# Fast and Trustworthy SMT Solving for String Constraints

Andrew Reynolds

Aug 24, 2022

The University of Iowa

# Overview

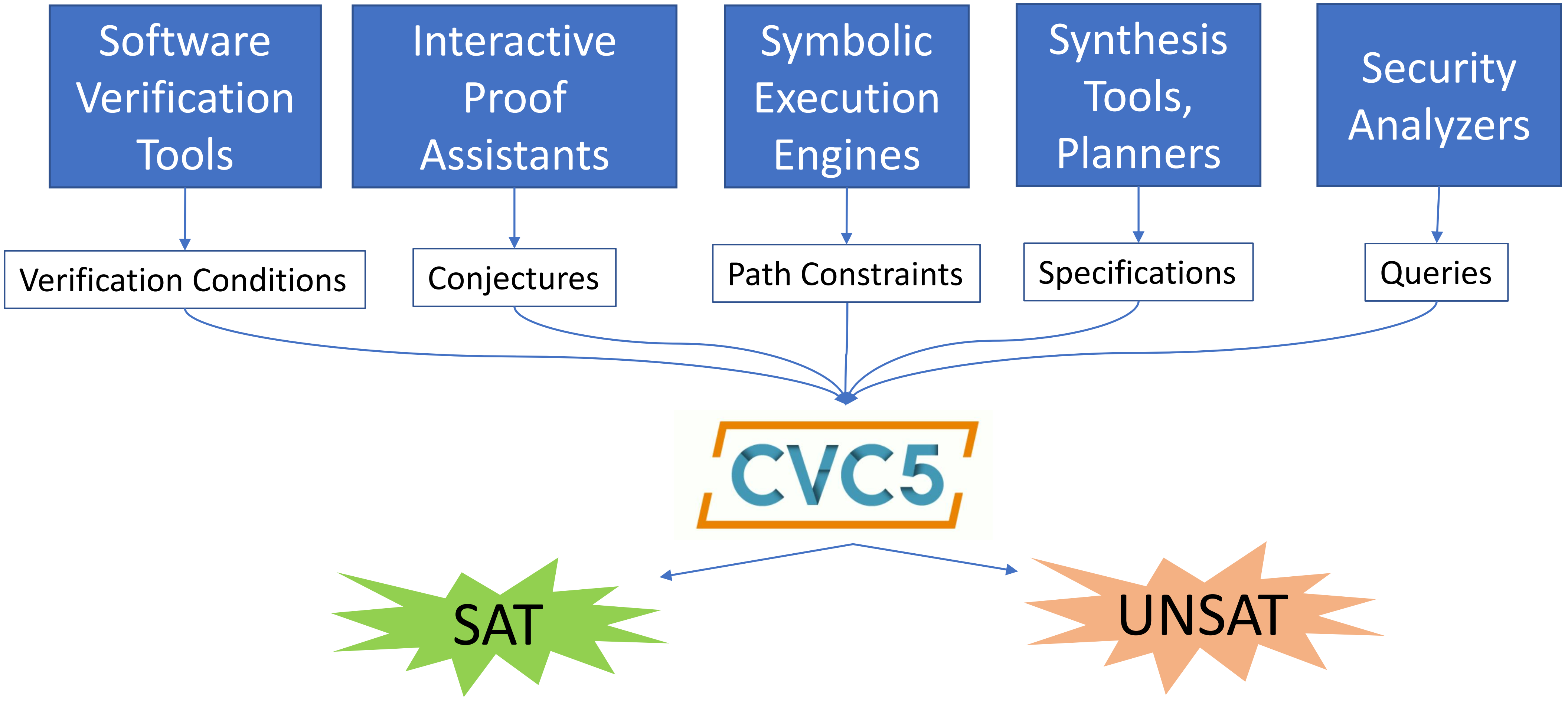
- Satisfiability Modulo Theories (SMT) solvers widely used tools

  $\Rightarrow$ the SMT solver cvc5

- cvc5 for strings and regular expressions
  - ***Fast***: new advances in SMT string solving
  - ***Trustworthy***: producing externally checkable proofs

- Future Directions

# Satisfiability Modulo Theories (SMT) Solvers




⇒ SMT solvers are fully automated reasoners, widely used in applications

# cvc5: A Versatile and Industrial-Strength SMT Solver*

Haniel Barbosa[3], Clark Barrett[1], Martin Brain[4], Gereon Kremer[1], Hanna Lachnitt[1], Makai Mann[1], Abdalrhman Mohamed[2], Mudathir Mohamed[2], Aina Niemetz[1(✉)], Andres Nötzli[1], Alex Ozdemir[1], Mathias Preiner[1], Andrew Reynolds[2], Ying Sheng[1], Cesare Tinelli[2], and Yoni Zohar[1,5]

