

Semantic Prefix Oracles for LLM Decoding

Contracts and Differential Validation

Paul Kronlund-Drouault

ENS de Lyon · Unsuspicious Industries

LMPL '26



UNSUSPICIOUS
—
INDUSTRIES

Three levels

Syntax (grammar)

Is the token string well-formed?

Semantics (logic)

Does it obey some logic rules ?

Meaning (reference)

Is it useful ?

We can help them for the first two !

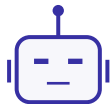
7

Whereof one cannot speak,
thereof one must be silent.

Ludwig Wittgenstein
Tractatus Logico-Philosophicus

What comes next?

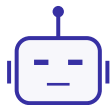
|



LLM

What comes next?

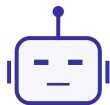
let



LLM

What comes next?

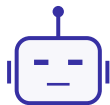
```
let y|
```



LLM

What comes next?

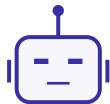
```
let y:|
```



LLM

What comes next?

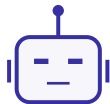
```
let y: Bool|
```



LLM

What comes next?

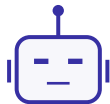
```
let y: Bool =|
```



LLM

What comes next?

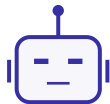
```
let y: Bool = true|
```



LLM

What comes next?

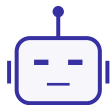
```
let y: Bool = true;
```



LLM

What comes next?

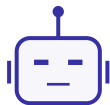
```
let y: Bool = true;  
let|
```



LLM

What comes next?

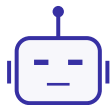
```
let y: Bool = true;  
let z|
```



LLM

What comes next?

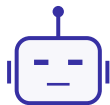
```
let y: Bool = true;  
let z:
```



LLM

What comes next?

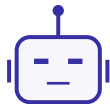
```
let y: Bool = true;  
let z: Int|
```



LLM

What comes next?

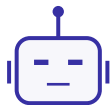
```
let y: Bool = true;  
let z: Int =
```



LLM

What comes next?

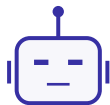
```
let y: Bool = true;  
let z: Int = 42|
```



LLM

What comes next?

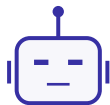
```
let y: Bool = true;  
let z: Int = 42;|
```



LLM

What comes next?

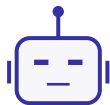
```
let y: Bool = true;  
let z: Int = 42;  
let
```



LLM

What comes next?

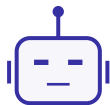
```
let y: Bool = true;  
let z: Int = 42;  
let x|
```



LLM

What comes next?

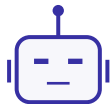
```
let y: Bool = true;  
let z: Int = 42;  
let x:
```



LLM

What comes next?

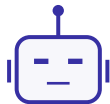
```
let y: Bool = true;  
let z: Int = 42;  
let x: Int
```



LLM

What comes next?

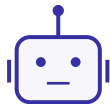
```
let y: Bool = true;  
let z: Int = 42;  
let x: Int =
```



LLM

What comes next?

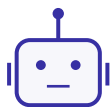
```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```



LLM

What comes next?

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```

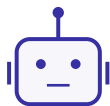


LLM

```
)  
true;  
m;  
y;  
z;  
67;
```

What comes next?

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```

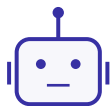


LLM

```
)  
true;  
m;  
y;  
z;  
▶ 67;
```

What comes next?

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```



LLM

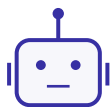


```
)  
true;  
m;  
y;  
z;  
67;
```



What comes next?

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```



LLM

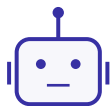


```
)  
true;  
m;  
y;  
z;  
67;
```



What comes next?

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```



LLM

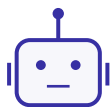


```
)  
true;  
m;  
y;  
z;  
67;
```



What comes next?

