Z3str4: A Multi-Armed String Solver

Federico Mora[1], Murphy Berzish[2], Mitja Kulczynski[3], Dirk Nowotka[3], and Vijay Ganesh[2]

[1] University of California, Berkeley

[2] University of Waterloo

[3] Kiel University

Wednesday, November 24, 2021

24th International Symposium Organized by Formal Methods Europe

String Solvers and Queries by Example

How Can We Automatically Reason About This Code?

```
def check_password(x: str) -> bool:
    if re.search(r"\d", x) is None:
        return False # no number
    if "12345" in x:
        return False # contains "123456”
    if len(x) < 5:
        return False # too short
    if x[:-len(x)//2] * 2 == x:
        return False # same string twice
    return True
```

String Solvers and Queries by Example

How Can We Automatically Reason About This Code?

```
def check_password(x: str) -> bool:
    if re.search(r"\d", x) is None:
        return False # no number
    if "12345" in x:
        return False # contains "123456”
    if len(x) < 5:
        return False # too short
    if x[:-len(x)//2] * 2 == x:
        return False # same string twice
    return True
```

String Solvers and Queries by Example

How Can We Automatically Reason About This Code?

```
def check_password(x: str) -> bool:
    if re.search(r"\d", x) is None:
        return False # no number
    if "12345" in x:
        return False # contains "123456"
    if len(x) < 5:
        return False # too short
    if x[:-len(x)//2] * 2 == x:
        return False # same string twice
    return True
```

Model

$$x \in .^*[\backslash d]^+.^*$$

$$\text{“12345”} \in x$$

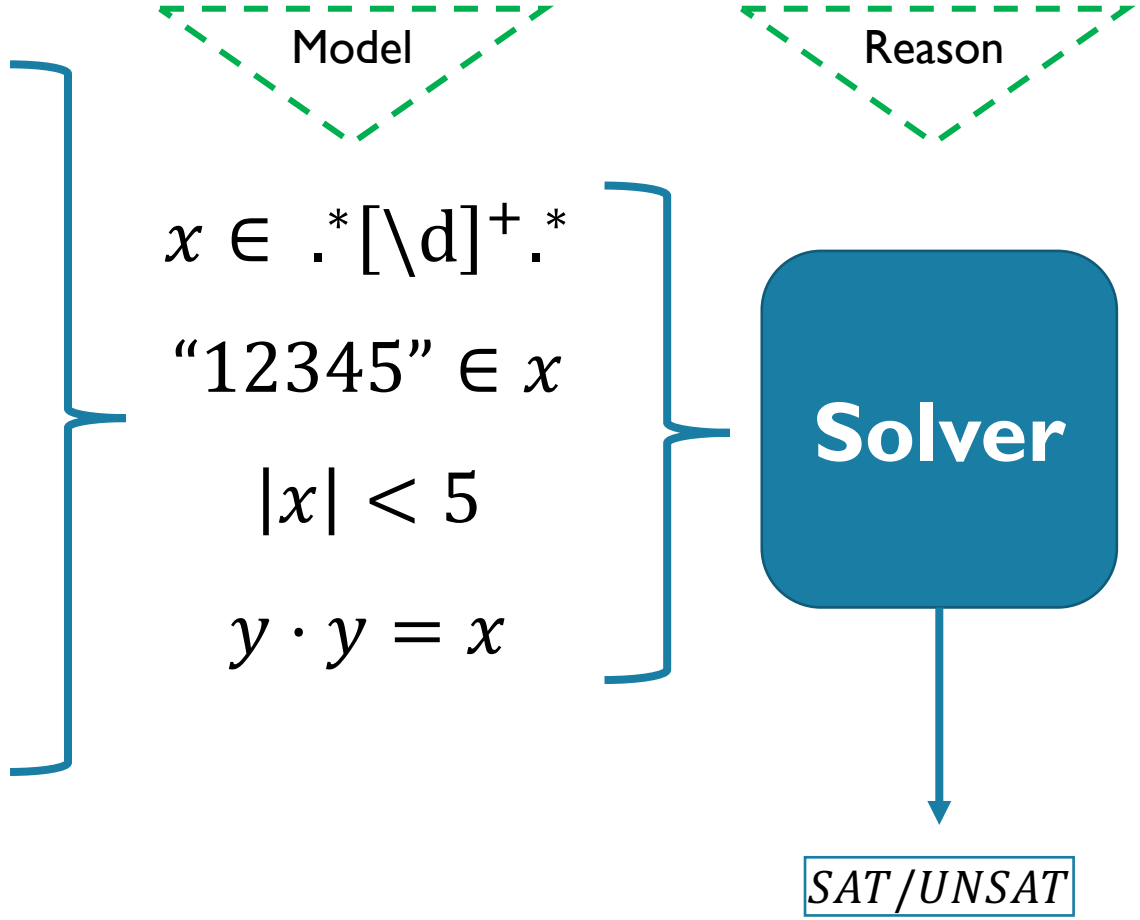
$$|x| < 5$$

$$y \cdot y = x$$

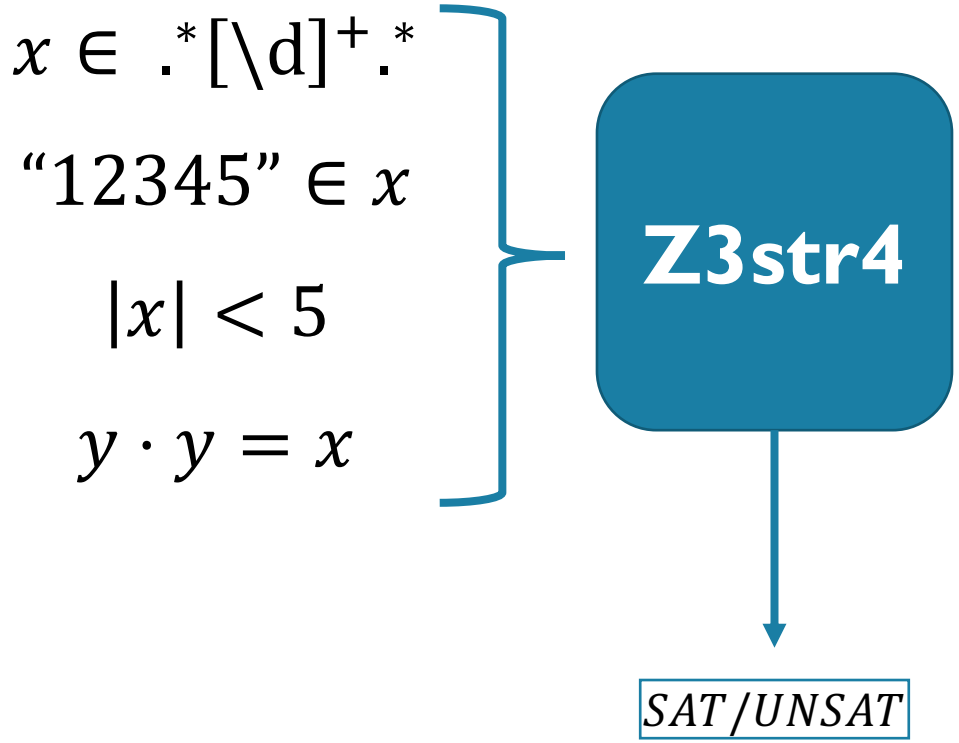
String Solvers and Queries by Example

How Can We Automatically Reason About This Code?

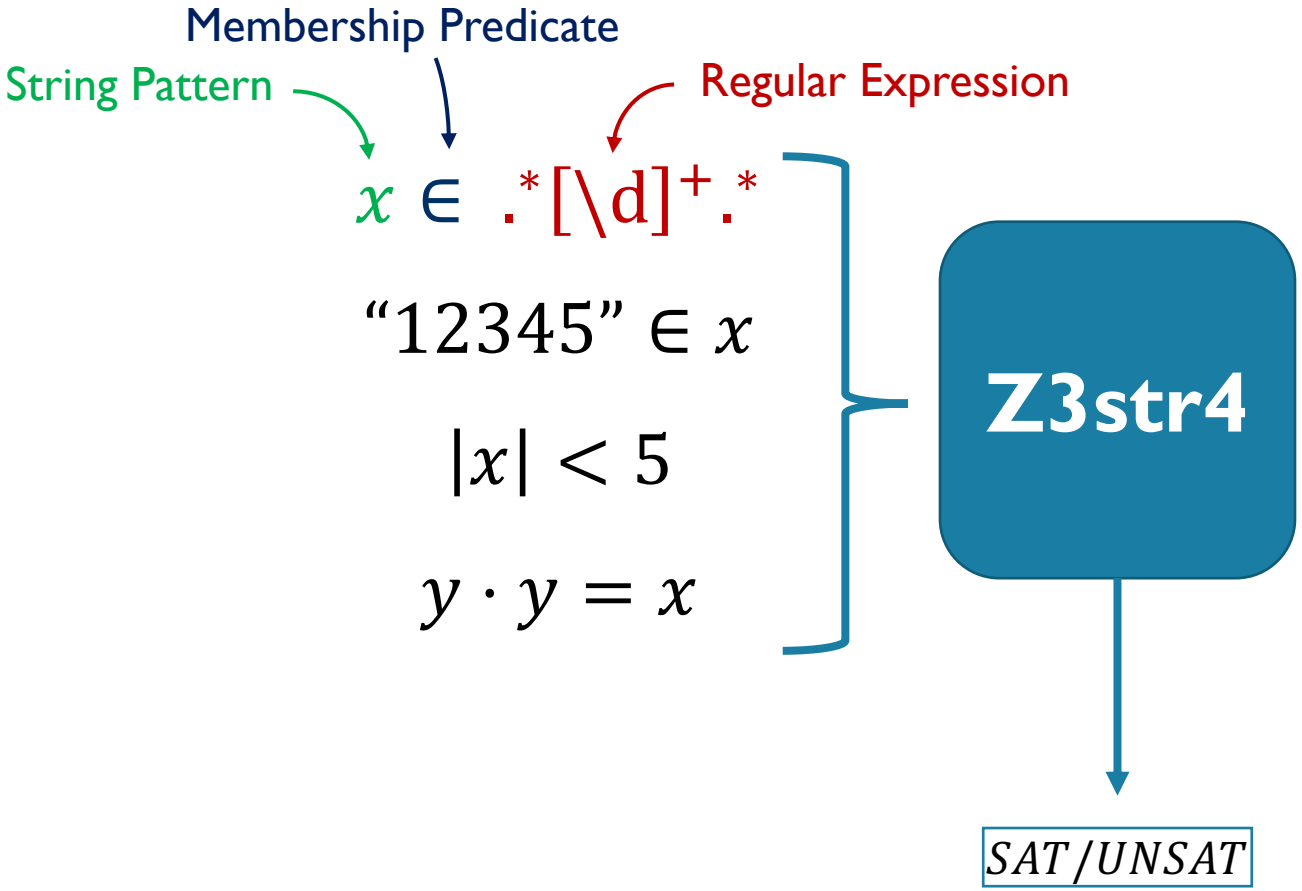
```python
def check_password(x: str) -> bool:
    if re.search(r"\d", x) is None:
        return False # no number
    if "12345" in x:
        return False # contains "123456"
    if len(x) < 5:
        return False # too short
    if x[:-len(x)//2] * 2 == x:
        return False # same string twice
    return True
```



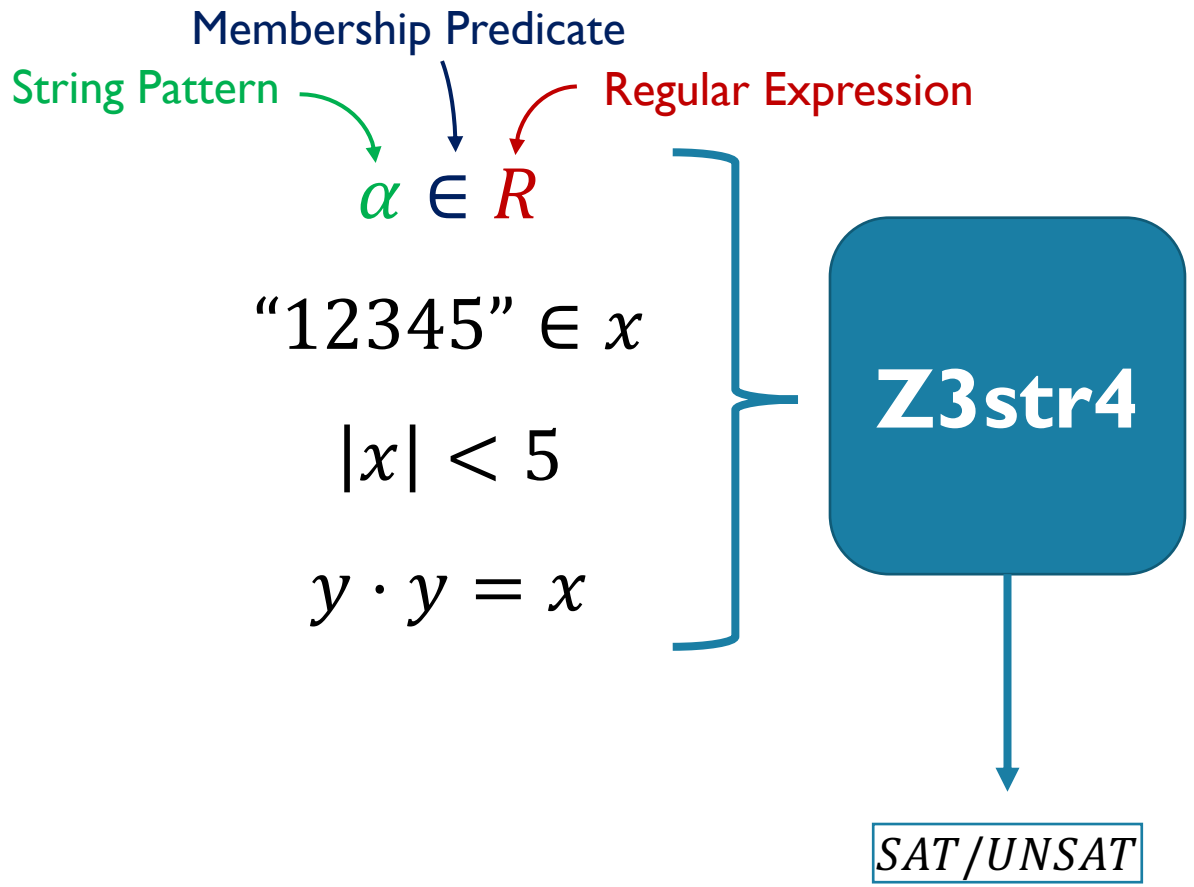
Goal of Z3str4: Solve as many queries as possible, as fast as possible.



