$$

string constraint that  
precisely captures  
parsing operations on s

# Grammar Inference

- problem: the string constraint is not a very practical grammar
- it is a first-order formula as big as the original program!

**parser** : {*s*:*S* |  $(\forall v_1. v_1 = 0 \Rightarrow v_1 \geq 0 \wedge v_1 < |s|) \wedge$   
 $(\forall x. x = s[0] \Rightarrow$   
 $(\forall p_1. p_1 = \text{true} \Leftrightarrow x = 'a' \Rightarrow$   
 $(p_1 = \text{true} \Rightarrow$   
 $(\forall n. n \geq 0 \wedge n = |s| \Rightarrow n = 1)) \wedge$   
 $(p_1 = \text{false} \Rightarrow$   
 $(\forall v_2. v_2 = 1 \Rightarrow v_2 \geq 0 \wedge v_2 < |s|) \wedge$   
 $(\forall y. y = s[1] \Rightarrow y = 'b'))))$  }  $\rightarrow$  *T*

**AI**

# Abstract Interpretation

# Abstract Interpretation of ad hoc parsers represented by first-order formulas

$$\overbrace{\Gamma}^{\text{axioms}} \vdash P \nearrow \overbrace{\{s : \mathbb{S} \mid \kappa(s)\} \rightarrow \mathbb{1}}^{\text{incomplete refinement}} \exists \quad \overbrace{\forall s. \kappa(s) \Rightarrow \varphi}^{\text{verification condition}}$$

# Abstract Interpretation of ad hoc parsers represented by first-order formulas

$$\underbrace{\Gamma \vdash P \nearrow \{s : \mathbb{S} \mid \kappa(s)\} \rightarrow \perp}_{\text{axioms}} \quad \underbrace{\exists \quad \forall s. \kappa(s) \Rightarrow \varphi}_{\text{incomplete refinement}} \quad \underbrace{\text{verification condition}}$$

$$\mathfrak{M}, [s \mapsto t] \models \varphi \iff P \text{ successfully parses } t \iff t \in L(P)$$

# Abstract Interpretation of ad hoc parsers represented by first-order formulas

$$\overbrace{\Gamma \vdash P \nearrow \{s : \mathbb{S} \mid \kappa(s)\} \rightarrow \mathbb{1}}^{\text{incomplete refinement}} \exists \quad \overbrace{\forall s. \kappa(s) \Rightarrow \varphi}^{\text{verification condition}}$$

$$\mathfrak{M}, [s \mapsto t] \models \varphi \iff P \text{ successfully parses } t \iff t \in L(P)$$

$$\mathfrak{M}, \sigma \models \varphi \iff \sigma \in \llbracket \varphi \rrbracket \iff \sigma(s) \in L(P)$$

# Abstract Interpretation of ad hoc parsers represented by first-order formulas

$$\overbrace{\Gamma \vdash P \nearrow \{s : \mathbb{S} \mid \kappa(s)\} \rightarrow \mathbb{1}}^{\text{incomplete refinement}} \exists \overbrace{\forall s. \kappa(s) \Rightarrow \varphi}^{\text{verification condition}}$$

$$\mathfrak{M}, [s \mapsto t] \models \varphi \iff P \text{ successfully parses } t \iff t \in L(P)$$

$$\mathfrak{M}, \sigma \models \varphi \iff \sigma \in \llbracket \varphi \rrbracket \iff \sigma(s) \in L(P)$$

$$\mathfrak{M}, \hat{\sigma} \models \varphi \iff \hat{\sigma} = \llbracket \varphi \rrbracket^\# \implies \hat{\sigma}(s) \subseteq L(P)$$

# Abstract Semantics of String Constraints

$$\llbracket \varphi \rrbracket^\# \doteq \{x \mapsto \llbracket \varphi \rrbracket \uparrow_x \mid x \in \text{vars}(\varphi)\}$$

$$\llbracket \varphi_1 \wedge \varphi_2 \rrbracket \uparrow_x \doteq \llbracket \varphi_1 \rrbracket \uparrow_x \sqcap \llbracket \varphi_2 \rrbracket \uparrow_x \quad \text{if } \varphi_1, \varphi_2 \text{ quantifier-free}$$

$$\llbracket \varphi_1 \vee \varphi_2 \rrbracket \uparrow_x \doteq \llbracket \varphi_1 \rrbracket \uparrow_x \sqcup \llbracket \varphi_2 \rrbracket \uparrow_x \quad \text{if } \varphi_1, \varphi_2 \text{ quantifier-free}$$