```
let y: Bool = true;
let z: Int = 42;
let x: Int =
```



LLM

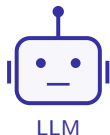
```

)
true;
m;
y;
z;
67;

```

What comes next?

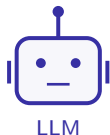
```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```



►)
true;
m;
y;
z;
67;

What comes next?

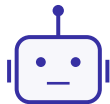
```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = ) | invalid
```



►)
true;
m;
y;
z;
67;

Add syntax.

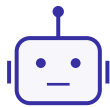
```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```



LLM

Add syntax.

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```

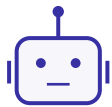


LLM

```
)  
true;  
m;  
y;  
z;  
67;
```

Add syntax.

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```



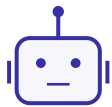
LLM

→ -

true;	
m;	
y;	
z;	
67;	

Add syntax.

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```



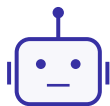
LLM

→ -

true;	
m;	
y;	
z;	
▶ 67;	

Add syntax.

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```



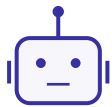
LLM

➤

→	-	
true;		
m;		
y;		
z;		
67;		

Add syntax.

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```



LLM



true;

m;

y;

z;

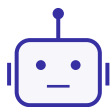
67;

-



Add syntax.

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```



LLM



true;

m;

y;

z;

67;

-

[purple bar]

[purple bar]

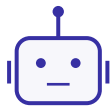
[purple bar]

[purple bar]

[purple bar]

Add syntax.

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```



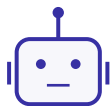
LLM

→ -

▶ true;	
m;	
y;	
z;	
67;	

Add syntax.

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = true;
```



LLM

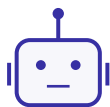
→ -

▶ true;	
m;	
y;	
z;	
67;	

Add syntax.

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = true;
```

Grammatical. But true has type Bool, not Int.



LLM

→ -

▶ true;	
m;	
y;	
z;	
67;	

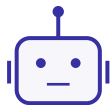
Use the context.

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```

Γ

y : Bool

z : Int

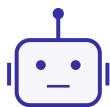


LLM

Use the context.

```
let y: Bool = true;
let z: Int = 42;
let x: Int =
```

Γ
 $y : \text{Bool}$
 $z : \text{Int}$



LLM

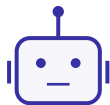
Value	Frequency
true;	10
m;	7
y;	5
z;	4
67;	3

Use the context.

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```

Γ

y : Bool
z : Int



LLM

```
)  
true;  
m;  
y;  
z;  
67;
```

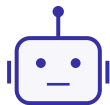


Use the context.

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```

 Γ

y : Bool
z : Int



LLM

~~)~~ -

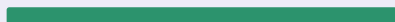
~~true;~~ -

~~m;~~ -

~~y;~~ -

z;

67;

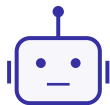


Use the context.

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```

Γ

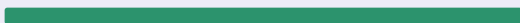
y : Bool
z : Int



LLM



~~)~~ -
~~true;~~ -
~~m;~~ -
~~y;~~ -
z;
67;

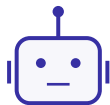


Use the context.

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = |
```

Γ

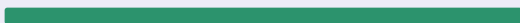
y : Bool
z : Int



LLM



~~)~~ -
~~true;~~ -
~~m;~~ -
~~y;~~ -
z;
67;

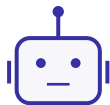


Use the context.

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = z;
```

 Γ

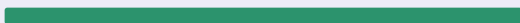
y : Bool
z : Int



LLM



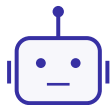
~~)~~ -
~~true;~~ -
~~m;~~ -
~~y;~~ -
z;
67;



Use the context.

