Scan the QR code to download the preprint

An Eager Satisfiability Modulo Theories Solver for Algebraic Datatypes

*Amar Shah, **Federico Mora**, Sanjit A. Seshia*

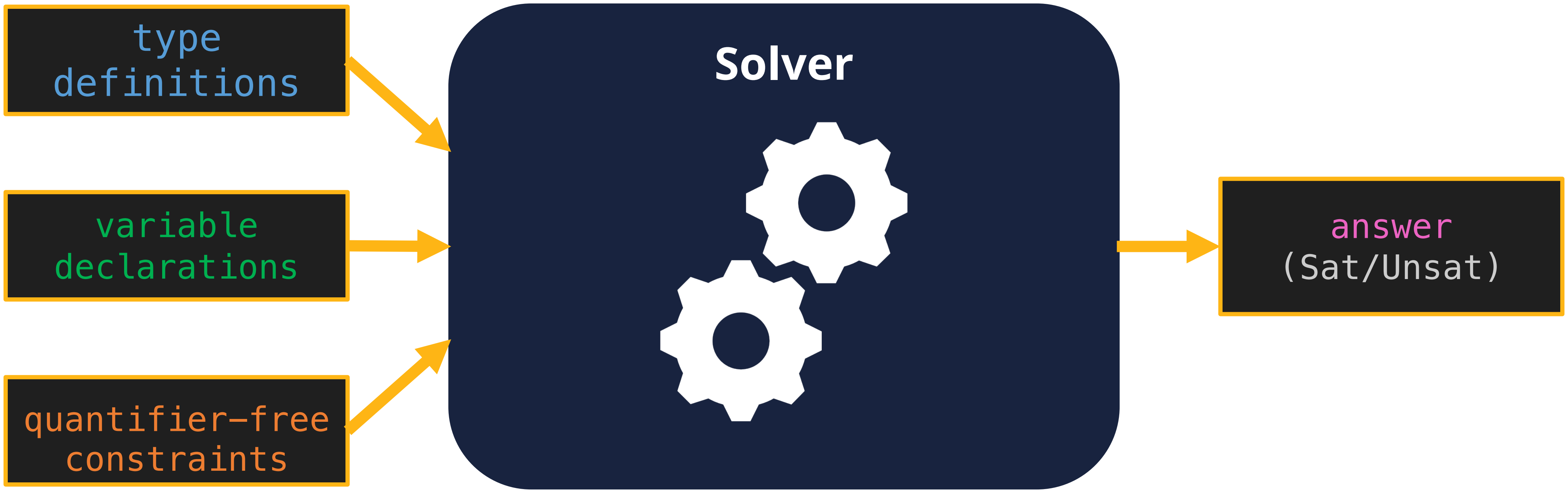
Berkeley
UNIVERSITY OF CALIFORNIA

https://github.com/uclid-org/algaroba

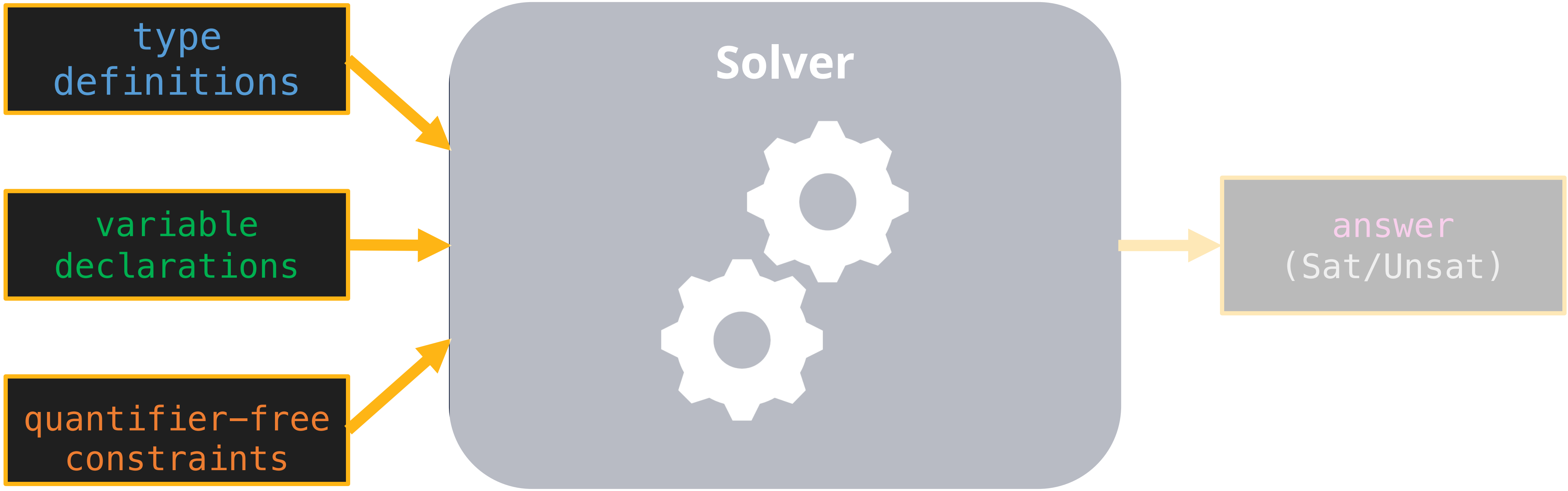
What are Satisfiability Modulo Theories (SMT) Solvers?

(for quantifier-free algebraic datatypes)

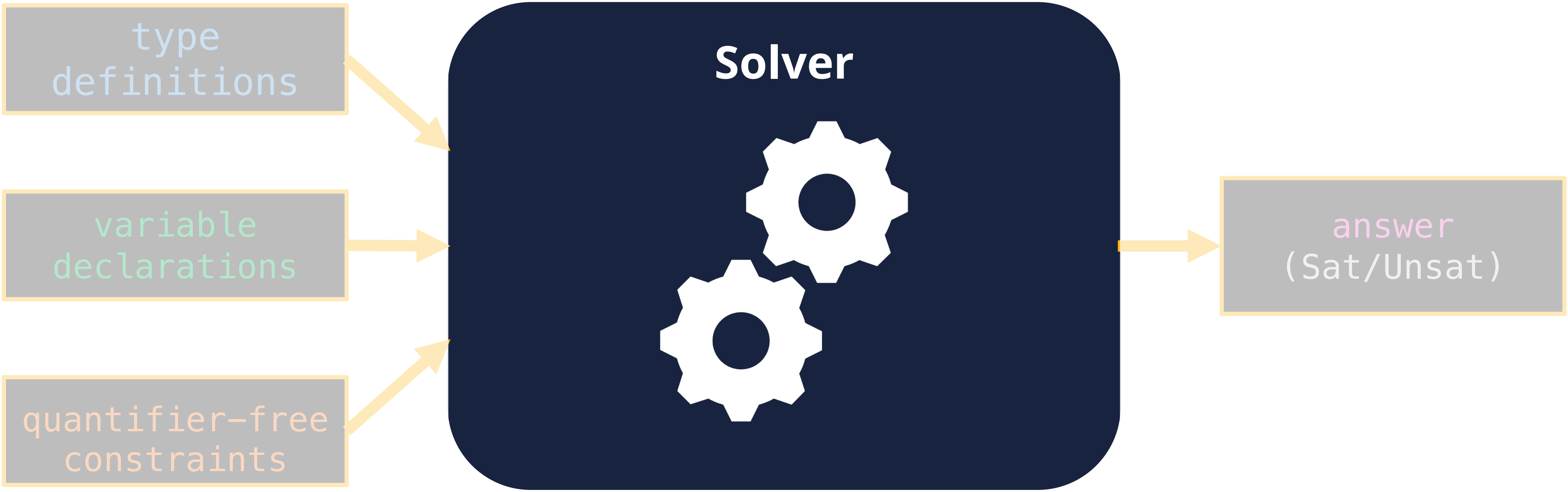
Solver Interface (SMT-LIB)