[1] Stanford University, Stanford, USA
[2] The University of Iowa, Iowa City, USA
[3] Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
[4] City, University of London, London, UK
[5] Bar-Ilan University, Tel Aviv, Israel

- cvc5 is latest SMT solver in CVC line of tools
  - Open source, builds on code base of CVC4
  - 1.0 launched in April 2022

⇒ Best tool paper, ETAPS 2022

# cvc5: A Versatile and Industrial-Strength SMT Solver*

Haniel Barbosa[3], Clark Barrett[1], Martin Brain[4], Gereon Kremer[1], Hanna Lachnitt[1], Makai Mann[1], Abdalrhman Mohamed[2], Mudathir Mohamed[2], Aina Niemetz[1(✉)], Andres Nötzli[1], Alex Ozdemir[1], Mathias Preiner[1], Andrew Reynolds[2], Ying Sheng[1], Cesare Tinelli[2], and Yoni Zohar[1,5]

[1] Stanford University, Stanford, USA
[2] The University of Iowa, Iowa City, USA
[3] Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
[4] City, University of London, London, UK
[5] Bar-Ilan University, Tel Aviv, Israel

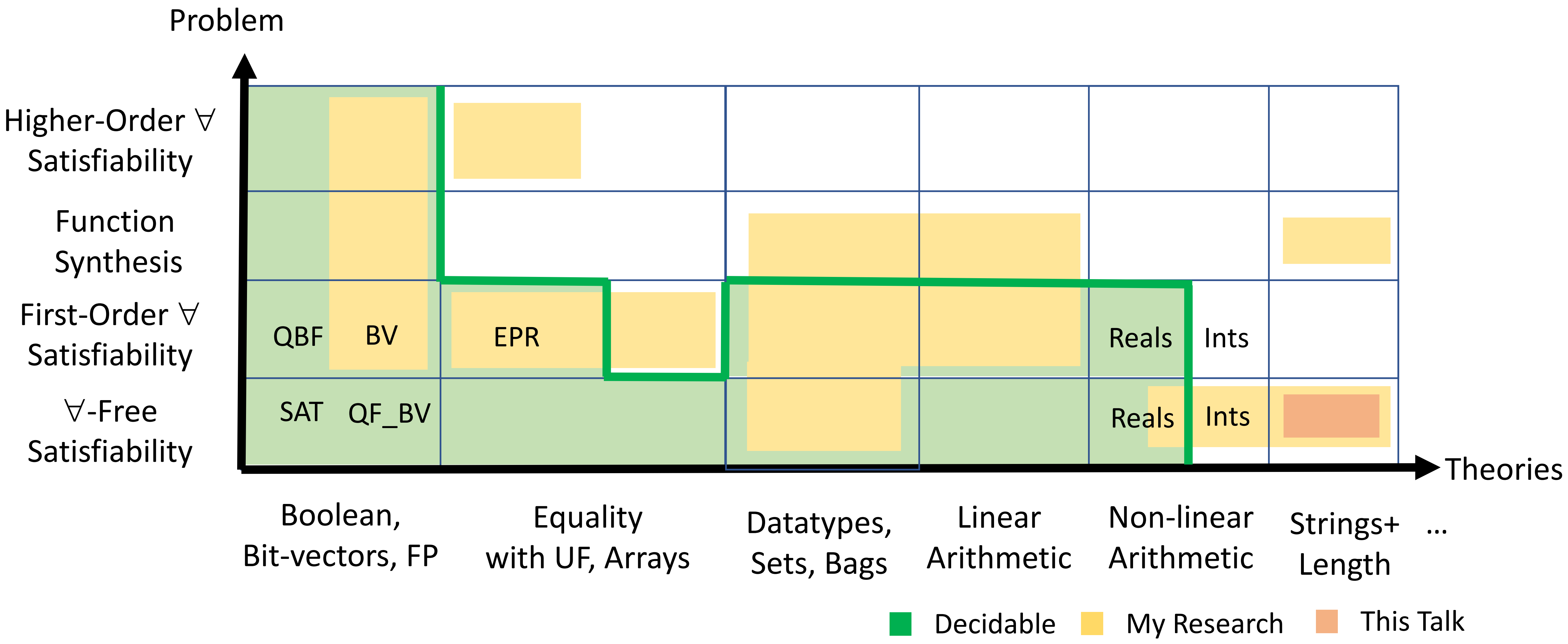
# cvc5: A Versatile SMT Solver

- Support for many theories
  - Arithmetic, Bit-vectors, Arrays, Datatypes, Floating-Points, *Strings*
  - **Extended:** Sets, Multisets, Finite Fields

- Many features:
  - `get-model`, `get-unsat-core`, `get-proof`
  - **Extended:** syntax-guided synthesis, `get-interpolant`, `get-abduct`, `get-quant-elim`

$\Rightarrow$ If you have a new problem domain, we'd like to support it!