$$\llbracket \neg \varphi \rrbracket \uparrow_x \doteq \neg \llbracket \varphi \rrbracket \uparrow_x \quad \text{if } \varphi \text{ quantifier-free}$$

$$\llbracket \varphi \rrbracket \uparrow_x \doteq \llbracket \text{qelim}(\varphi) \rrbracket \uparrow_x$$

$$\llbracket \rho \rrbracket \uparrow_x \doteq \begin{cases} \omega, & \text{as defined by domain;} \\ \langle \rho[x := \bullet] \rangle, & \text{otherwise.} \end{cases}$$

**quantifier elimination (excerpt)**

$$\text{qelim1}(x, R) \doteq \hat{R} \wedge R_{\bar{x}}$$

where

$$\hat{R} = \left\{ \omega_1 \between \omega_2 \mid (\omega_1, \omega_2) \in \binom{E}{2} \right\}$$

$$E = \bigcup \{\llbracket \rho \rrbracket \uparrow_x \mid \rho \in R_x\}$$

$$R_x = \{\rho \in R \mid x \in \text{vars}(\rho)\}$$

$$R_{\bar{x}} = \{\rho \in R \mid x \notin \text{vars}(\rho)\}$$

**relational string semantics (excerpt)**

$$\llbracket |s| \between \hat{n} \rrbracket \uparrow_s \doteq \Sigma^{\hat{n}}$$

$$\llbracket s[\hat{i}] \between \hat{c} \rrbracket \uparrow_s \doteq \Sigma^{\hat{i}} \hat{c} \Sigma^*$$

$$\llbracket s[|s| - \hat{i}] \between \hat{c} \rrbracket \uparrow_s \doteq \Sigma^* \hat{c} \Sigma^{\hat{i}-1}$$

$$\llbracket s[\hat{i}..\hat{j}] \between \hat{t} \rrbracket \uparrow_s \doteq \Sigma^{\hat{i}} (\Sigma^{\hat{j}-\hat{i}+1} \sqcap \hat{t}) \Sigma^*$$

# Abstract Domains for Grammar Inference

unit	$\wp(\mathbb{1})$	=	$\hat{1}$	the two-element lattice
Booleans	$\wp(\mathbb{B})$	=	$\hat{\mathbb{B}}$	Boolean subset lattice
integers	$\wp(\mathbb{Z})$	$\supseteq$	$\hat{\mathbb{Z}}$	open-ended interval lists
characters	$\wp(\mathbb{Ch})$	=	$\hat{\mathbb{C}}$	Unicode character sets
strings	$\wp(\mathbb{S})$	$\supseteq$	$\hat{\mathbb{S}}$	regular expressions over $\hat{\mathbb{C}}$

we under-approximate  
super-regular languages

# Grammar Inference

```
parser : {s:$ | [[ (forall v1. v1 = 0 => v1 ≥ 0 ∧ v1 < |s|) ∧  
          (forall x. x = s[0] =>  
            (forall p1. p1 = true ⇔ x = 'a') =>  
            (p1 = true =>  
              (forall n. n ≥ 0 ∧ n = |s| => n = 1)) ∧  
            (p1 = false =>  
              (forall v2. v2 = 1 => v2 ≥ 0 ∧ v2 < |s|) ∧  
              (forall y. y = s[1] => y = 'b'))))) ]# } → 1
```

# Grammar Inference

parser : {s:\$ |  $s \in (a + (\Sigma \setminus a)b\Sigma^*)^*$ } → 1

# Evaluation

- benchmark dataset of 204 regular ad hoc parsers written in Python
- classified into *straight-line*, *conditional*, and *loop* programs
- manually derived ground truth grammars to compute precision and recall

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

a | [^a]b.\*

```
def getAddrSpec(email):
    b1 = email.index('<',0)+1
    b2 = email.index('>',b1)
    return email[b1:b2]
```

[^<>]\*<[^>]\*>.\*

```
def lsb_check(s):
    i = 0
    while i < len(s)-1:
        assert s[i] == '0'
        i += 1
    assert s[i] == '1'
```

0\*1

# Evaluation

- 100% precision: *PANINI never over-approximates!*
- 93% overall recall (100% for non-loop programs)

