Synthesizing Program Input Grammars

Osbert Bastani (Stanford)
Rahul Sharma (Microsoft Research)
Alex Aiken (Stanford)
Percy Liang (Stanford)

Program Input Grammars

```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;
  size_t cc;
  int opt, prepended;
  int prev_optind, last_recursive;
  int fread_errno;
  intmax_t default_context;
  FILE *fp;
  exit_failure = EXIT_TROUBLE;
  initialize_main (&argc, &argv);
  set_program_name (argv[0]);
  program_name = argv[0];
  // ...
}
```

Program Input Grammars

```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;
```

$$A_{\mathrm{XML}} \rightarrow (\mathtt{a} + \cdots + \mathtt{z})$$
$$A_{\mathrm{XML}} \rightarrow \mathtt{<a>} A_{\mathrm{XML}} \mathtt{}$$
$$A_{\mathrm{XML}} \rightarrow A^*_{\mathrm{XML}}$$

```
  set_program_name (argv[0]);
  program_name = argv[0];
  // ...
}
```

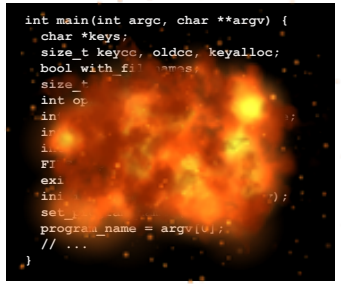
Applications

Applications

Fuzz testing

- Security vulnerabilities
- Differential testing

```
<a>hihi</a>
<a><a></a></a>
<a></a>
...
```

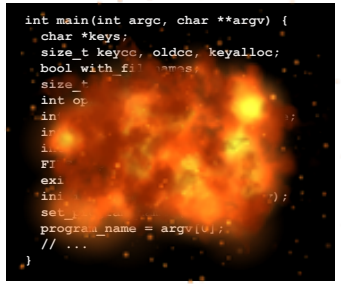


Applications

Fuzz testing

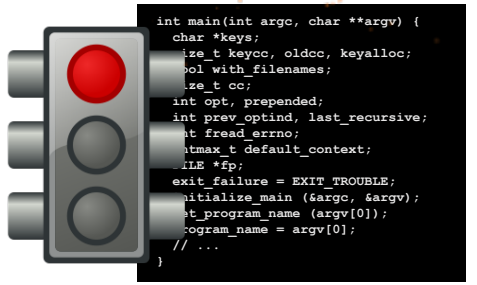
- Security vulnerabilities
- Differential testing

<a>hihi
<a><a>
<a>
...



Whitelisting inputs

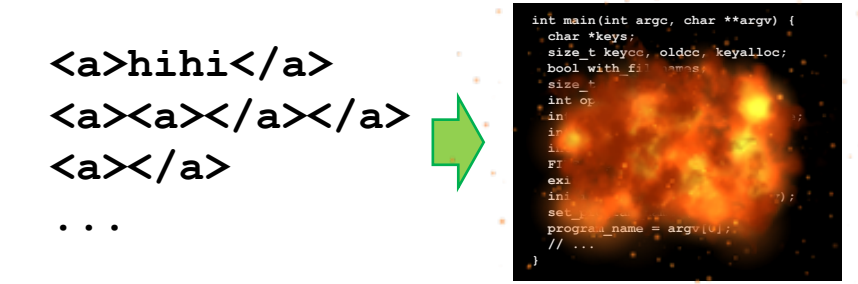
aa>hi



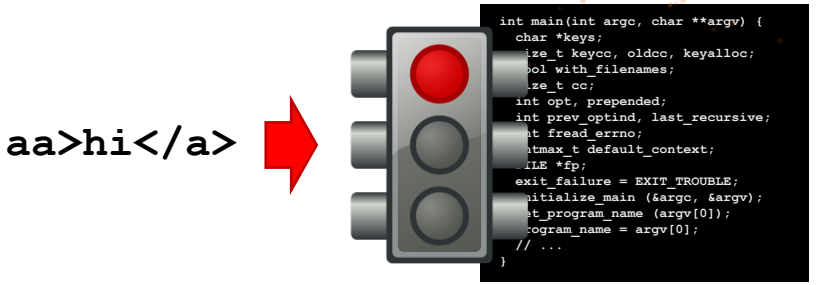
Applications

Fuzz testing

- Security vulnerabilities
- Differential testing



Whitelisting inputs



Reverse engineering

Program Input Grammars

```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;
```

$$A_{\mathrm{XML}} \rightarrow (\mathtt{a} + \cdots + \mathtt{z})$$
$$A_{\mathrm{XML}} \rightarrow \mathtt{<a>} A_{\mathrm{XML}} \mathtt{}$$
$$A_{\mathrm{XML}} \rightarrow A^*_{\mathrm{XML}}$$

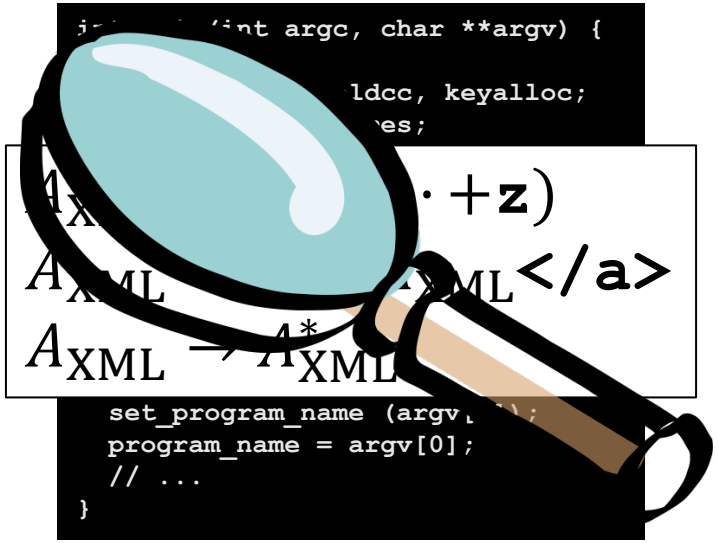
```
  set_program_name (argv[0]);
  program_name = argv[0];
  // ...
}
```

Program Input Grammars

```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;
```

```
  set_program_name (argv[0]);
  program_name = argv[0];
  // ...
}
```

Program Input Grammars



Grammar Synthesis Algorithm

Grammar Synthesis Algorithm

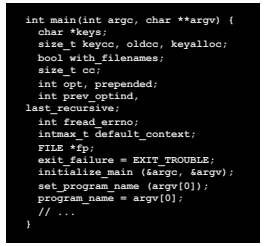
Grammar Synthesis Algorithm

$\alpha_{\text{in}} =$ `<a>hi</a>`

input example

Grammar Synthesis Algorithm

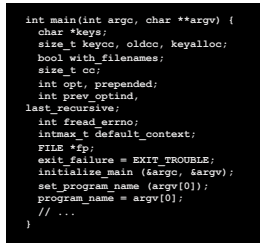
$\alpha_{\text{in}} =$ `<a>hi</a>`



input example &
blackbox access to program

Grammar Synthesis Algorithm

$\alpha_{\text{in}} =$ `<a>hi</a>`

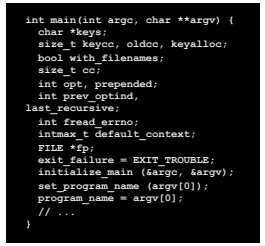
α 

```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;
  size_t cc;
  int opt, prepended;
  int prev_optind,
last_recursive;
  int fread_errno;
  intmax_t default_context;
  FILE *fp;
  exit_failure = EXIT_TROUBLE;
  initialize_main (&argc, &argv);
  set_program_name (argv[0]);
  program_name = argv[0];
  // ...
}
```

input example &
blackbox access to program

Grammar Synthesis Algorithm

$\alpha_{\text{in}} =$ `<a>hi</a>`

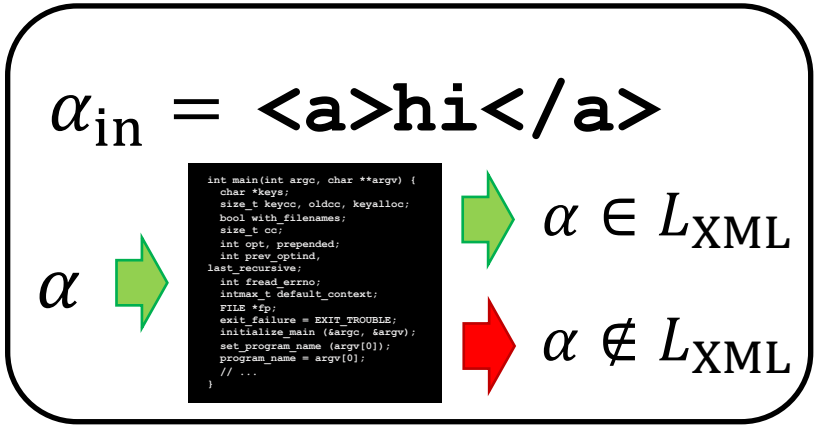
α 

```
int main(int argc, char **argv) {
  char *keys;
  size_t keycc, oldcc, keyalloc;
  bool with_filenames;
  size_t cc;
  int opt, prepended;
  int prev_optind,
last_recursive;
  int fread_errno;
  intmax_t default_context;
  FILE *fp;
  exit_failure = EXIT_TROUBLE;
  initialize_main (&argc, &argv);
  set_program_name (argv[0]);
  program_name = argv[0];
  // ...
}
```

$\alpha \notin L_{\text{XML}}$

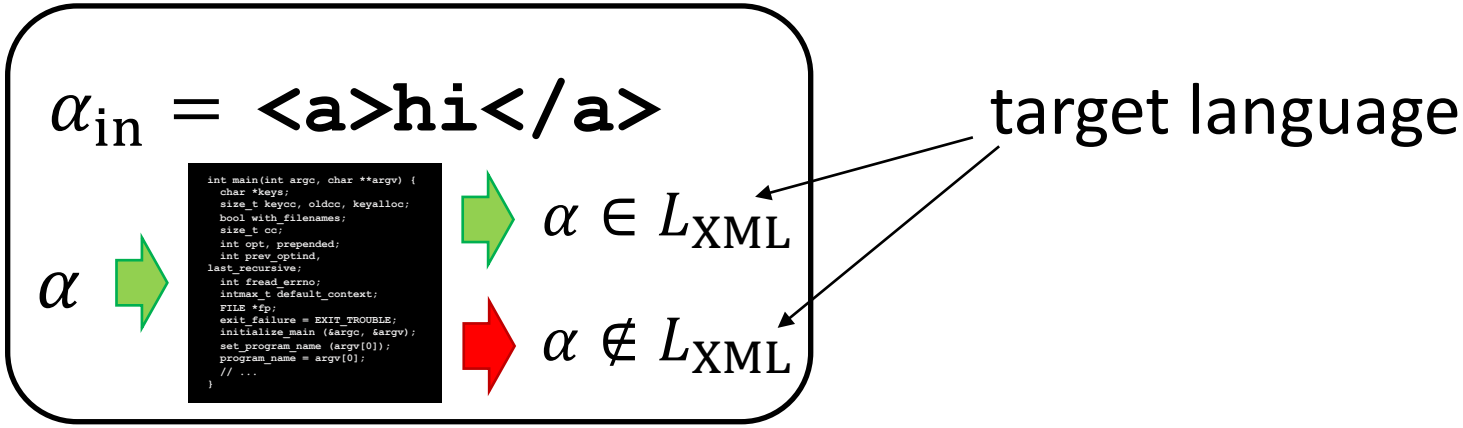
input example &
blackbox access to program

Grammar Synthesis Algorithm



input example &
blackbox access to program

Grammar Synthesis Algorithm



input example &
blackbox access to program

Grammar Synthesis Algorithm

$\alpha_{\text{in}} =$ `<a>hi</a>`

$$\mathcal{O}_{\text{XML}}(\alpha) = \begin{cases} 1 & \text{if } \alpha \in L_{\text{XML}} \\ 0 & \text{otherwise} \end{cases}$$

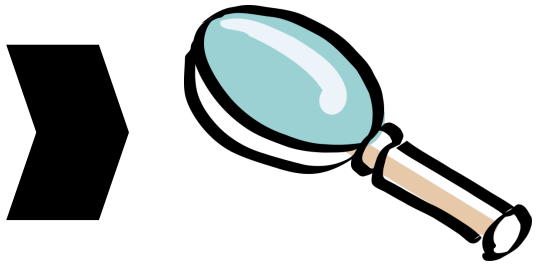
input example &
membership oracle

Grammar Synthesis Algorithm

$\alpha_{\text{in}} =$ `<a>hi</a>`

$$\mathcal{O}_{\text{XML}}(\alpha) = \begin{cases} 1 & \text{if } \alpha \in L_{\text{XML}} \\ 0 & \text{otherwise} \end{cases}$$

input example &
membership oracle



grammar synthesis
algorithm

Grammar Synthesis Algorithm

$\alpha_{\text{in}} = \texttt{<a>hi}$

$$\mathcal{O}_{\text{XML}}(\alpha) = \begin{cases} 1 & \text{if } \alpha \in L_{\text{XML}} \\ 0 & \text{otherwise} \end{cases}$$

input example &
membership oracle

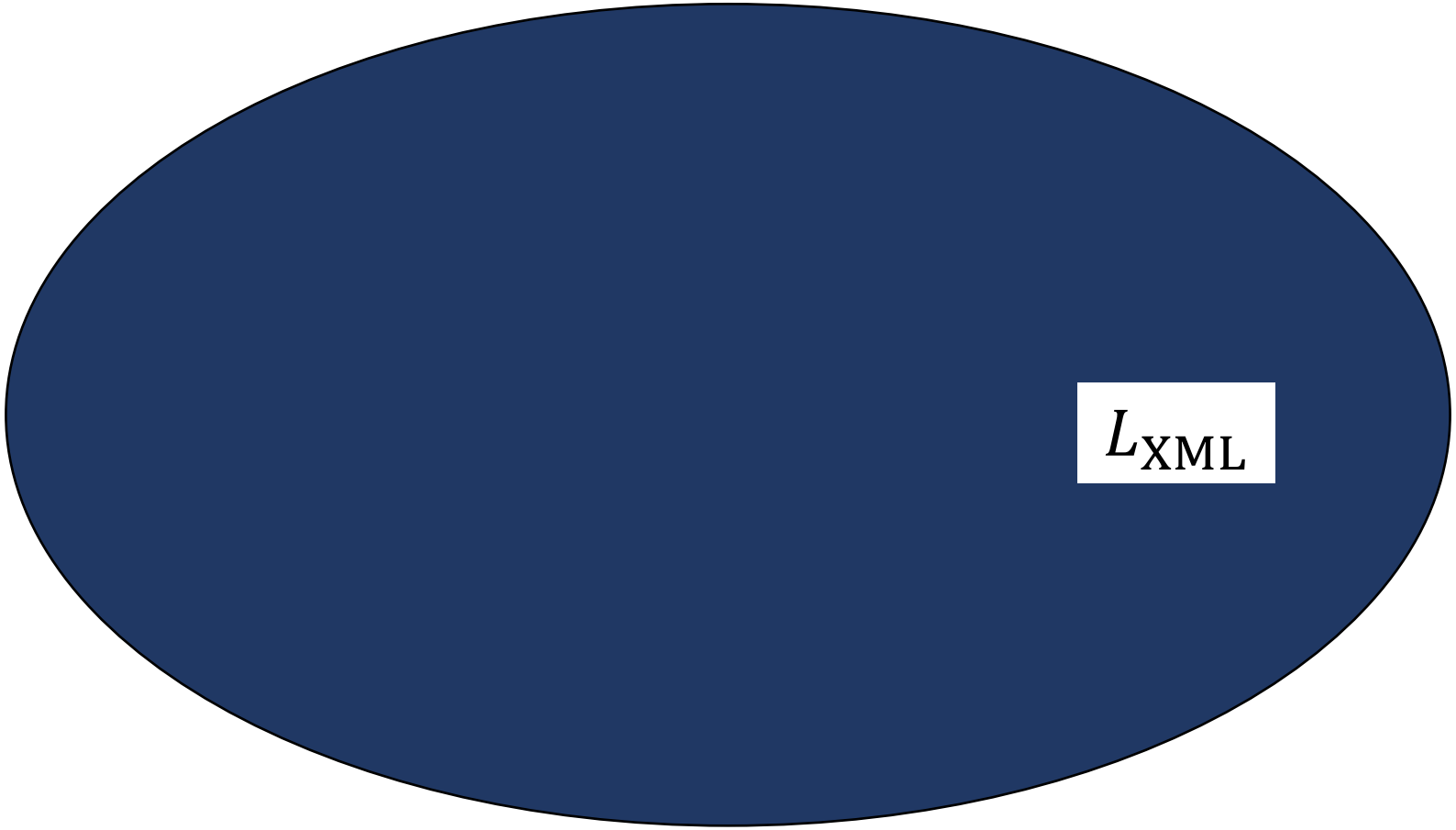
grammar synthesis
algorithm

$$A_{\text{XML}} \rightarrow (\texttt{a} + \cdots + \texttt{z})$$
$$A_{\text{XML}} \rightarrow \texttt{<a>} A_{\text{XML}} \texttt{}$$
$$A_{\text{XML}} \rightarrow A^*_{\text{XML}}$$

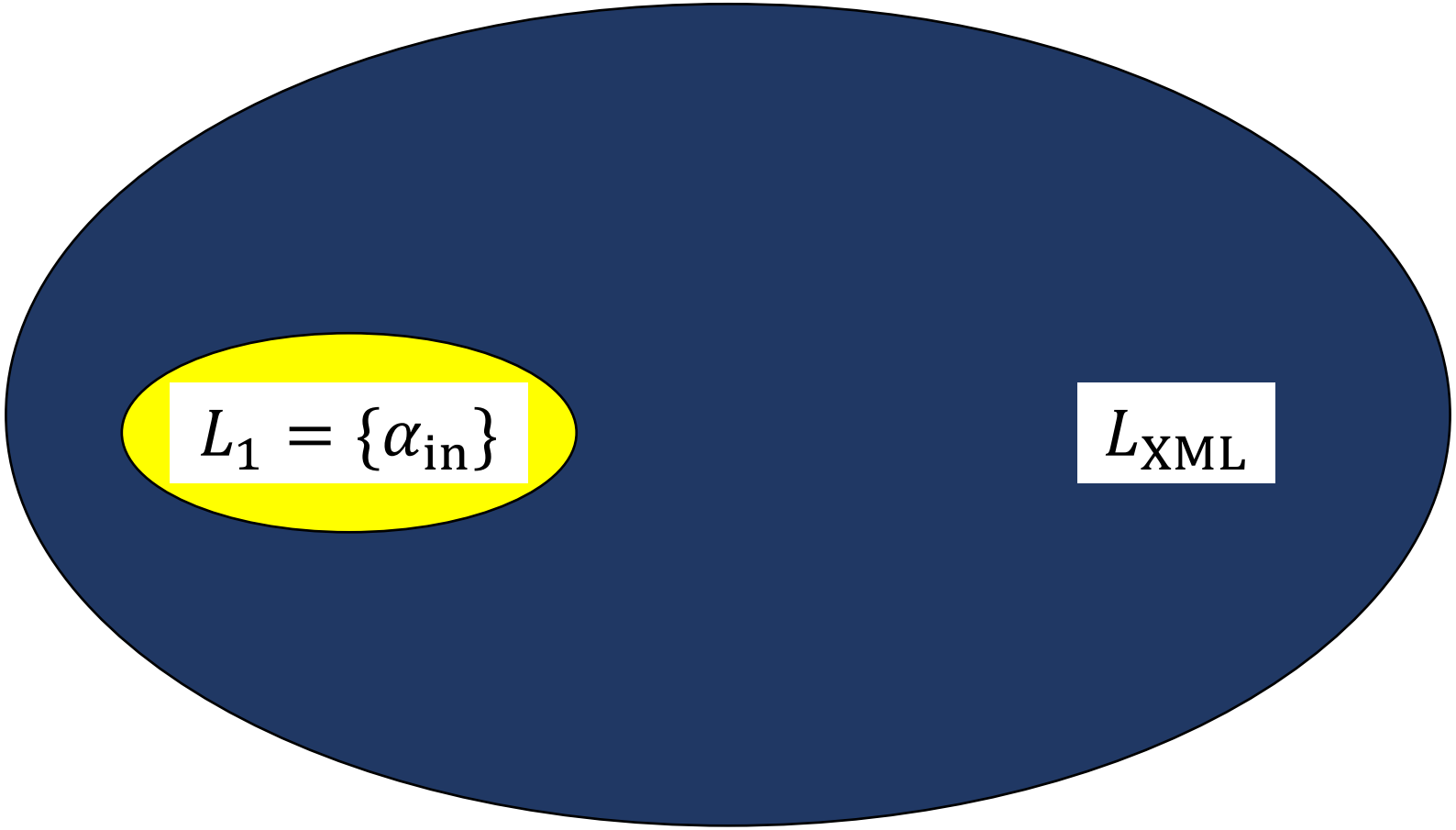
grammar approximating
the target language