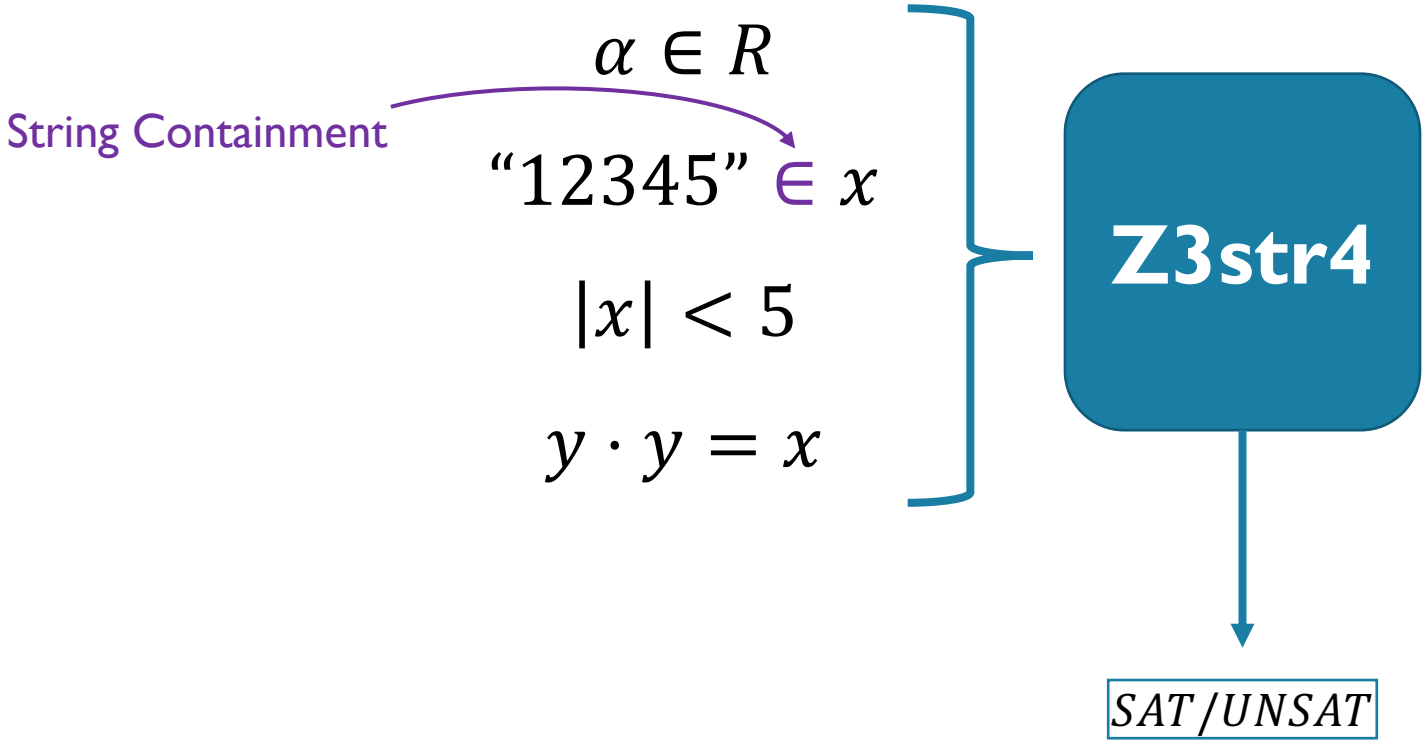
Goal of Z3str4: Solve as many queries as possible, as fast as possible.



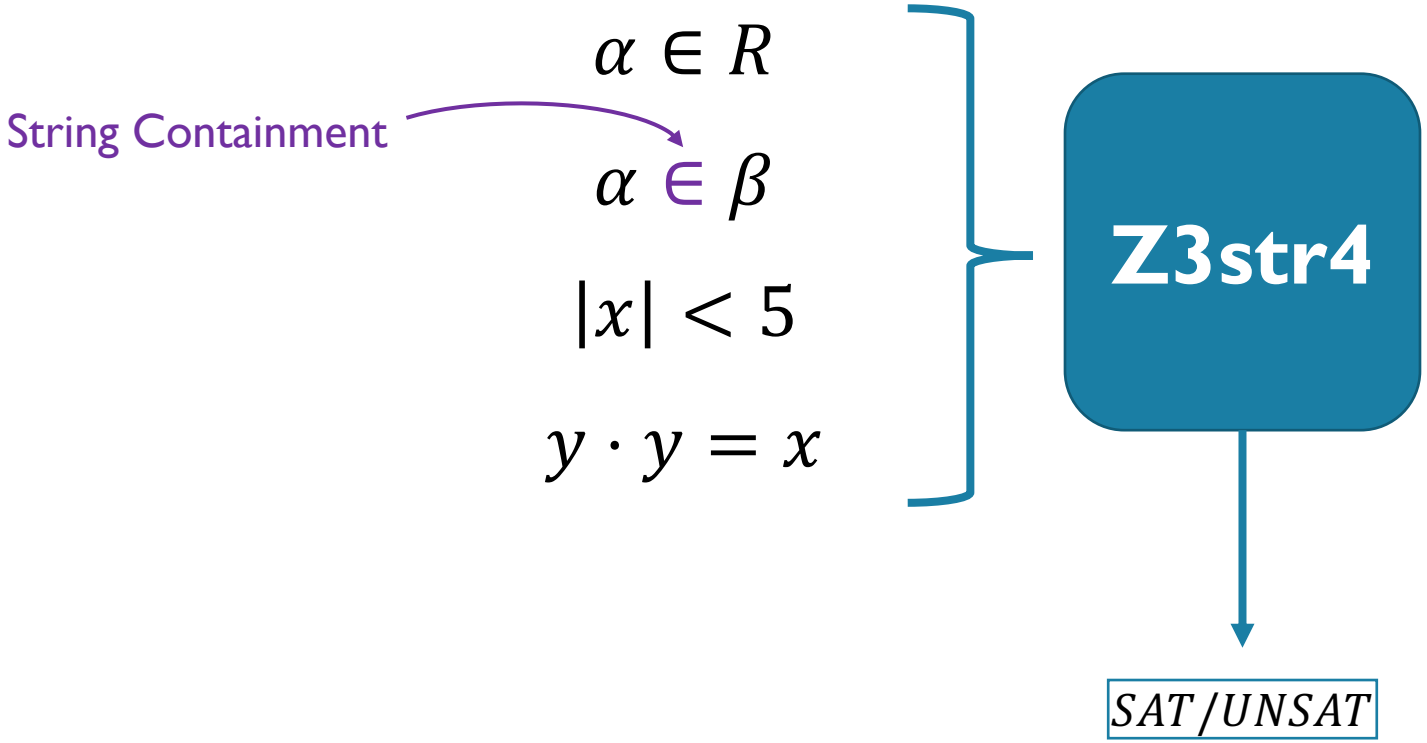
Goal of Z3str4: Solve as many queries as possible, as fast as possible.



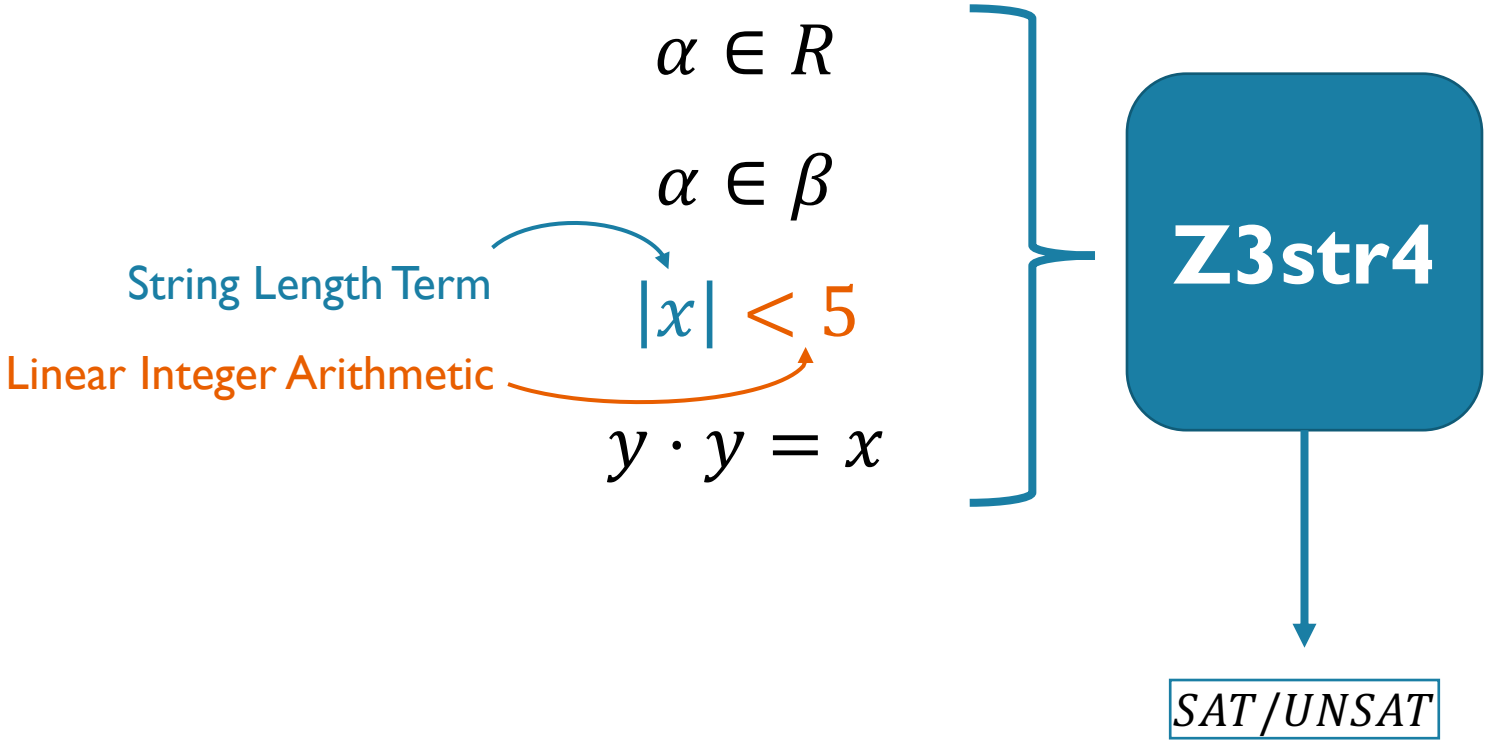
Goal of Z3str4: Solve as many queries as possible, as fast as possible.



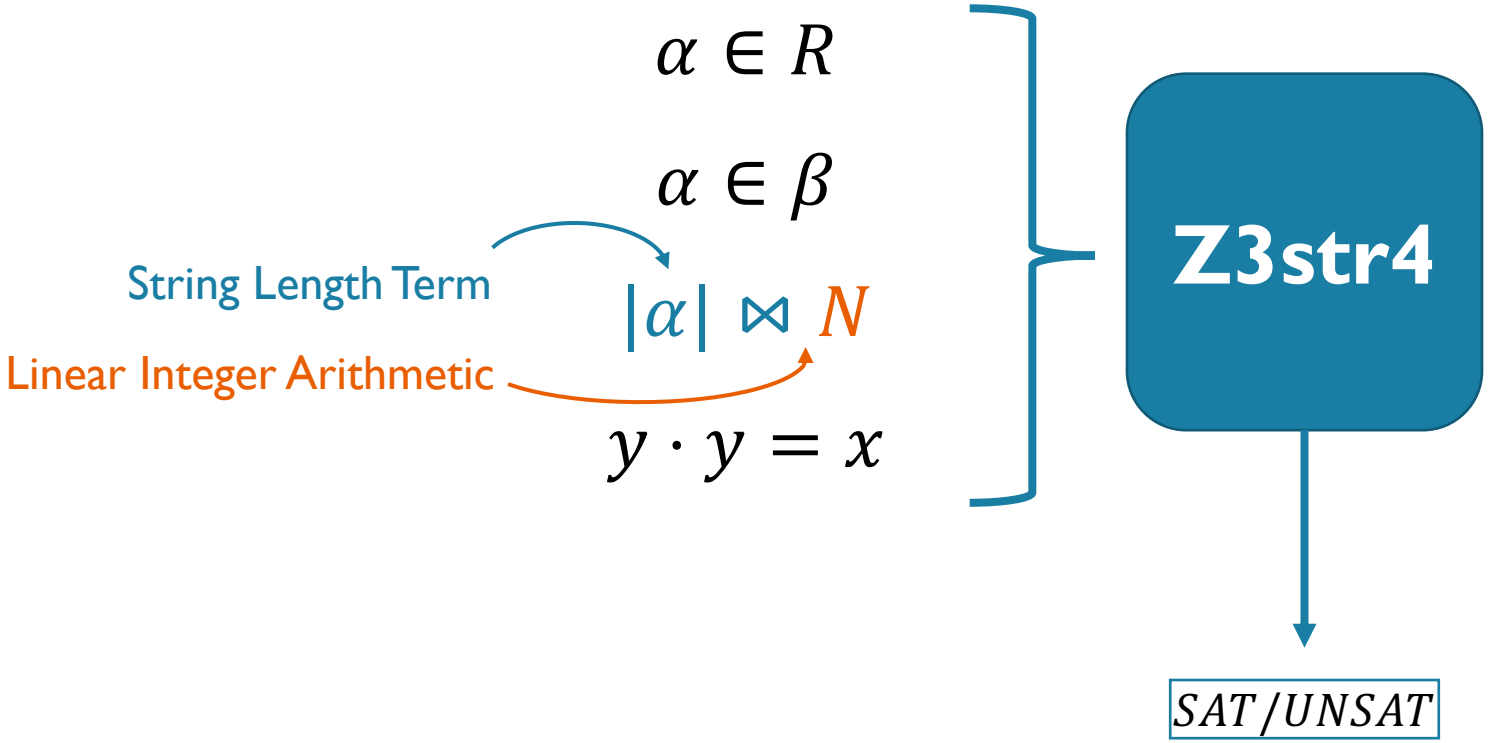
Goal of Z3str4: Solve as many queries as possible, as fast as possible.



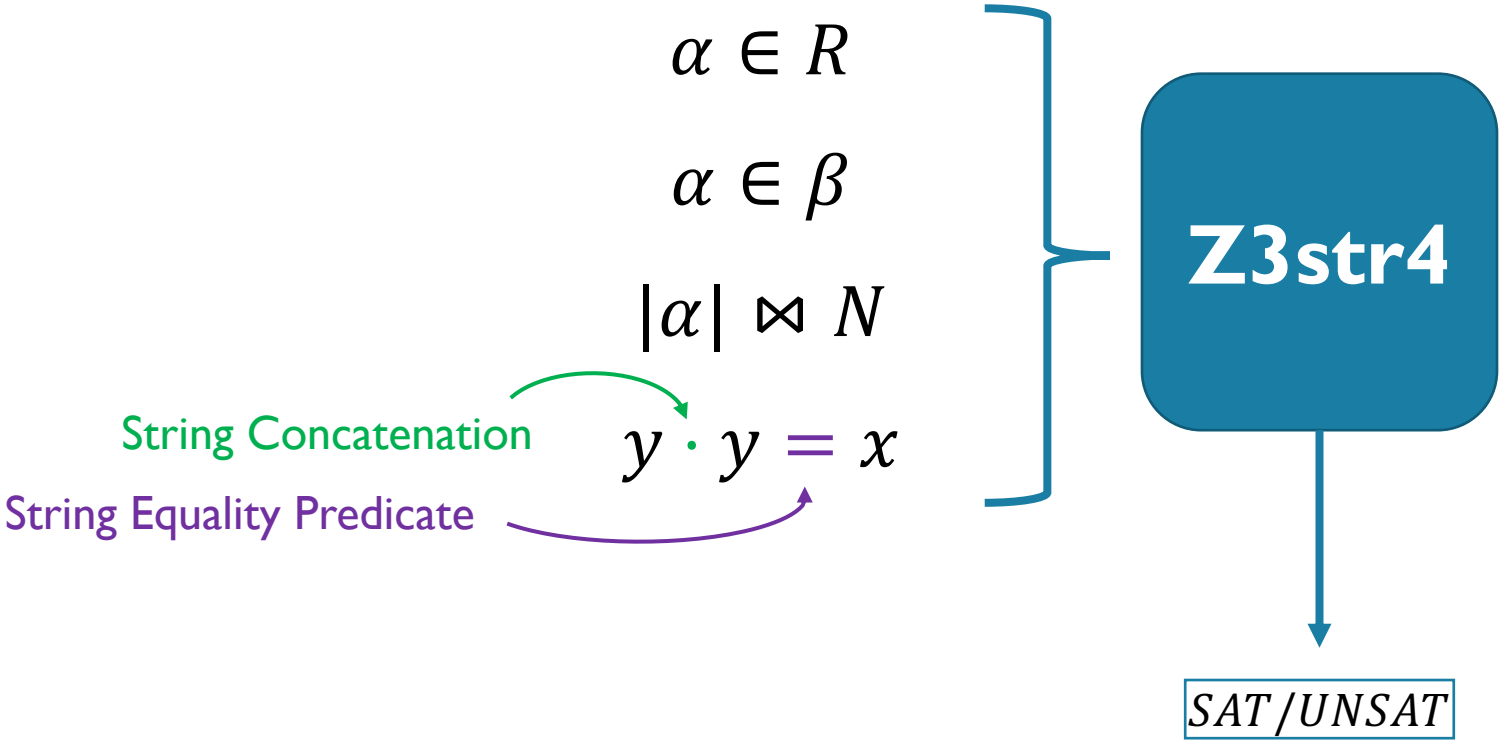
Goal of Z3str4: Solve as many queries as possible, as fast as possible.