Type	Parser Example	#	Inferred Language				SR	P	R	Time (s)	
			=	<	$\emptyset$	Error					
Straight	getAddrSpec	[^<>]*<[^>]*>.*	89	84	0	0	5	.94	1.00	1.00	0.16 ±0.27
+ Cond.	Figure 3	a [^\n]b.*	51	50	0	0	1	.98	1.00	1.00	0.09 ±0.05
+ Loops	lsb_check	0*1	64	40	7	7	10	.84	1.00	.76	2.32 ±4.77
			204	174	7	7	16	.92	1.00	.93	0.82 ±2.85

SR = success rate, P = precision, R = recall

# Evaluation

- 100% precision: *PANINI never over-approximates!*
- 93% overall recall (100% for non-loop programs)

Type	Parser Example	#	Inferred Language				SR	P	R	Time (s)
			=	<	$\emptyset$	Error				
Straight + Cond. + Loops	getAddrSpec	89	84	0	0	5	.94	1.00	1.00	0.16 ±0.27
	Figure 3	51	50	0	0	1	.98	1.00	1.00	0.09 ±0.05
	lsb_check	64	40	7	7	10	.84	1.00	.76	2.32 ±4.77
		204	174	7	7	16	.92	1.00	.93	0.82 ±2.85

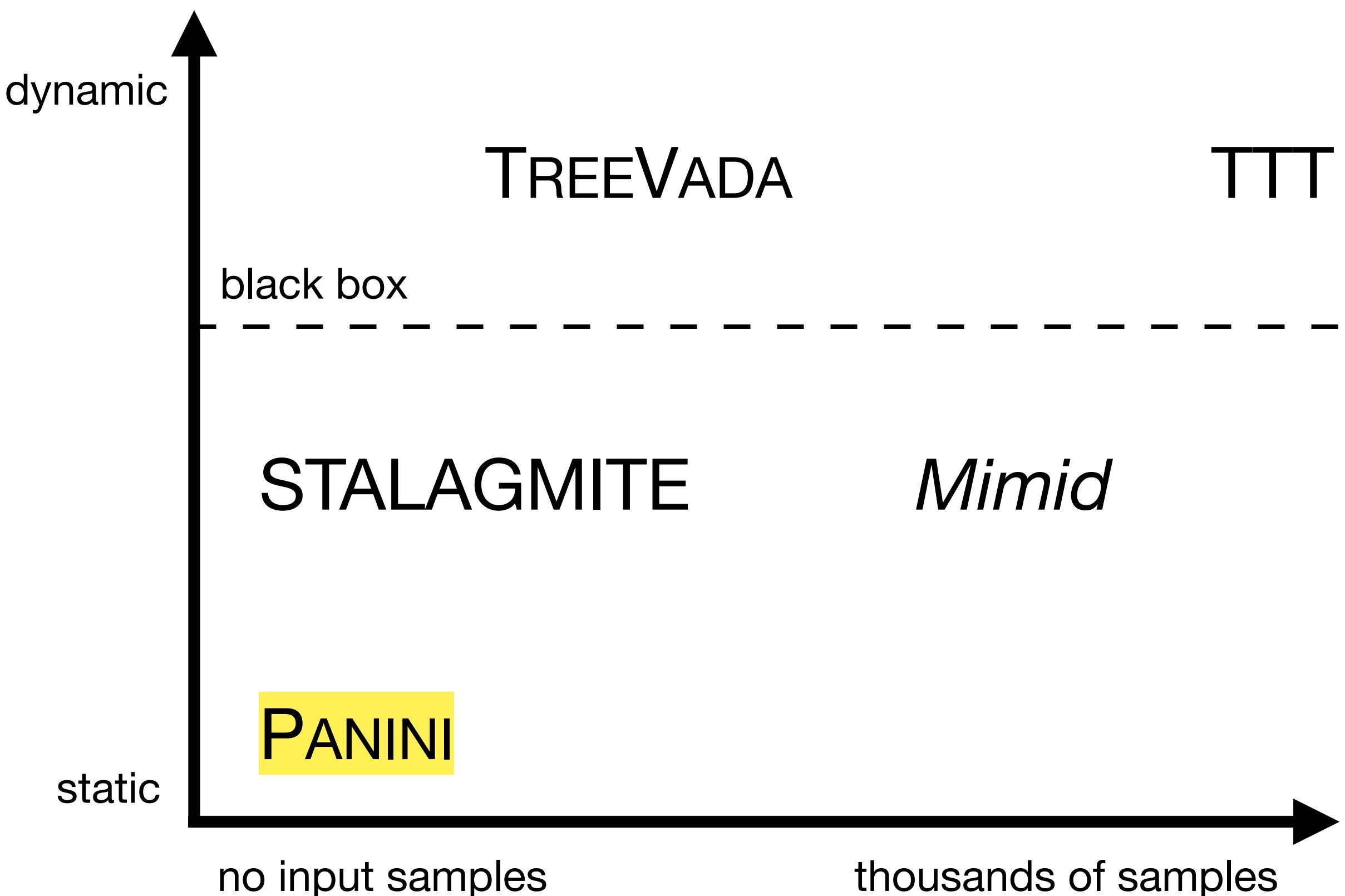
SR = success rate, P = precision, R = recall