# cvc5: SMT and beyond



# cvc5: A Versatile and Industrial-Strength SMT Solver*

Haniel Barbosa[3], Clark Barrett[1], Martin Brain[4], Gereon Kremer[1], Hanna Lachnitt[1], Makai Mann[1], Abdalrhman Mohamed[2], Mudathir Mohamed[2], Aina Niemetz[1](✉), Andres Nötzli[1], Alex Ozdemir[1], Mathias Preiner[1], Andrew Reynolds[2], Ying Sheng[1], Cesare Tinelli[2], and Yoni Zohar[1,5]

[1] Stanford University, Stanford, USA
[2] The University of Iowa, Iowa City, USA
[3] Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
[4] City, University of London, London, UK
[5] Bar-Ilan University, Tel Aviv, Israel

# cvc5: An Industrial-Strength SMT Solver

- Efficient solving algorithms
  - Best overall performance* SMT-COMP 2018, 2019, 2020, 2021, 2022

  * shared with predecessor CVC4

- High coding standards
  - Streamlined API, high code coverage, code reviews

- Extensively tested
  - 3000+ hand-crafted regressions and counting
  - Fuzzed internally (Murxla [Niemetz et al CAV22]) and by external groups

- **New:** Produces externally checkable proofs

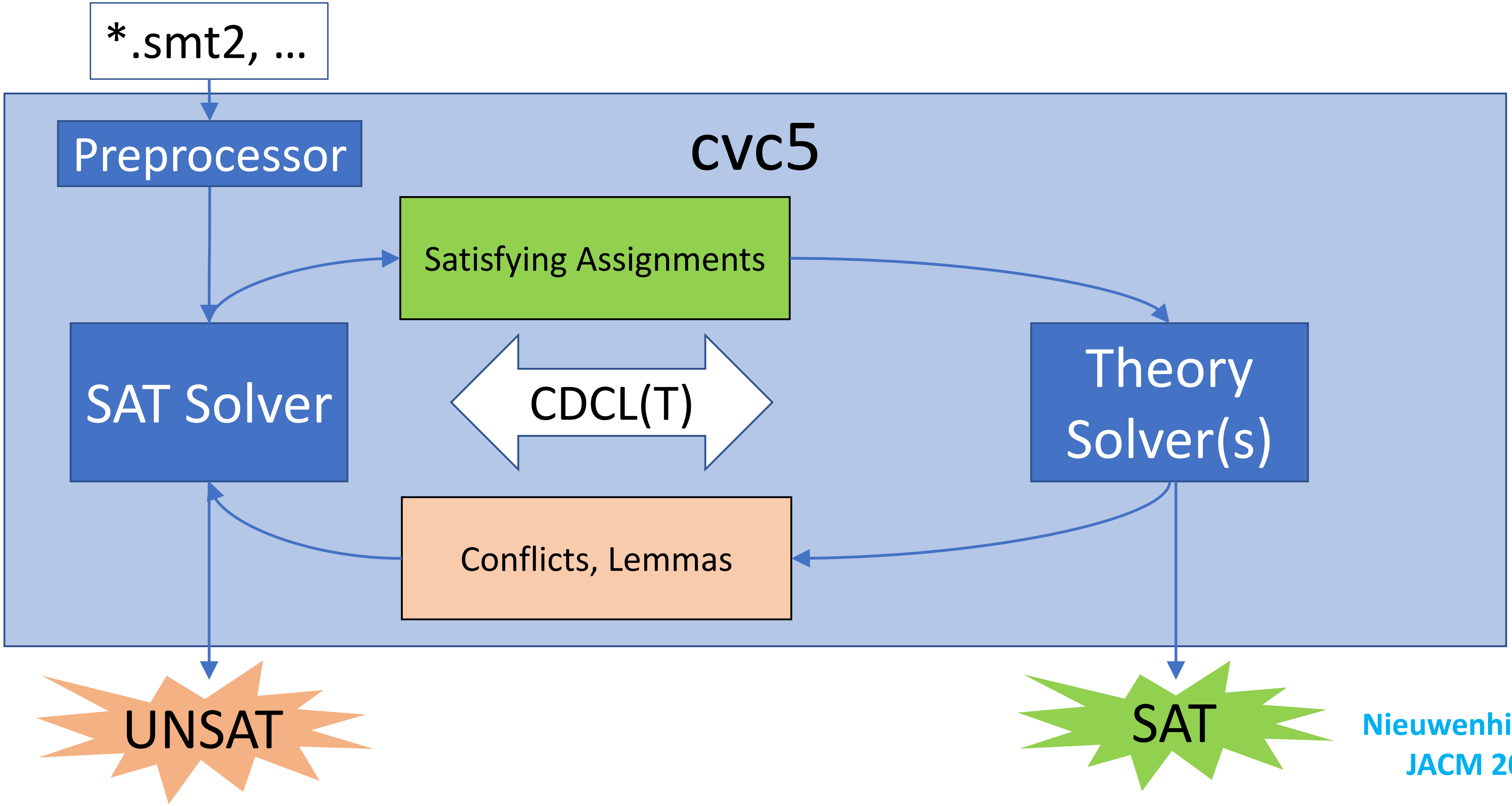
# cvc5: state-of-the-art SMT solver *for Strings*