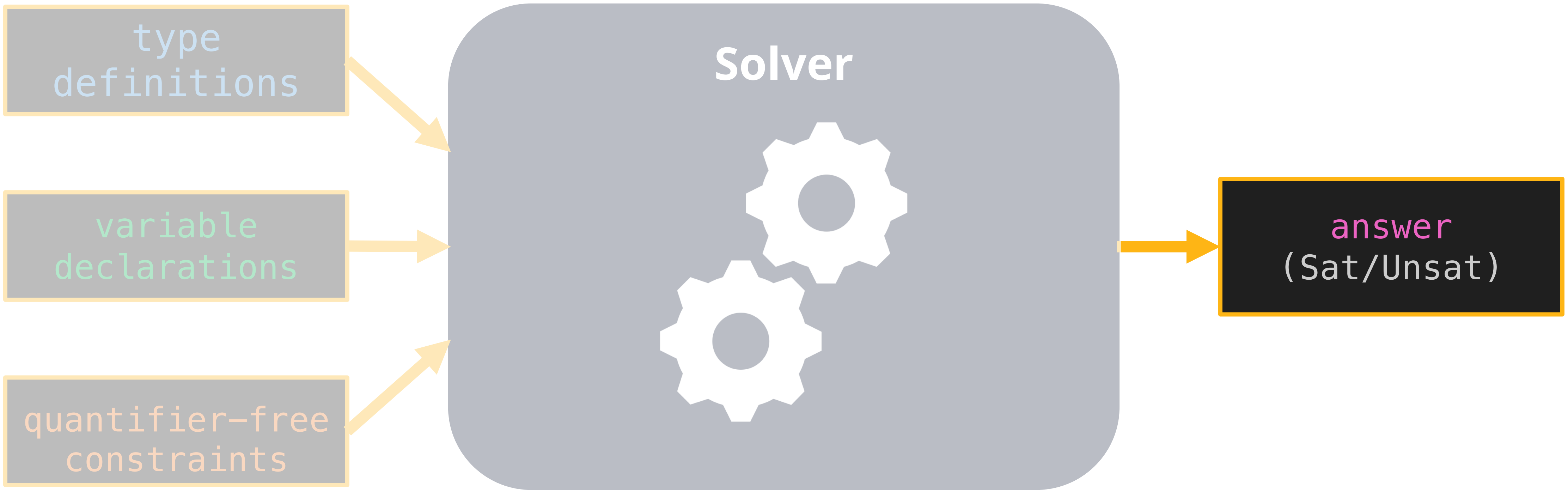
Solver Interface (SMT-LIB)



Solver Interface (SMT-LIB)



Solver Interface (SMT-LIB)



What are Algebraic Datatypes?

ADTs for short

Running Example: Blocks World

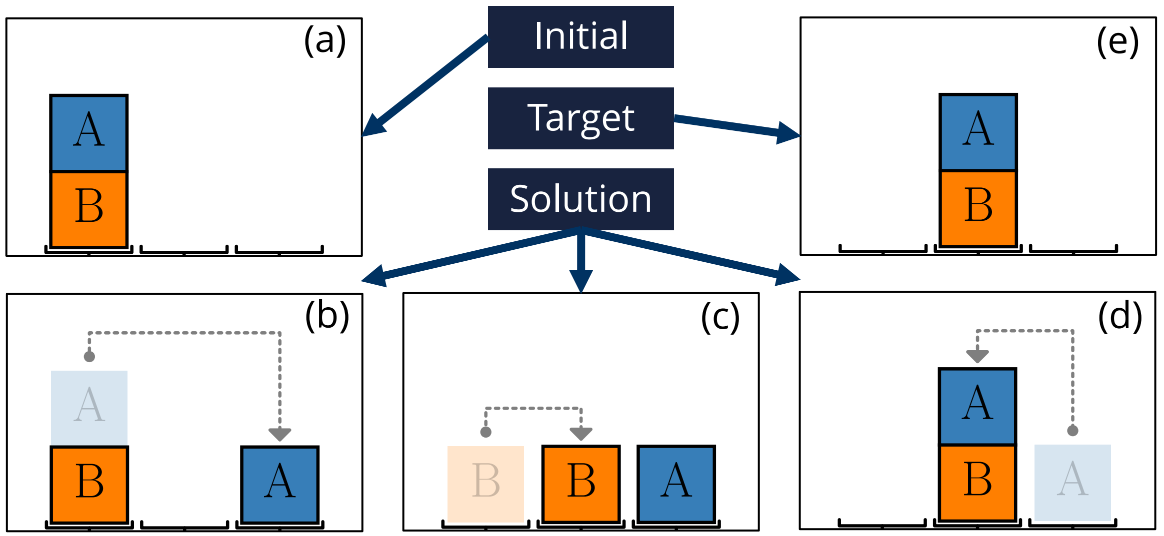


Figure 1. Solution (1b, 1c, 1d) to a simple blocks world puzzle. 1a is the initial configuration; 1e is the target configuration.

Winograd (1971); Sussman (1973); Gupta and Nau (1992)

Running Example: Blocks World

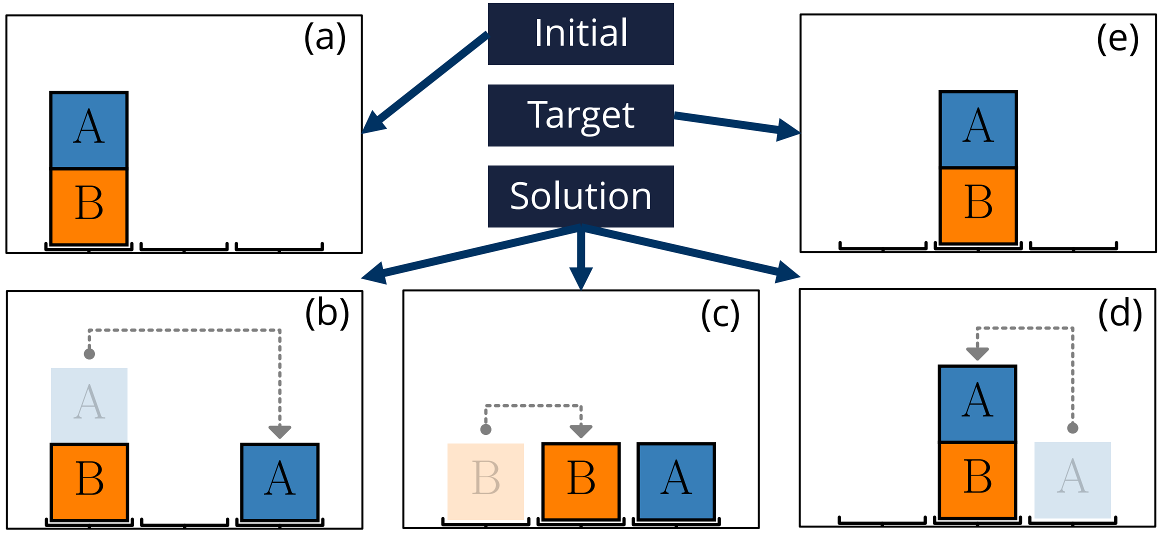


Figure 1. Solution (1b, 1c, 1d) to a simple blocks world puzzle. 1a is the initial configuration; 1e is the target configuration.

Winograd (1971); Sussman (1973); Gupta and Nau (1992)

Running Example: Blocks World

Is there a sequence of *k* legal moves that leads from the initial to the target configuration?