insufficient invariants  
weak abstract domains

# Comparison with Other Approaches

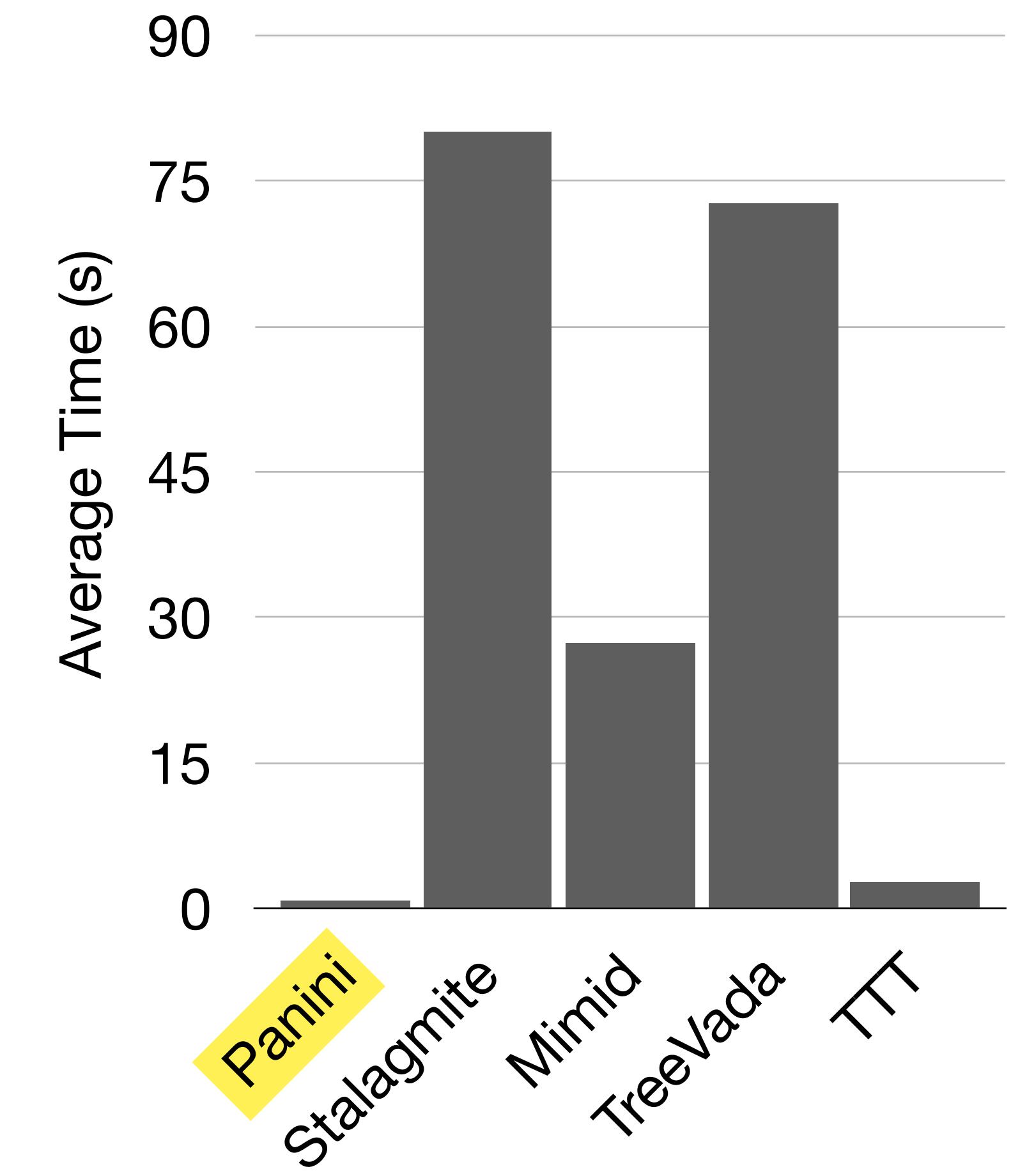