- **Fast** solving techniques
  - CDCL(T)
  - Core calculus for strings and length constraints
  - Extensions to extended functions and regular expressions

- **Trustworthy** results
  - Proof production for the full theory of strings

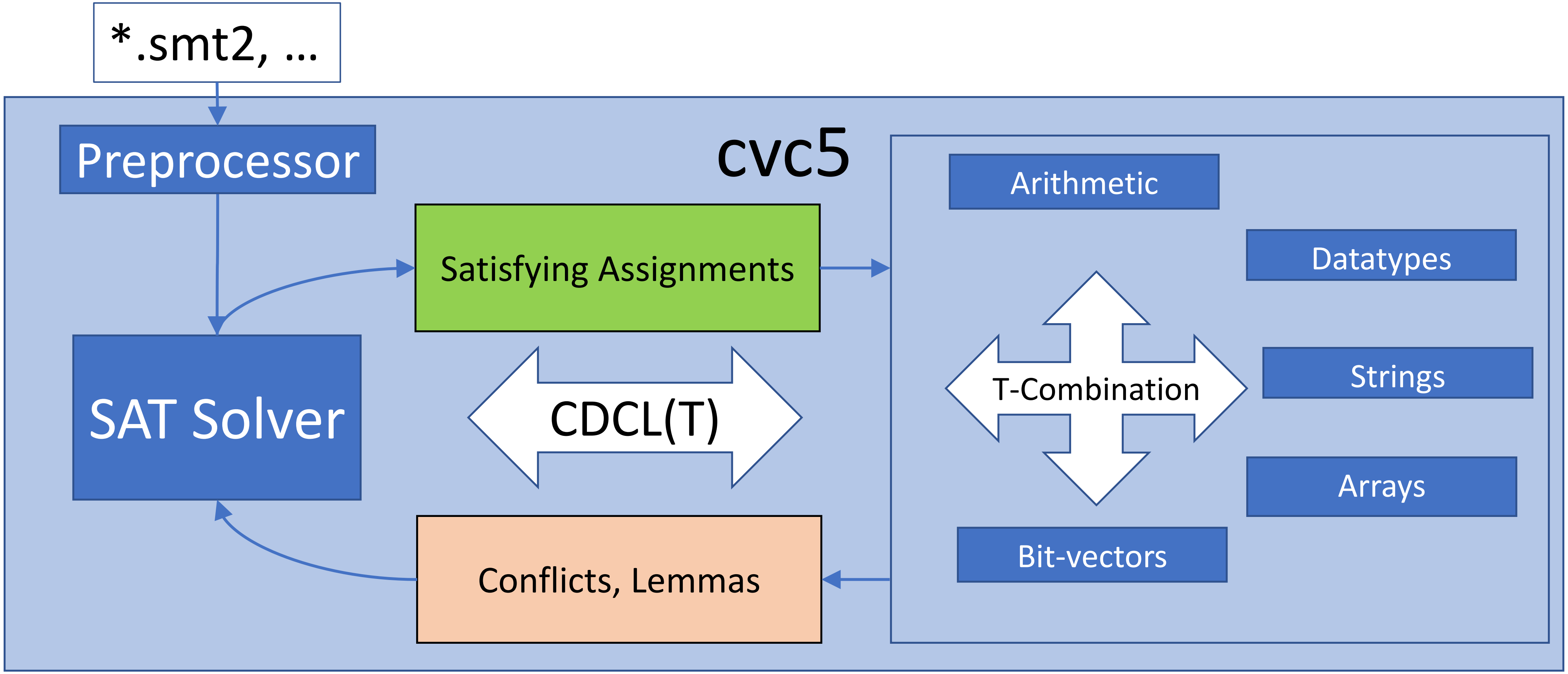
# Designing a **Fast** String Solver

# Architecture of cvc5



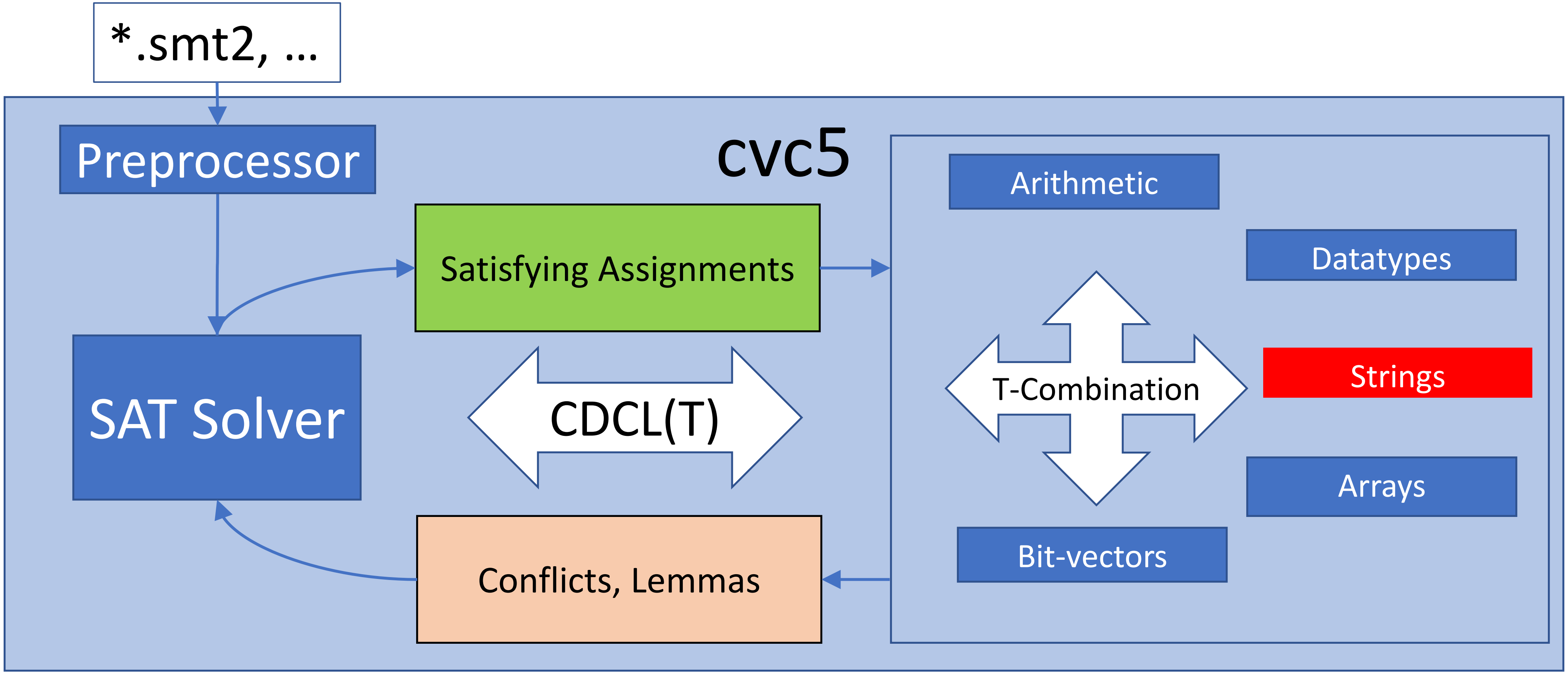

Nieuwenhius et al
JACM 2006

# Architecture of cvc5



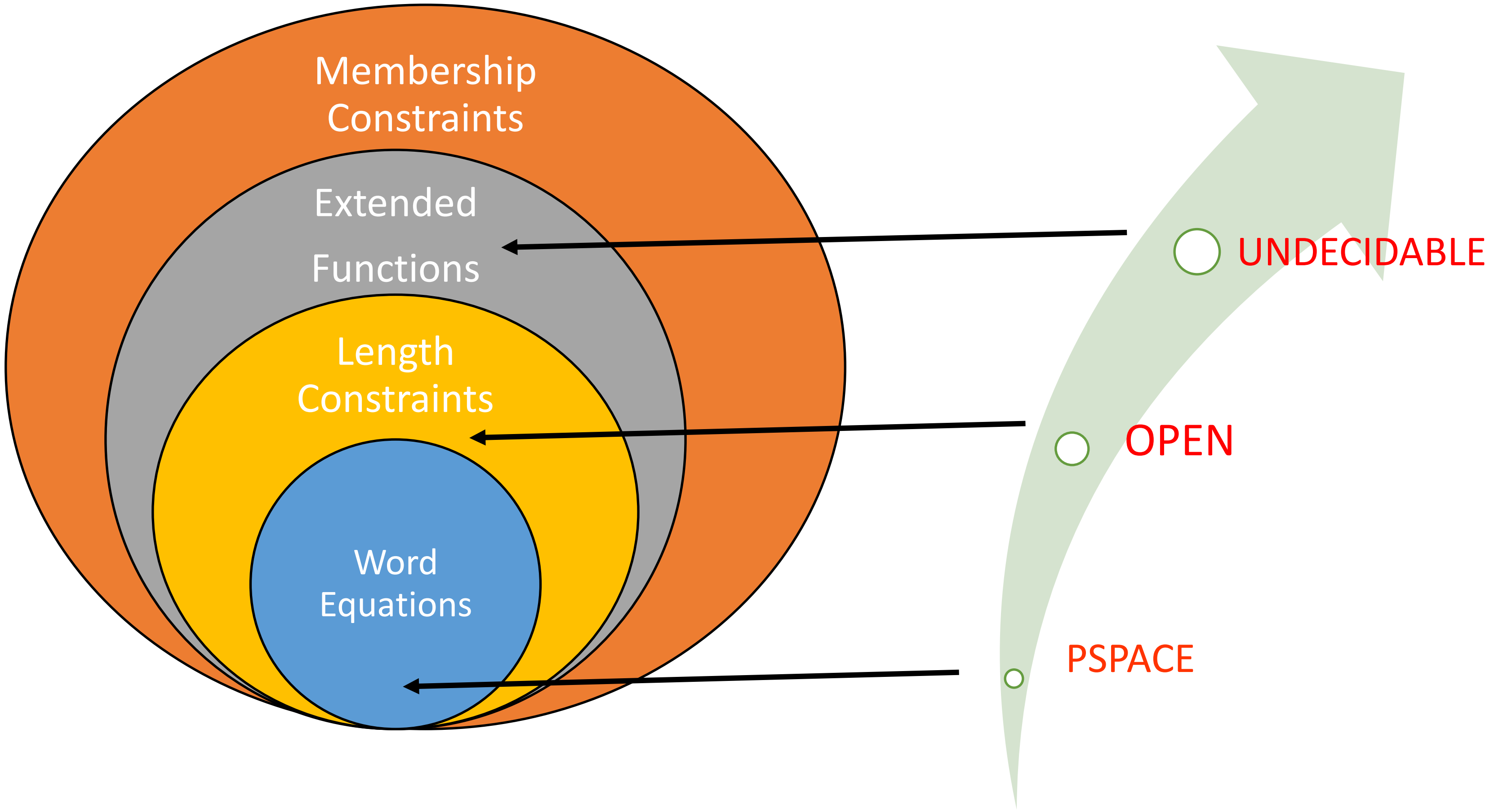