```
let y: Bool = true;  
let z: Int = 42;  
let x: Int = z;
```

A valid complete program.



LLM



```
)  
true;  
m;  
y;  
z;  
67;
```

Γ

y : Bool
z : Int

Semantic prefix grammars

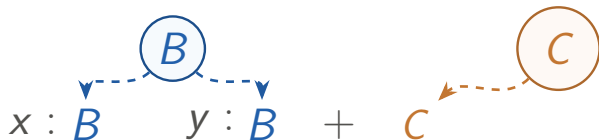
Attributed context-free grammars with first-order unification.

Application(app) ::= Expression[l] Expression[r]
Variable(var) ::= Identifier[x]
Type* ::= Identifier | Type $\rightarrow$ Type | (Type)

$$\frac{\Gamma \vdash l : \mathfrak{A} \rightarrow \mathfrak{B} \quad \Gamma \vdash r : \mathfrak{A}}{\mathfrak{B}} \quad (\text{APP})$$

User-defined semantic constraints over context-free grammars.

User-defined type algebra

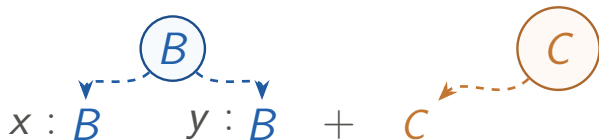
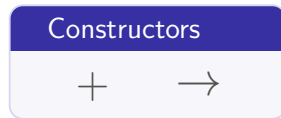


Constructors

+

$\rightarrow$

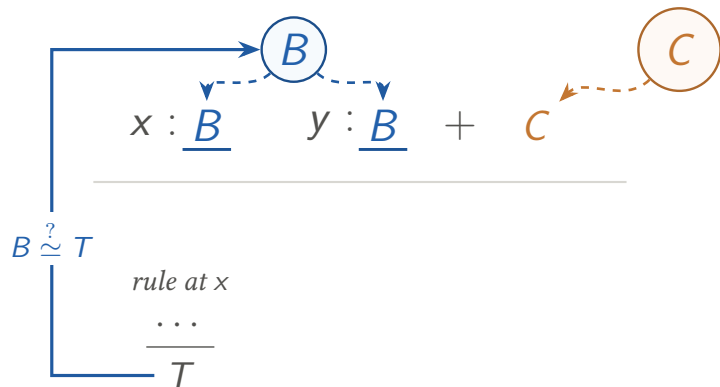
User-defined type algebra



rule at x

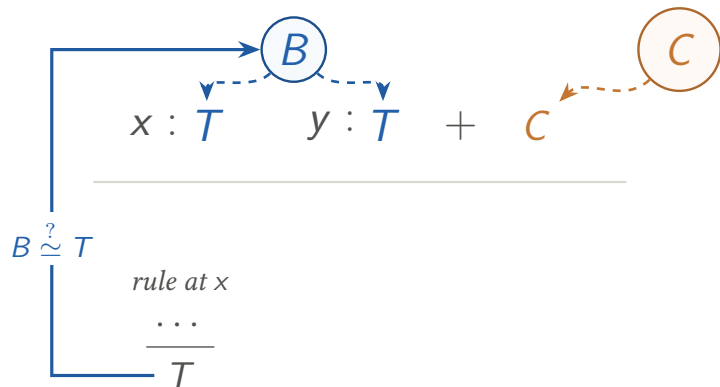
$$\frac{\dots}{T}$$

User-defined type algebra



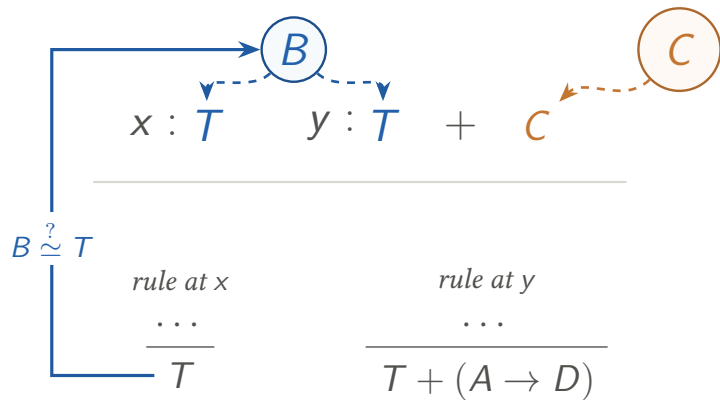
Constructors	
+	→

User-defined type algebra



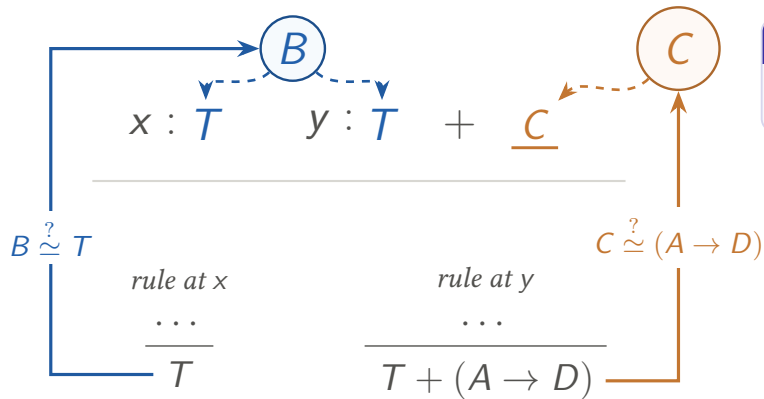
Constructors	
+	→

User-defined type algebra



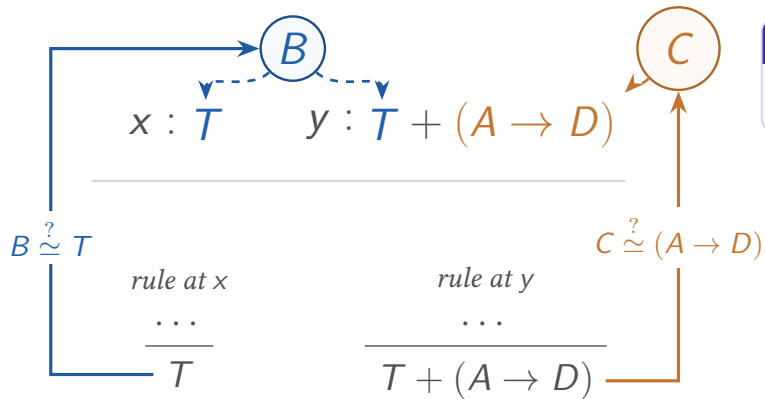