1. blocks can only be taken from the top of a stack;
2. blocks can only be placed on the top of a stack; and
3. only one block can be moved at a time.

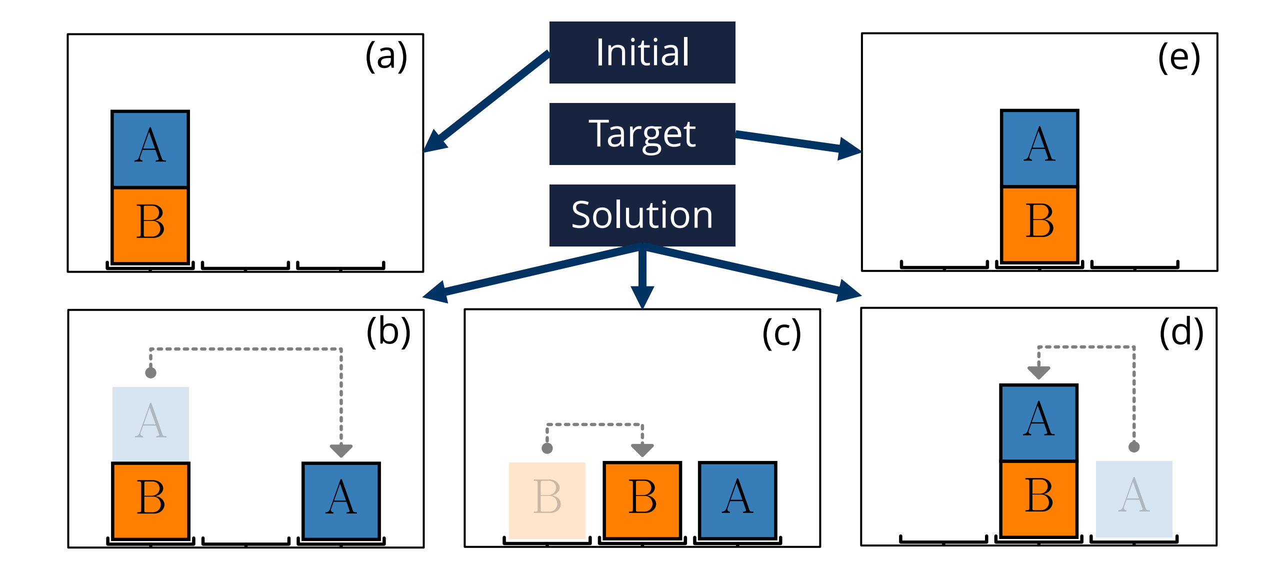
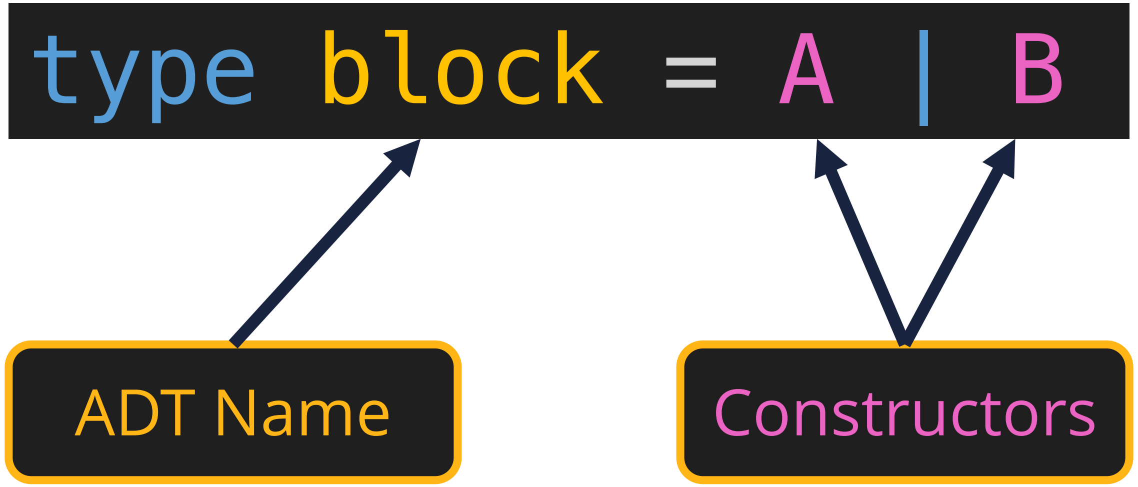


Figure 1. Solution (1b, 1c, 1d) to a simple blocks world puzzle. 1a is the initial configuration; 1e is the target configuration.

Winograd (1971); Sussman (1973); Gupta and Nau (1992)

Algebraic Datatypes Example 1



- Variables of type block can take on one of two values:
  - A or B

Algebraic Datatypes Example 2

```
type tower =
  | Empty
  | Stack of {top: block; rest: tower}
```

ADT Name
Constructor
Selector

- Variables of type tower can be one of:
  - Empty;
  - Stack(A, Empty); Stack(B, Empty);
  - Stack(A, Stack(A, Empty)); Stack(B, Stack(A, Empty)); …
  - …
  - Stack(A, Stack(A, Stack(A, Stack(A, Stack(A, Stack(A, Empty)))))); …
  - …

Running Example: Blocks World

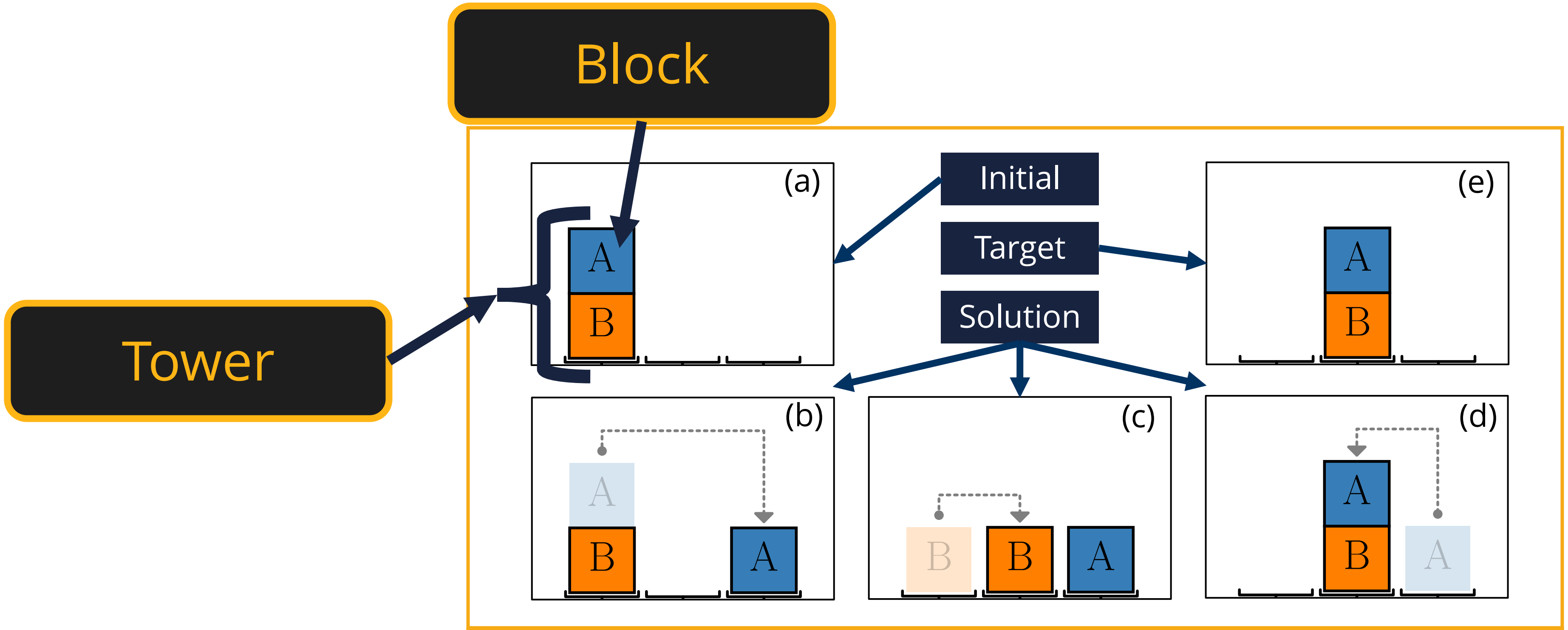


Figure 1. Solution (1b, 1c, 1d) to a simple blocks world puzzle. 1a is the initial configuration; 1e is the target configuration.