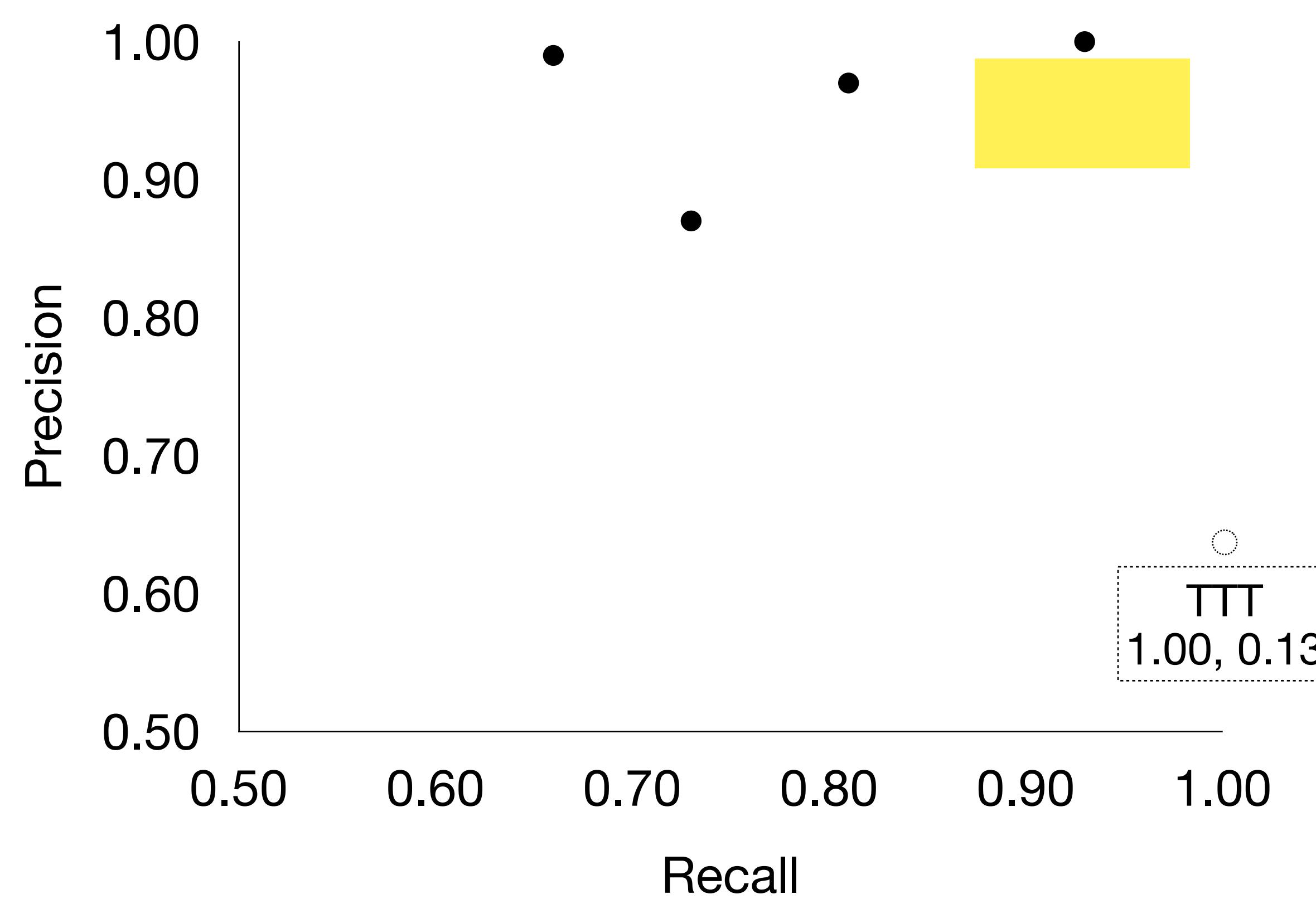
Panini is purely static and requires no pre-existing input samples