Constructors	
$+$	$\rightarrow$

User-defined type algebra



Constructors	
+	→

User-defined type algebra



Constructors	
$+$	$\rightarrow$

Three verdicts

$\lambda x:\text{int}. x$ **Satisfied**
LAM (lam) : $\text{int} \rightarrow \text{int}$
 body VAR (var) : int
 $x \mapsto \text{int}$
in $\Gamma = [x:\text{int}]$

(a)

$\lambda x:\text{int}. \text{Live}$
LAM (lam) : $\text{int} \rightarrow \mathfrak{B}$
 body body e
Pending

(b)

$\lambda x:\text{int}. y$ **Lost**
LAM (lam) : int
 body VAR (var)
 $y \notin \Gamma$
 $\Gamma = [x:\text{int}]$

(c)

y fails lookup in $\Gamma = [x:\text{int}]$: contradiction

Correctness contracts

Safe pruning Never reject a prefix with a valid completion.

Dead-end freedom Every retained prefix has a valid completion.

Together, these contracts make decoding complete and sound.

Safe pruning

**Rigid constructor
clash** { $\rightarrow$ `let x: Int = true;`

Empty
regular leaf

Occurs check
failure

Failed
lookup

Safe pruning

Rigid constructor
clash

{ $\rightarrow$ let x: Int = true;

Empty
regular leaf

expected

Int

?
~
X

produced

Bool

Occurs check
failure

Failed
lookup

Safe pruning

Rigid constructor
clash