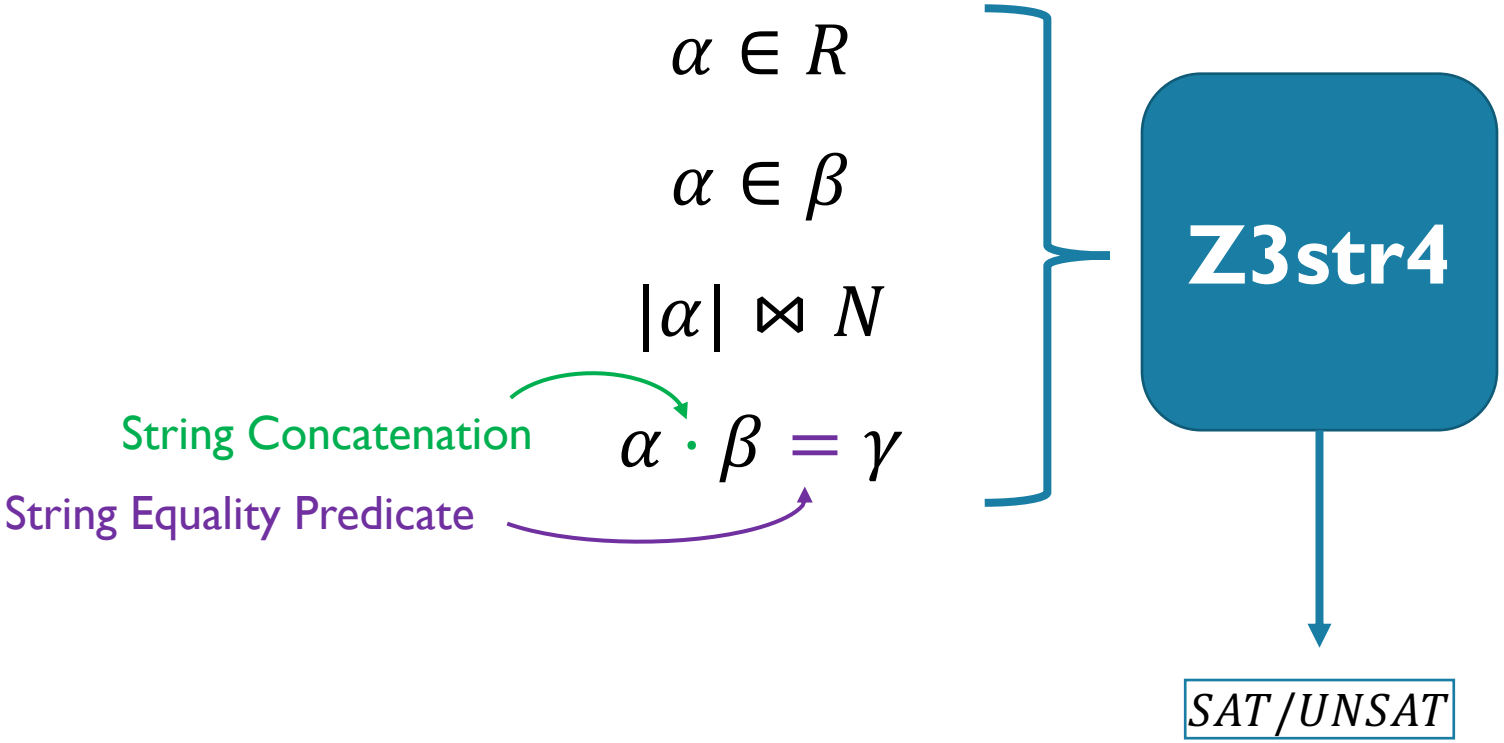
Goal of Z3str4: Solve as many queries as possible, as fast as possible.



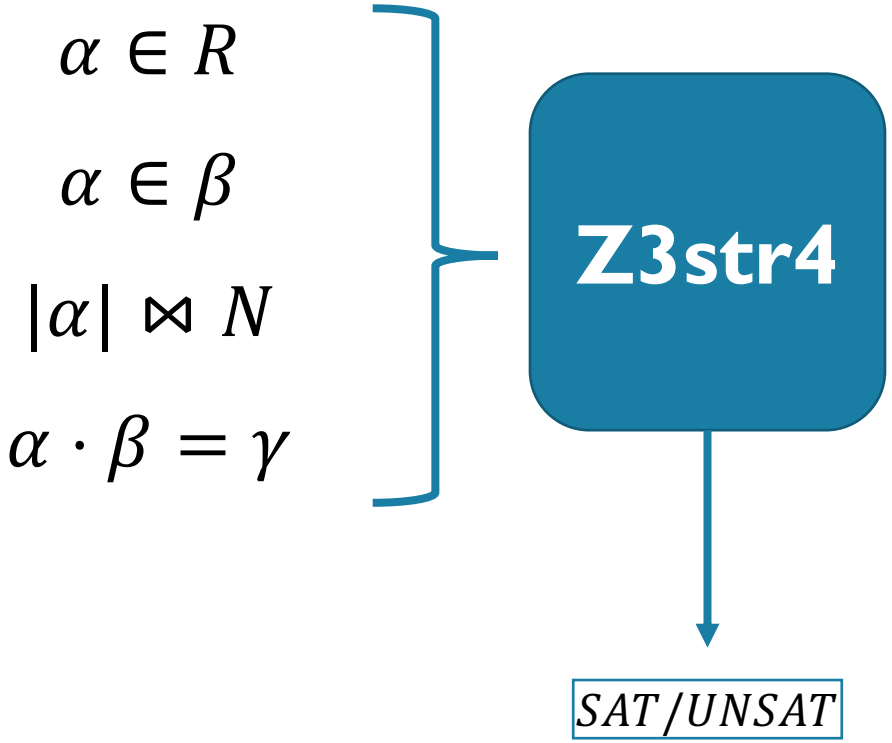
Goal of Z3str4: Solve as many queries as possible, as fast as possible.



Goal of Z3str4: Solve as many queries as possible, as fast as possible.

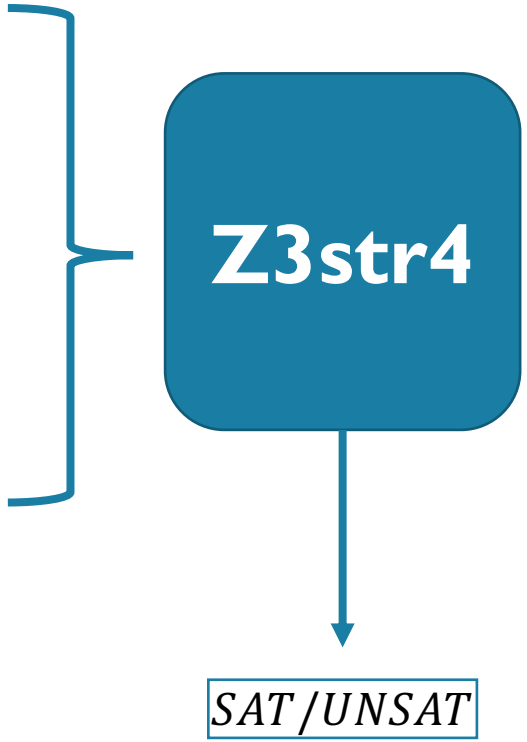


Goal of Z3str4: Solve as many queries as possible, as fast as possible.

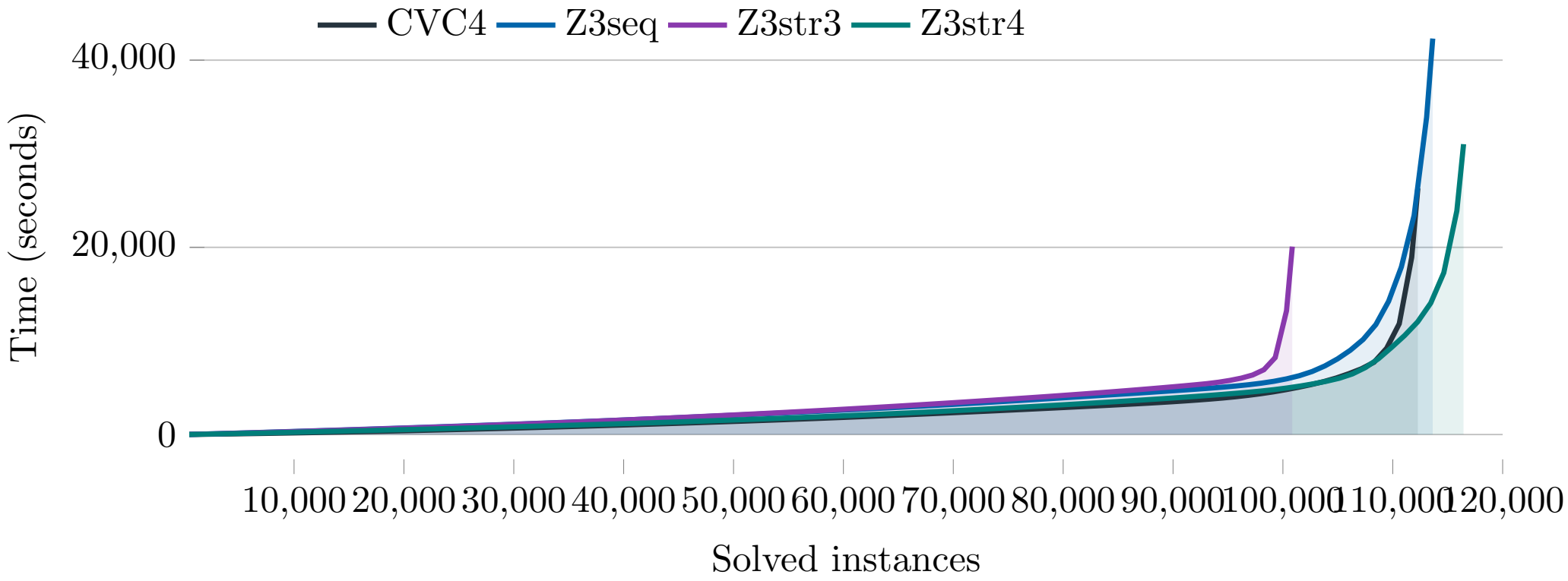


Goal of Z3str4: Solve as many queries as possible, as fast as possible.

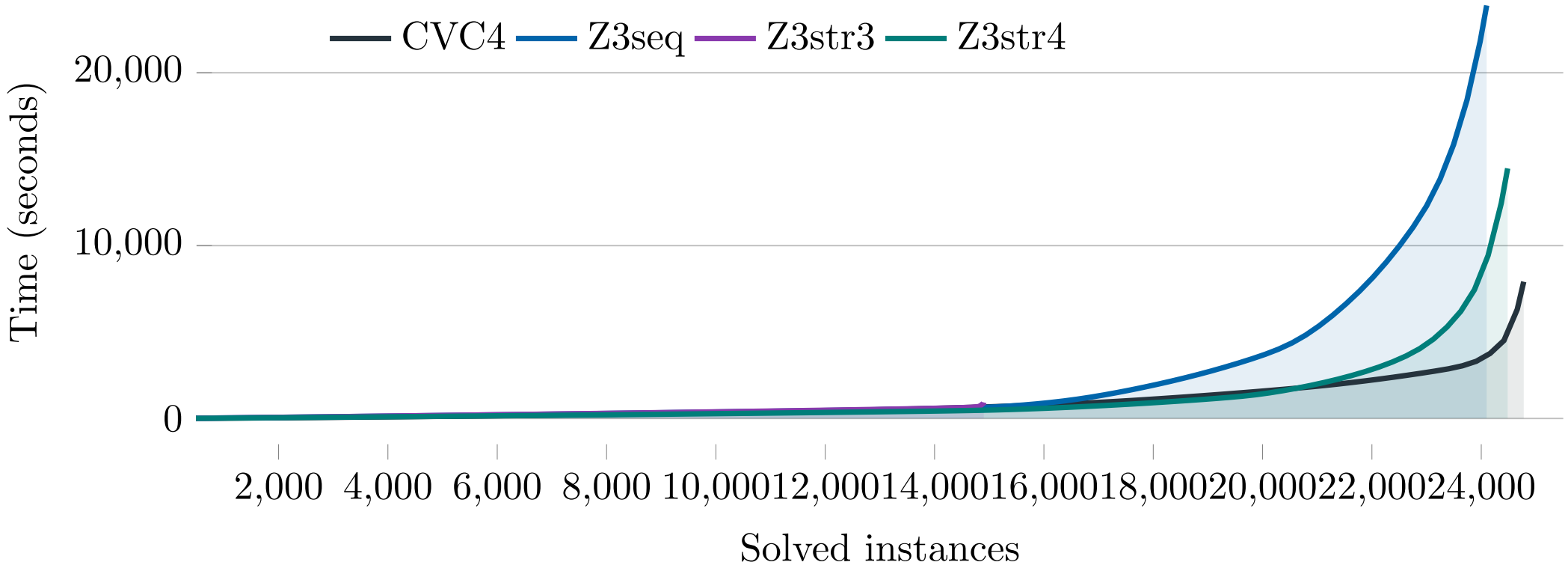
$$
\begin{array}{lll}
F & ::= & Atom \mid F \wedge F \mid F \vee F \mid \neg F \\
Atom & ::= & t_{str} = t_{str} \mid A_{int} \mid A_{ext} \mid A_{re} \\
A_{re} & ::= & t_{str} \in RE \\
A_{int} & ::= & t_{int} = t_{int} \mid t_{int} < t_{int} \\
A_{ext} & ::= & contains(t_{str}, t_{str}) \mid prefix(t_{str}, t_{str}) \mid suffix(t_{str}, t_{str}) \\
t_{int} & ::= & m \mid v \mid len(t_{str}) \mid t_{int} + t_{int} \mid m \cdot t_{int} \mid indexof(t_{str}, t_{str}, t_{int}) \mid \\
 & & str.to\_int(t_{str}) \text{ where } m \in Con_{int} \ \& \ v \in Var_{int} \\
t_{str} & ::= & s \mid v \mid t_{str} \cdot t_{str} \mid str.from\_int(t_{int}) \mid replace(t_{str}, t_{str}, t_{str}) \mid \\
 & & charAt(t_{str}, t_{int}) \mid substr(t_{str}, t_{int}, t_{int}) \text{ where } s \in Con_{str} \ \& \ v \in Var_{str}
\end{array}
$$