Idea: Construct a series of increasingly general languages

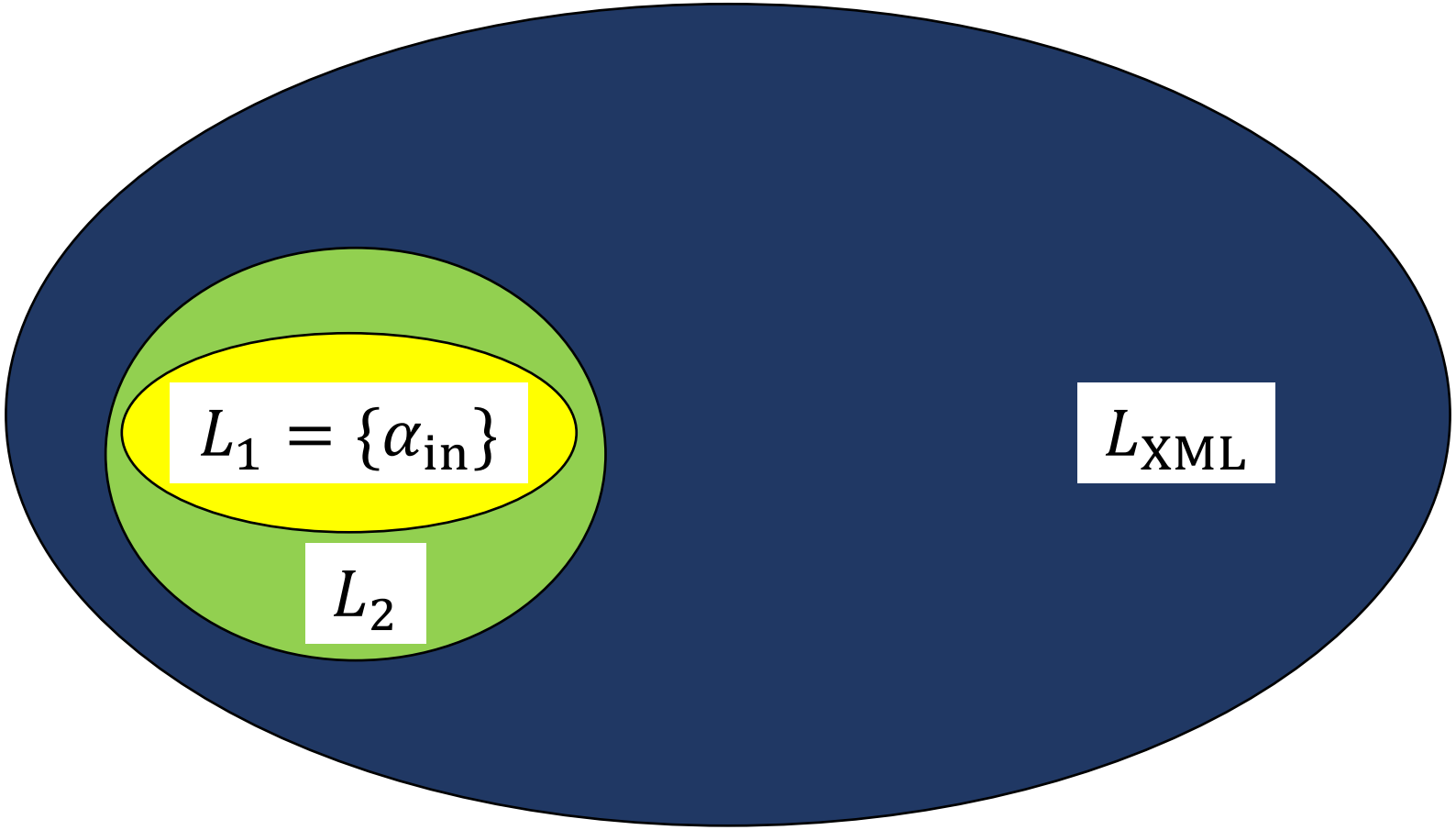
Idea: Construct a series of increasingly general languages



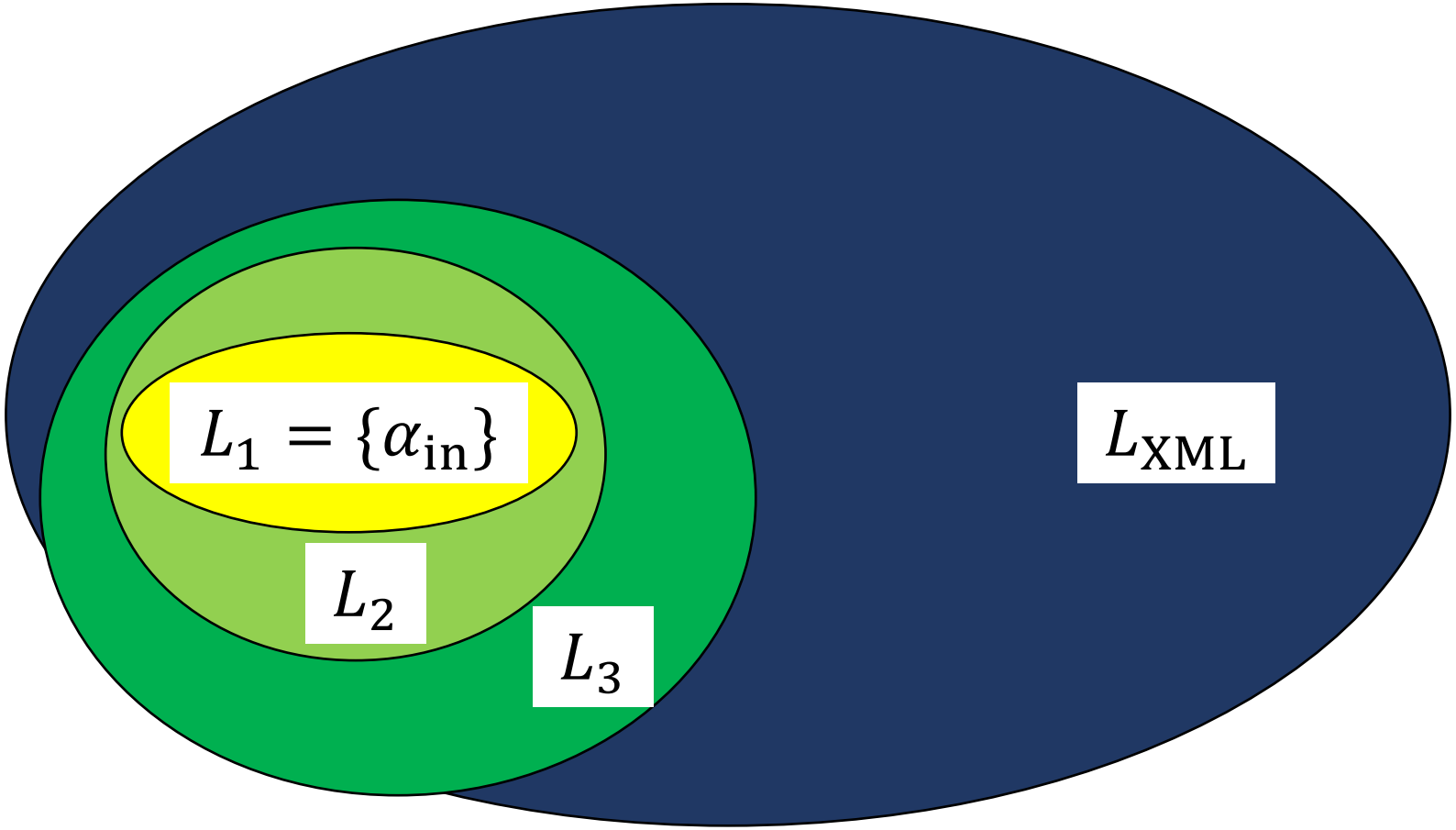
Idea: Construct a series of increasingly general languages



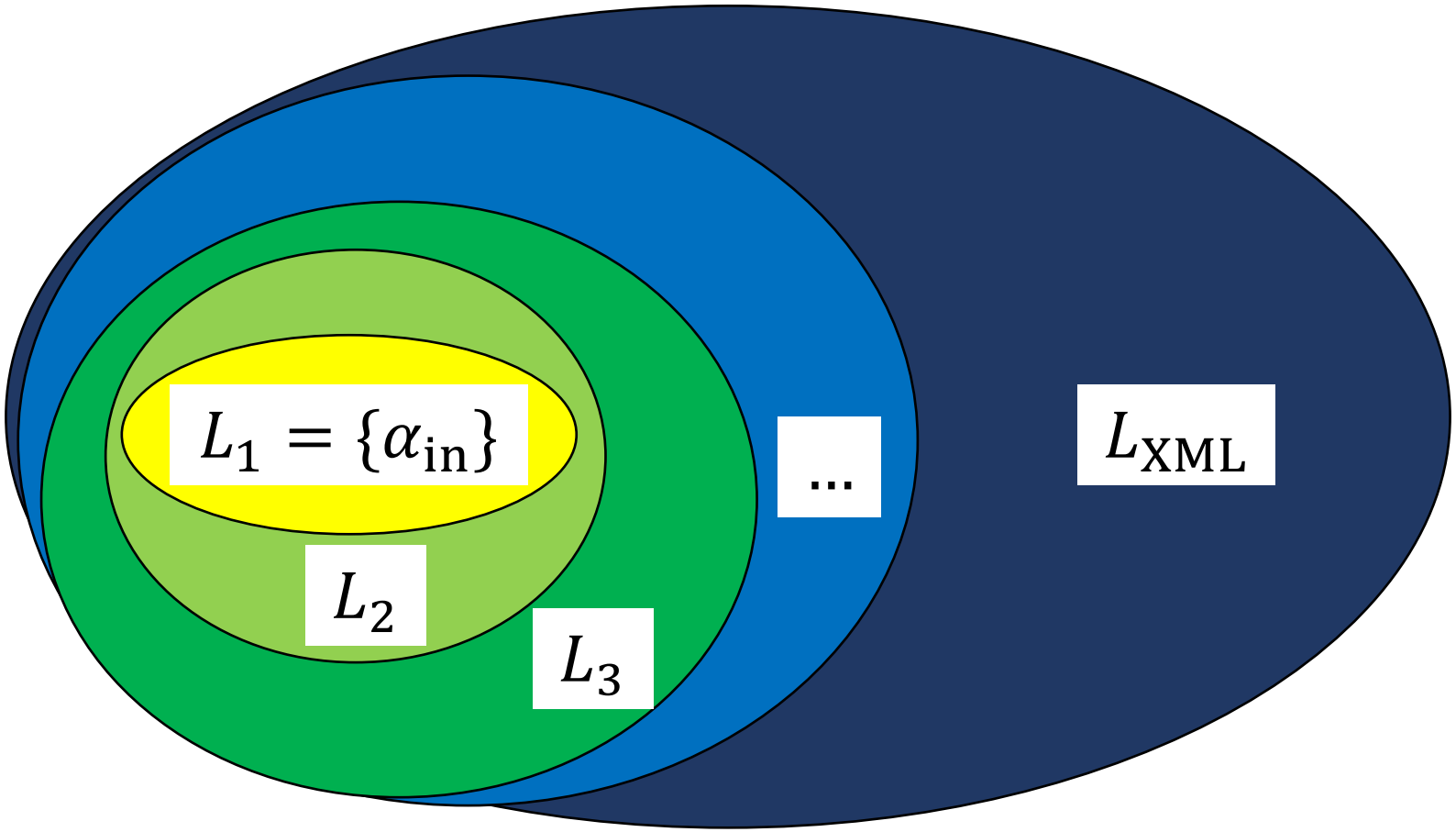
Idea: Construct a series of increasingly general languages



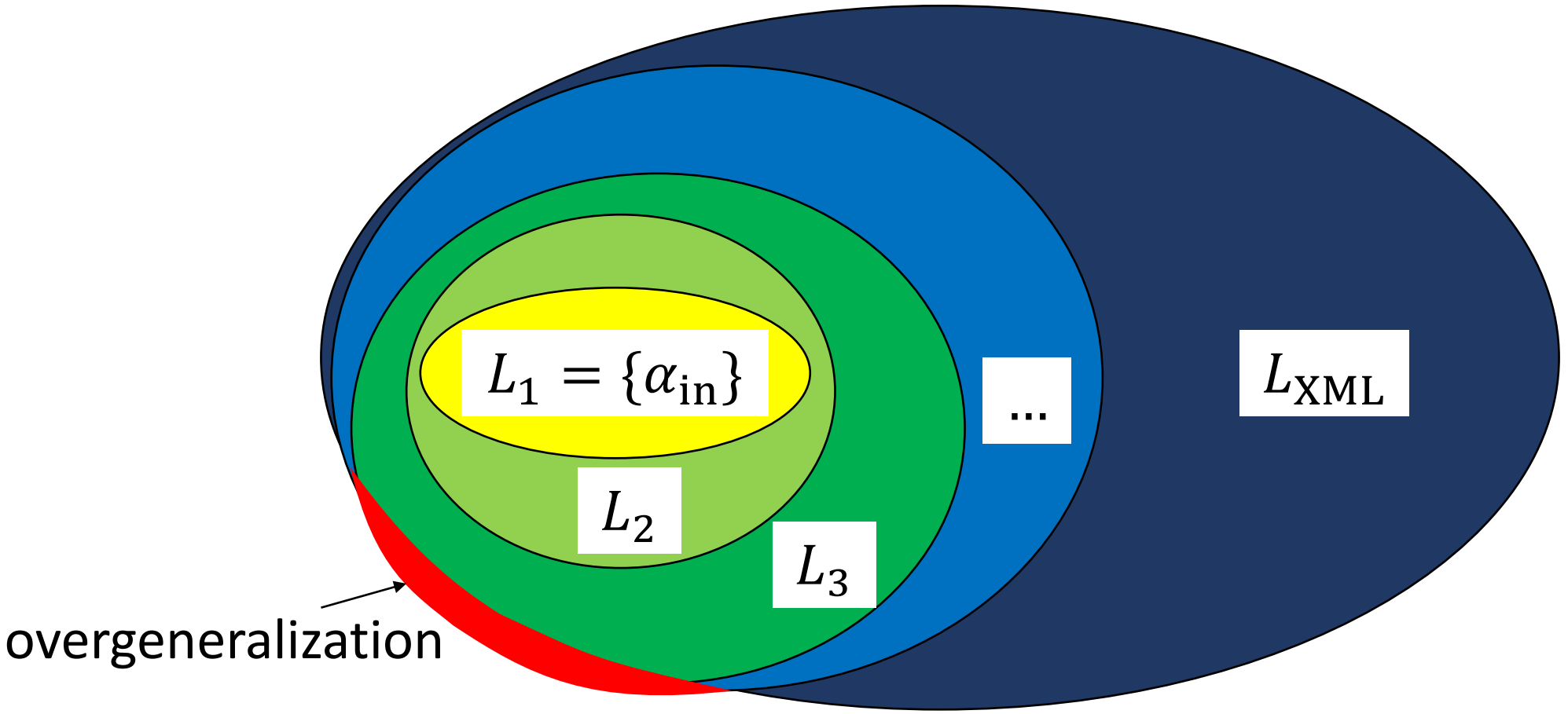
Idea: Construct a series of increasingly general languages



Idea: Construct a series of increasingly general languages



Idea: Construct a series of increasingly general languages



Idea: Construct a series of increasingly general languages

Idea: Construct a series of increasingly general languages

$\alpha_{\text{in}} =$ `<a>hi</a>`

Idea: Construct a series of increasingly general languages

$\alpha_{\text{in}} = \texttt{<a>hi} \subseteq (\texttt{<a>hi})^*$

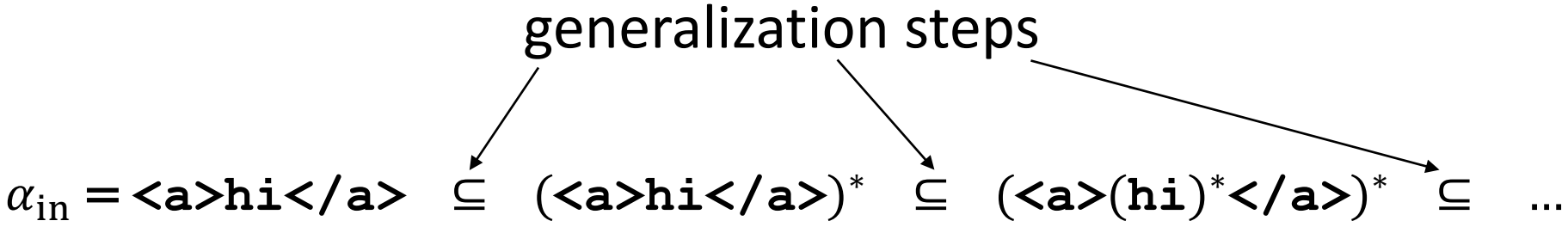
Idea: Construct a series of increasingly general languages

$$\alpha_{\text{in}} = \texttt{<a>hi} \subseteq (\texttt{<a>hi})^* \subseteq (\texttt{<a>}(\texttt{hi})^*\texttt{})^*$$

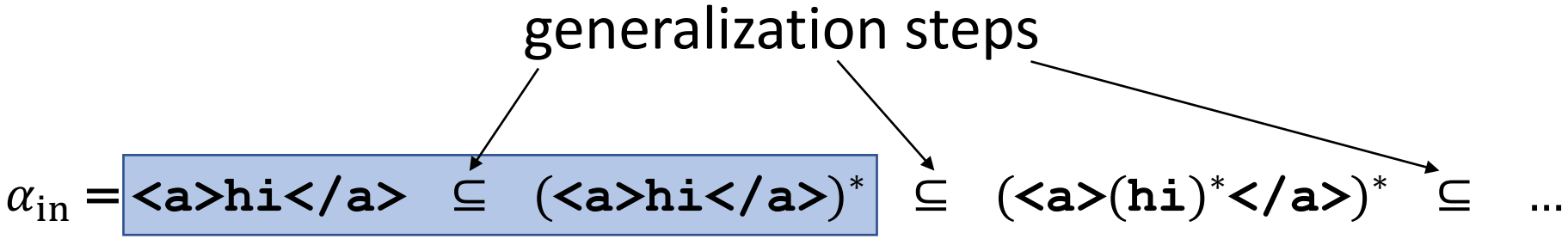
Idea: Construct a series of increasingly general languages

$$\alpha_{\text{in}} = \texttt{<a>hi} \subseteq (\texttt{<a>hi})^* \subseteq (\texttt{<a>}(\texttt{hi})^*\texttt{})^* \subseteq \ldots$$

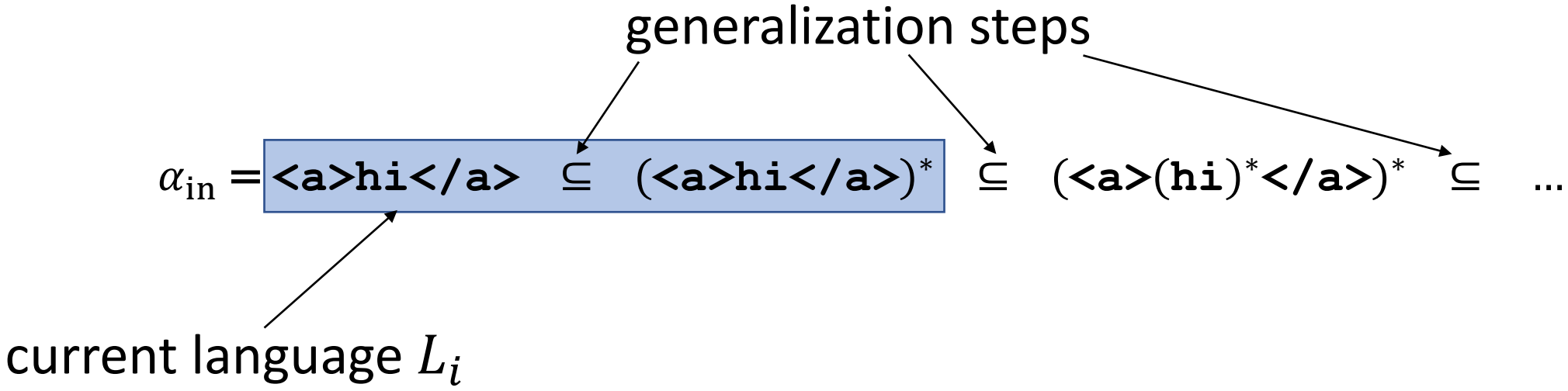
Idea: Construct a series of increasingly general languages



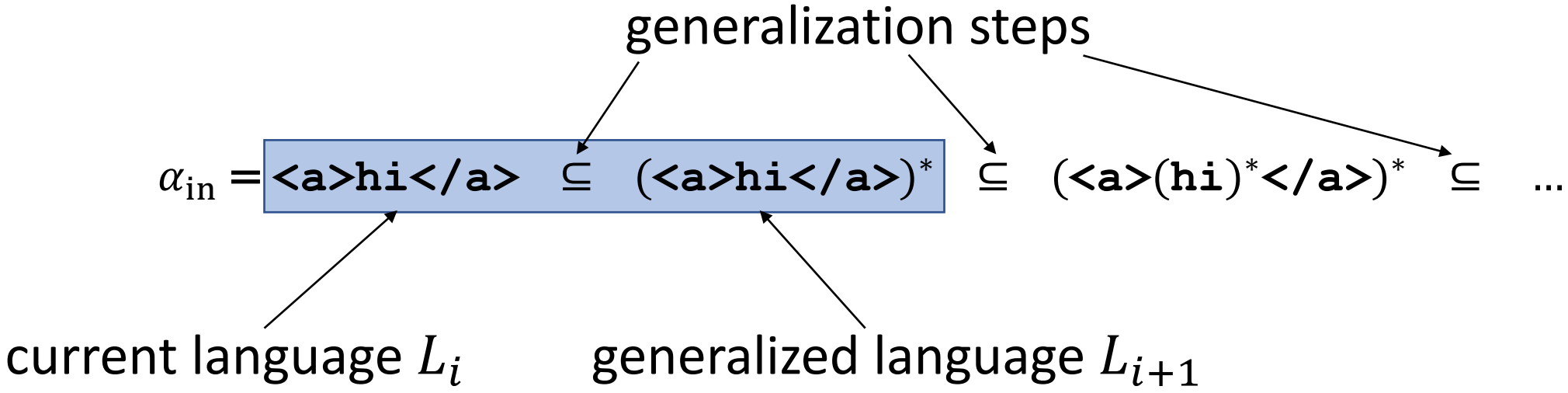
Idea: Construct a series of increasingly general languages