Winograd (1971); Sussman (1973); Gupta and Nau (1992)

Definition: Algebraic Datatypes

Algebraic datatypes consist of

- constructors (e.g., `Stack` is a function from `block * tower` to `tower`),
- selectors (e.g., `rest` is a function from `tower` to `tower`),
- testers (e.g., `is_Empty` is a function from `tower` to `boolean`).

Barrett, Fontaine, and Tinelli (2017)

Definition: Algebraic Datatypes

Algebraic datatypes consist of

- constructors (e.g., `Stack` is a function from `block * tower` to `tower`),
- selectors (e.g., `rest` is a function from `tower` to `tower`),
- testers (e.g., `is_Empty` is a function from `tower` to `boolean`).

The following informal axioms govern their behaviour:

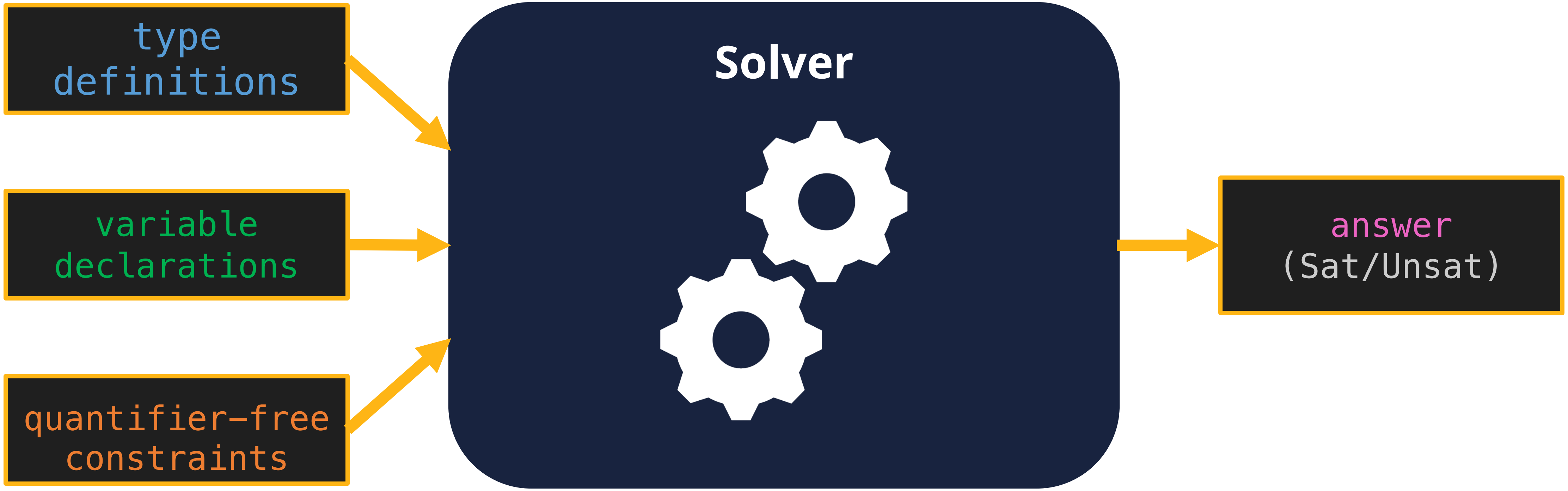
- Selectors and constructors play nicely (e.g., `Stack(A, Empty).rest` returns `Empty`)
- Testers behave as expected (e.g., `is_Empty(Stack(A, Empty))` returns false).
- **Every instance of an algebraic datatype is acyclic.**

Barrett, Fontaine, and Tinelli (2017)

What are Satisfiability Modulo Theories Solvers? Revisited

(for quantifier-free algebraic datatypes)

Solver Interface (SMT-LIB)



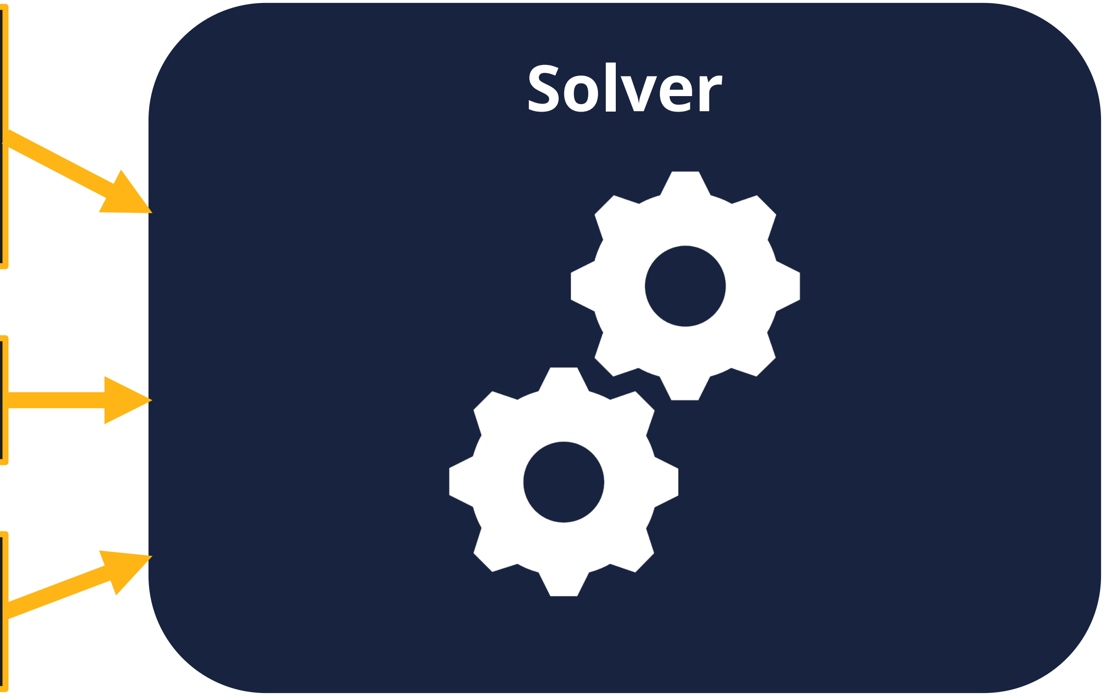
Solver Interface (SMT-LIB)

```
type block = A | B

type tower =
  | Empty
  | Stack of {top: block;
              rest: tower}
```

```
let x: tower;
let y: tower;
```

```
assert x == y.rest;
assert y == x.rest;
```




```
ADT Name; Constructor; Selector; Variable; Constraint
```