```
let x: Int = true;
```

Empty
regular leaf

expected

Int

produced



Bool

Occurs check
failure

These constructors cannot unify.

Failed
lookup

More input cannot change either one.

Safe pruning

Rigid constructor
clash

Empty
regular leaf

Occurs check
failure

Failed
lookup


$$\begin{array}{lll} R = [a-z]^+ & \text{grammar} & \\ S = a(b|c) & \text{leaf constraint} & a| \\ D_a(R) & = & [a-z]^* \\ D_a(S) & = & b|c \\ D_a(R) \cap D_a(S) & = & b|c \end{array}$$

Safe pruning

Rigid constructor
clash

Empty
regular leaf

Occurs check
failure

Failed
lookup



$R = [a-z]^+$

$S = a(b|c)$

grammar

leaf constraint

ad|

$D_{ad}(R) = [a-z]^*$

$D_{ad}(S) = \emptyset$

$D_{ad}(R) \cap D_{ad}(S) = \emptyset$

Safe pruning

Rigid constructor
clash

Empty
regular leaf

Occurs check
failure

Failed
lookup



$R = [a-z]^+$

$S = a(b|c)$

grammar

leaf constraint

ad|

$$D_{ad}(R) = [a-z]^*$$

$$D_{ad}(S) = \emptyset$$

$$D_{ad}(R) \cap D_{ad}(S) = \emptyset$$

$$D_w(\emptyset) = \emptyset \quad \text{for any suffix } w$$

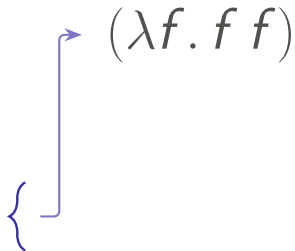
Safe pruning

Rigid constructor
clash

Empty
regular leaf

**Occurs check
failure**

Failed
lookup



$(\lambda f. f f)$

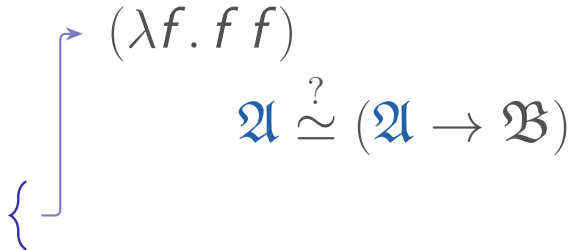
Safe pruning

Rigid constructor
clash

Empty
regular leaf

**Occurs check
failure**

Failed
lookup


$$(\lambda f. f f)$$
$$\mathfrak{A} \stackrel{?}{\simeq} (\mathfrak{A} \rightarrow \mathfrak{B})$$

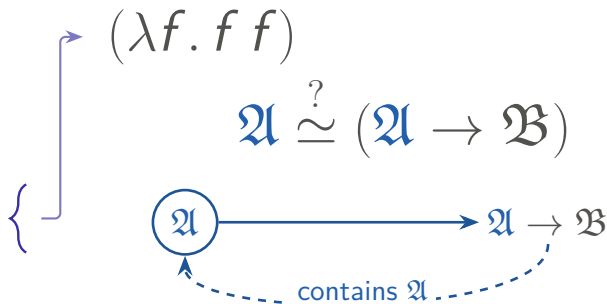
Safe pruning

Rigid constructor
clash

Empty
regular leaf

**Occurs check
failure**

Failed
lookup



This would require an infinite type.

Safe pruning

Rigid constructor
clash

Empty

regular leaf

Occurs check
failure

**Failed
lookup**