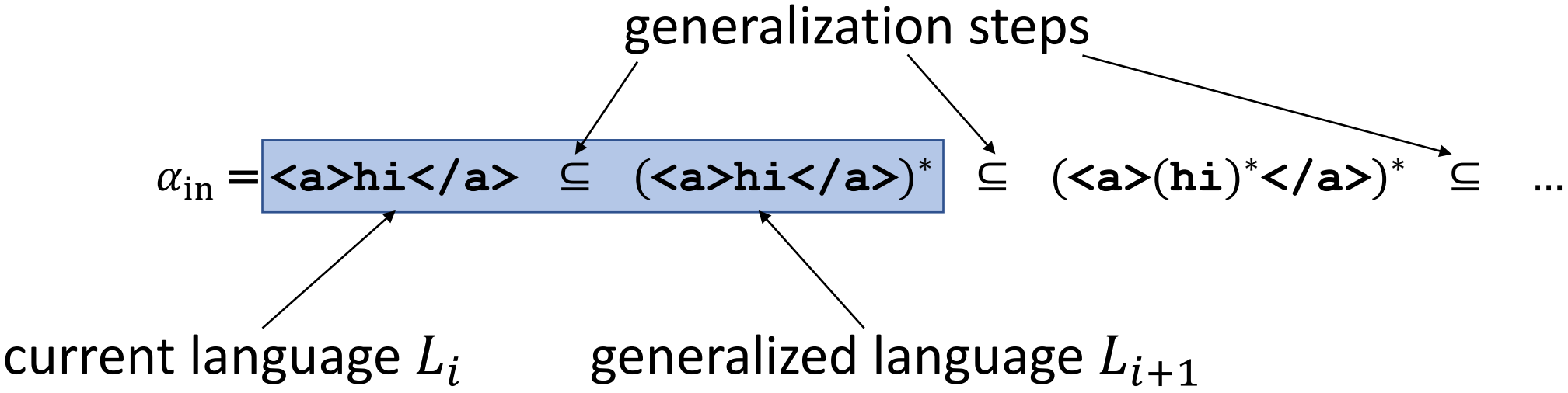
Idea: Construct a series of increasingly general languages