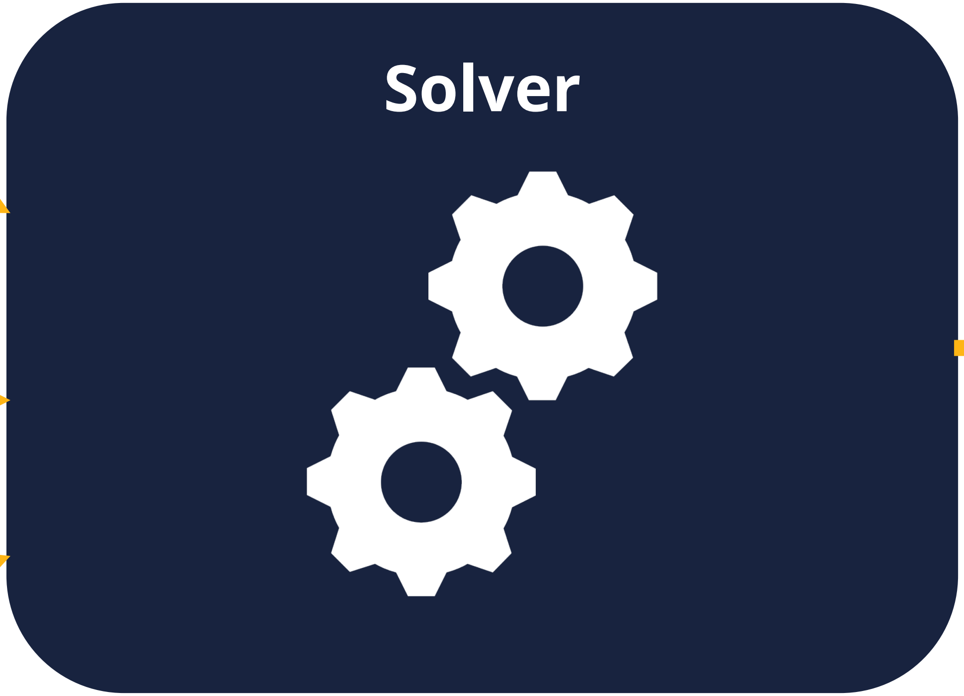
Solver Interface (SMT-LIB)

```
type block = A | B

type tower =
  | Empty
  | Stack of {top: block;
              rest: tower}
```

```
let x: tower;
let y: tower;
```

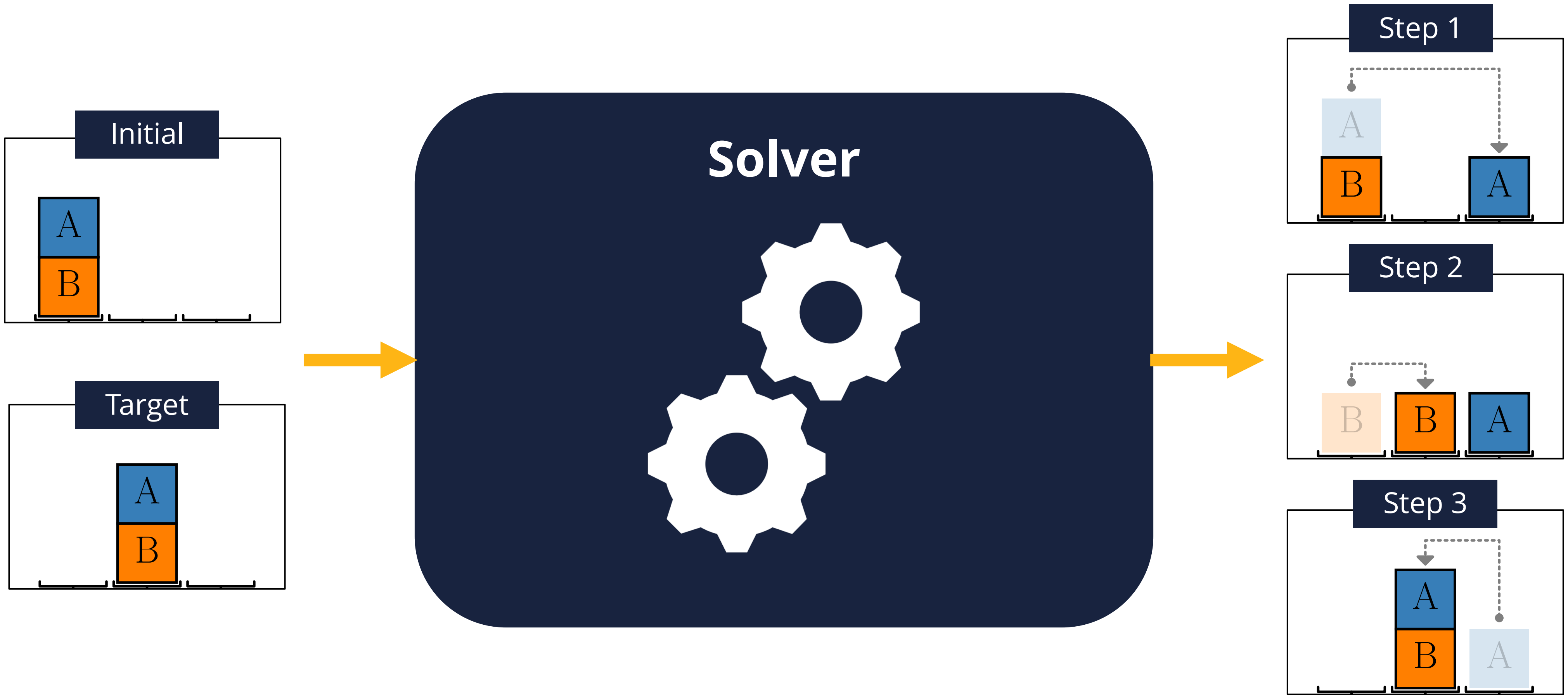
```
assert x == y.rest;
assert y == x.rest;
```




```
Unsat
```

ADT Name; Constructor; Selector; Variable; Constraint

Solver Interface (SMT-LIB)



* Model construction is a work in progress, for now we would just say “sat”

Other Applications of ADTs

Distributed Systems:

- We used ADTs to verify distributed systems
  - node states are records,
  - messages are records, and
  - sequences of messages are an inductive type (like a list).

Hardware:

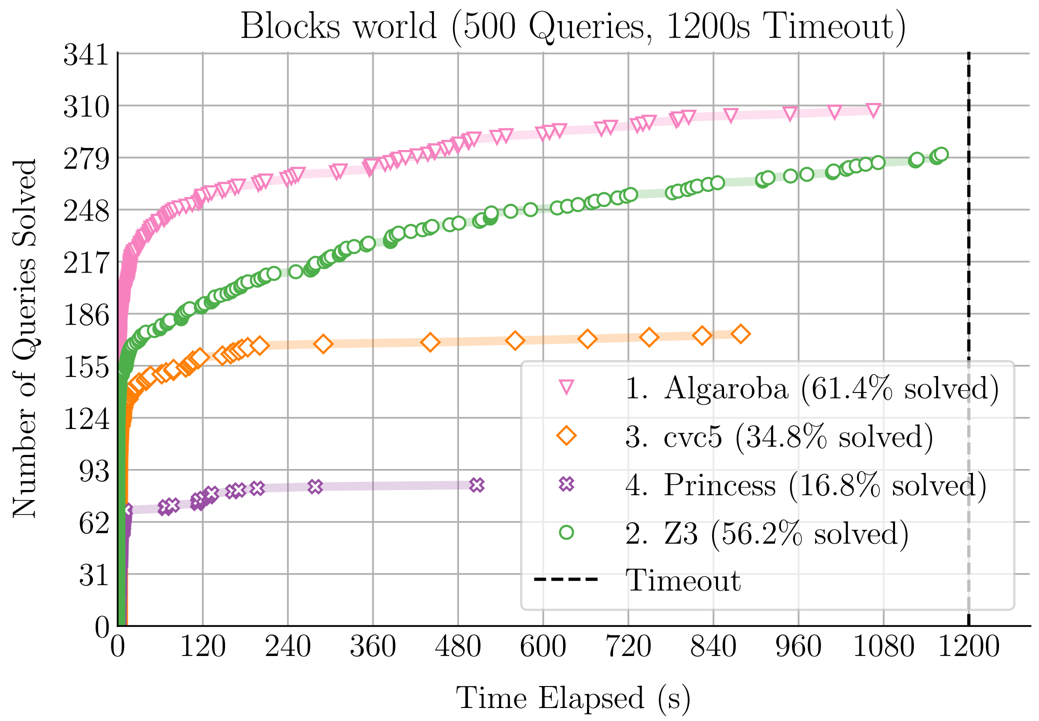
- We are using ADTs to model encryption in trusted enclaves
  - encryption with a constructor,
  - decryption with a selector, and
  - garbled text with a sum type.