```
let x: Int = m;
```

fixed left context

Γ
y : Bool
z : Int

Safe pruning

Rigid constructor
clash

Empty

regular leaf

Occurs check
failure

**Failed
lookup**



```
let x: Int = m;
```

fixed left context

Γ
y : Bool
z : Int

$m \notin \Gamma$

Safe pruning

Rigid constructor
clash

Empty

regular leaf

Occurs check
failure

**Failed
lookup**

let x: Int = m;

fixed left context

Γ
y : Bool
z : Int

$m \notin \Gamma$

let m: Int = 67;

Too late for that earlier lookup.

Safe pruning

Rigid constructor

clash

Empty

regular leaf

Occurs check

failure

Failed

lookup

A prefix is Lost only when **every** compatible branch is Lost.

Never discard a prefix with a valid completion.

Dead-end freedom

We found that it holds under the following conditions on the grammar:

- (P) Surface productivity Every reachable nonterminal can terminate.
- (T) Type coverage Every demanded type has a closed realizer.
- (F) Left-to-right flow Child obligations depend on the parent, context, and earlier siblings.

Some evaluated grammars violate these conditions.

Type coverage (T)

(T) fails

$\lambda f:A \rightarrow B. f($

A has no closed term

Live, no completion

(T) holds

$(f: \text{Int} \Rightarrow \text{Bool}) \Rightarrow f(0)$

literals cover every base type

Int, Bool, Float

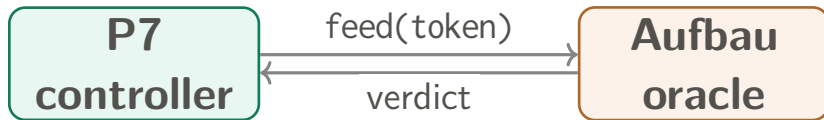
Base-type literals establish (T).

Compiler cross-check

- Compilers (ocamlc, cc) grade whole programs.
- The oracle grades prefixes.
- The harness detects valid programs rejected by the oracle.

Compilers share no code with the oracle.

Python interface



Controller verdicts

Lost

Reject. Resample.

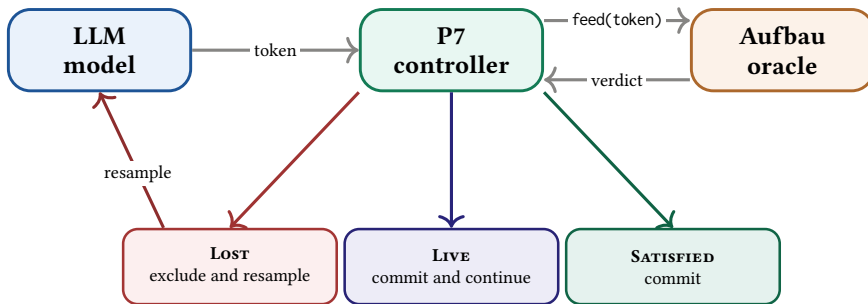
Live

Accept. Continue.

Satisfied

Accept.

Full architecture



Aligning to tokens

TOKEN SPELLING

GENERATED TEXT

"le"

le

Aligning to tokens

TOKEN SPELLING

GENERATED TEXT

"le"

le

"t x: I"

let x: I

Aligning to tokens

TOKEN SPELLING