Idea: Construct a series of increasingly general languages

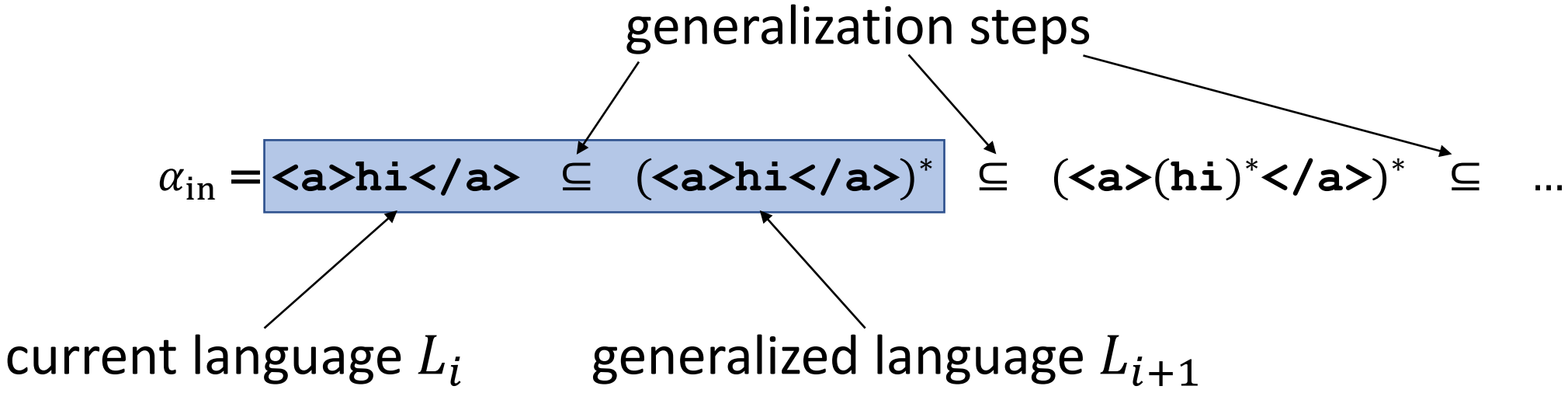


Idea: Construct a series of increasingly general languages



Monotone: $L_{i+1} \supseteq L_i$

Idea: Construct a series of increasingly general languages

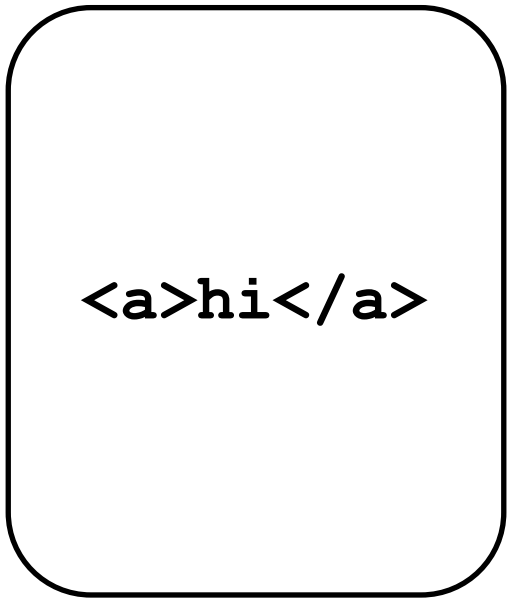


Monotone: $L_{i+1} \supseteq L_i$

Precise: $L_{i+1} \subseteq L_{\text{XML}}$

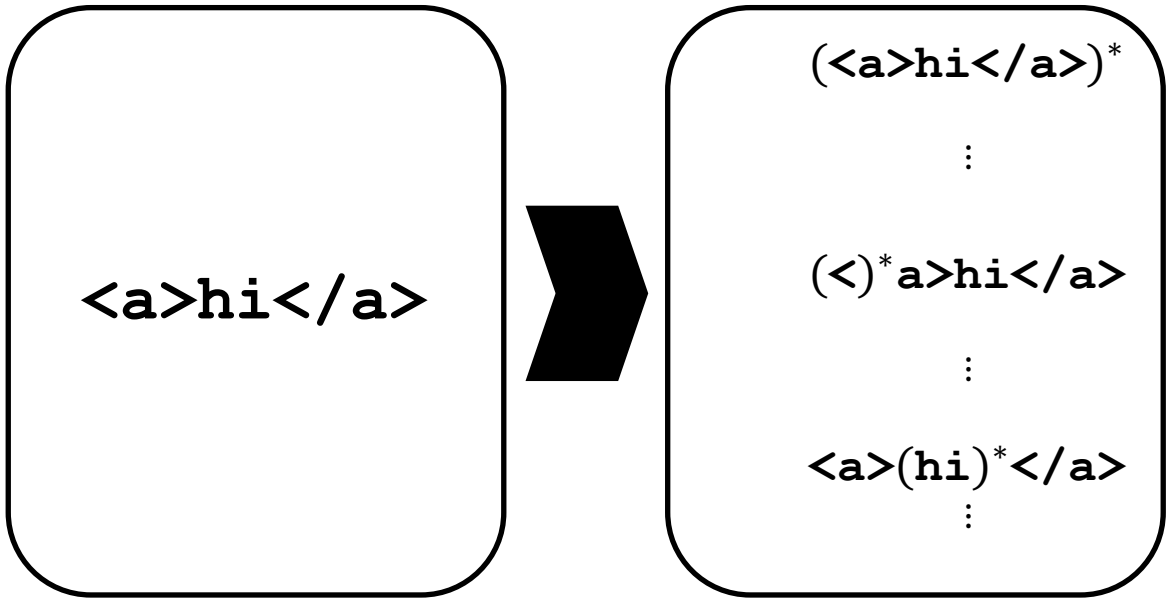
Generalization Step

Generalization Step



current
language

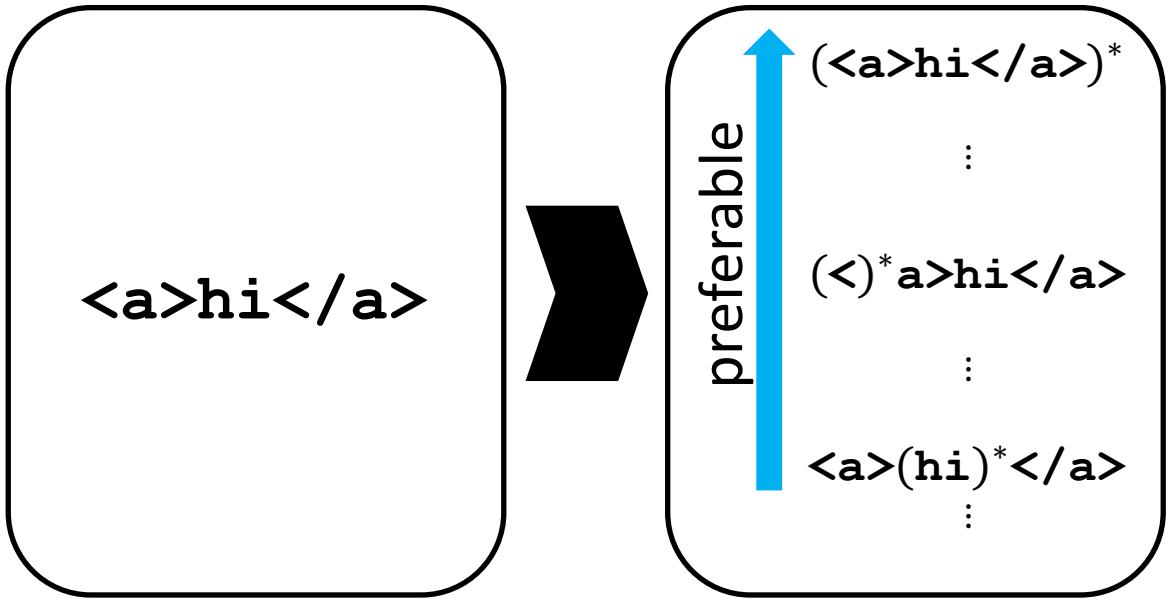
Generalization Step



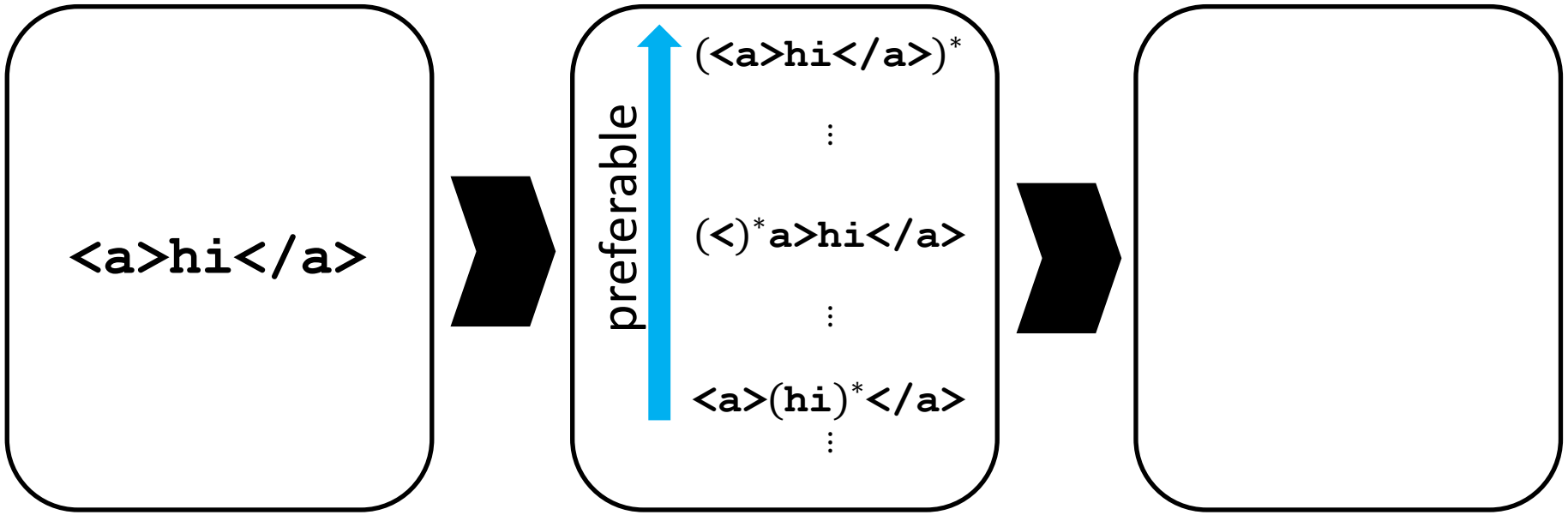
current language

candidates

Generalization Step



Generalization Step

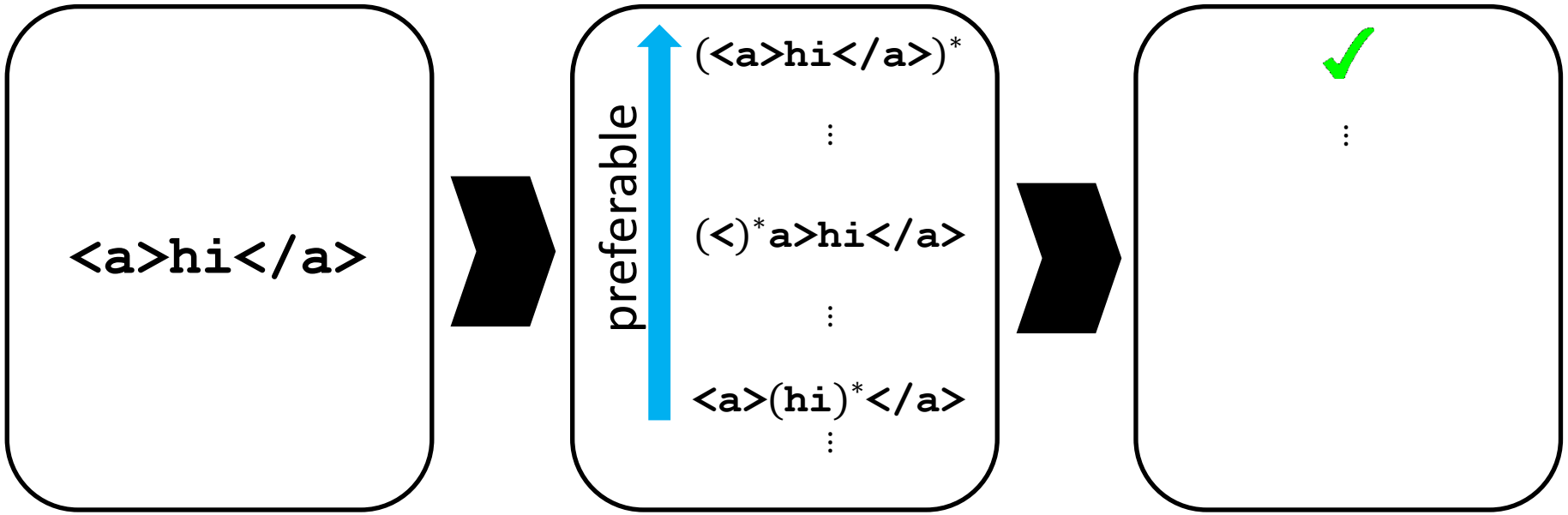


current language

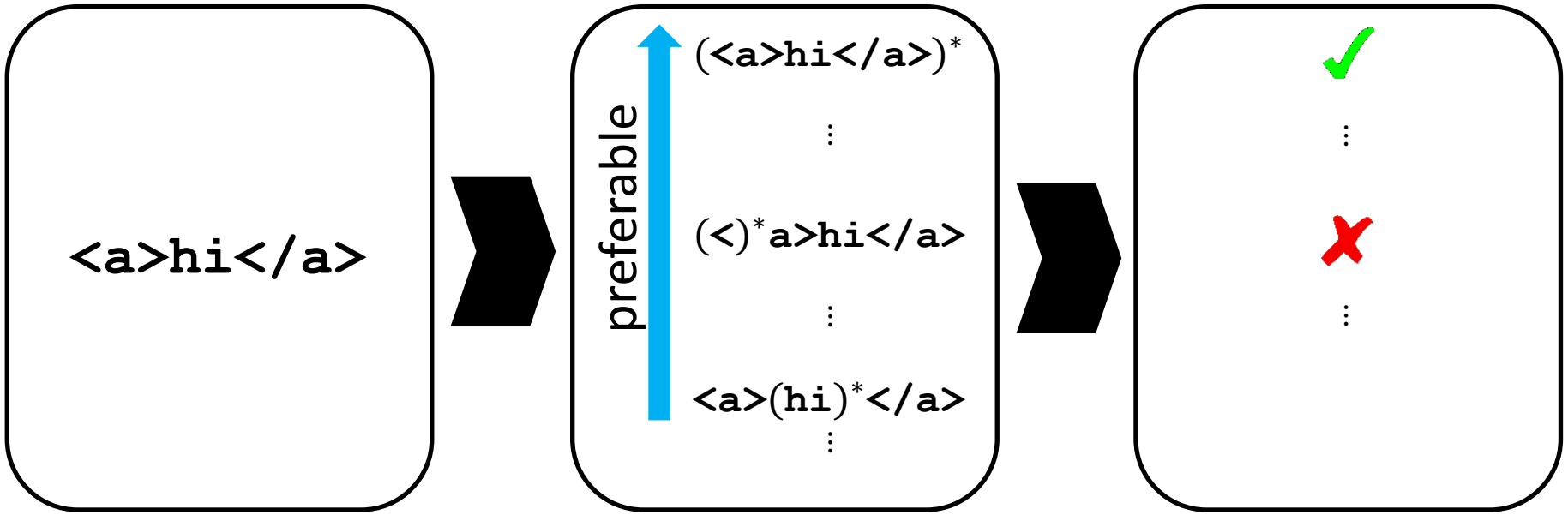
candidates

check precision

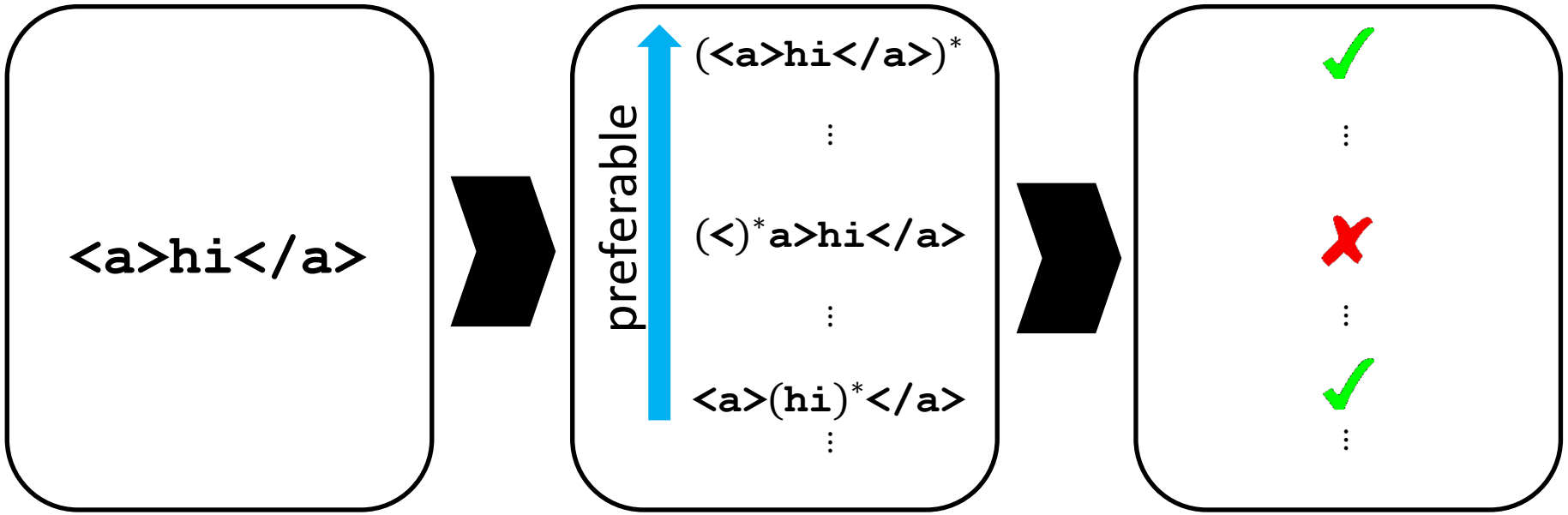
Generalization Step



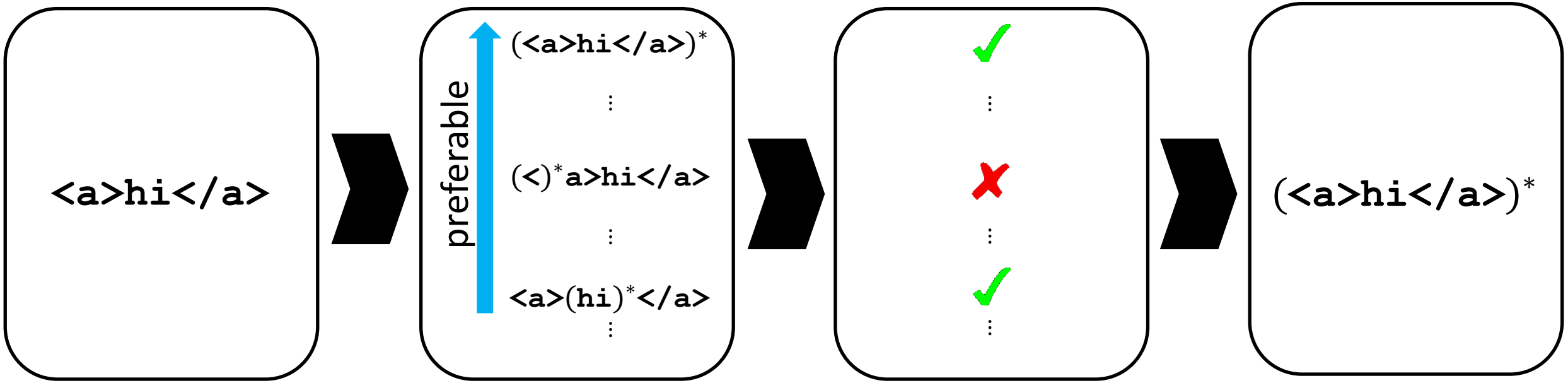
Generalization Step



Generalization Step



Generalization Step



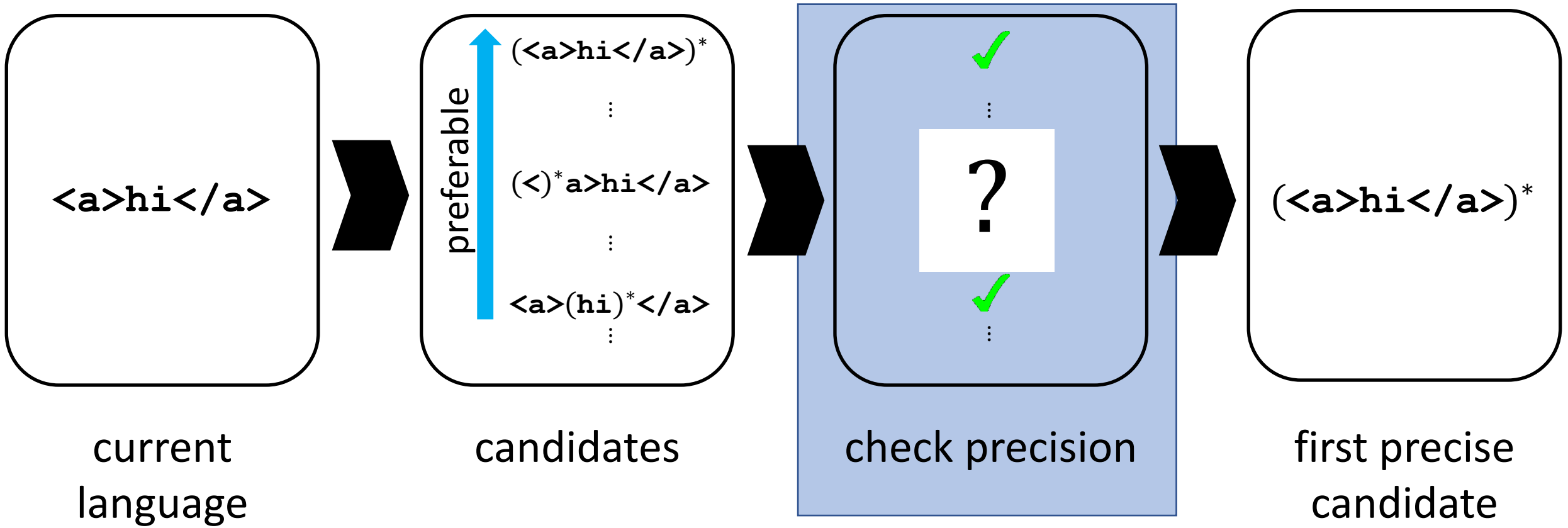
current language

candidates

check precision

first precise candidate

Generalization Step



Check Precision

For every $\alpha \in$ (<a>hi)*:

$$\alpha \in L_{\text{XML}}$$

Check Precision

For every $\alpha \in$ (<a>hi)*:

$$\mathcal{O}_{\mathrm{XML}}(\alpha) = 1$$

Check Precision

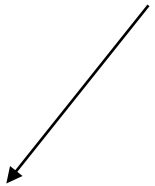
infinite!

For every $\alpha \in$ `(<a>hi</a>)`*:

$$\mathcal{O}_{\mathrm{XML}}(\alpha) = 1$$

Check Potential Precision

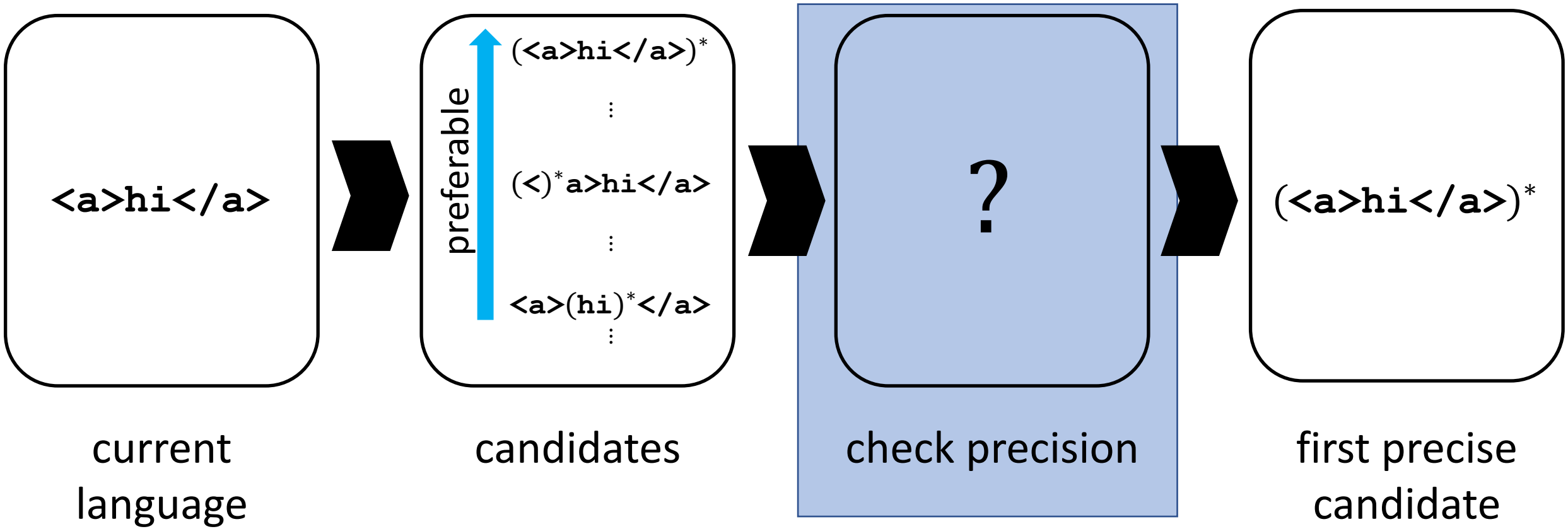
finite subset of **checks**



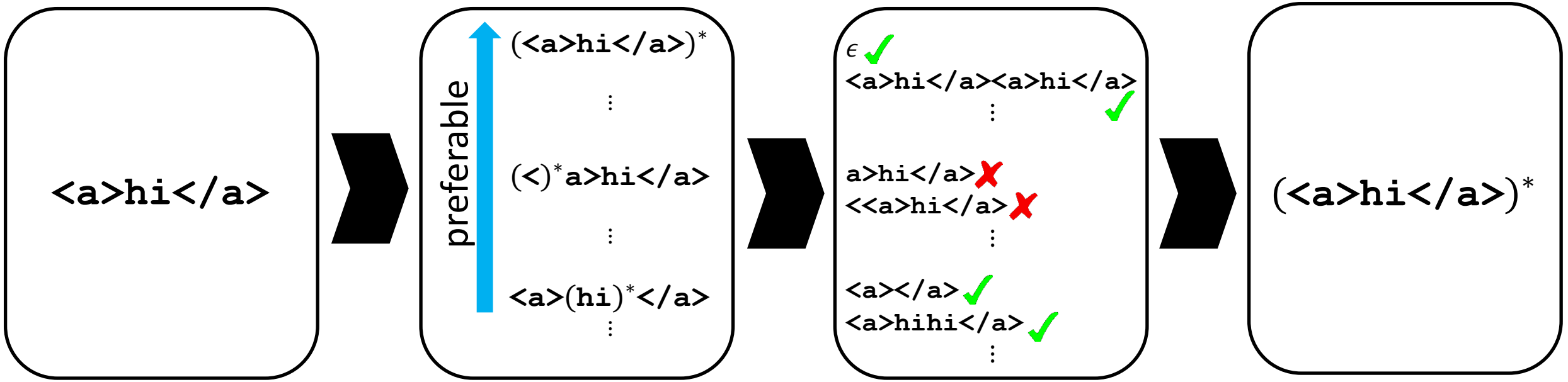
For every $\alpha \in S \subseteq (\texttt{<a>hi})^*$:

$$\mathcal{O}_{\mathrm{XML}}(\alpha) = 1$$

Generalization Step



Generalization Step



current language

candidates

check potential precision

first potentially precise candidate

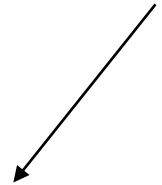
Check Potential Precision

For every $\alpha \in S \subseteq (\texttt{<a>hi})^*$:

$$\mathcal{O}_{\mathrm{XML}}(\alpha) = 1$$

Check Potential Precision

what if we made mistakes earlier?



For every $\alpha \in S \subseteq (\texttt{<a>hi})^*$:

$$\mathcal{O}_{\mathrm{XML}}(\alpha) = 1$$

Precise: $L_{i+1} \subseteq L_{\mathrm{XML}}$

ignore past mistakes

Precise: $L_{i+1} \setminus L_i \subseteq L_{\mathrm{XML}}$

Check Potential Precision

For every $\alpha \in S \subseteq (\texttt{<a>hi})^*$:

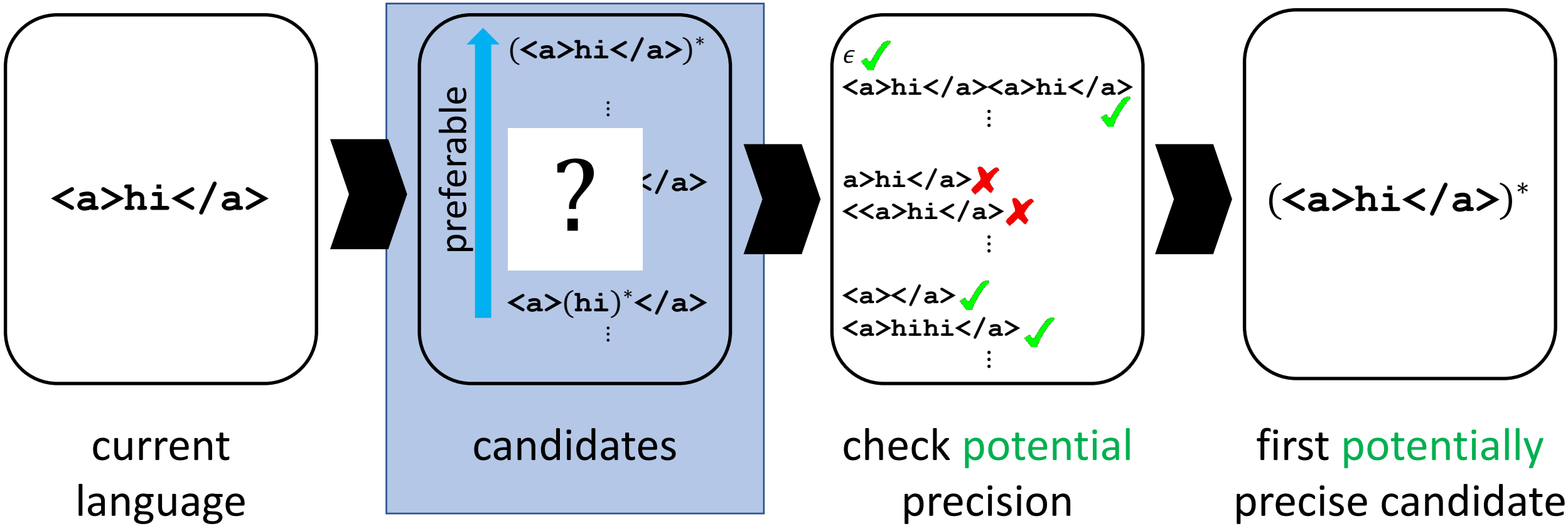
$$\mathcal{O}_{\mathrm{XML}}(\alpha) = 1$$

Check Potential Precision

For every $\alpha \in S \subseteq (\texttt{<a>hi})^* \setminus \texttt{<a>hi}$:

$$\mathcal{O}_{\mathrm{XML}}(\alpha) = 1$$

Generalization Step



input example `<a>hi</a>`

input example { **<a>hi**

regular expression { $\Rightarrow$ (**<a>hi**)*

$\Rightarrow$ (**<a>**(**hi**)*****)*

$\Rightarrow$ (**<a>**(**h**+**i**)*****)*

input example

<a>hi

regular expression

$\Rightarrow$ $(\texttt{<a>hi})^*$

$\Rightarrow$ $(\texttt{<a>}(\texttt{hi})^*\texttt{})^*$

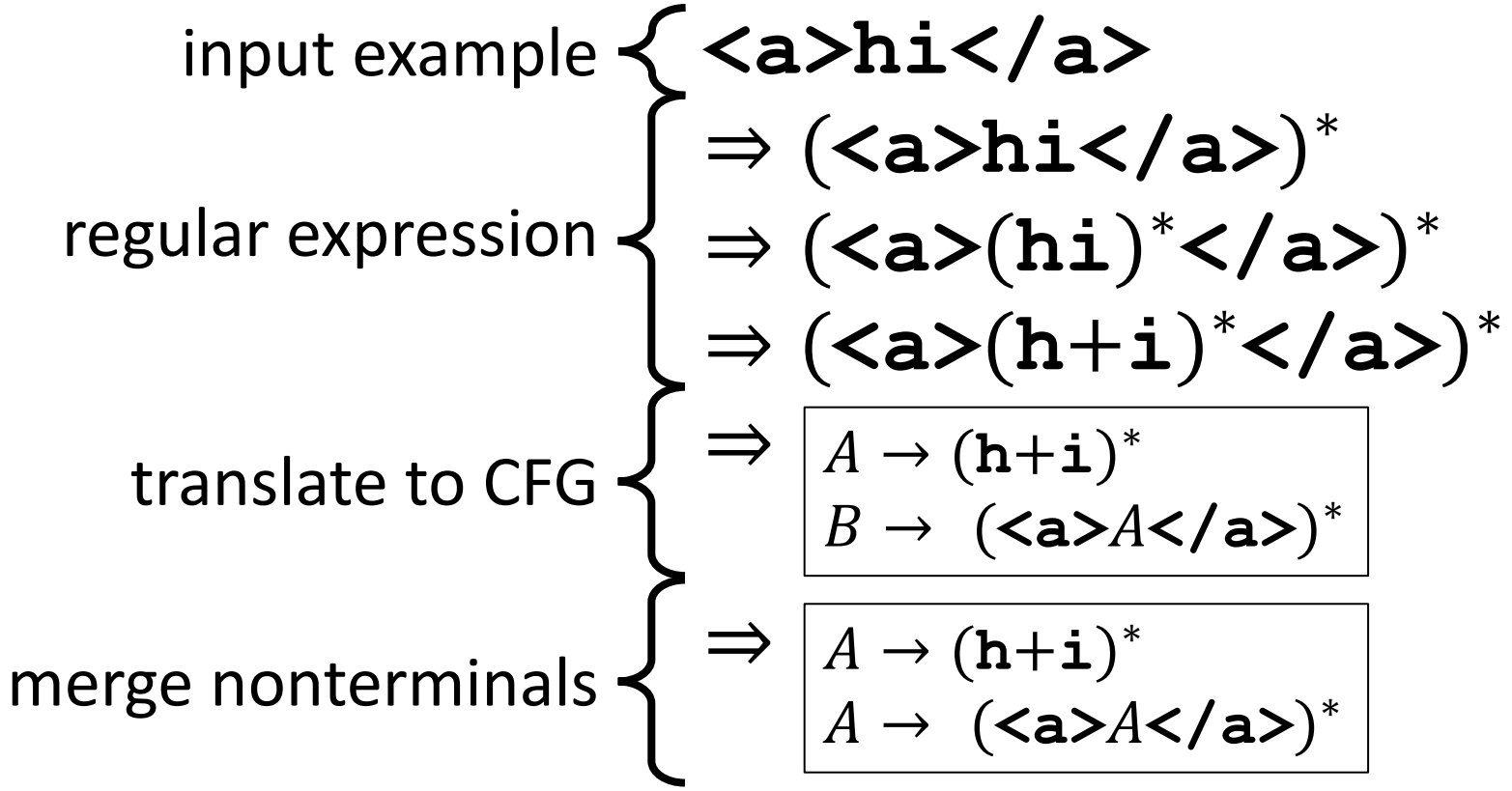
$\Rightarrow$ $(\texttt{<a>}(\texttt{h}+\texttt{i})^*\texttt{})^*$

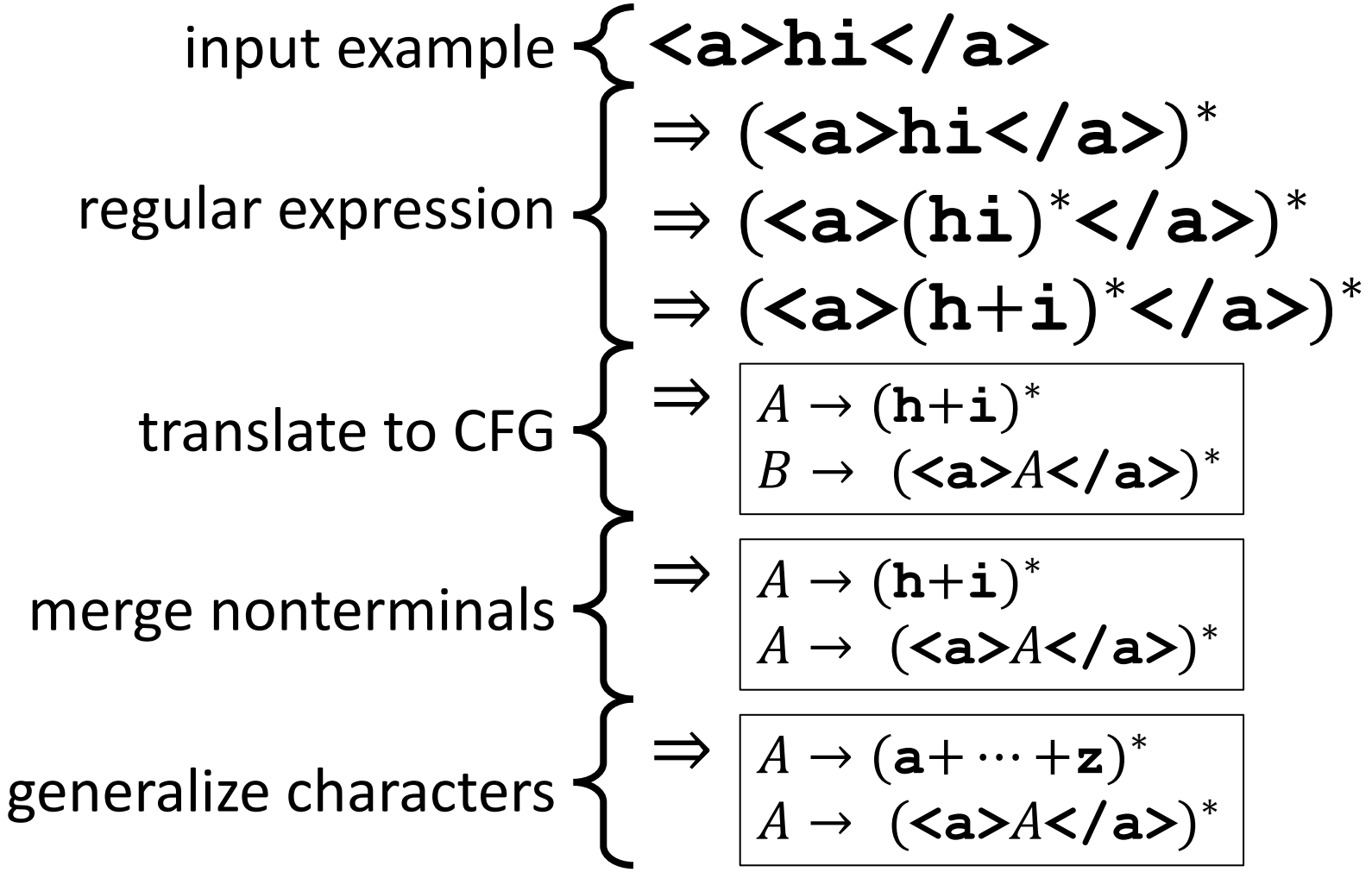
translate to CFG

$\Rightarrow$

$A \rightarrow (\texttt{h}+\texttt{i})^*$

$B \rightarrow (\texttt{<a>}A\texttt{})^*$


input example
regular expression
translate to CFG
merge nonterminals
<a>hi
⇒ (<a>hi)*
⇒ (<a>(hi)*)*
⇒ (<a>(h+i)*)*
⇒
A → (h+i)*
B → (<a>A)*
⇒
A → (h+i)*
A → (<a>A)*


input example
<a>hi
regular expression
⇒ (<a>hi)*
⇒ (<a>(hi)*)*
⇒ (<a>(h+i)*)*
translate to CFG
⇒
A → (h+i)*
B → (<a>A)*
merge nonterminals
⇒
A → (h+i)*
A → (<a>A)*
generalize characters
⇒
A → (a+ ⋯ +z)*
A → (<a>A)*