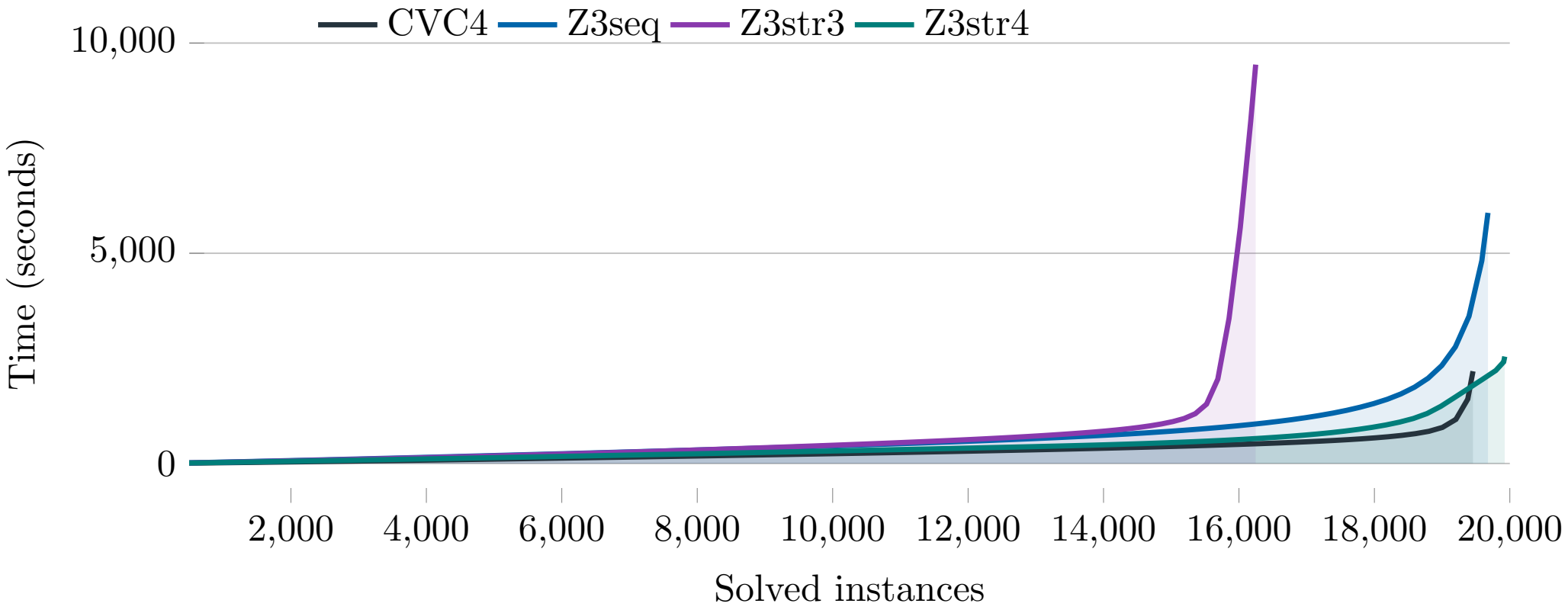
Results Overview



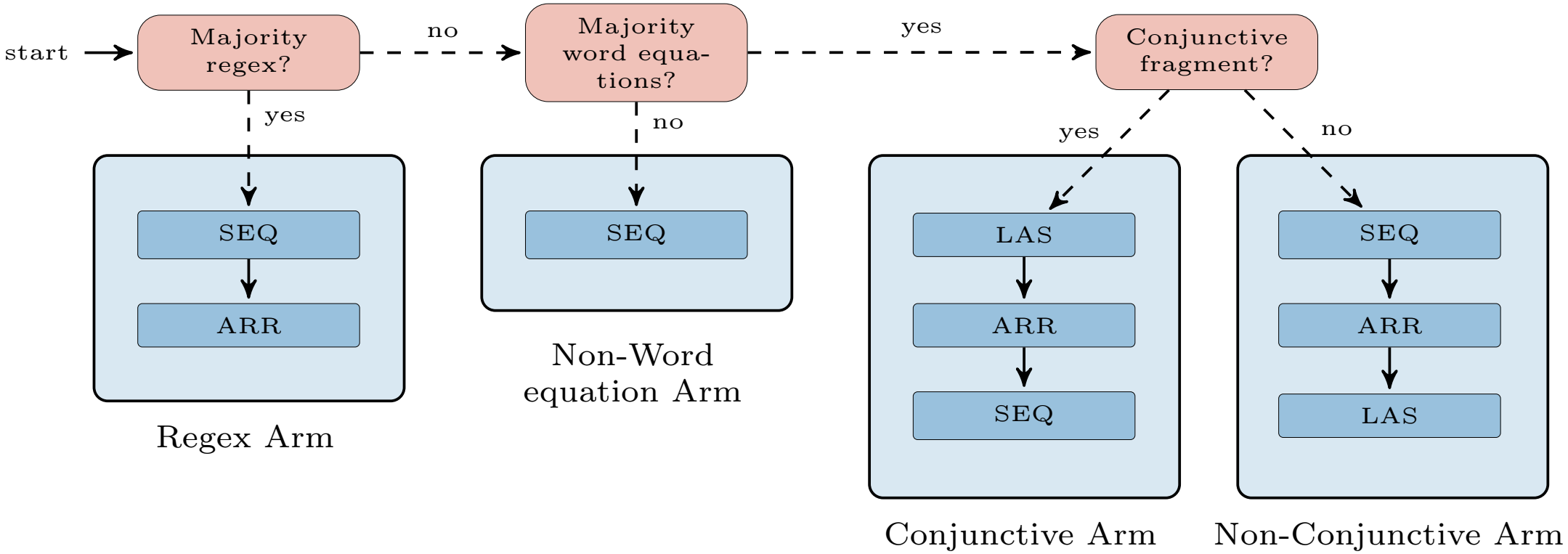
Results on PyEx



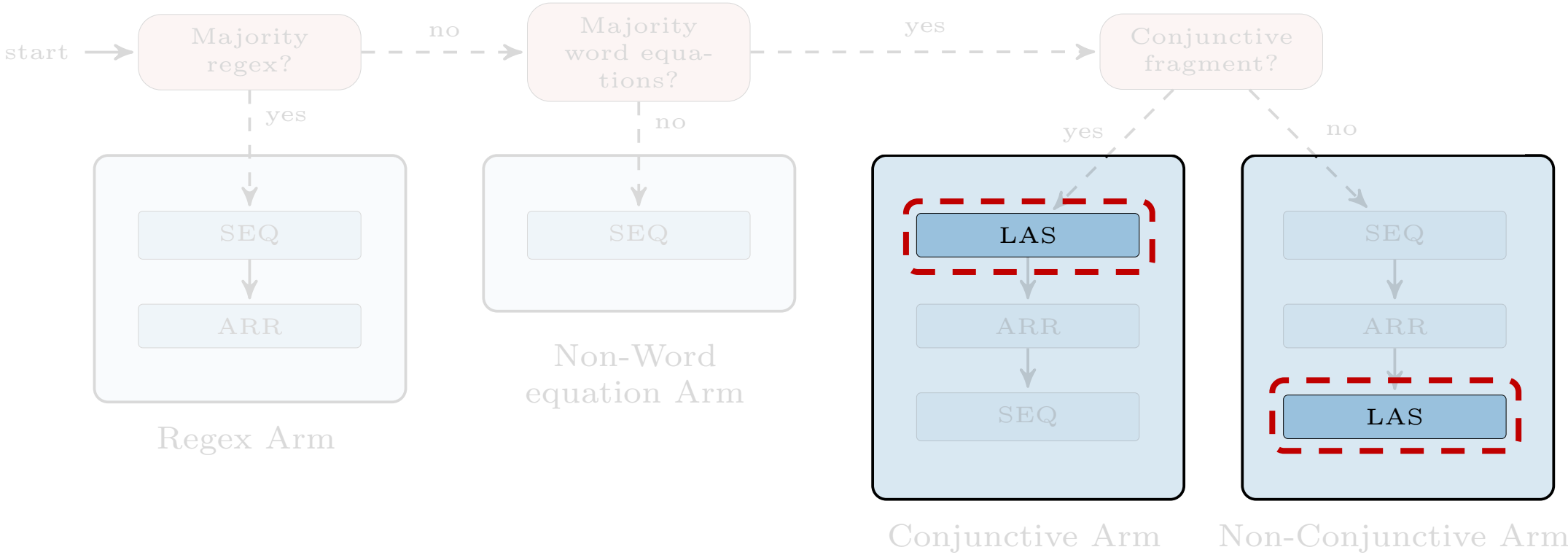
Results on Automark25



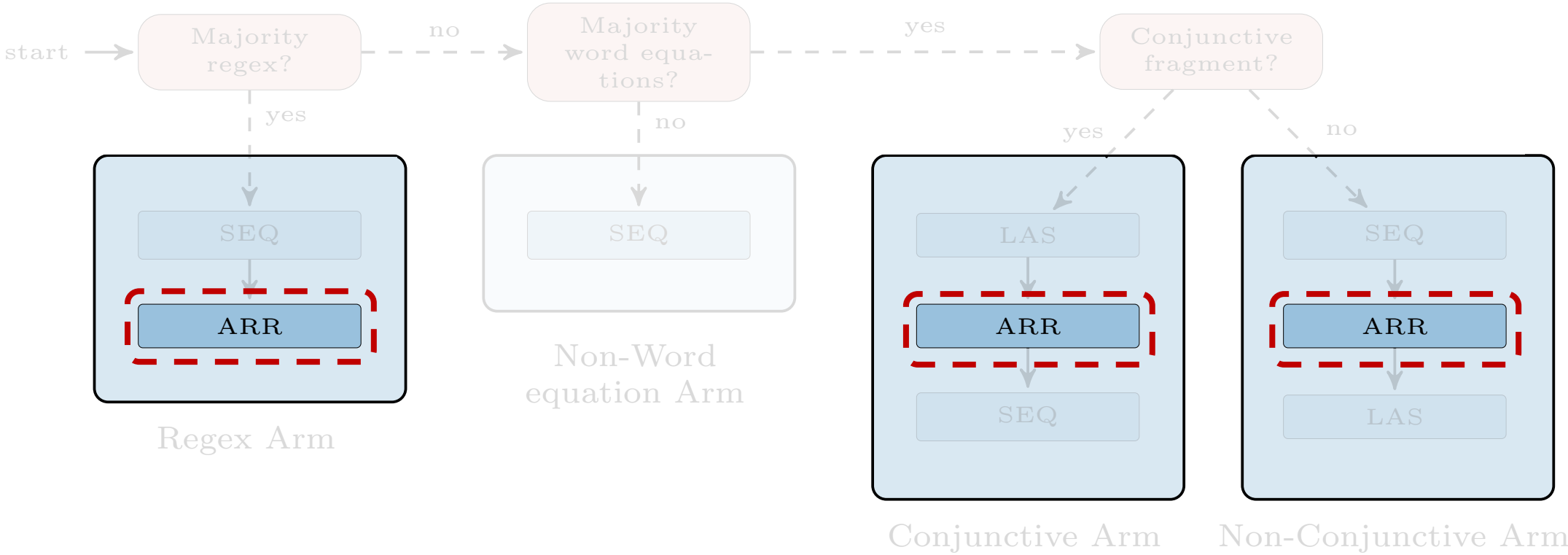
Contributions Overview



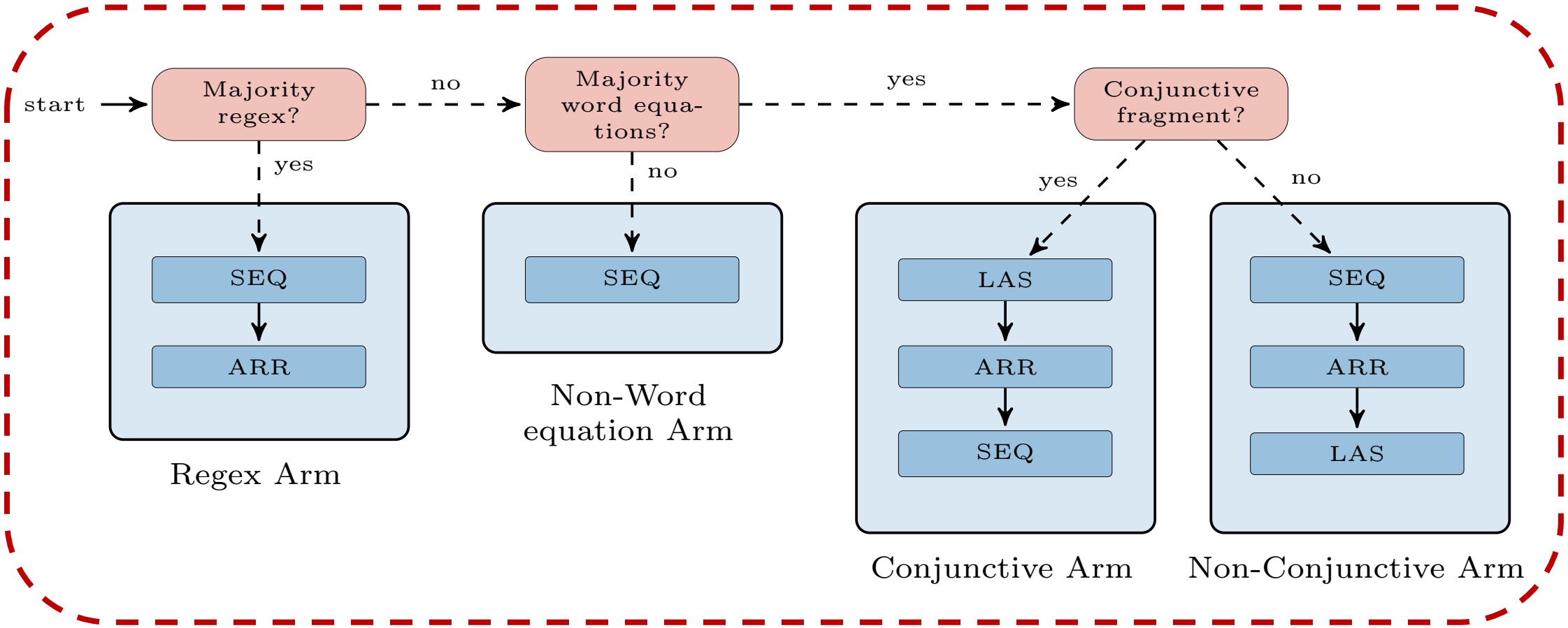
Contribution 1: Length Abstraction Solver