GENERATED TEXT

"le"

le

"t x: I"

let x: I

"nt = 4"

let x: Int = 4

Aligning to tokens

TOKEN SPELLING

GENERATED TEXT

"le"

le

"t x: I"

let x: I

"nt = 4"

let x: Int = 4

"2;"

let x: Int = 42;

Token coverage

V : ordinary text tokens

Σ_G : grammar characters

Exact spelling: $\ell : V \longrightarrow \Sigma_G^*$

$$L_V = \{\ell(v) \mid v \in V\}$$

Token coverage

V : ordinary text tokens

Exact spelling: $\ell : V \longrightarrow \Sigma_G^*$

Σ_G : grammar characters

$$L_V = \{\ell(v) \mid v \in V\}$$

Coverage: $\Sigma_G \subseteq L_V^*$

$$\Sigma_G^* \subseteq L_V^*$$

Every grammar character is spellable.

So is every character-level completion.

Token coverage

V : ordinary text tokens

Σ_G : grammar characters

Exact spelling: $\ell : V \longrightarrow \Sigma_G^*$

$$L_V = \{\ell(v) \mid v \in V\}$$

Coverage: $\Sigma_G \subseteq L_V^*$

Every grammar character is spellable.

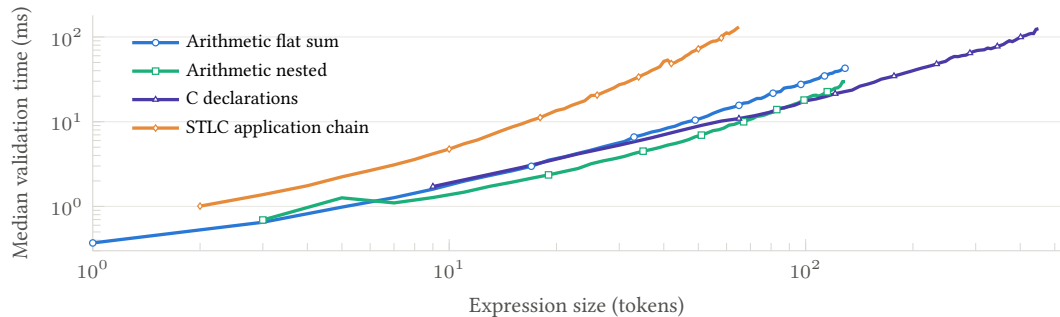
$$\Sigma_G^* \subseteq L_V^*$$

So is every character-level completion.

Example: $\Sigma_G = \{1, e, t\}$, $L_V = \{1, e, t, le\}$.

$$\text{let} = 1 \cdot e \cdot t = le \cdot t$$

Oracle cost



Log-log slopes 1.1–1.4. Largest inputs: about 130 ms.

Models and modes

12 models



Qwen3.5

Instruct: 0.8B, 2B, 4B, 9B

Base: 2B, 4B, 9B



Qwen3.6-27B



Phi-4-mini



gpt-oss-20b



gemma-4-26b



Pythia-1.4B

no-instruction floor

5 decoding modes

52 tasks · 3,693 graded attempts · greedy

Models and modes

12 models



Qwen3.5

Instruct: 0.8B, 2B, 4B, 9B

Base: 2B, 4B, 9B



Qwen3.6-27B



Phi-4-mini



gpt-oss-20b



gemma-4-26b



Pythia-1.4B

no-instruction floor

5 decoding modes

Unconstrained

Free emission

52 tasks · 3,693 graded attempts · greedy

Models and modes

12 models



Qwen3.5

Instruct: 0.8B, 2B, 4B, 9B

Base: 2B, 4B, 9B



Qwen3.6-27B



Phi-4-mini



gpt-oss-20b



gemma-4-26b



Pythia-1.4B

no-instruction floor

5 decoding modes

Unconstrained