Multiple Input Examples

Multiple Input Examples

Input examples: $\alpha_1, \ldots, \alpha_k$

Multiple Input Examples

Input examples: $\alpha_1, \ldots, \alpha_k$

Regular expressions: $R_1, \ldots, R_k$

Multiple Input Examples

Input examples: $\alpha_1, \ldots, \alpha_k$

Regular expressions: $R_1, \ldots, R_k$

Combine: $R = R_1 + \cdots + R_k$

Multiple Input Examples

Input examples:	$\alpha_1, \ldots, \alpha_k$
Regular expressions:	$R_1, \ldots, R_k$
Combine:	$R = R_1 + \cdots + R_k$
Merging nonterminals:	C

Multiple Input Examples

Input examples:	$\alpha_1, \ldots, \alpha_k$
Regular expressions:	$R_1, \ldots, R_k$
Combine:	$R = R_1 + \cdots + R_k$
Merging nonterminals:	C
Generalize constants:	C'

Evaluation

Evaluation

Grammar learning: Compare to language learners

Evaluation

Grammar learning: Compare to language learners

Fuzz testing: Compare to fuzzers

Evaluation: Grammar Learning

Evaluation: Grammar Learning

Baselines: L-Star, RPNI

Evaluation: Grammar Learning

Baselines: L-Star, RPNI

Grammars: URL, Grep, LISP, XML

Evaluation: Grammar Learning

Baselines: L-Star, RPNI

Grammars: URL, Grep, LISP, XML

Inputs: membership oracle $\mathcal{O}$
50 random strings $E_{\text{in}} \subseteq L_*$
5 minutes

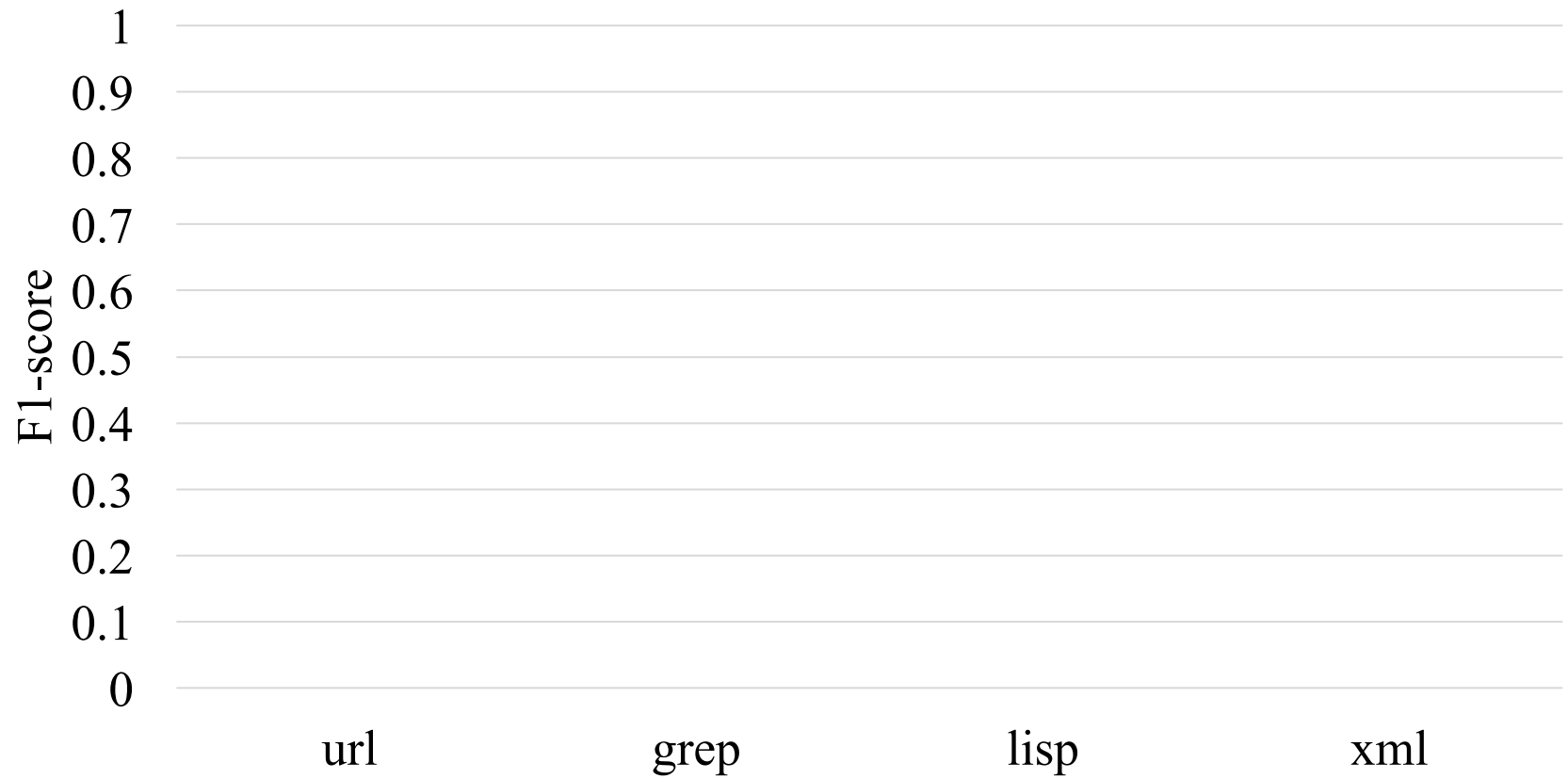
Evaluation: Grammar Learning

Precision: $$\frac{\#\text{ valid sampled inputs}}{\#\text{ sampled inputs}}$$

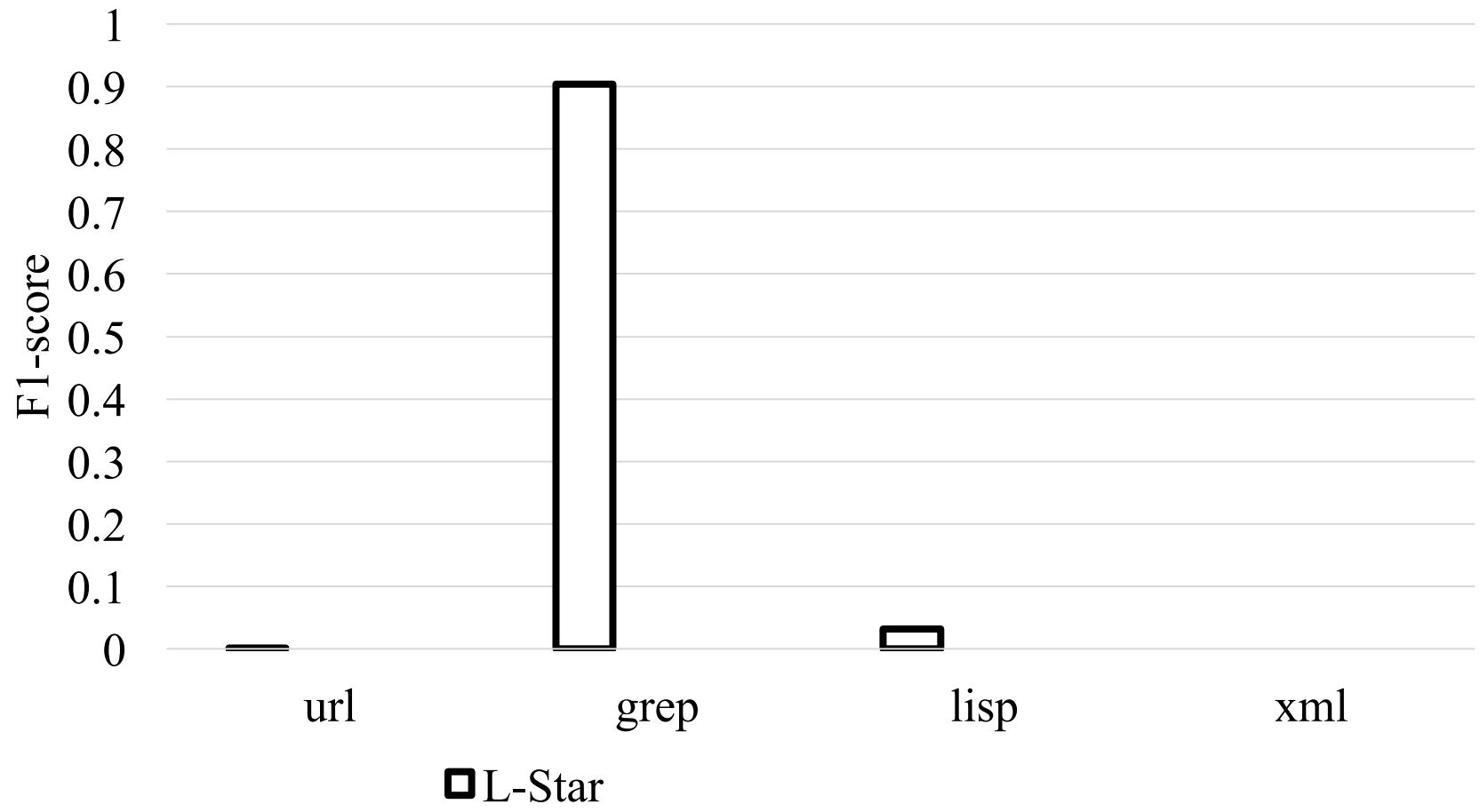
Recall: $$\frac{\#\text{ true inputs that might be sampled}}{\#\text{ true inputs}}$$