- **STALAGMITE**  
(Bettscheider and Zeller 2024)
- *Mimid*  
(Gopinath et al. 2020)
- **TREEVADA**  
(Arefin et al. 2024)
- **TTT**  
(Isberner 2015; Isberner et al. 2014)

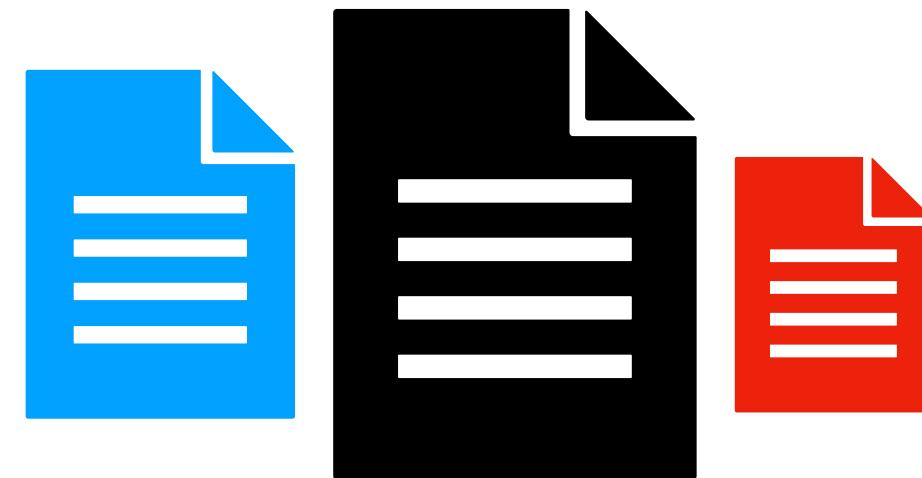


# Comparison with Other Approaches

Panini produces better grammars in less time

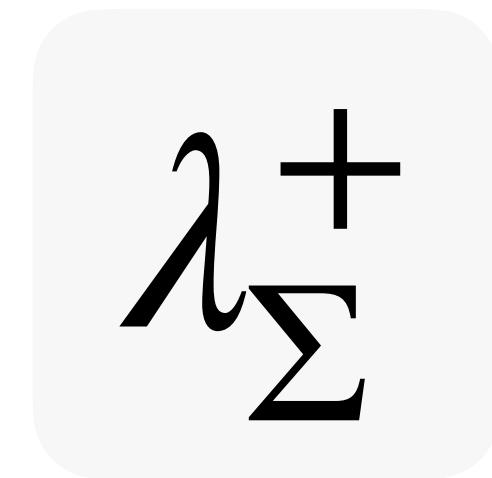


# Future Work



## Improved Front End

- Increase support for various Python features
- Add more source languages
- Systematic and trustworthy axiomatization of string functions



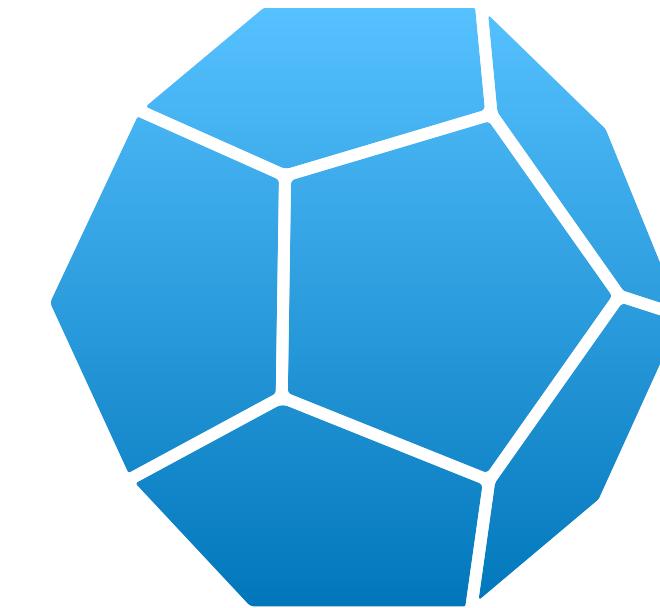
## Extended IR

- Add polymorphism and user-defined data types (waiting to be merged)