Free emission

Syntactic only

Grammar mask

52 tasks · 3,693 graded attempts · greedy

Models and modes

12 models



Qwen3.5

Instruct: 0.8B, 2B, 4B, 9B

Base: 2B, 4B, 9B



Qwen3.6-27B



Phi-4-mini



gpt-oss-20b



gemma-4-26b



Pythia-1.4B

no-instruction floor

5 decoding modes

Unconstrained

Free emission

Syntactic only

Grammar mask

Constrained direct

Grammar + typing

52 tasks · 3,693 graded attempts · greedy

Models and modes

12 models



Qwen3.5

Instruct: 0.8B, 2B, 4B, 9B

Base: 2B, 4B, 9B



Qwen3.6-27B



Phi-4-mini



gpt-oss-20b



gemma-4-26b



Pythia-1.4B

no-instruction floor

5 decoding modes

Unconstrained

Free emission

Syntactic only

Grammar mask

Constrained direct

Grammar + typing

Unconstrained thinking

Free reasoning → free emission

52 tasks · 3,693 graded attempts · greedy

Models and modes

12 models



Qwen3.5

Instruct: 0.8B, 2B, 4B, 9B

Base: 2B, 4B, 9B



Qwen3.6-27B



Phi-4-mini



gpt-oss-20b



gemma-4-26b



Pythia-1.4B

no-instruction floor

5 decoding modes

Unconstrained

Free emission

Syntactic only

Grammar mask

Constrained direct

Grammar + typing

Unconstrained thinking

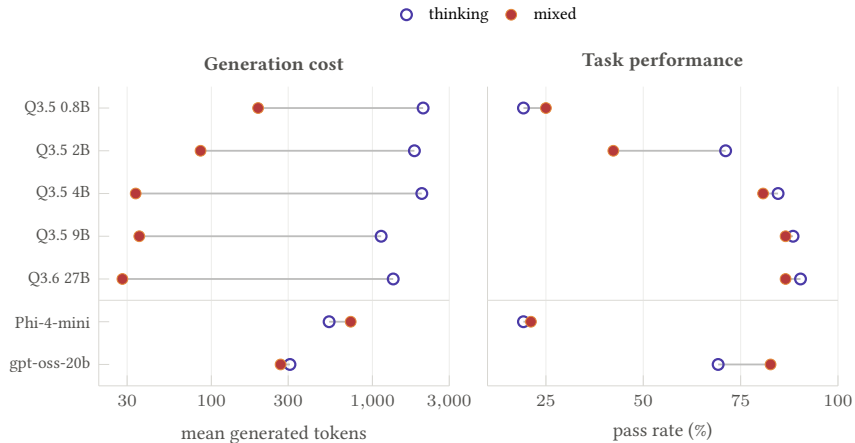
Free reasoning → free emission

Constrained mixed

Free reasoning → typed emission

52 tasks · 3,693 graded attempts · greedy

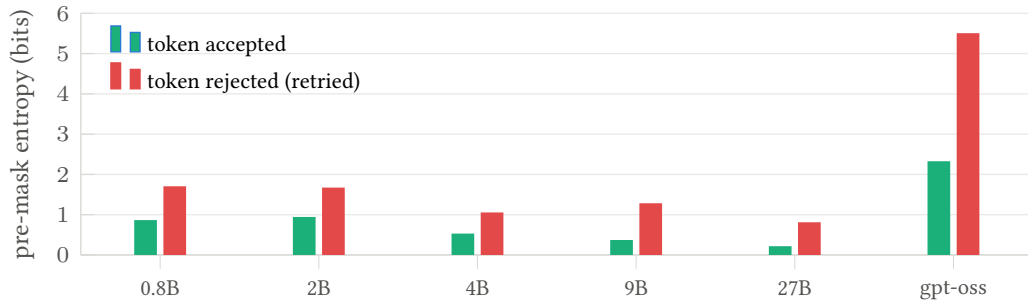
Generation cost and task performance



Thinking: free emission. Mixed: constrained emission.

Rejection and entropy

(a) model uncertainty: entropy where the mask fires



Future work

1. More types (Sum, Refinement, maybe proof types even)
2. Mechanizing the contract proofs
3. Constrained chain-of-thought ?

Thank you

Paul Kronlund-Drouault · ENS de Lyon · Unsuspicious Industries

*Semantic Prefix Oracles for LLM Decoding:
Contracts and Differential Validation*

ACM DL · 10.1145/3843750.3843841

P7 · github.com/Unsuspicious-Industries/p7



UNSUSPICIOUS
—
INDUSTRIES