F_1-Score: $$\frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

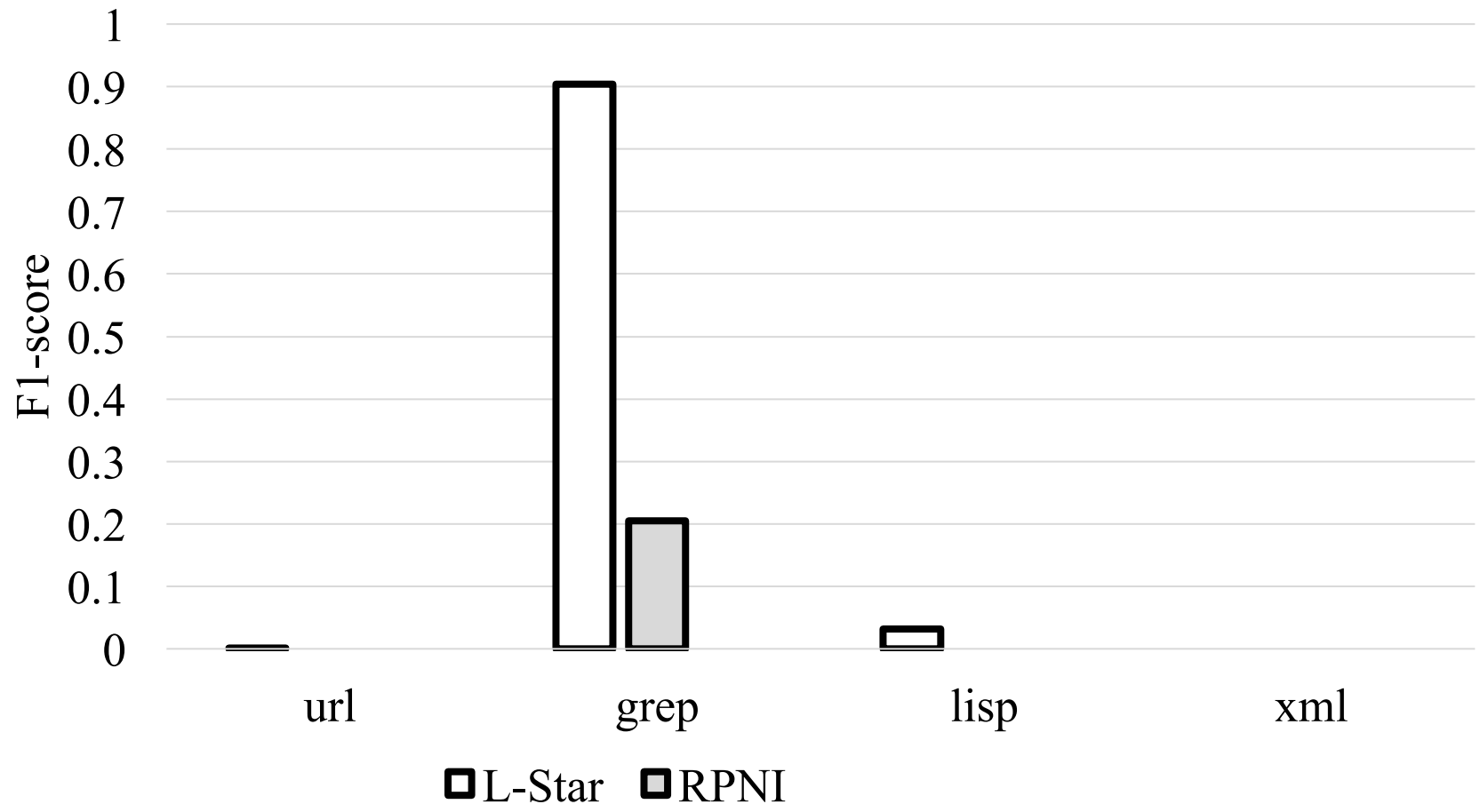
Evaluation: Grammar Learning



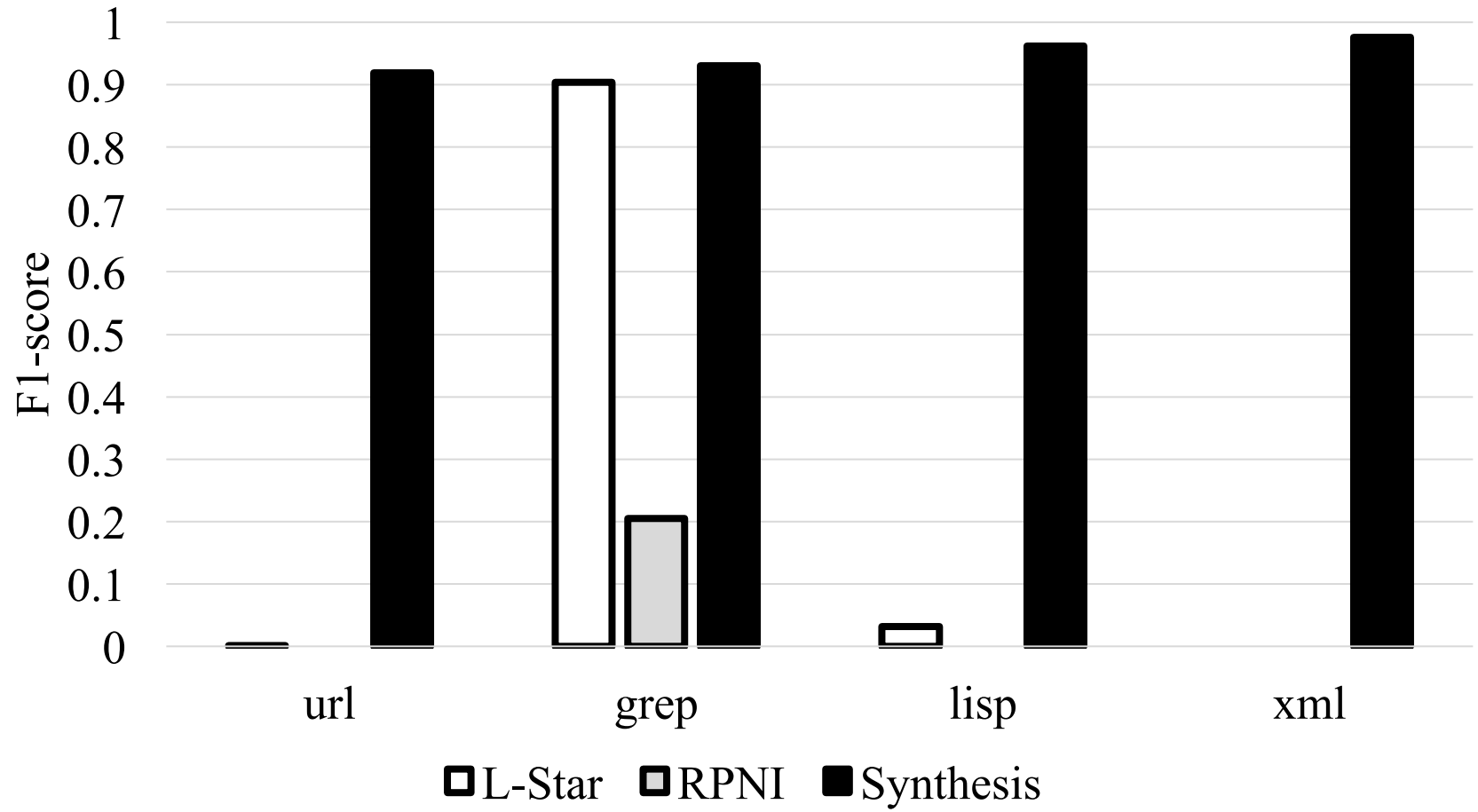
Evaluation: Grammar Learning



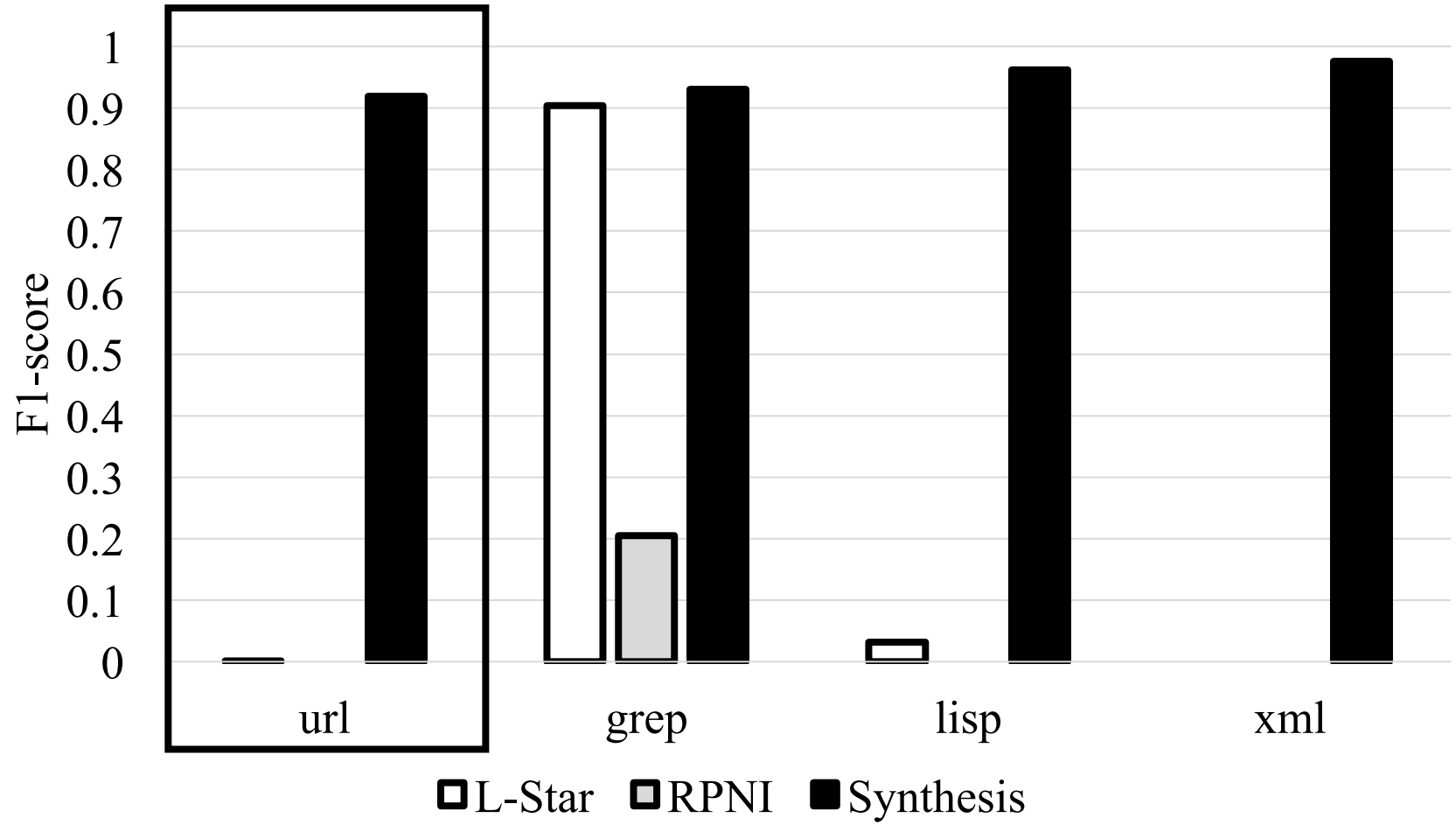
Evaluation: Grammar Learning



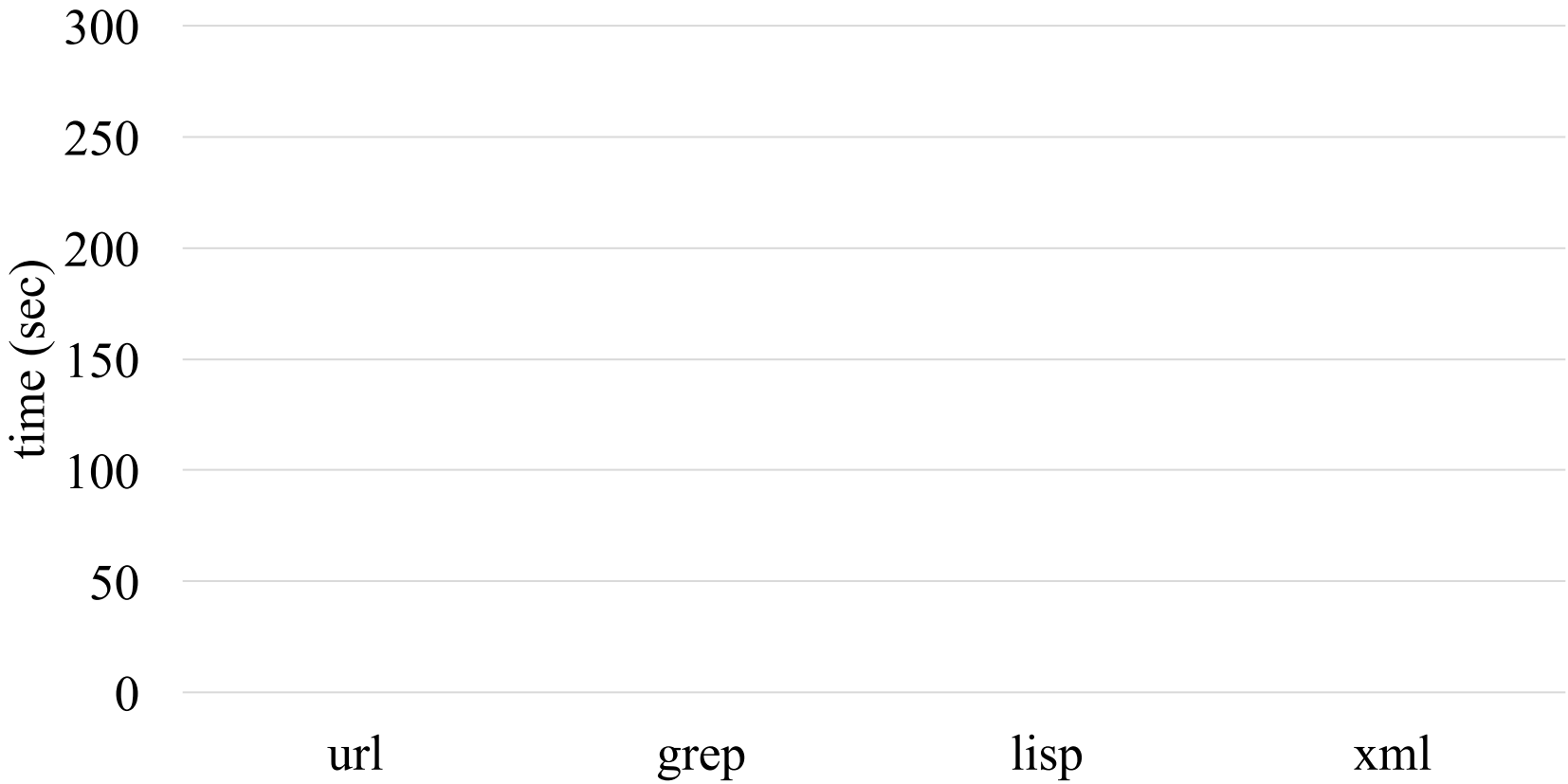
Evaluation: Grammar Learning



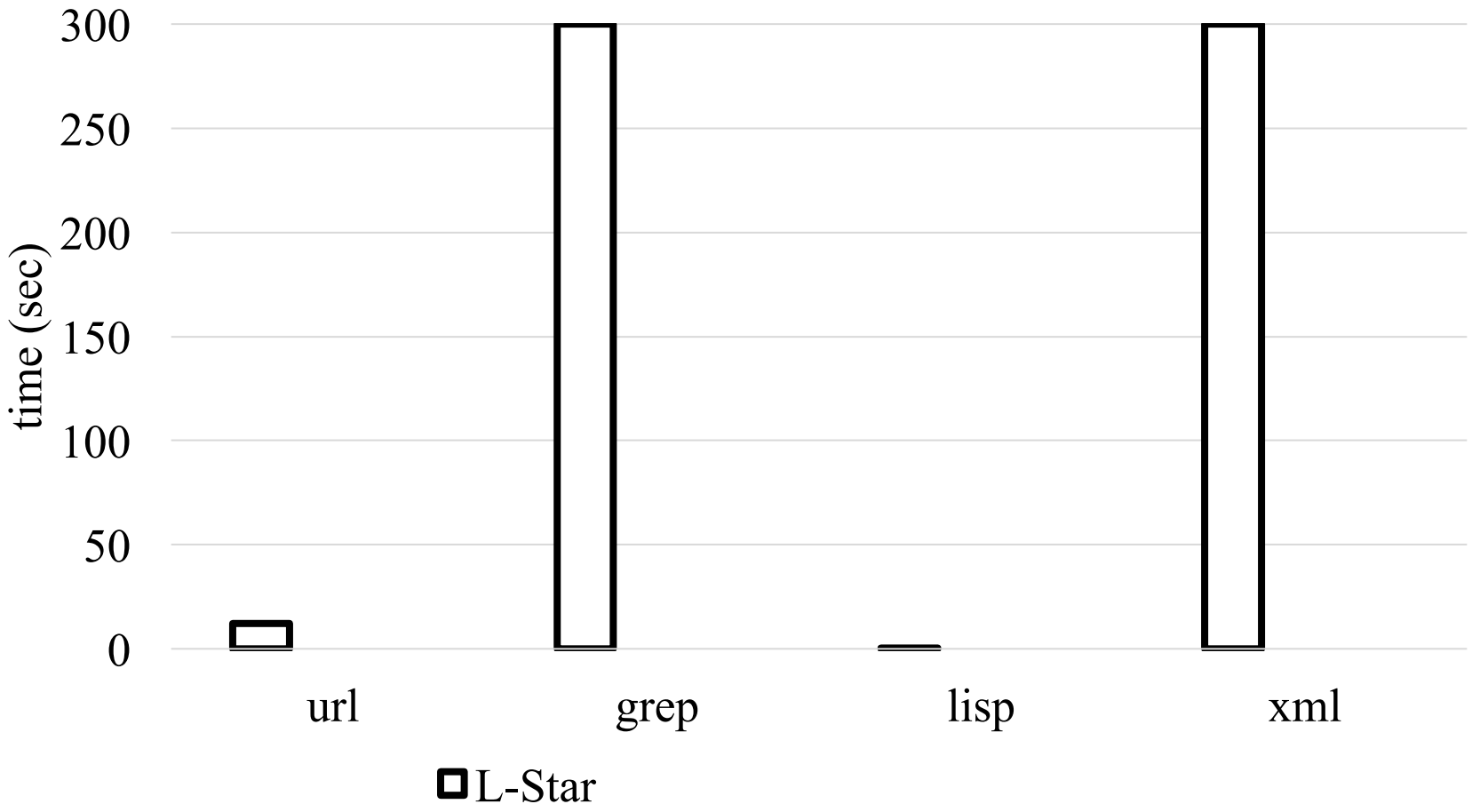
Evaluation: Grammar Learning



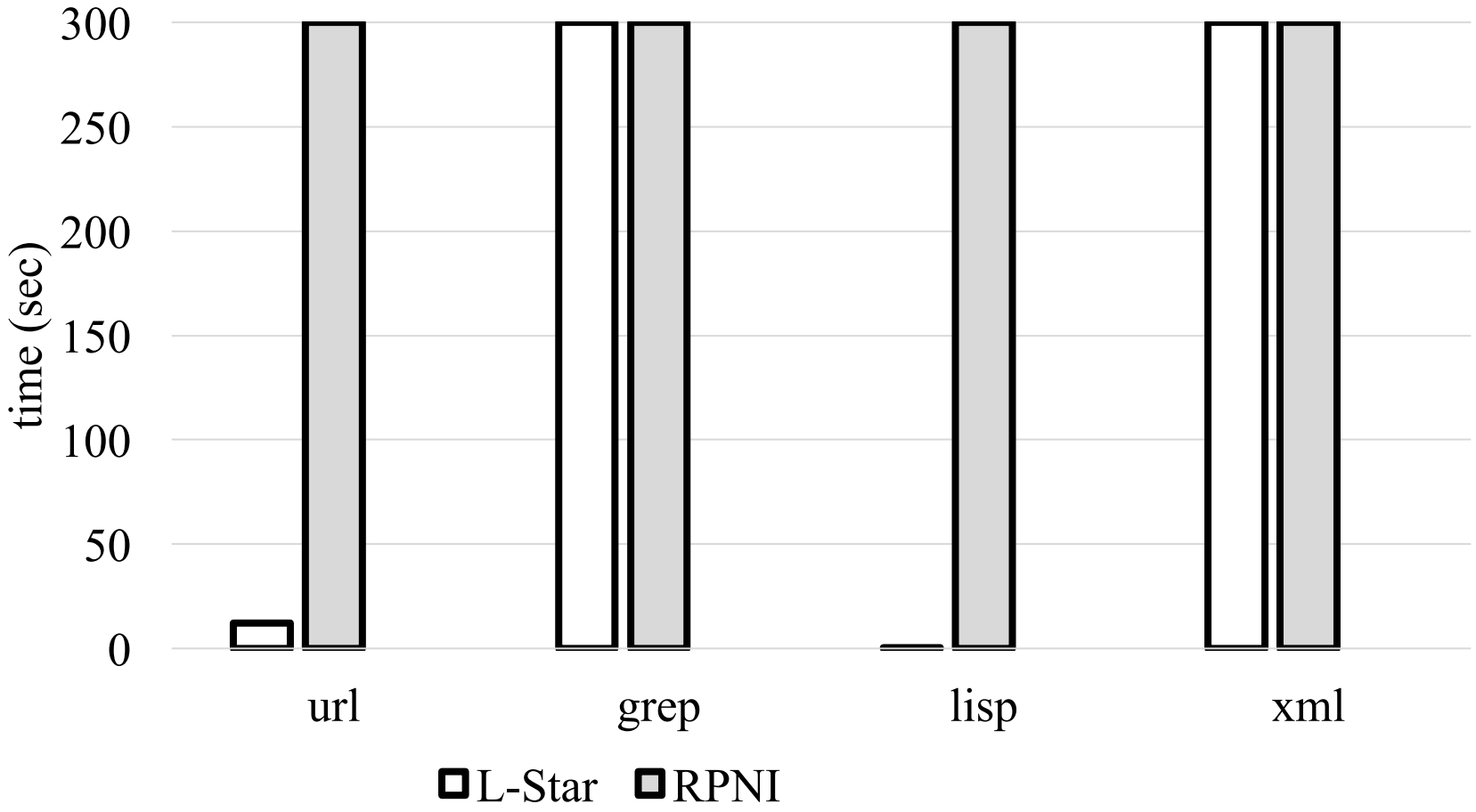
Evaluation: Grammar Learning



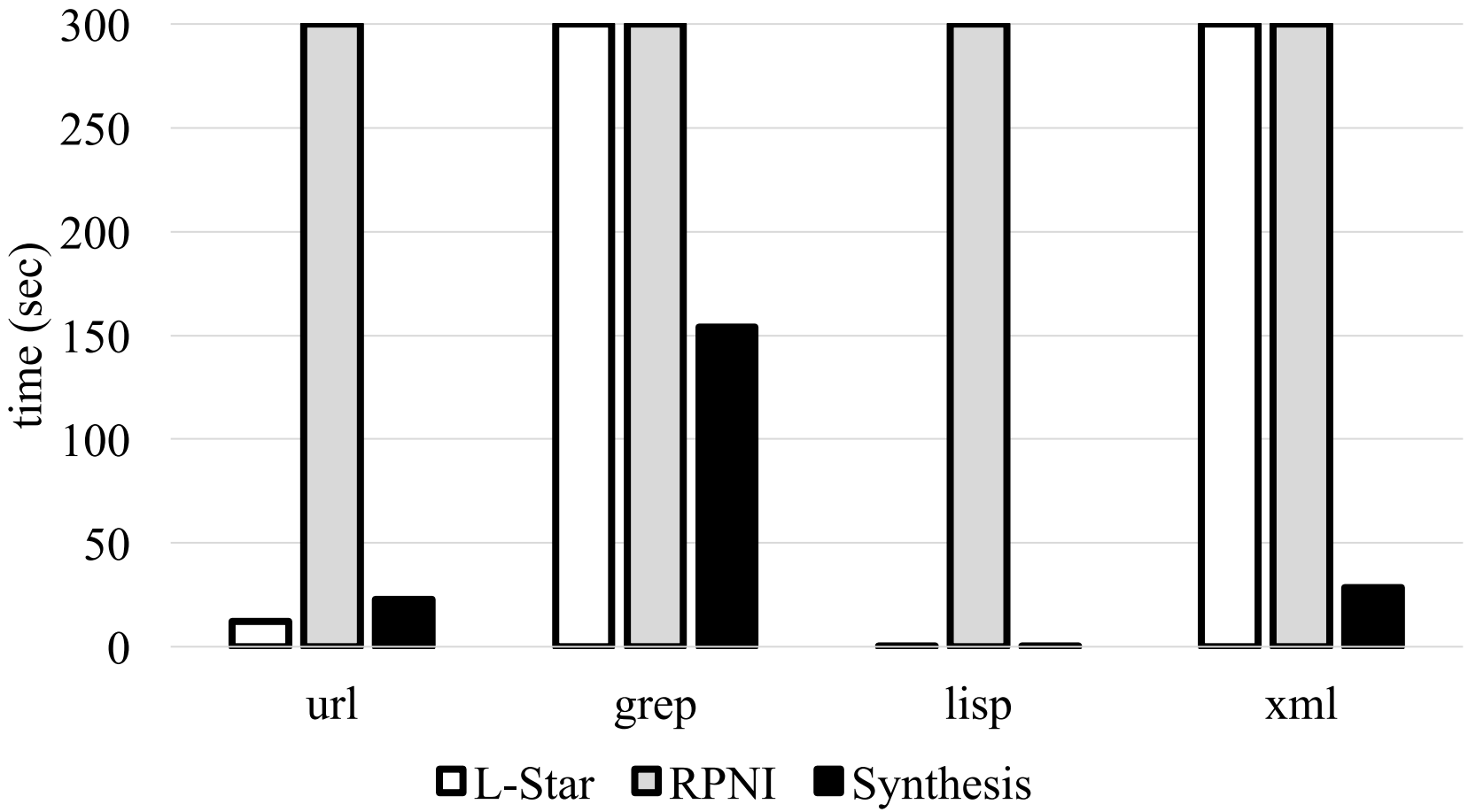
Evaluation: Grammar Learning



Evaluation: Grammar Learning



Evaluation: Grammar Learning



Evaluation: Fuzz Testing

Evaluation: Fuzz Testing

Fuzzer: synthesize grammar,
randomly resample subtrees of parse tree

Evaluation: Fuzz Testing

Fuzzer: synthesize grammar,
randomly resample subtrees of parse tree

Baselines: naïve (random insertions/deletions)
afl-fuzz (production fuzzer)

Evaluation: Fuzz Testing

Fuzzer: synthesize grammar,
randomly resample subtrees of parse tree

Baselines: naïve (random insertions/deletions)
afl-fuzz (production fuzzer)

Programs: Grep, Sed, Flex, Bison, XML Parser
Python, Ruby, SpiderMonkey (parser only)

Evaluation: Fuzz Testing

Fuzzer:	synthesize grammar, randomly resample subtrees of parse tree
Baselines:	naïve (random insertions/deletions) afl-fuzz (production fuzzer)
Programs:	Grep, Sed, Flex, Bison, XML Parser Python, Ruby, SpiderMonkey (parser only)
Inputs:	50,000 samples

Evaluation: Fuzz Testing

Evaluation: Fuzz Testing

Valid coverage: $\mathrm{Cov}(E) = \#(\text{lines covered by } E \cap L_*)$

Evaluation: Fuzz Testing

Valid coverage: $\mathrm{Cov}(E) = \#(\text{lines covered by } E \cap L_*)$

Incremental coverage: $\mathrm{IncCov}(E) = \mathrm{Cov}(E) - \mathrm{Cov}(\alpha_{\mathrm{in}})$

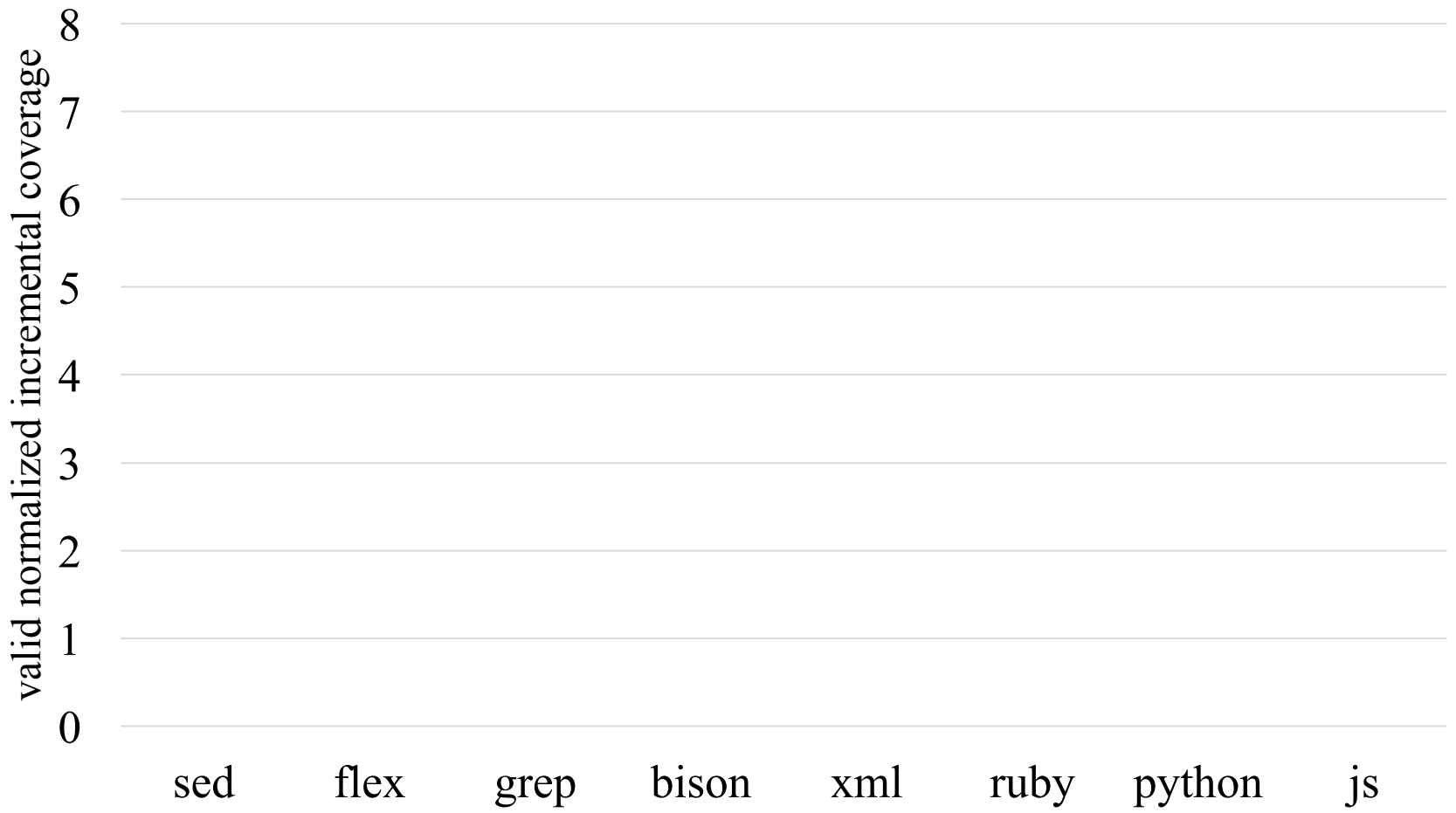
Evaluation: Fuzz Testing

Valid coverage: $\mathrm{Cov}(E) = \#(\text{lines covered by } E \cap L_*)$

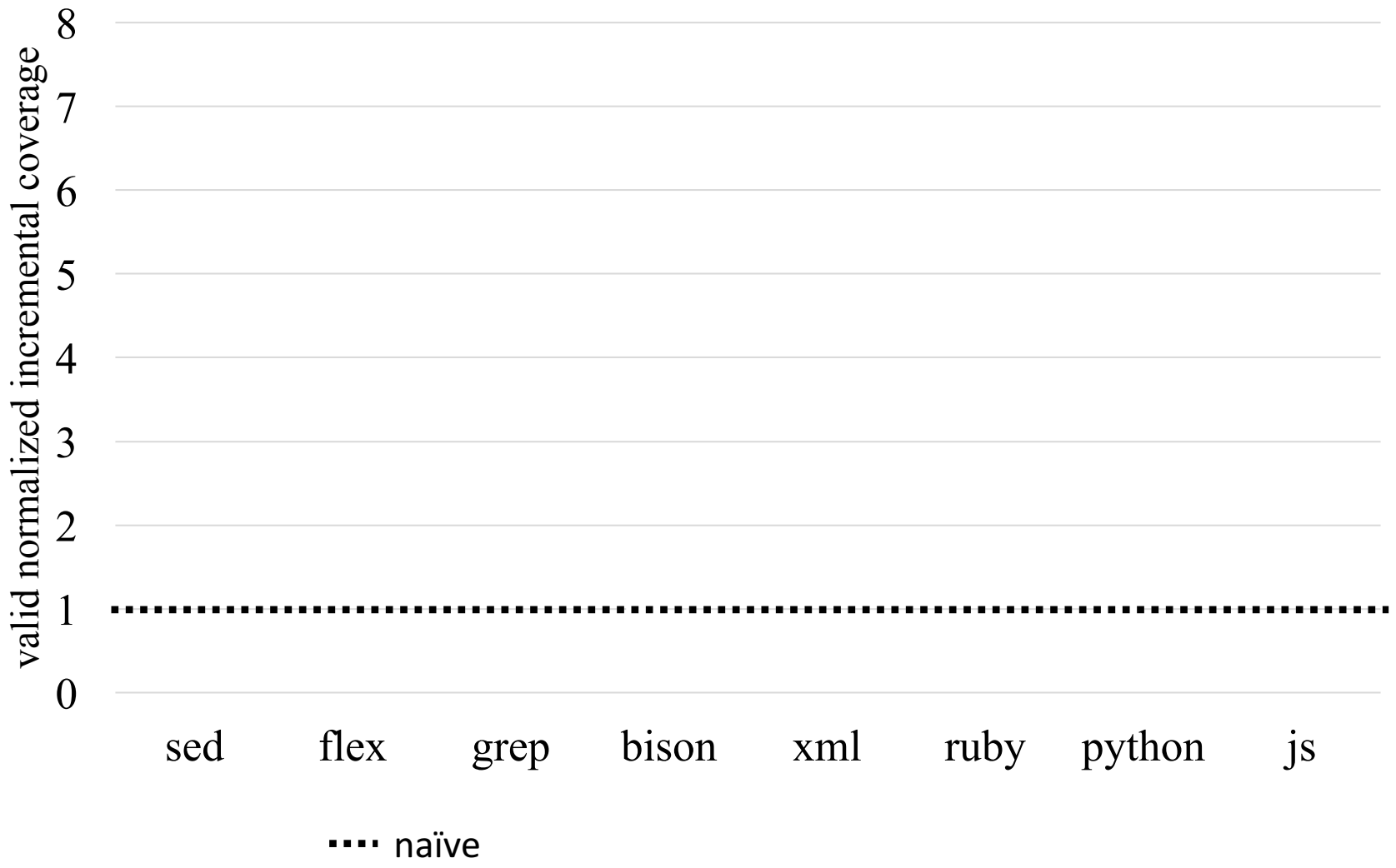
Incremental coverage: $\mathrm{IncCov}(E) = \mathrm{Cov}(E) - \mathrm{Cov}(\alpha_{\mathrm{in}})$