- Centralized methods (Nelson-Oppen, polite) for combining theories

# Architecture of cvc5




- Focus of this talk: solver for the *theory of strings and regular expressions*

# Strings and RegExp: Theoretical Challenges




- Many applications require *extended string functions* and *RegEx memberships*
  - `ctn(x,“a”), to_lower(x)=“abc”, x∈range(“A”,”Z”)`

# SMT Solvers for Strings: Timeline

| 2009 | 2012 | 2014 | 2017 | 2020 | 2022 |
|---|---|---|---|---|---|
| CVC4 project begins | First release CVC4 1.0 | CVC4 support for strings + RE | Support for Extended String Functions | Support For Proofs | cvc5 1.0 release |

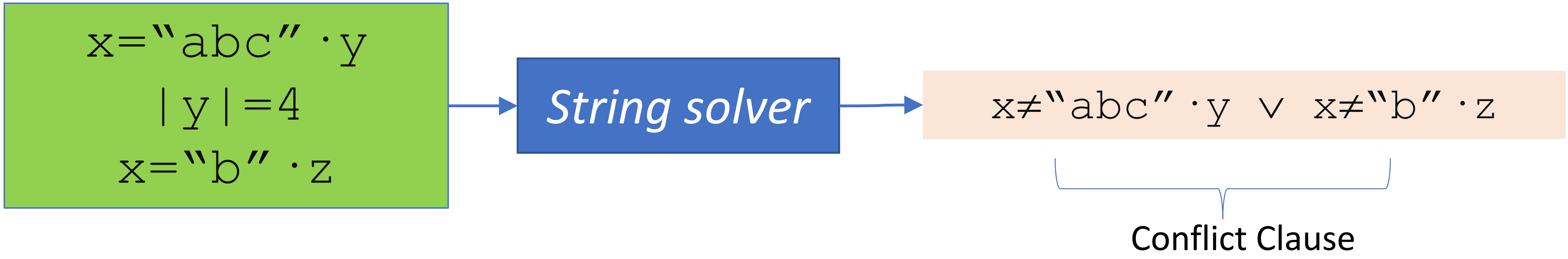
**CVC papers on strings**

- [Liang et al CAV 14]
- [Reynolds et al CAV 17]
- [Reynolds et al CAV 19]
- [Reynolds et al FMCAD 20]
- [Reynolds et al IJCAR 20]
- [Noetzli et al CAV 22]
- [Noetzli et al FMCAD 22]