Contribution 2: Extended Arrangement Solver

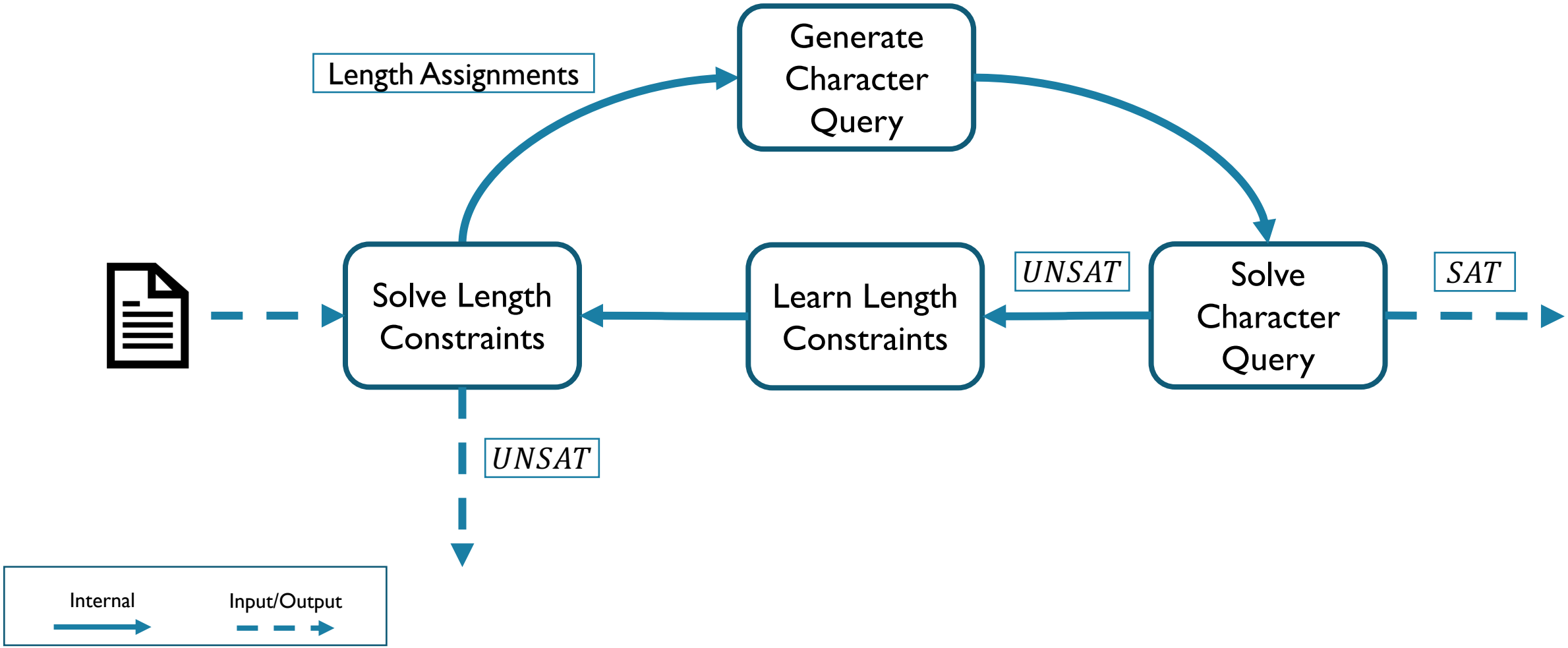


Contribution 3: Selection Architecture

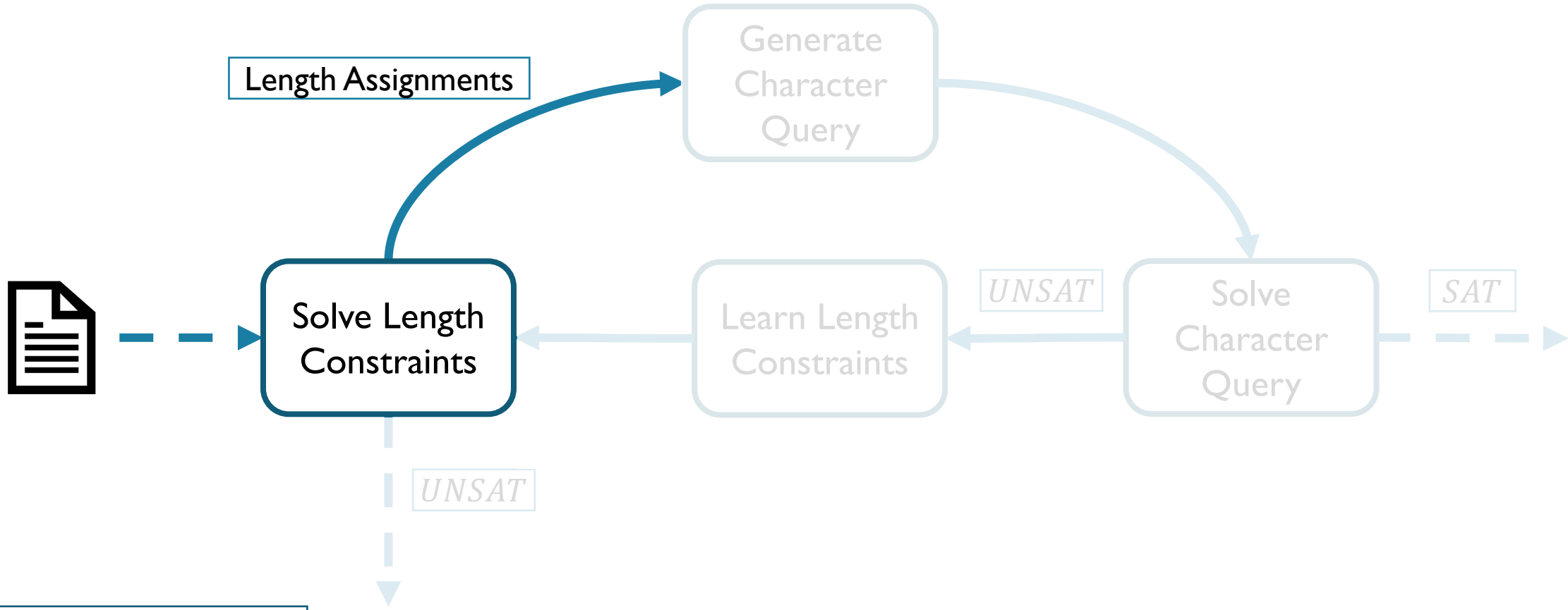


Length Abstraction Solver

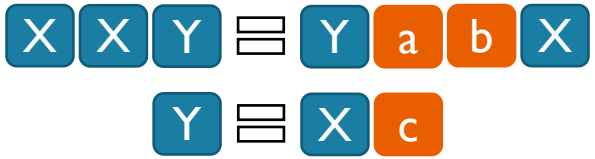
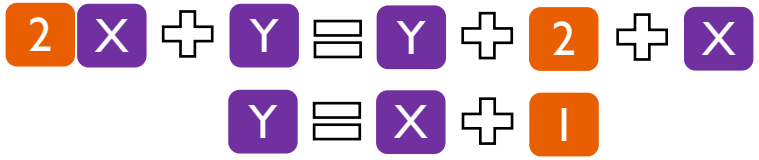
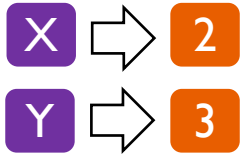
Length Abstraction Solver: Refinement Loop

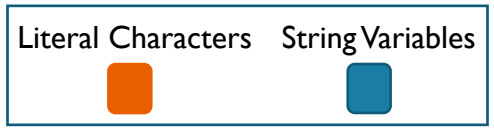


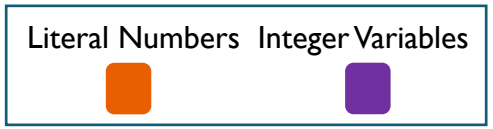
Length Abstraction Solver: Solve Length Constraints

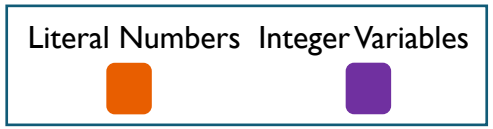


Length Abstraction Solver: Solve Length Constraints

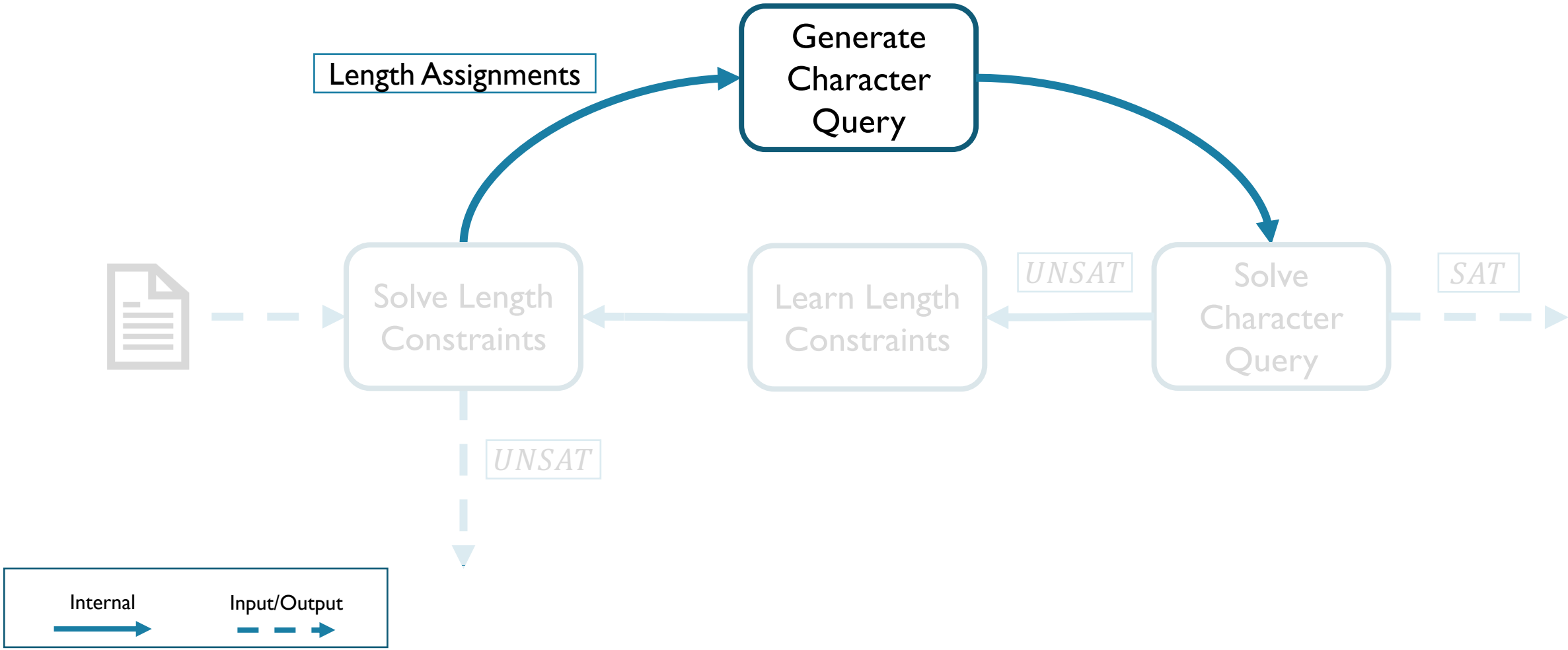
Input String Query	Implied Length Constraints	Length Solution
		