## Better Invariants

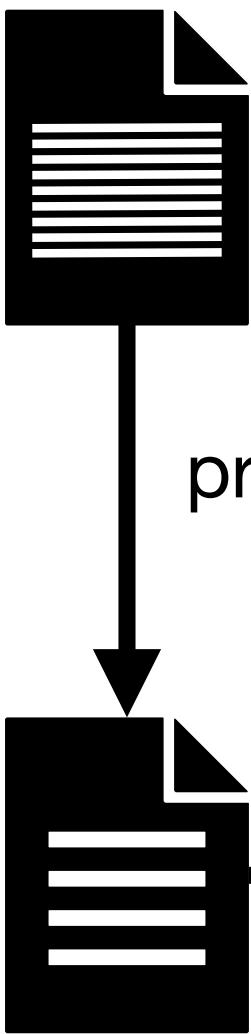
- Improve qualifier extraction heuristics
- Integrate external invariant generators
- Solve certain invariants using relational abstraction



## Stronger Domains

- Support more complex numerical domains (active work in progress)
- Support super-regular languages

## Full Source Code



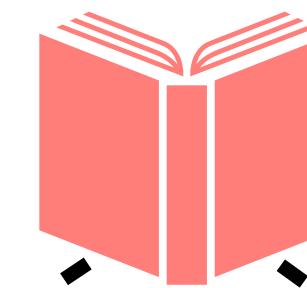
program slicing



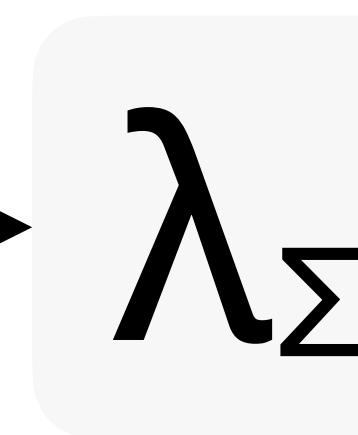
## Ad Hoc Parser Source Code

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

## String Function Specifications



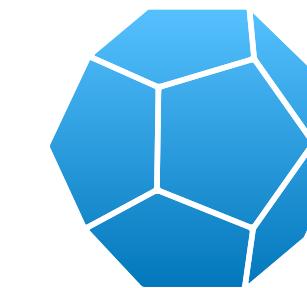
SSA/ANF transform



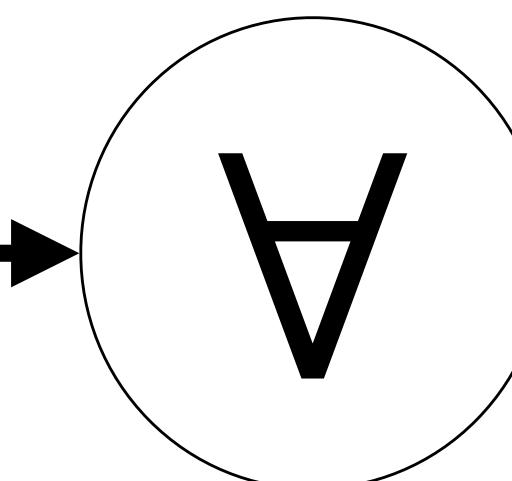
## Intermediate Representation

```
parser : {s:$ | ?} → 1
parser = λs:$.
  let x = charAt s 0 in
  let p1 = eqChar x 'a' in
  if p1 then
    let n = length s in
    let p2 = eq n 1 in
```

## Abstract Domains

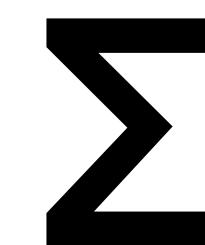


refinement inference



## String Constraints

grammar inference



## Grammar

$$(\forall v_1. v_1 = 0 \Rightarrow v_1 \geq 0 \wedge v_1 < |s|) \wedge$$
$$(\forall x. x = s[0] \Rightarrow$$
$$(\forall p_1. p_1 = \text{true} \Leftrightarrow x = 'a' \Rightarrow$$
$$(p_1 = \text{true} \Rightarrow$$
$$(\forall n. n \geq 0 \wedge n = |s| \Rightarrow n = 1)) \wedge$$
$$(n = \text{false} \Rightarrow$$
$$a + (\Sigma \setminus a)b\Sigma^*$$

[github.com/mcschroeder/panini](https://github.com/mcschroeder/panini)