**Other SMT String Solvers**

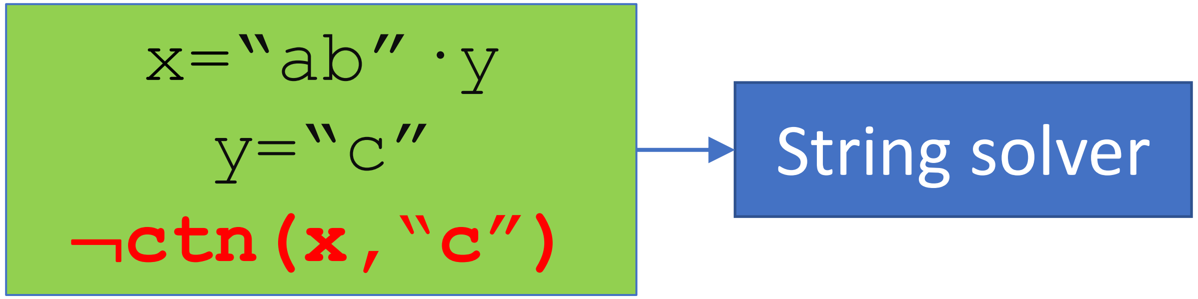
Z3Str, S3, Norn, Z3Str2, Z3Seq, Trau, Z3Str3, Z3Str4, OSTRICH, ....

# A Theory Solver for Strings [Liang et al, CAV 14]




- Designed a string solver for concat+length that is:
    - **Refutation and model sound** ("unsat" and "sat" can be trusted)
    - **Not terminating** in general
    - **Efficient** in practice

# *Extended* Theory of Strings [Reynolds et al CAV17]




- Support *extended* string functions commonly used in applications
  - `substr(x,n,m)` substring of `x` at position `n` of length at most `m`
  - `ctn(x,y)` true if `x` contains substring `y`
  - `indexof(x,y,n)` position of `y` in `x` starting from position `n`
  - `replace(x,y,z)` result of replacing first occurrence of `y` in `x` by `z`

- For example: **¬ctn(x,“c”)** denotes `x` does not contain substring “c”

# *Extended* Theory of Strings [Reynolds et al CAV17]

$x = \text{"ab"} \cdot y$
$y = \text{"c"}$
$\neg \mathbf{ctn}(x, \text{"c"})$

→ String solver →

Reduction lemma

$\forall 0 \le n < |x| . \text{substr}(x, n, 1) \neq \text{"c"}$

Bound length of $x$

$\text{substr}(x, 0, 1) \neq \text{"c"} \wedge \ldots \wedge \text{substr}(x, n, 1) \neq \text{"c"}$

Reduce substr

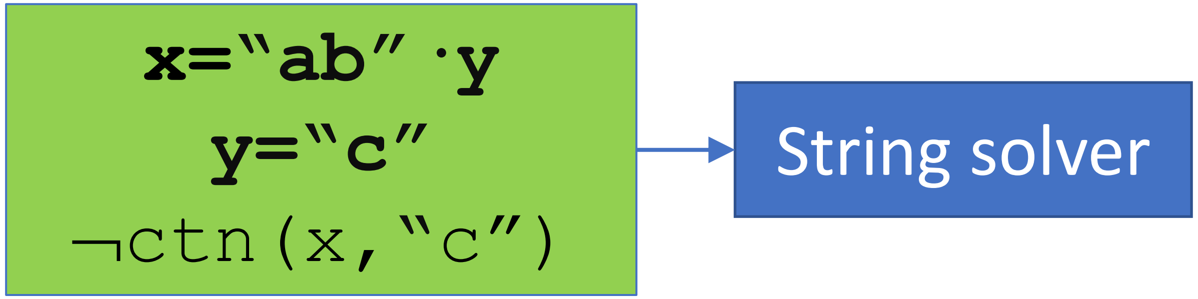
$x = z_{11} \cdot k_1 \cdot z_{21} \wedge$
$|z_{11}| = 0 \wedge$
$k_1 \neq \text{"c"} \wedge$

...

$x = z_{1n} \cdot k_n \cdot z_{2n} \wedge$
$|z_{1n}| = n \wedge$
$k_n \neq \text{"c"}$

- Use *reduction* lemmas
- Expensive:
  Introduces $3 * |x|$ string vars

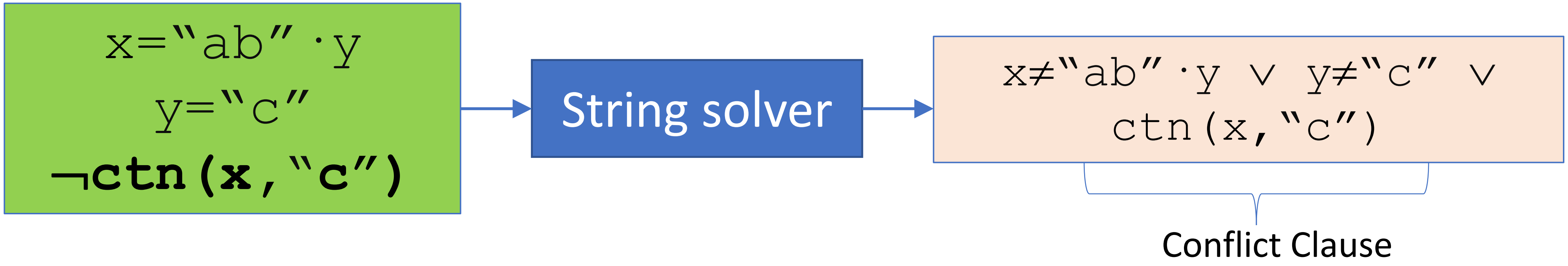
# Context-Dependent Simplification [Reynolds et al CAV17]