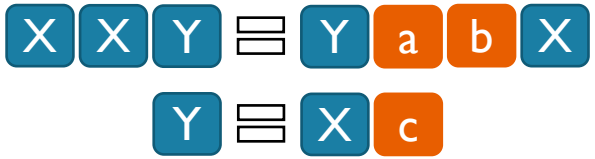


Length Abstraction Solver: Generate Character Query

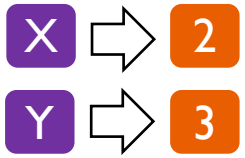


Length Abstraction Solver: Generate Character Query

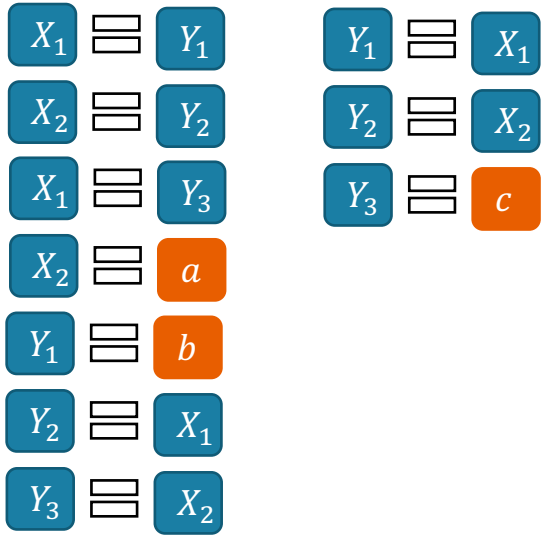
Input String Query

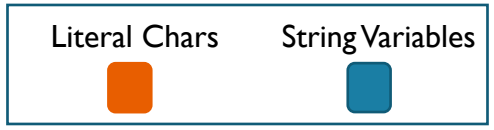


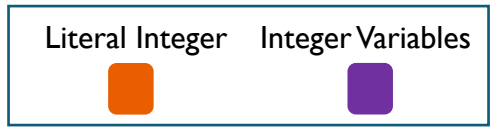
Length Solution

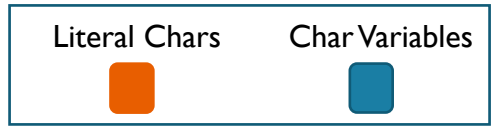


Character Query

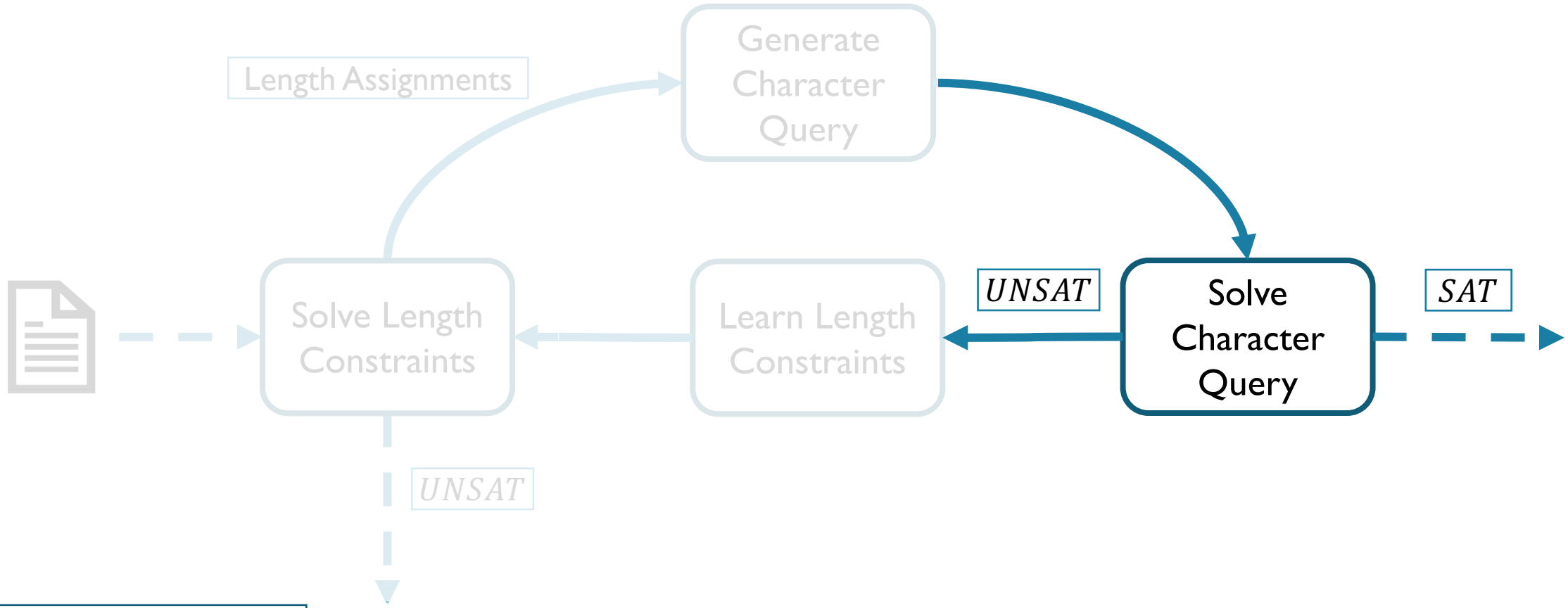






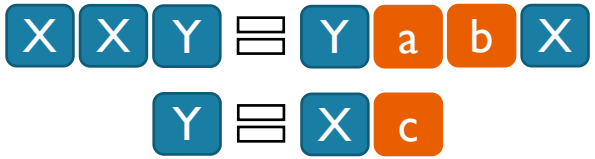


Length Abstraction Solver: Solve Character Query

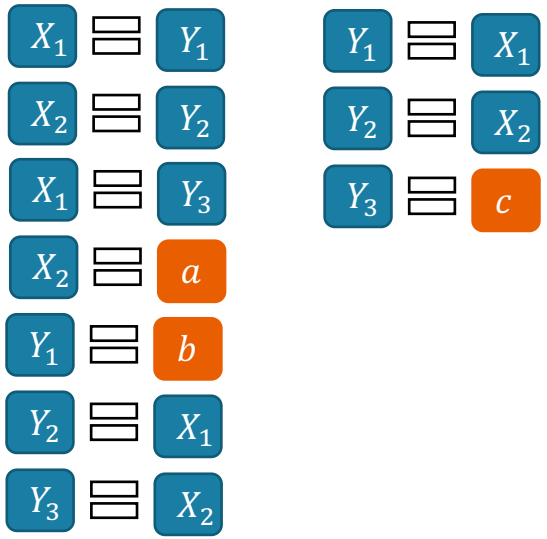


Length Abstraction Solver: Solve Character Query

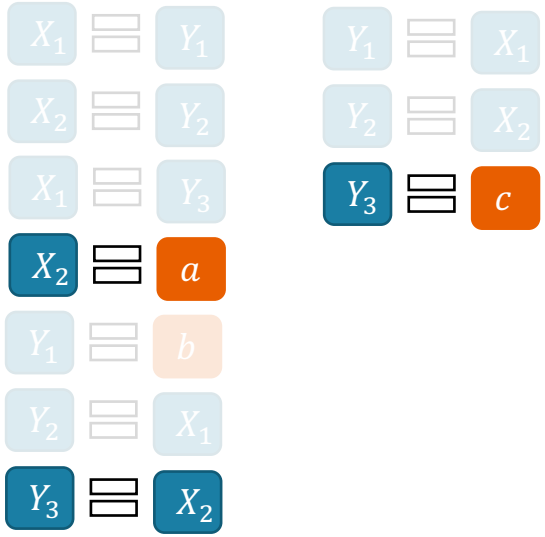
Input String Query

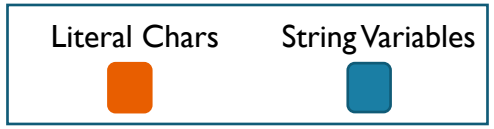


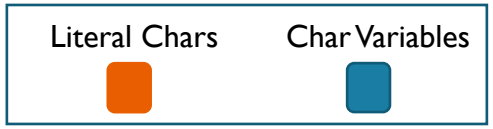
Character Query

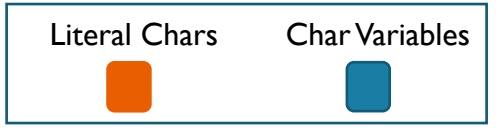


UNSAT Core



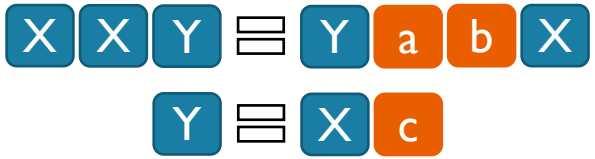






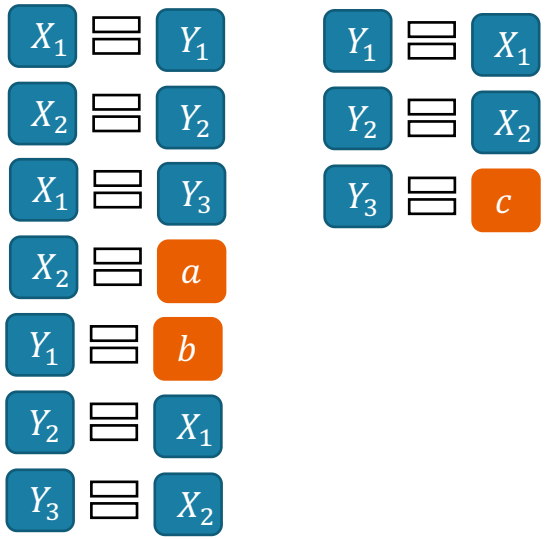
Length Abstraction Solver: Solve Character Query

Input String Query



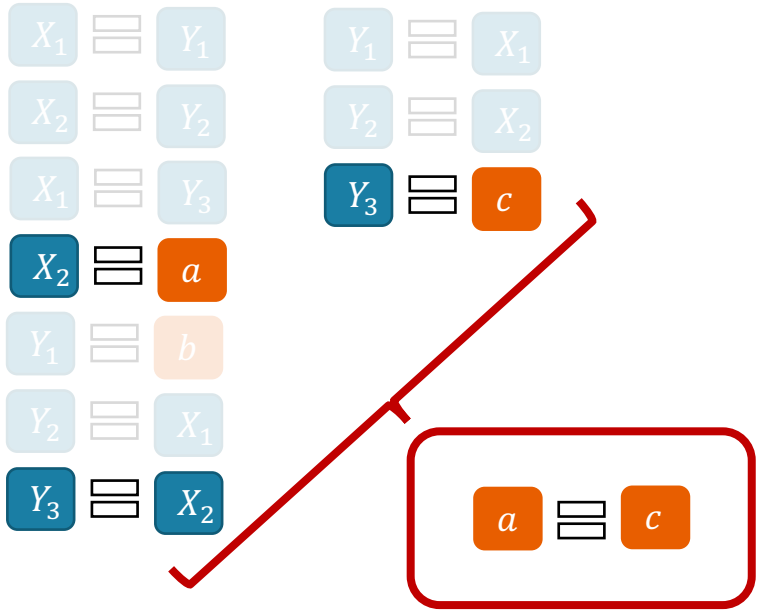
Literal Chars: ■ String Variables: ■

Character Query



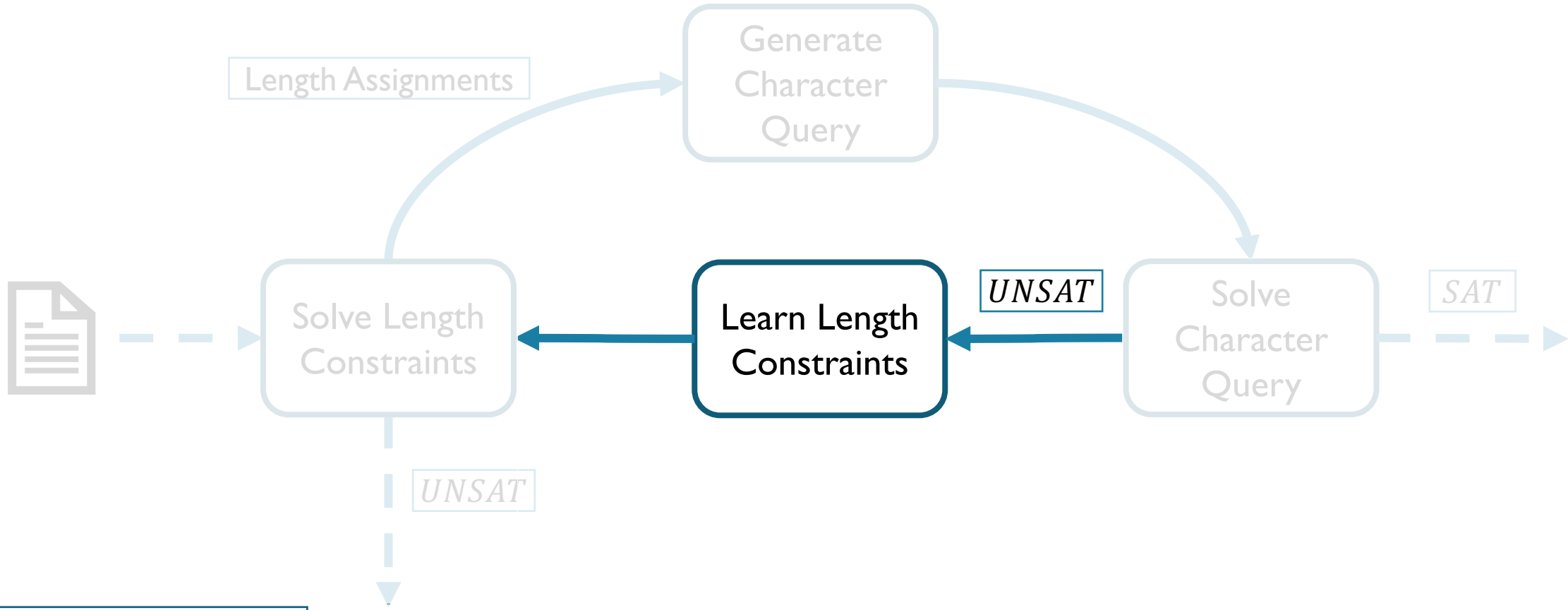
Literal Chars: ■ Char Variables: ■

UNSAT Core



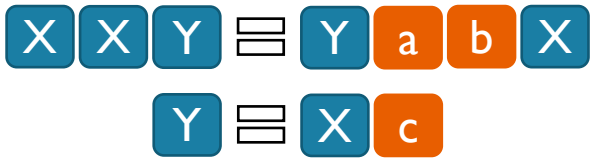
Literal Chars: ■ Char Variables: ■

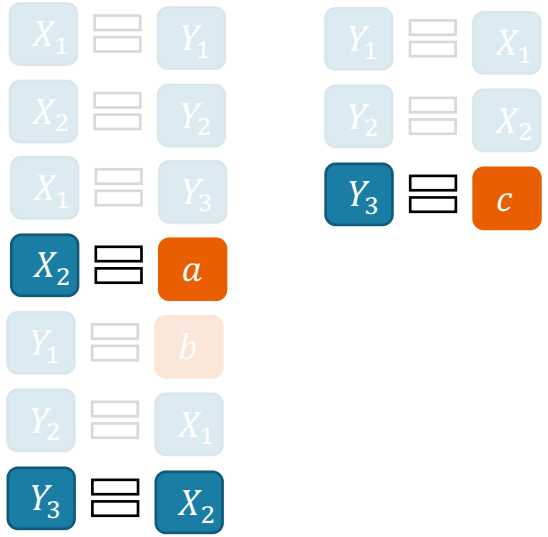
Length Abstraction Solver: Learn Length Constraint

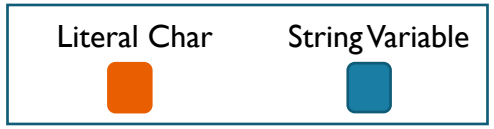


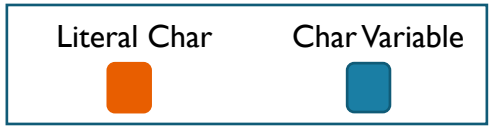
Length Abstraction Solver: Learn Length Constraint

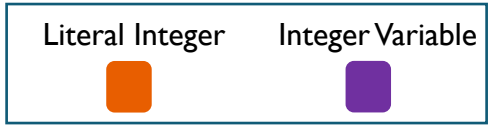
Input String Query	UNSAT Core	Learnt Length Constraint