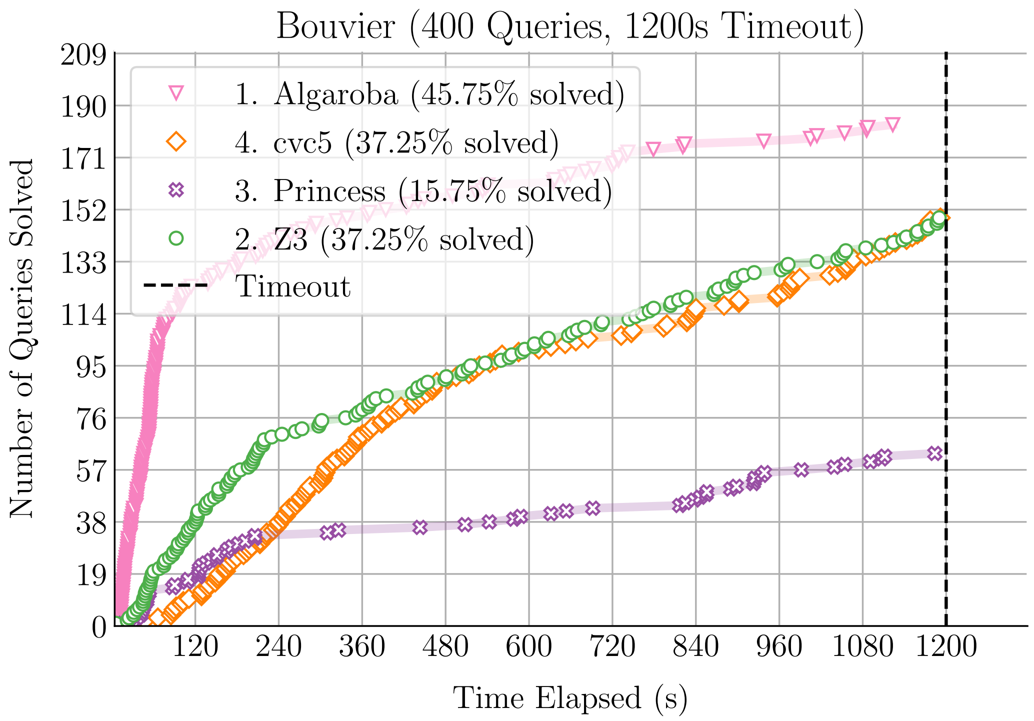
Empirical Evaluation

Implementation and Tool Links

- Try out
  - Algaroba, our prototype solver!
    - https://github.com/uclid-org/algaroba
  - UCLID5, our formal modeling and verification engine with (coming) ADT support!
    - https://github.com/uclid-org/uclid
  - The UPVerifier, our tool for distributed systems verification based on ADTs!
    - https://github.com/uclid-org/upverifier

Results: Overall Performance

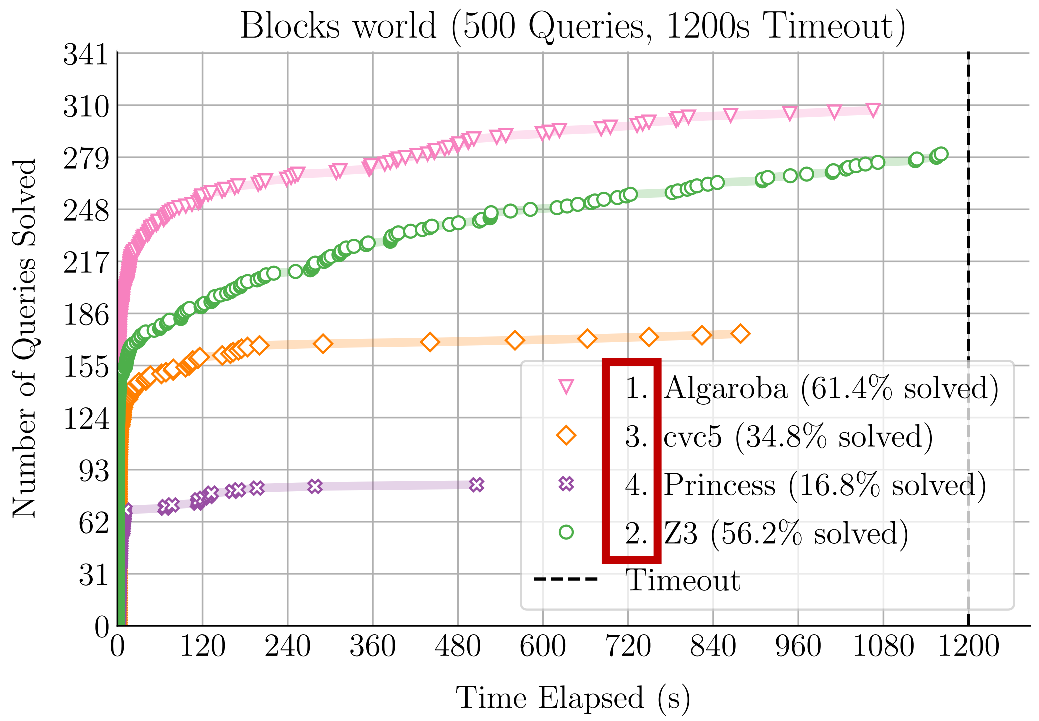


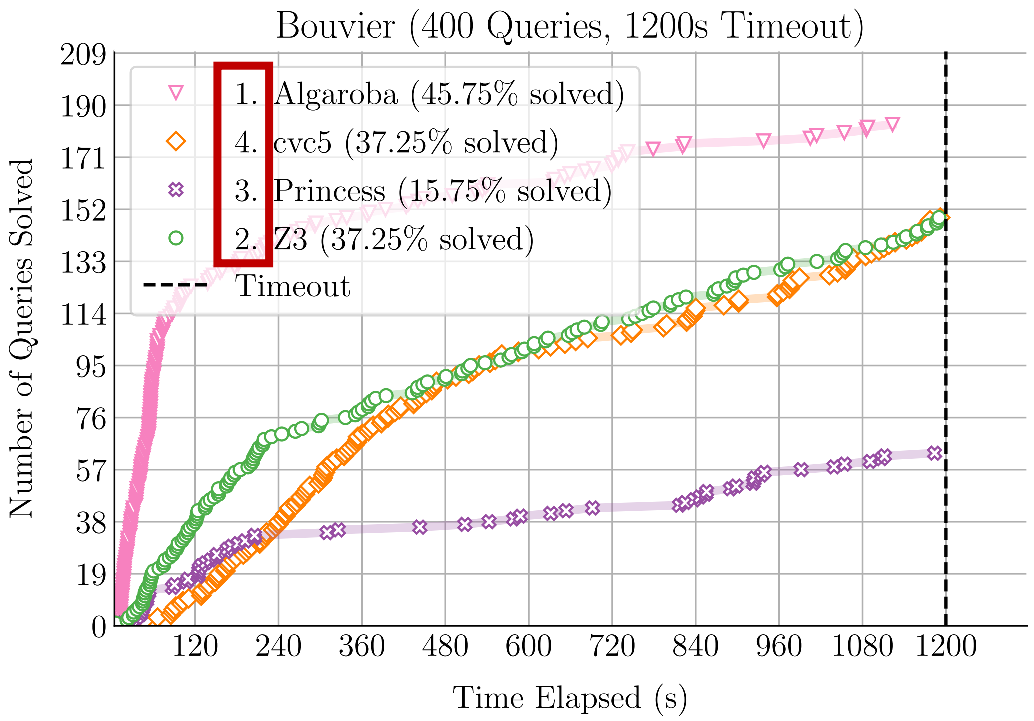


Our tool (Algaroba) solves more queries in less time (higher left is better)