Normalized: $\mathrm{NormIncCov}(E) = \dfrac{\mathrm{IncCov}(E)}{\mathrm{IncCov}(E_{\mathrm{naïve}})}$

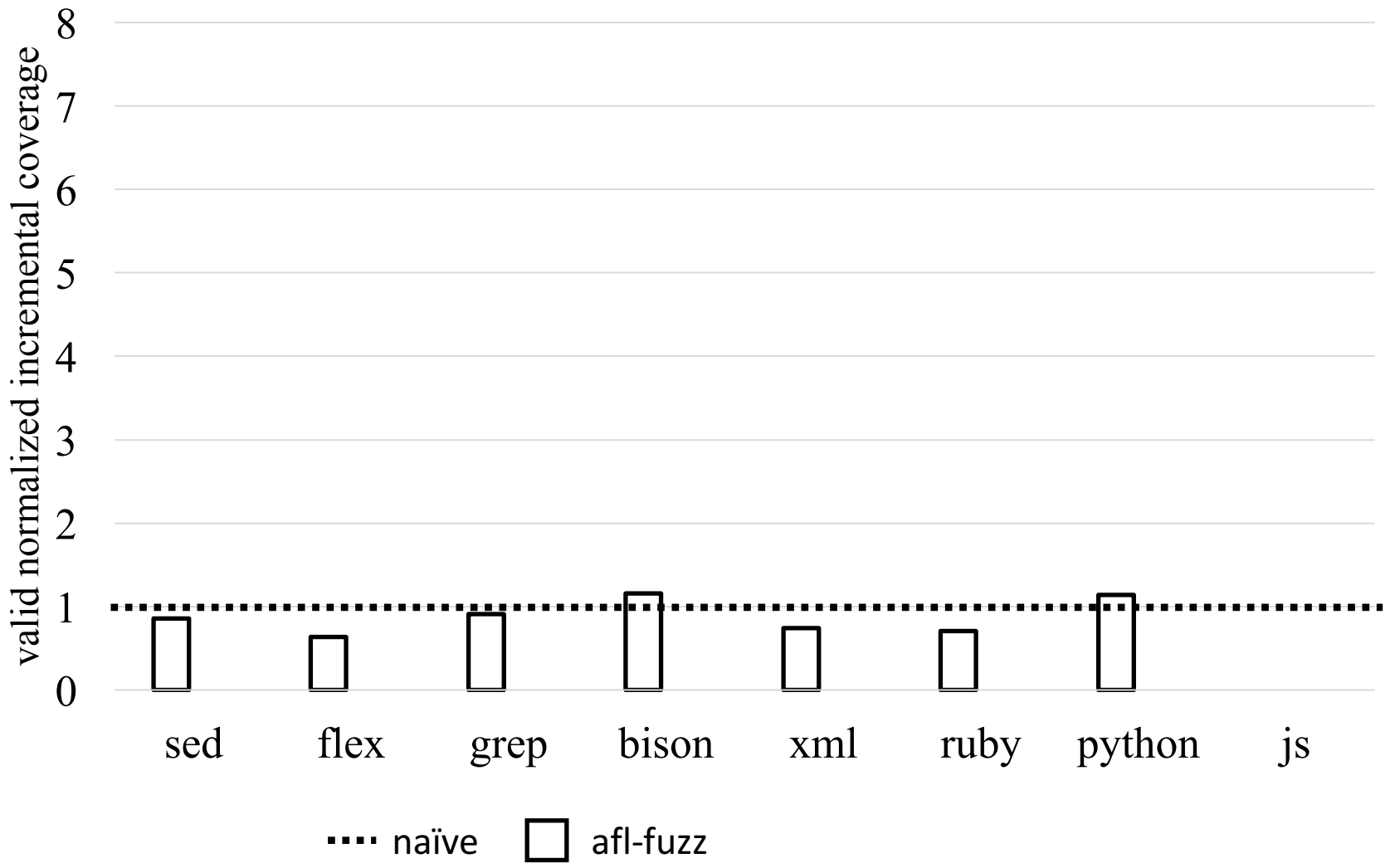
Evaluation: Fuzz Testing



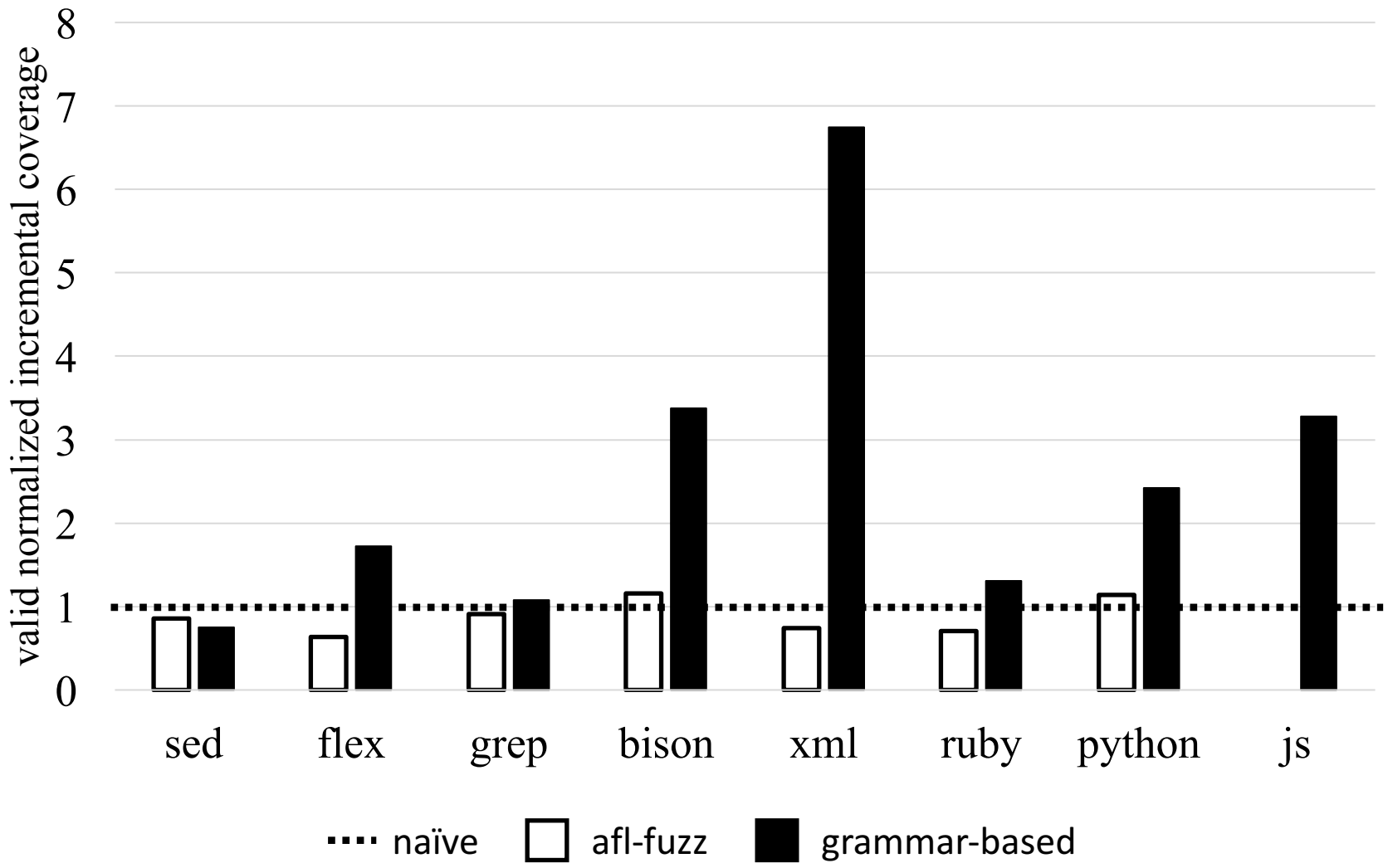
Evaluation: Fuzz Testing



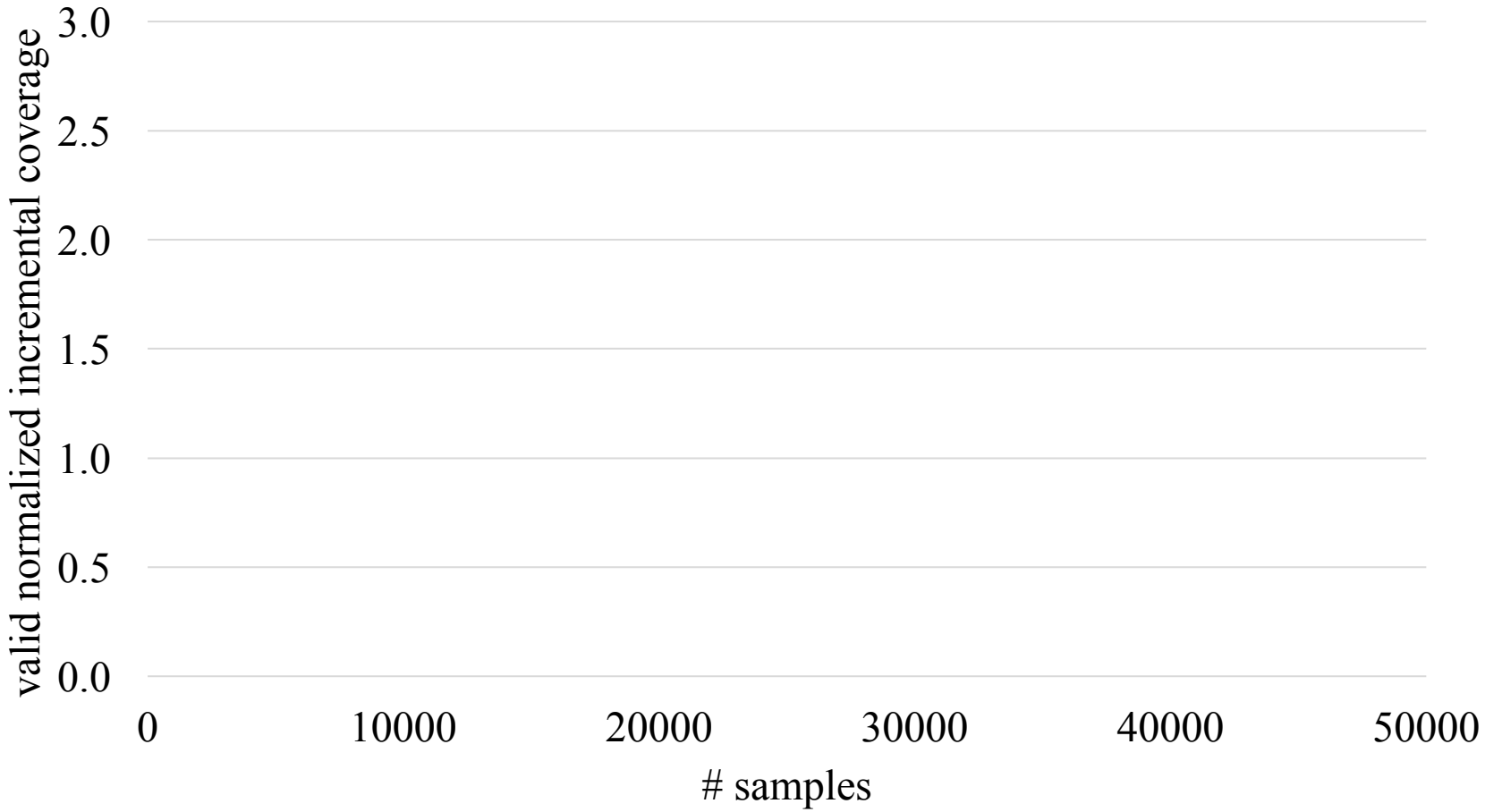
Evaluation: Fuzz Testing



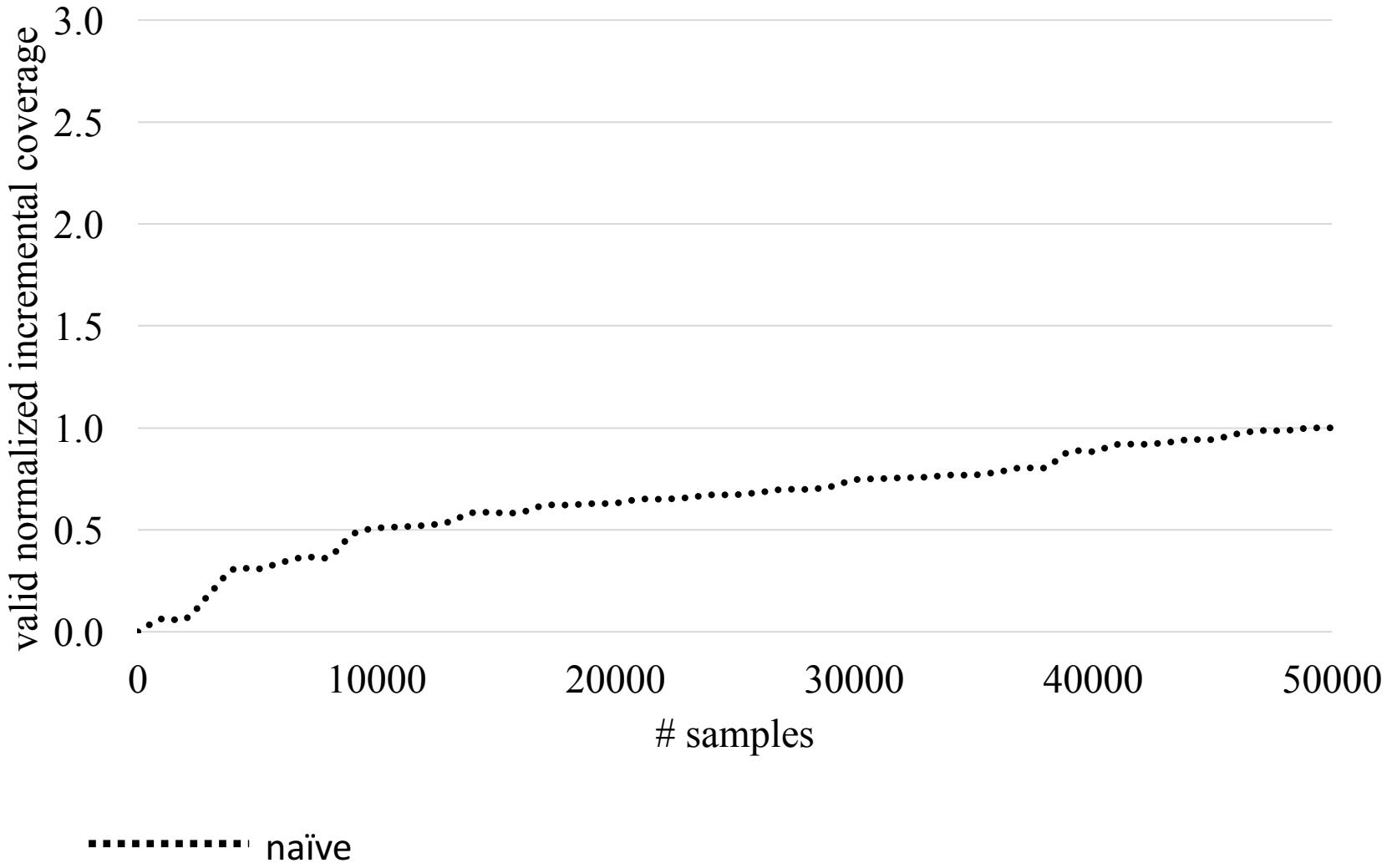
Evaluation: Fuzz Testing



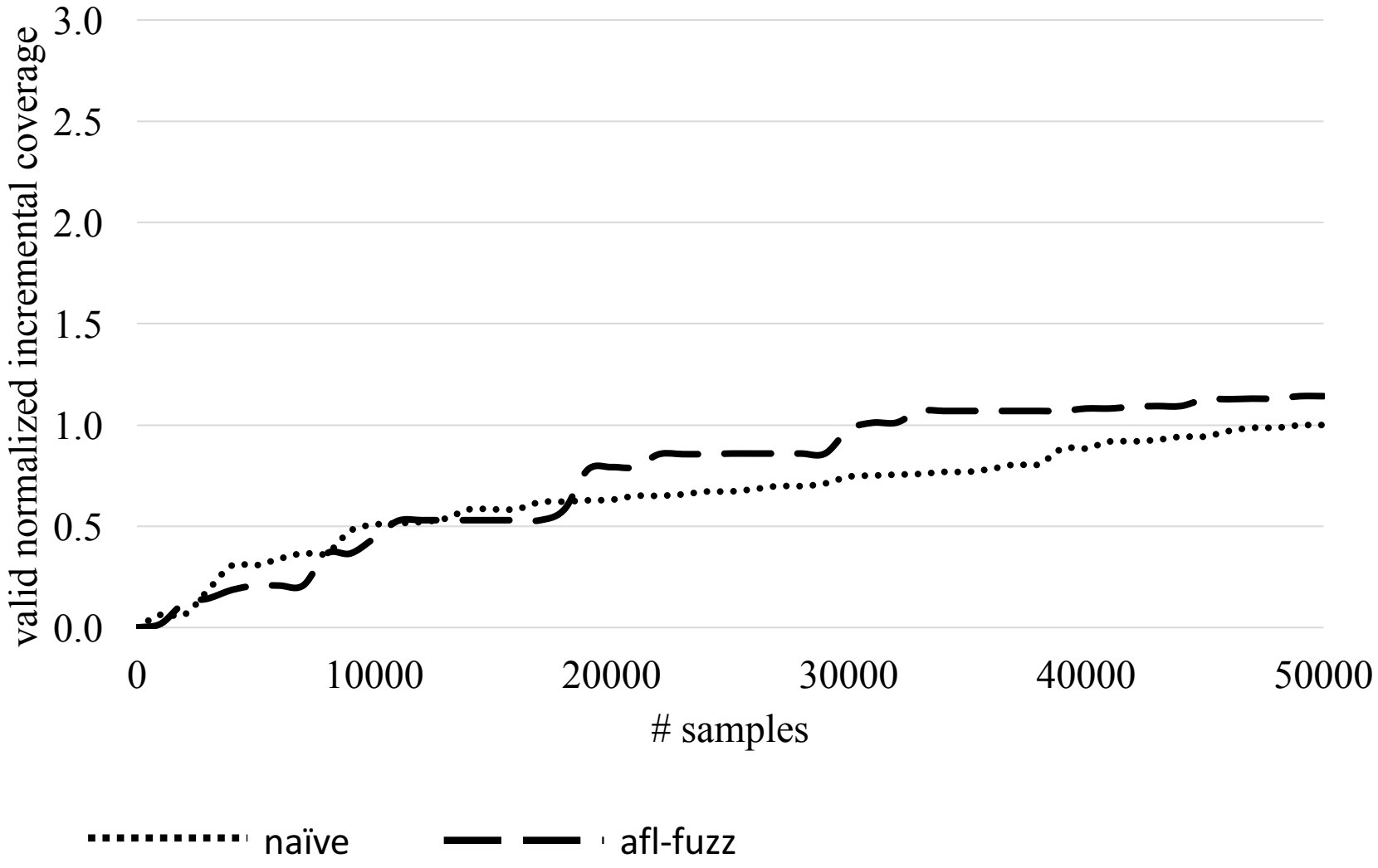
Evaluation: Fuzz Testing



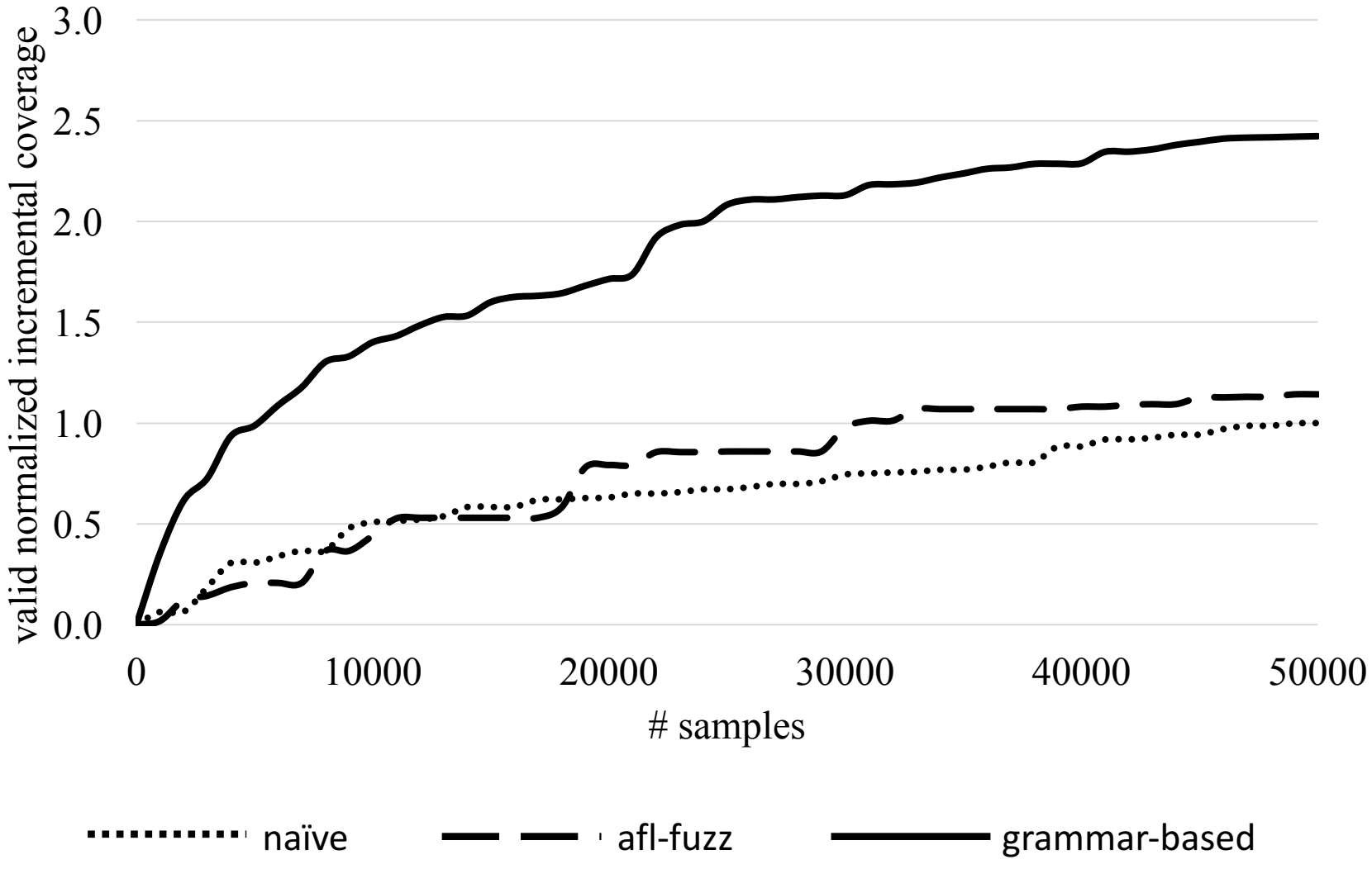
Evaluation: Fuzz Testing



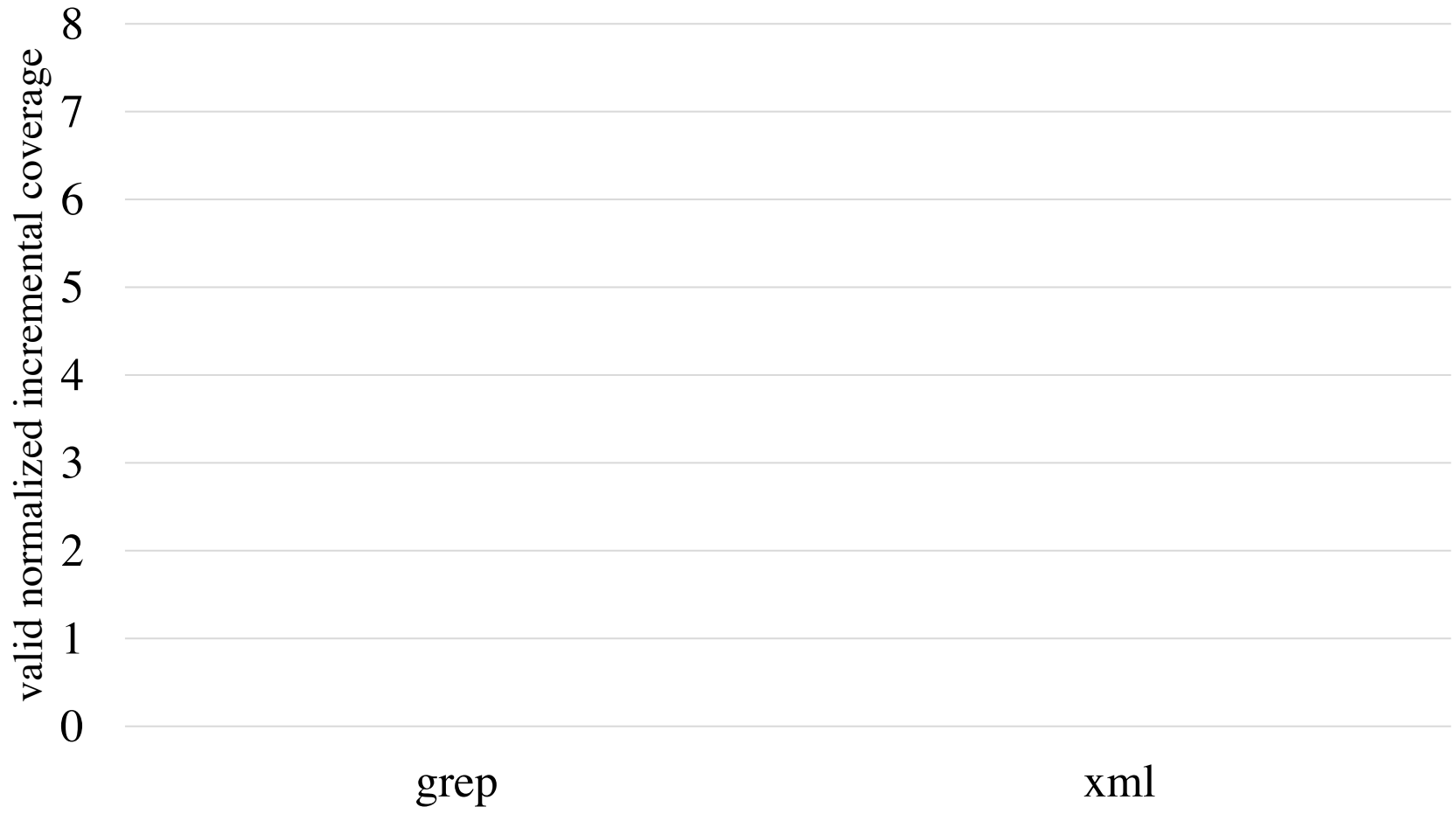
Evaluation: Fuzz Testing



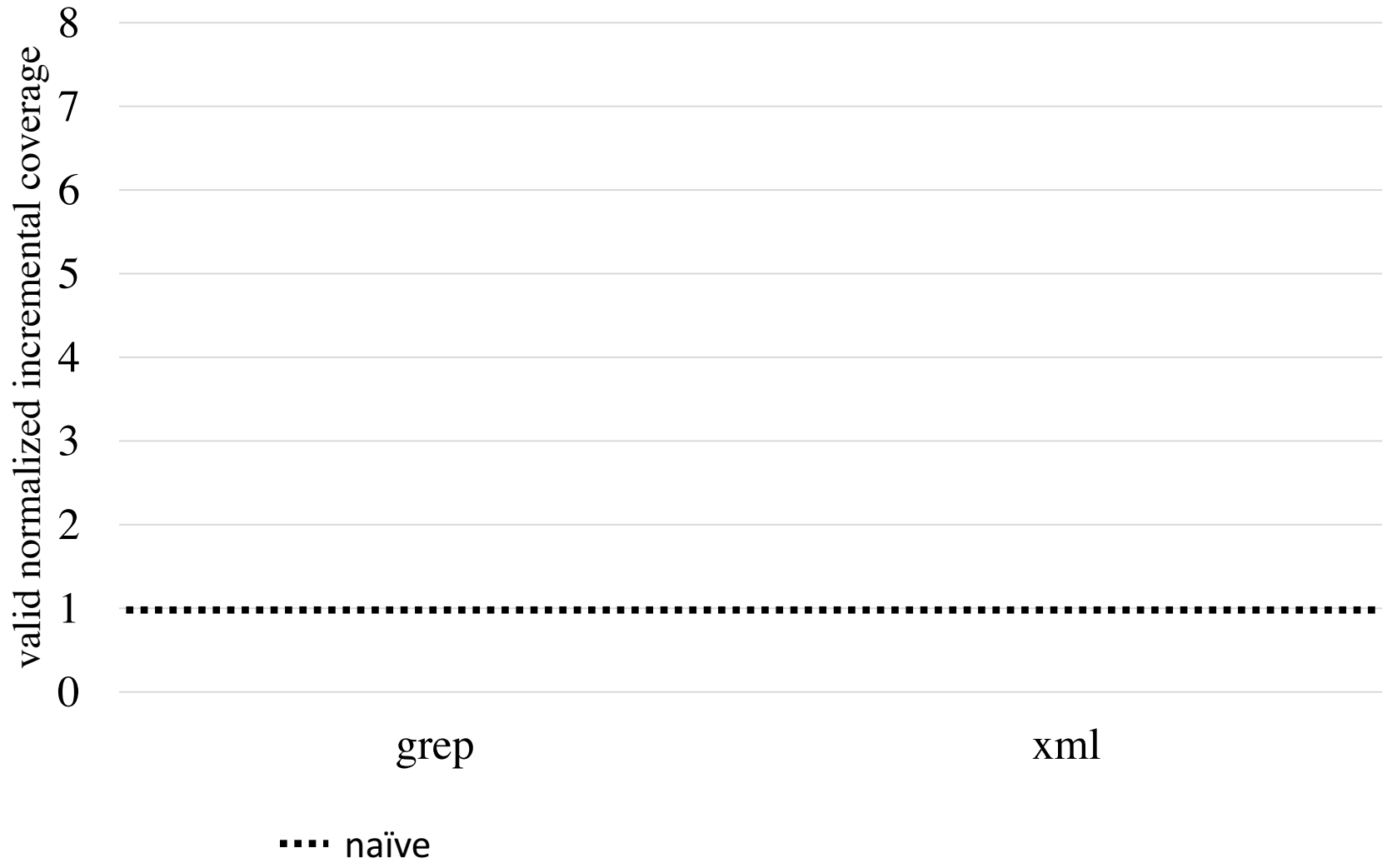
Evaluation: Fuzz Testing



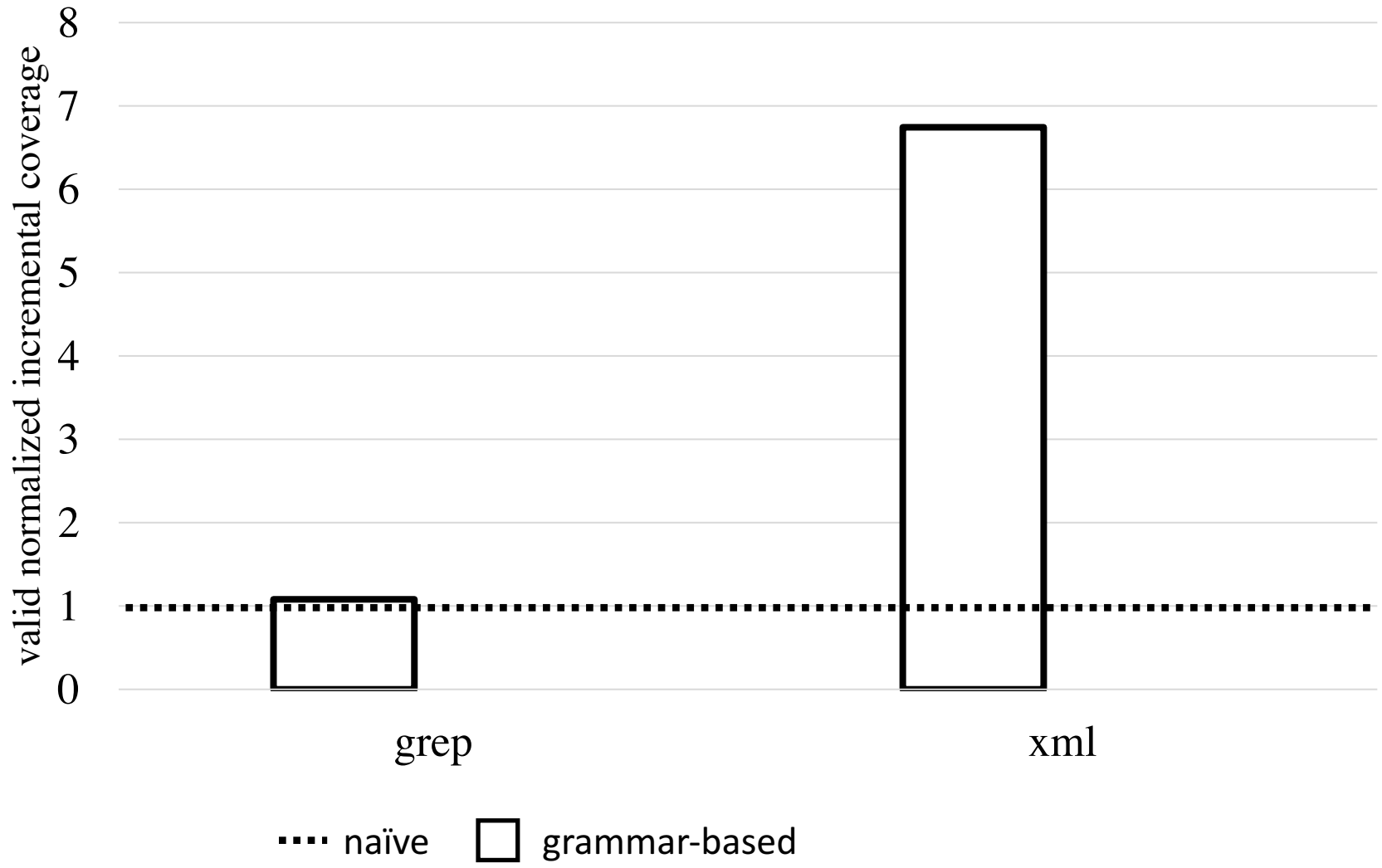
Evaluation: Fuzz Testing



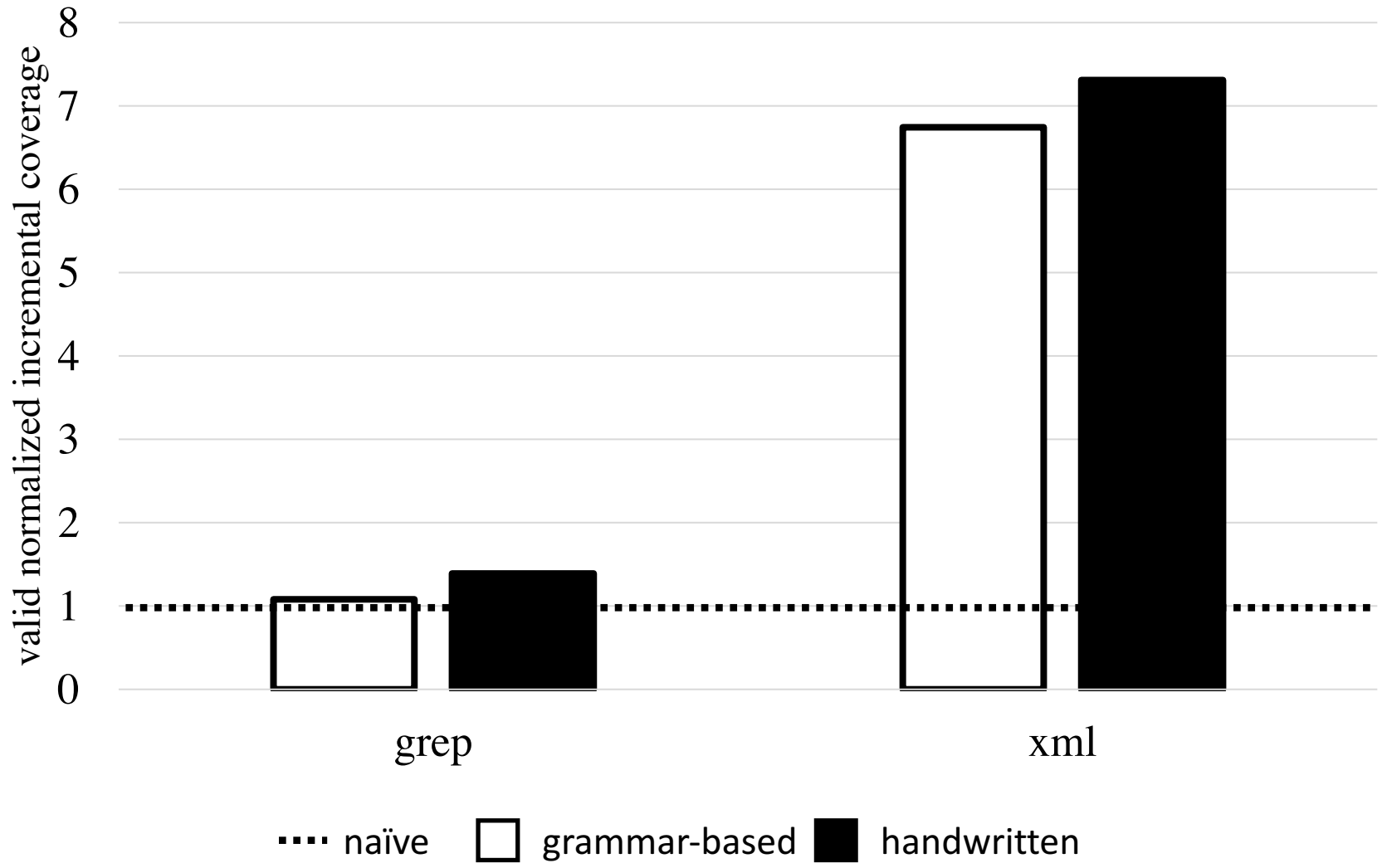
Evaluation: Fuzz Testing



Evaluation: Fuzz Testing



Evaluation: Fuzz Testing



Learn program properties from input-output examples

- “Extreme” form of active learning

References

- B. Livshits, A. Nori, S. Rajamani, A. Banerjee, Merlin: Specification Inference for Explicit Information Flow Problems , in PLDI, 2009.
- M. Höschele, A. Zeller, Mining input grammars from dynamic taints. In ASE, 2016.
- B. P. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of unix utilities. Communications of the ACM, 33(12):32–44, 1990.
- V. Ganesh, T. Leek, and M. Rinard. Taint-based directed whitebox fuzzing. In Proceedings of the 31st International Conference on Software Engineering, pages 474–484. IEEE Computer Society, 2009.
- P. Godefroid, A. Kiezun, and M. Y. Levin. Grammar-based whitebox fuzzing. In ACM Sigplan Notices, volume 43, pages 206–215. ACM, 2008.
- M. Zalewski. American fuzzy lop, 2015. http://lcamtuf.coredump.cx/afl.
- D. Angluin. Learning regular sets from queries and counterexamples. Information and computation, 75(2):87–106, 1987.