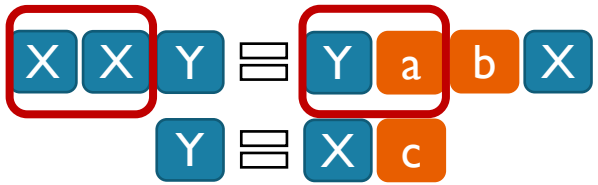




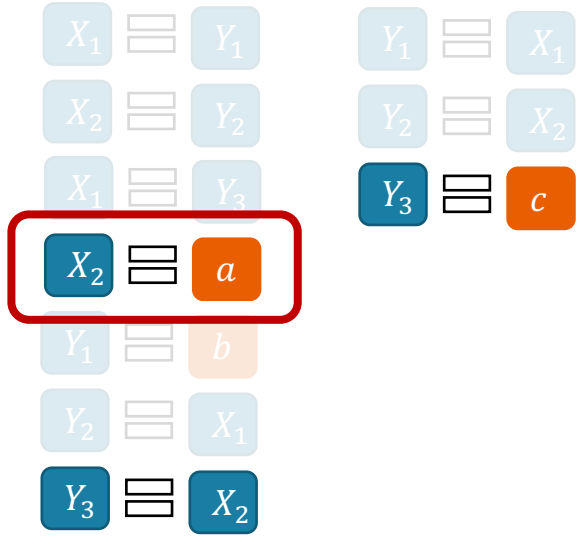


Length Abstraction Solver: Learn Length Constraint

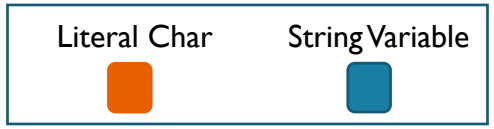
Input String Query

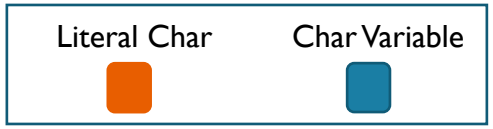


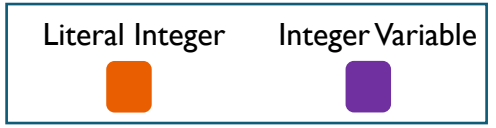
UNSAT Core



Learnt Length Constraint

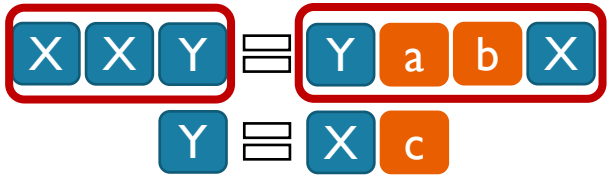




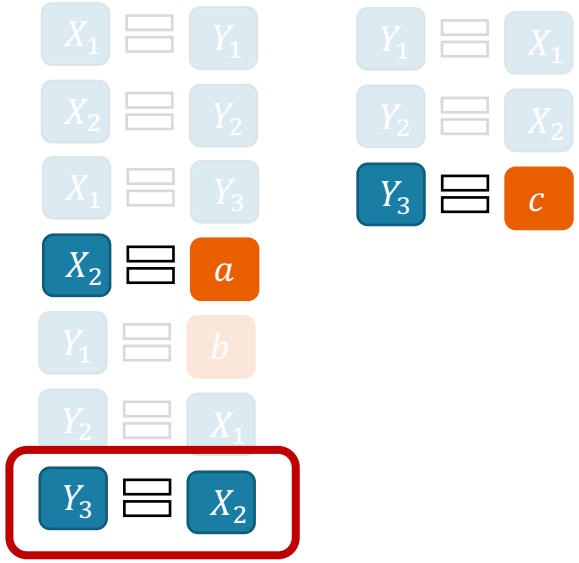


Length Abstraction Solver: Learn Length Constraint

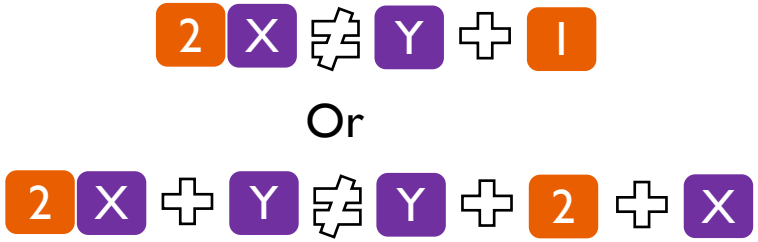
Input String Query

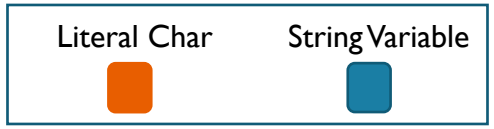


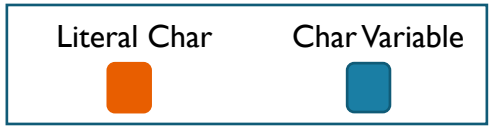
UNSAT Core

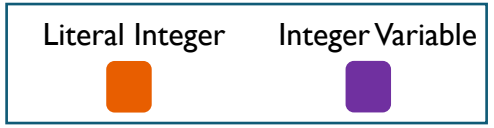


Learnt Length Constraint



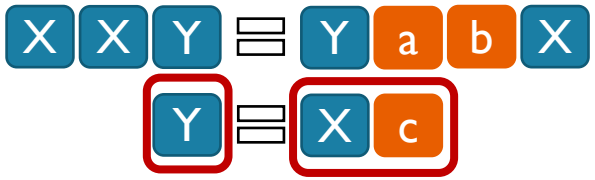




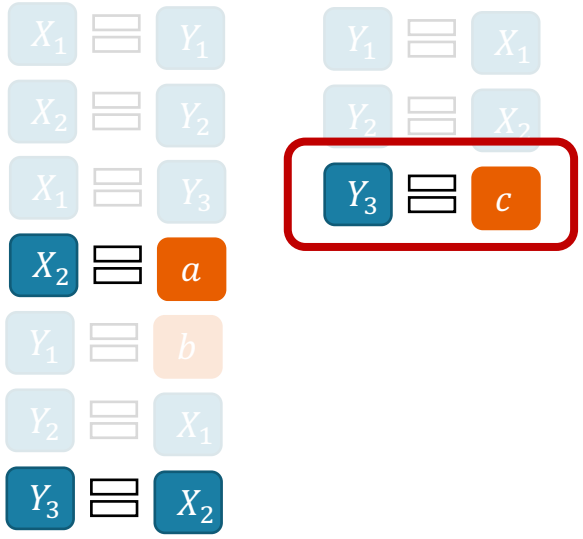


Length Abstraction Solver: Learn Length Constraint

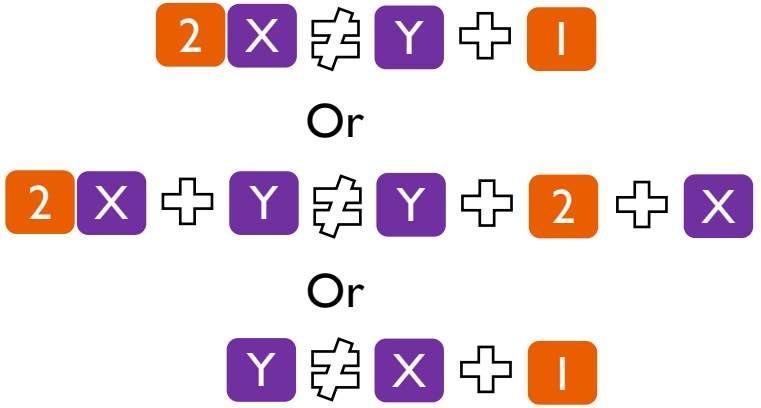
Input String Query

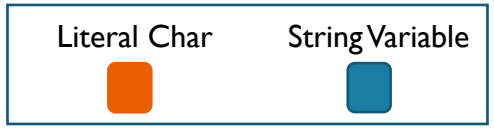


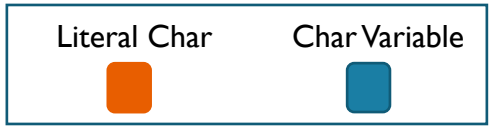
UNSAT Core

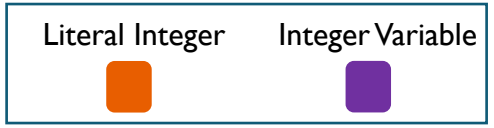


Learnt Length Constraint



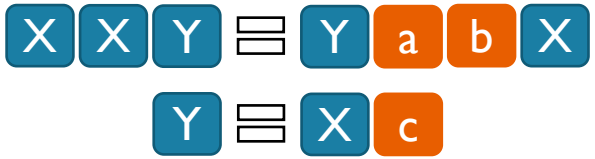




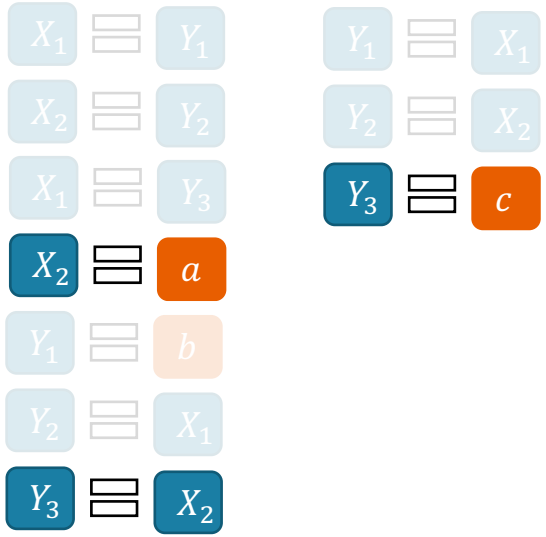


Length Abstraction Solver: Learn Length Constraint

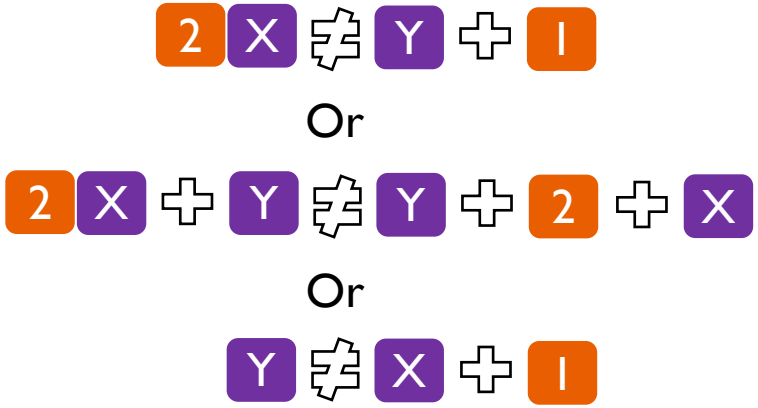
Input String Query

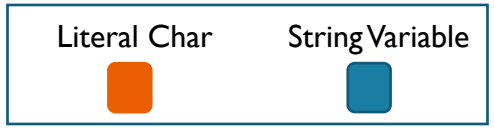


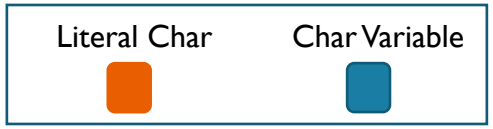
UNSAT Core

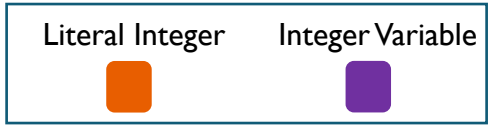


Learnt Length Constraint



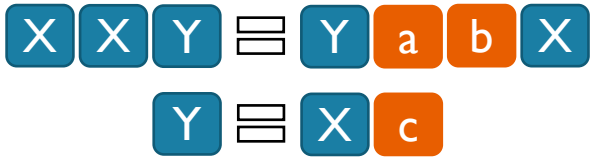






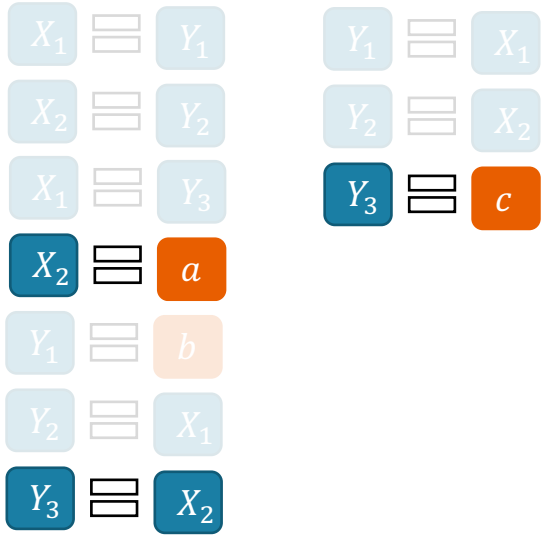
Length Abstraction Solver: Learn Length Constraint

Input String Query



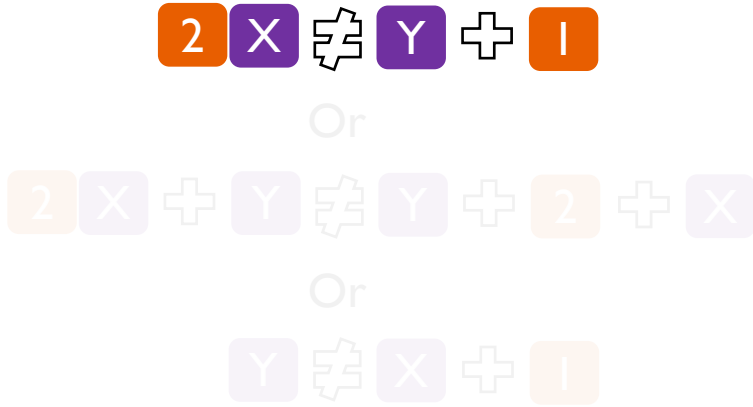
Literal Char: ■ String Variable: ■

UNSAT Core



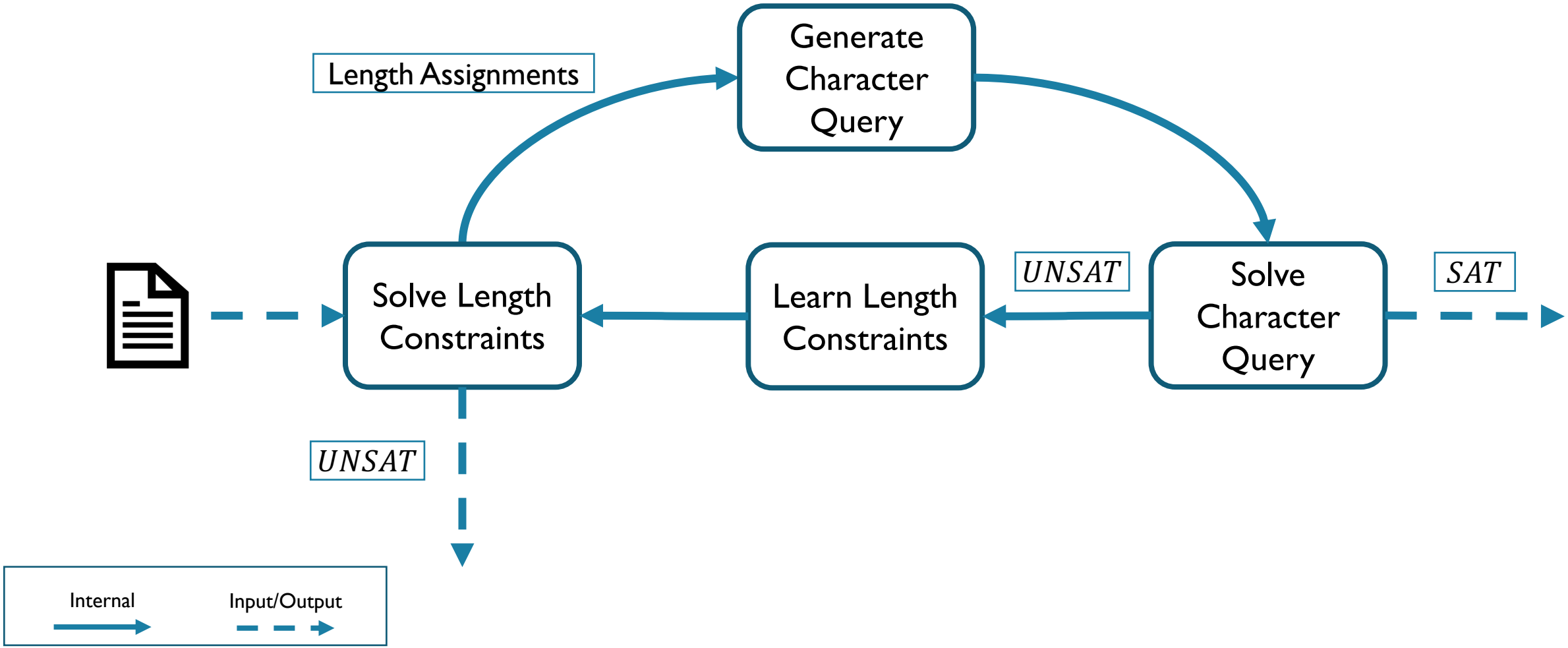
Literal Char: ■ Char Variable: ■

Learnt Length Constraint

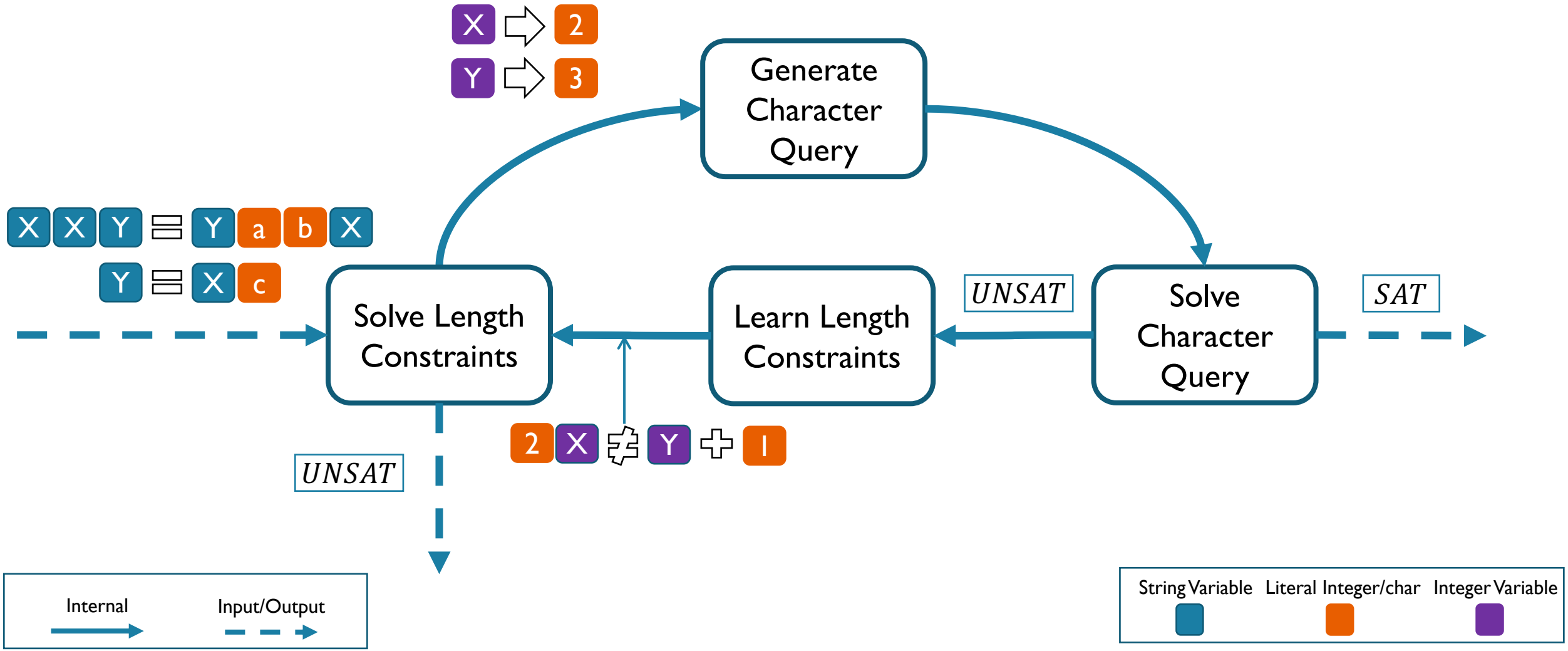


Literal Integer: ■ Integer Variable: ■

Length Abstraction Solver: Refinement Loop

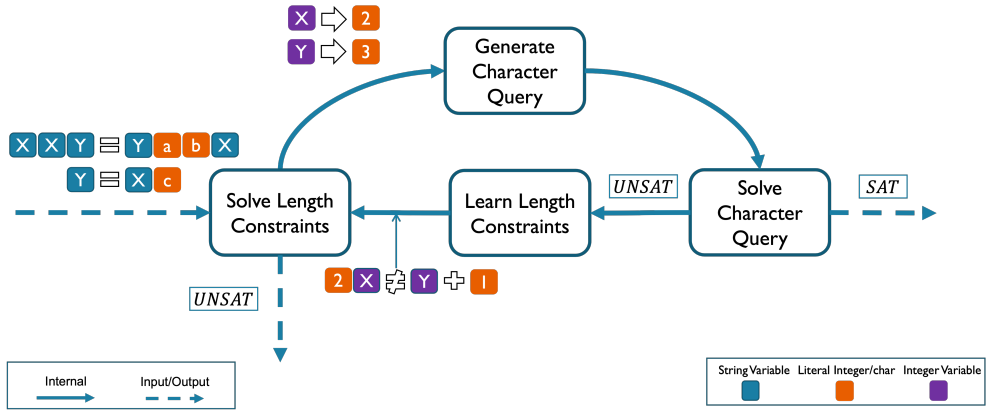


Length Abstraction Solver: Refinement Loop



Length Abstraction Solver Summary

- Returns UNSAT when there are no length solutions left to explore
- Returns SAT when character solver returns SAT
  - For a given length solution
- Learns length constraints that block character UNSAT Cores
- Works well for character queries that are only conjunctions
  - Faster solving
  - Better learning



Arrangement Solver Extension

Existing Arrangement Solver

Basic Idea:

- split equations into simpler ones (arrangements)
- until their satisfiability is "easily decided" (solved form)
- backtrack on conflicts

Existing Arrangement Solver

Basic Idea:

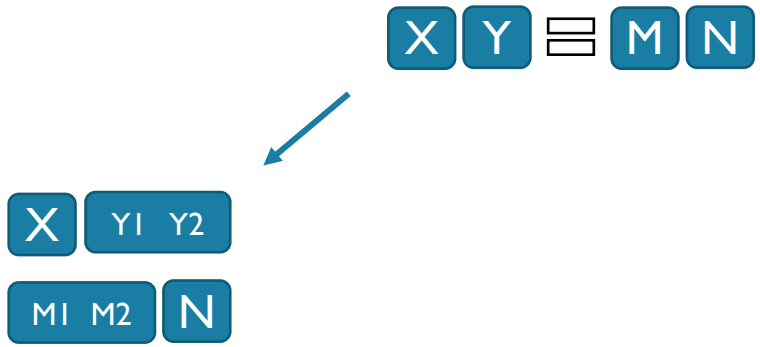
- split equations into simpler ones (arrangements)
- until their satisfiability is "easily decided" (solved form)
- backtrack on conflicts

String Variable

Existing Arrangement Solver

Basic Idea:

- split equations into simpler ones (arrangements)
- until their satisfiability is "easily decided" (solved form)
- backtrack on conflicts

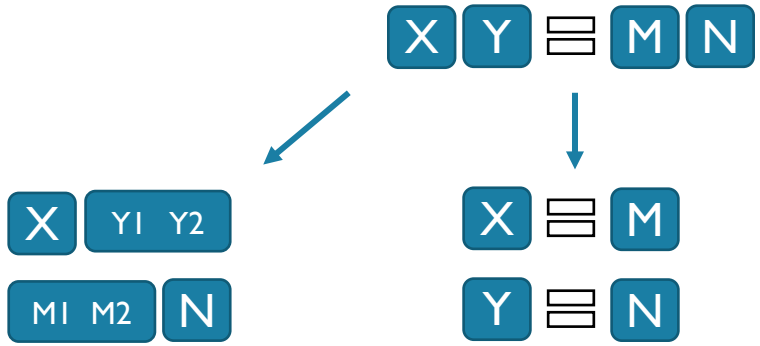


String Variable

Existing Arrangement Solver

Basic Idea:

- split equations into simpler ones (arrangements)
- until their satisfiability is "easily decided" (solved form)
- backtrack on conflicts

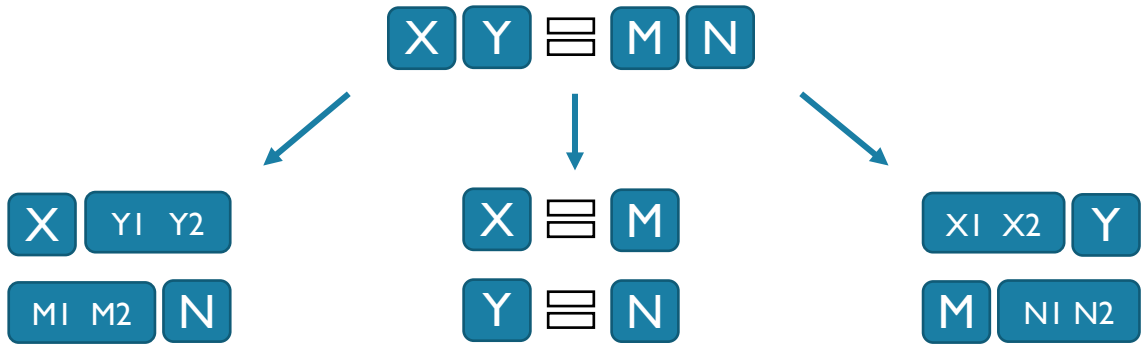


String Variable

Existing Arrangement Solver

Basic Idea:

- split equations into simpler ones (arrangements)
- until their satisfiability is "easily decided" (solved form)
- backtrack on conflicts

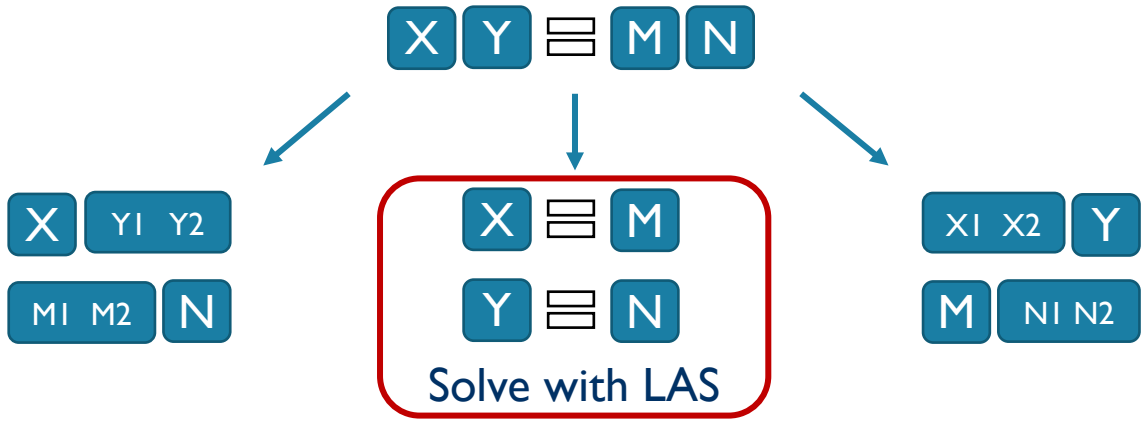


String Variable

Arrangement Solver Extension

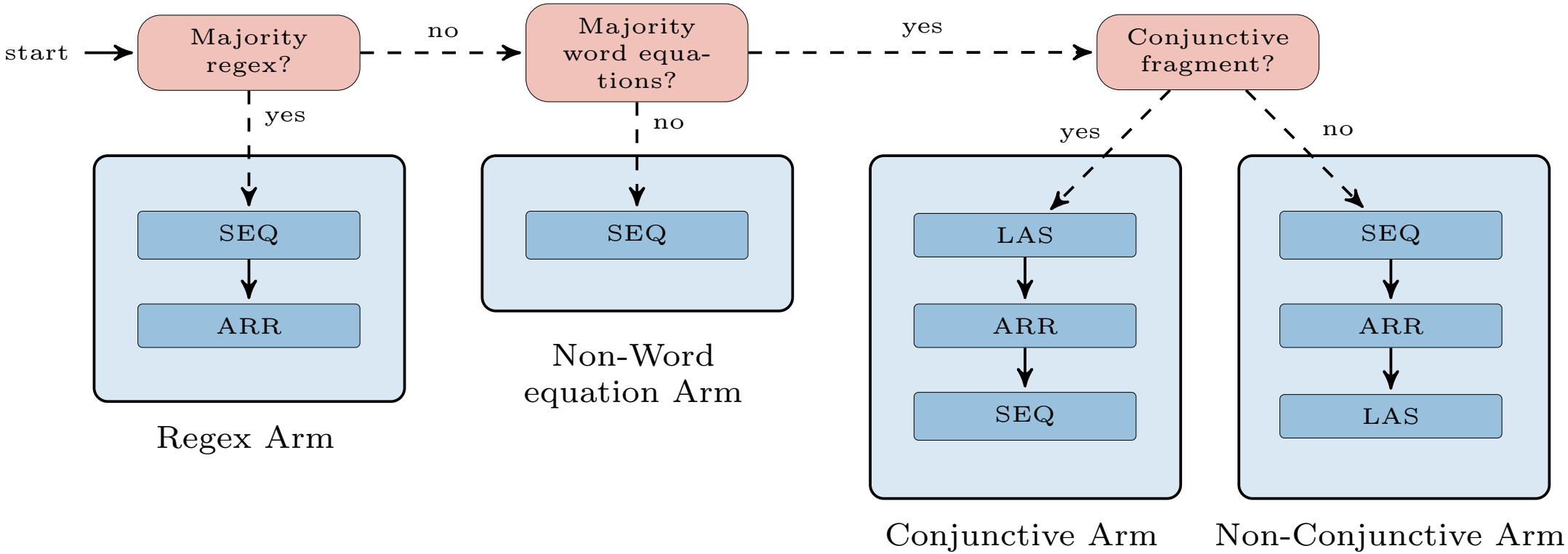
Basic Idea:

- check some arrangements with length-abstraction solver (LAS)
- helps block infinite loops
- helps model generation

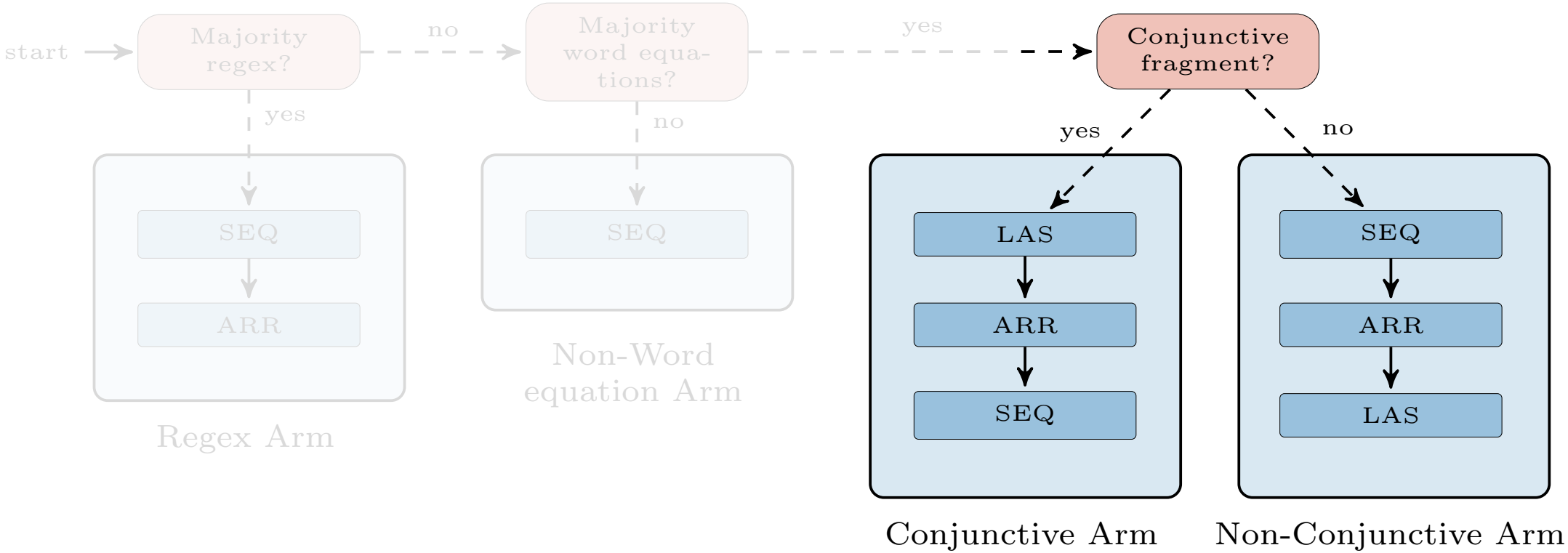


Selection Architecture

Selection Architecture



Selection Architecture: Conjunctive Fragment



Conjunctive Fragment Static Analysis

$$
\begin{array}{lll}
F & ::= & Atom \mid F \wedge F \mid \neg G \mid A_{int} \vee A_{int} \mid \neg A_{int} \\
G & ::= & G \vee G \mid \neg F \\
Atom & ::= & t_{str} = t_{str} \mid A_{int} \mid A_{ext} \\
A_{int} & ::= & t_{int} = t_{int} \mid t_{int} < t_{int} \\
A_{ext} & ::= & prefix(t_{str}, t_{str}) \mid suffix(t_{str}, t_{str}) \\
t_{int} & ::= & m \mid v \mid len(t_{str}) \mid t_{int} + t_{int} \mid m \cdot t_{int} \\
 & & \text{where } m \in Con_{int} \ \& \ v \in Var_{int} \\
t_{str} & ::= & s \mid v \mid t_{str} \cdot t_{str} \mid charAt(t_{str}, t_{int}) \\
 & & \text{where } s \in Con_{str} \ \& \ v \in Var_{str}
\end{array}
$$

Conjunctive Fragment Static Analysis

Theorem 2 (Conjunctive Fragment). *Let L be the language generated by the grammar in Figure 3. If $\varphi \in L$ is an input query, then* LAS *will always call* `ReduceToBV` *such that it produces a conjunction of bit-vector equations.*

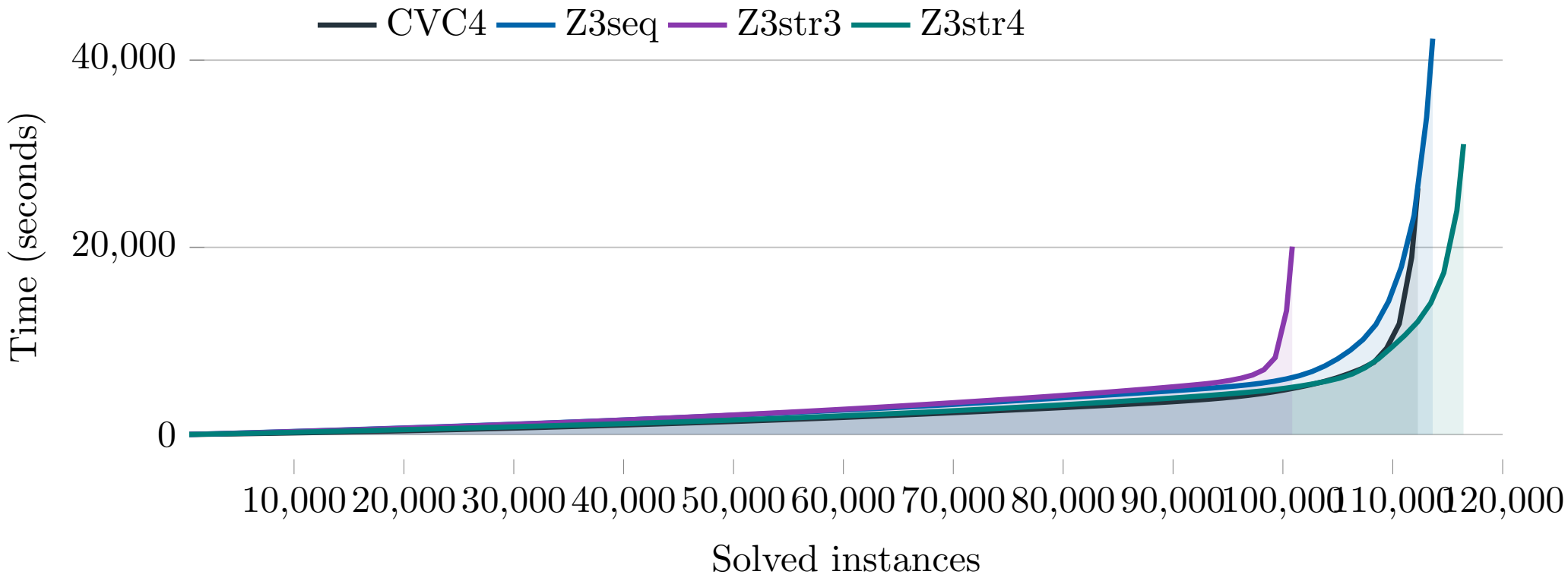
Evaluation

Experimental Setup

All experiments were performed on a server running

- Ubuntu 18.04.4 LTS
- with two AMD EPYC 7742 processors and
- 2TB RAM
- using the ZaligVinder benchmarking framework.
- The timeout for solving an instance was set at 20 seconds.

Overall Evaluation



Length Abstraction Solver Analysis

- LAS solves more queries per second compared to the arrangement solver in the conjunctive fragment (1128.5%)
- than it does outside the conjunctive fragment (904.6%)
- High percentages due to dynamic difficulty estimation
  - Really good at cutting off LAS

Arrangement Solver Analysis

- Without the extension, the arrangement solver
  - solves 72417 instances
  - in 423949.163 seconds
- With the extension, the arrangement solver
  - solves 107401 instances (148.3% of the queries without)
  - in 262047.893 seconds (61.8% of the time without)

Thank You!

z3str4.github.io