Bouvier (2021); Barbosa et al. (2022); de Moura and Bjørner (2008); Hojjat and Rümmer (2017)

Results: Contribution Score



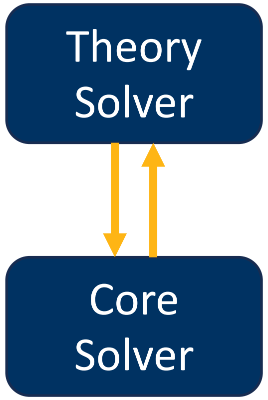


Algaroba solves many queries that no other solver can (108/900), achieves the highest contribution score (rank in legend).

Related Work

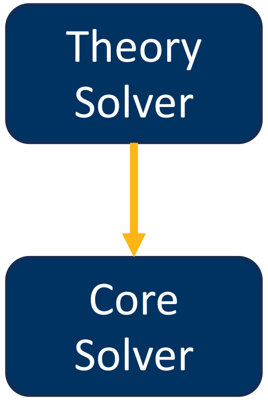
Lazy Approaches (Axioms as Needed):

- cvc5, SMTInterpol
  - Theory solver based on Oppen
- z3
  - (Unpublished but similar)



Eager Approaches (Axioms Upfront):

- Princess
  - Reduce to linear integer arithmetic
- Algaroba (our solver)

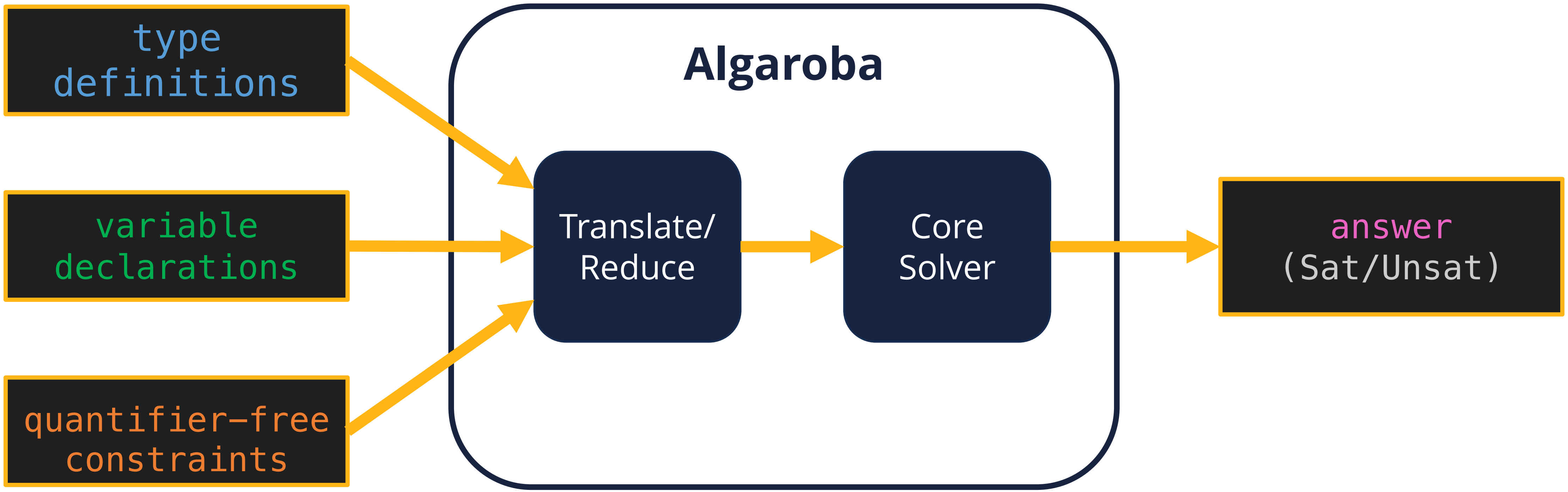


Sebastiani (2007); Seshia (2005); Oppen (1980); Barbosa et al. (2022); Christ, Hoenicke, and Nutz (2012); Hojjat and Rümmer (2017)

How Do We Do It?

Eager Reduction to Core Solver Explained

Approach Sketch: Eager Reduction



Seshia (2005)

Challenge: Finite Reduction

<u>**Well-Foundedness Axiom**</u>:

Let u and v be two ADT values. If $u = v.s_1.s_2 \ldots s_n \wedge \theta$ then $u \neq v$,

- where s_i are selectors and
- θ asserts that all s_i are correctly applied.

```
let x: tower;
let y: tower;

assert x == y.rest;
assert y == x.rest;
```

How can we have a finite, quantifier-free reduction if n is arbitrary?

Challenge: Finite Reduction

Well-Foundedness Axiom:

Let u and v be two ADT values. If $u = v.s_1.s_2 \ldots s_n \wedge \theta$ then $u \neq v$,

- where s_i are selectors and
- θ asserts that all s_i are correctly applied.

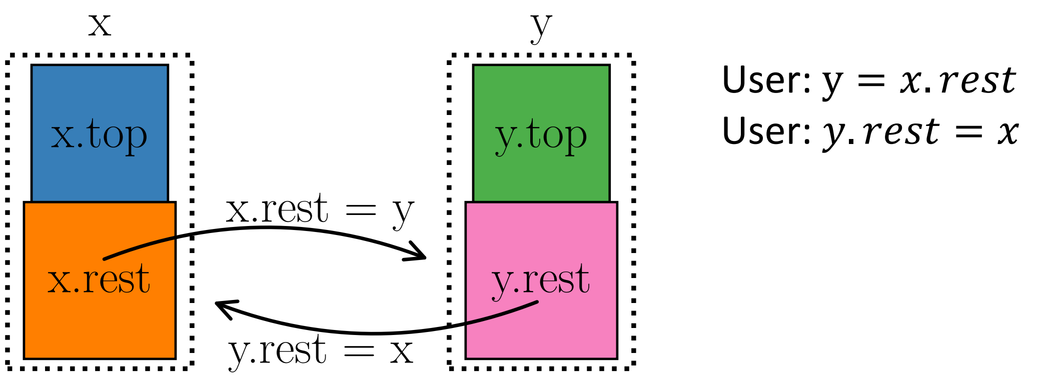


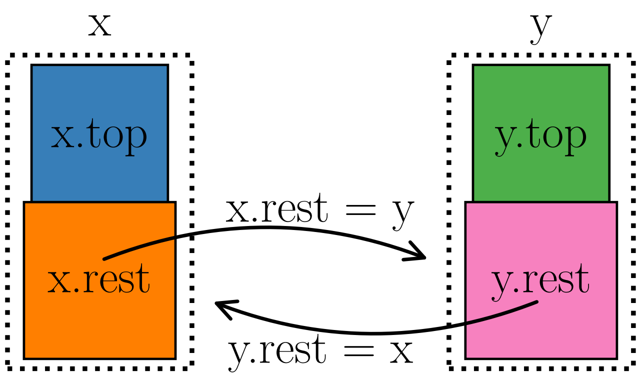
Figure 3. Visual representation (left) and proof (right) of an unsatisfiable query. θ ($\stackrel{\text{def}}{=}$ x *is Stack* $\wedge$ y *is Stack*) omitted as premise.