- X. Yang, Y. Chen, E. Eide, and J. Regehr. Finding and understanding bugs in c compilers. In ACM SIGPLAN Notices, volume 46, pages 283–294. ACM, 2011.

Questions?

Backup Slides

Phase One: Regular Expressions

Phase One: Regular Expressions

Repetitions: $S\alpha T$

Phase One: Regular Expressions

Repetitions: $S\alpha T \Rightarrow S\alpha^{*}T$

Phase One: Regular Expressions

Repetitions: $S\alpha T \Rightarrow S\alpha^{*}T$

Alternations: $S\alpha\beta T$

Phase One: Regular Expressions

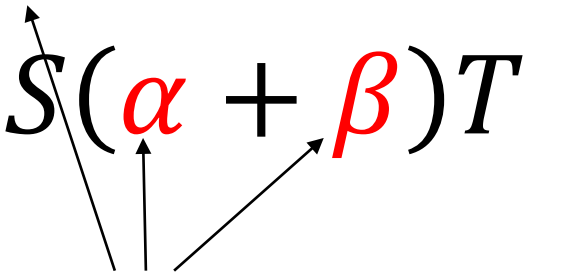
Repetitions: $S\alpha T \Rightarrow S\alpha^* T$

Alternations: $S\alpha\beta T \Rightarrow S(\alpha + \beta)T$

Phase One: Regular Expressions

Repetitions: $S\alpha T \Rightarrow S\alpha^* T$

Alternations: $S\alpha\beta T \Rightarrow S(\alpha + \beta)T$



recursively generalize

Phase Two: Merging CFG Nonterminals

Phase Two: Merging CFG Nonterminals

$$A \rightarrow \ldots A \ldots B \ldots$$
$$B \rightarrow \ldots A \ldots C \ldots$$

Phase Two: Merging CFG Nonterminals

$$\boxed{\begin{array}{l} A \rightarrow \ \ldots A \ldots B \ldots \\ B \rightarrow \ \ldots A \ldots C \ldots \end{array}} \Rightarrow \boxed{\begin{array}{l} A \rightarrow \ \ldots A \ldots A \ldots \\ A \rightarrow \ \ldots A \ldots C \ldots \end{array}}$$

Guarantees

Guarantees

Theorem 1: The search space includes all regular languages

Guarantees

Theorem 1: The search space includes all regular languages

Theorem 2: The search space includes all "generalized matching parentheses" languages

Generalized Matching Parentheses

$$T \to R(T_1 + \cdots + T_k)^* R'$$

Guarantees

Guarantees

Need to use the “right” candidate ordering

Guarantees

Need to use the “right” candidate ordering
We use an intuitive heuristic that works empirically