Challenge: Finite Reduction

Well-Foundedness Axiom:

Let u and v be two ADT values. If $u = v.s_1.s_2 \ldots s_n \land \theta$ then $u \neq v$,

- where s_i are selectors and
- θ asserts that all s_i are correctly applied.



User: $y = x.rest$
User: $y.rest = x$
Equality: $x.rest.rest = x$
WF Axiom: $x \neq x$
Reflexivity: $x = x$
Conclusion: $\bot$

Figure 3. Visual representation (left) and proof (right) of an unsatisfiable query. θ ($\stackrel{\text{def}}{=}$ *x is Stack* $\land$ *y is Stack*) omitted as premise.

Get $x \neq x$ from $x.rest.rest = x$, with $n = 2$

Approach: Sufficient Encoding

$\psi_1 \leftarrow \mathit{NNF}(\psi)$
$\psi_2 \leftarrow \mathit{Flatten}(\psi_1)$
$k \leftarrow$ Number of ADT variables in ψ_2
$\psi_3 \leftarrow$ Apply rewrite rules to ψ_2
$\phi_1, \ldots, \phi_m \leftarrow$ Add axioms using k to ψ_3
$\psi^* \leftarrow \psi_3 \wedge \phi_1 \wedge \ldots \wedge \phi_m$
return $\mathit{UF\text{-}SMT\text{-}Solver}(\psi^*)$

Think of k as the number of unique ADT terms in the query

Think of ϕ_i as instances of the cycle axiom for all $0 < n \leq k$

Let ψ be the input ADT query, k gives a bound that we use to compute ψ^*, a finite, quantifier-free UF query.

Burch and Dill (1994)

Approach: Sufficient Encoding

$[(x.rest.rest = x) \Rightarrow (x \neq \mathrm{x})]$ was one of the ϕ_i

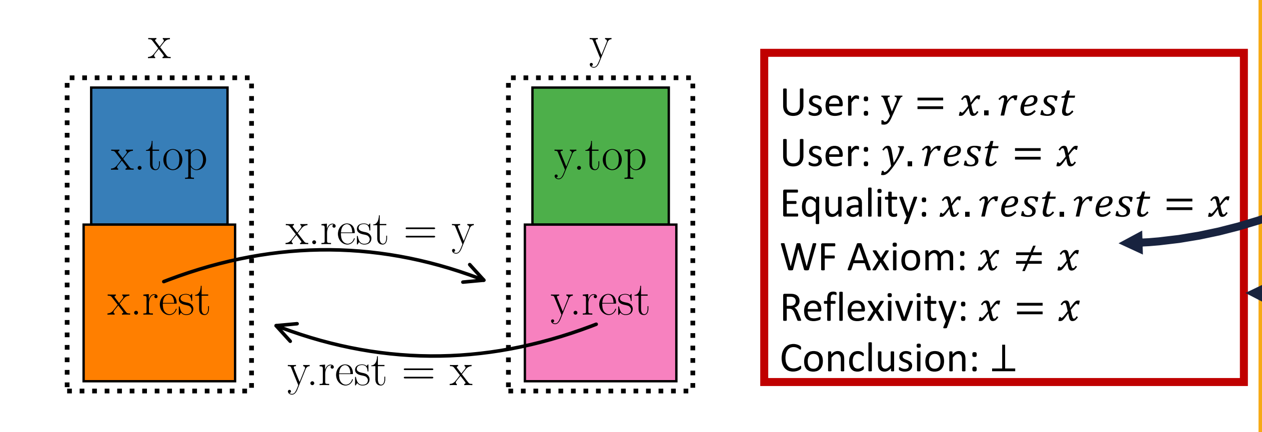
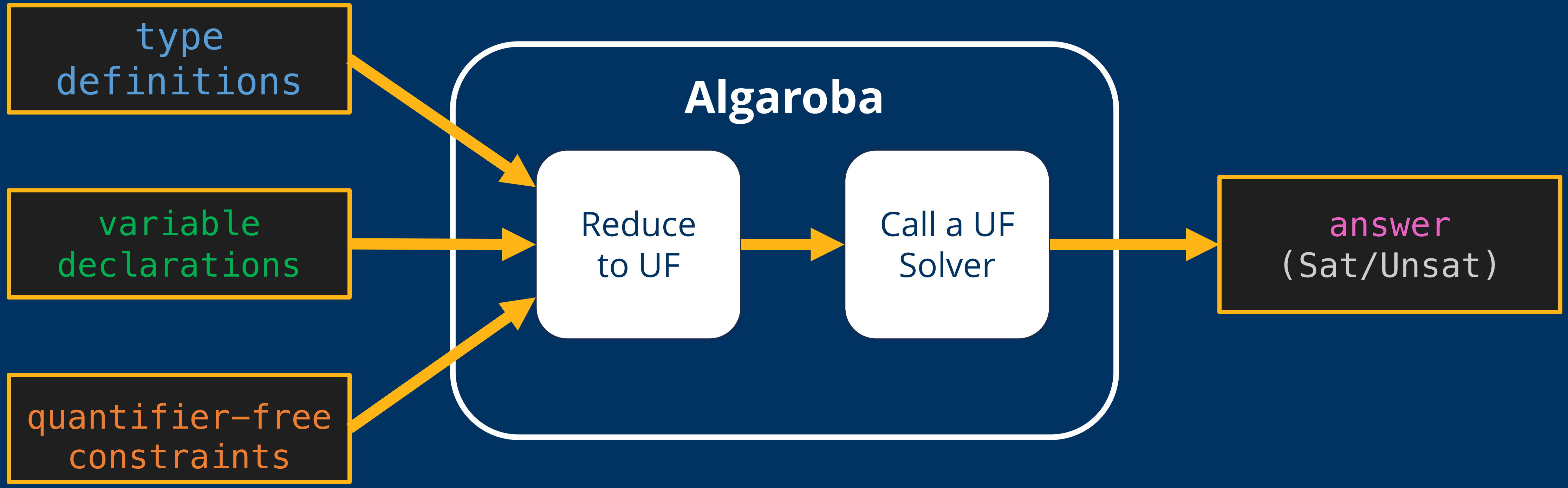


Figure 3. Visual representation (left) and proof (right) of an unsatisfiable query. θ ($\stackrel{\text{def}}{=}$ *x is Stack* $\wedge$ *y is Stack*) omitted as premise.

All of these are equality constraints that an off-the-shelf solver can handle!

Thank you!



Scan the QR code to download the **preprint**

https://github.com/uclid-org/algaroba

Works Cited In Presentation

- Winograd (1971);
- Sussman (1973);
- Gupta and Nau (1992);
- Barrett, Fontaine, and Tinelli (2017);
- Mora, Desai, Polgreen, and Seshia (2023);
- Bouvier (2021);
- Barbosa et al. (2022);
- de Moura and Bjørner (2008);
- Hojjat and Rümmer (2017);
- Seshia (2005);
- Burch and Dill (1994);
- Sebastiani (2007);
- Oppen (1980);
- Christ, Hoenicke, and Nutz (2012);