- Alternatively: use *context-dependent* simplification:

$$x=\text{“ab”}\cdot y \;\wedge\; y=\text{“c”} \;\models\; x=\text{“abc”}$$

# Context-Dependent Simplification [Reynolds et al CAV17]




- Alternatively: use *context-dependent* simplification:

$$x = \text{``ab''} \cdot y \wedge y = \text{``c''} \models x = \text{``abc''}$$

  - Thus:

$$\neg ctn(x, \text{``c''})\{x \rightarrow \text{``abc''}\} \Leftrightarrow \neg ctn(\text{``abc''}, \text{``c''}) \Leftrightarrow \bot$$

By substitution

By rewriting

# Recent Developments for Theory of Strings

- Context-dependent simplifications
  - Use aggressive rewriting **[Reynolds et al CAV 2019]**
  - Applied eagerly **[Noetzli et al CAV 2022]**

- Reduction lemmas
  - Leverage String-to-code point (`code`) conversion **[Reynolds et al IJCAR 2020]**
  - Improved encodings **[Reynolds et al FMCAD 2020]**
  - Applied lazily based on model **[Noetzli et al CAV 2022]**

# Rewriting based on High-Level Abstractions

- Unlike arithmetic:

`x+x+7*y=y-4` → `2*x+6*y+4=0`

…rewrite rules for strings are *highly non-trivial*:

`ctn("abcde","b"· x· "a")` → `⊥`

`substr(x· "abcd",1+len(x),2)` → `"bc"`

`indexof("abc"·x,"d"·x,0)` → `-1`

- Used syntax-guided synthesis to search for rewrite rules
  - Wrote 3000+ new LOC (C++) in cvc5's string rewriter module

**Reynolds et al CAV19**

# Rewriting based on High-Level Abstractions

- Rules based on high-level abstractions
  - Strings as #characters (e.g. reasoning about their length):

    `ctn(substr(x,i,j),x·“a”)` → $\bot$ …since the second argument is longer than the first

  - Strings as elements in containment lattice:

    `ctn(x·y,substr(x,i,j))` → $\top$ …since `x·y` contains `x`, which contains the second argument

  - Strings as multisets of characters:

    `x·x·y·“ab”= x·“bbbbbb”·y` → $\bot$ …since LHS contains at least 1 more occurrences of `“a”`

$\Rightarrow$ With more rewrites, context-dependent simplification applies *more often*

# A Decision Procedure for Code Points

- Even with aggressive simplification, still require reductions
  - Many extended function reductions require reasoning about characters
- **Idea**: extend core solver for strings to reason about *code points*
  - Assume ordering on characters of alphabet $\mathcal{A}$:
    - $c_1 < \ldots < c_{|\mathcal{A}|-1}$ where for each $c_i$, we call $i$ its code point
  - $\texttt{code} : \texttt{Str} \rightarrow \texttt{Int}$ is interpreted as:
    1. For $w$ in $\mathcal{A}^1$, $\texttt{code}(w)$ is the code point of the single character in $w$
    2. For all other $w$, $\texttt{code}(w)$ is $-1$
- Fragment with string length + string code point (w/o concatenation):
  - Procedure is **sound, complete, terminating**
  - Can be combined modularly with the existing string solver

**Reynolds et al IJCAR20**

# A Decision Procedure for Code Points

- *More efficient reductions* that leverage `code`, including:

- Conversion between strings and integers `to_int(x)`:
  - ⊗ `ite(x[i]="9",9,ite(x[i]="8",8, … ite(x[i]="0",0,-1)…)`
  - ⇒ `ite(48≤`**code**`(x[i])≤57,`**code**`(x[i])-48, -1)`

…note `48` is Unicode for "`0`"

- Regular expression ranges $x \in range(c_1, c_2)$:
  - ⊗ $len(x)=1 \wedge (x=c_1 \vee \ldots \vee x=c_2)$
  - ⇒ $\mathbf{code}(c_1) \leq \mathbf{code}(x) \leq \mathbf{code}(c_2)$

  - Similar for conversions to lower/upper case, lexicographic ordering

⇒ Reasoning about code points is deferred to cvc5's linear arithmetic solver

# Revisiting Reductions for Strings

- **Observation:** there exist equivalent ways of expressing the same constraint
  - For strings `x`,`y`:

$$\exists z. x=z \cdot y \wedge \mathrm{len}(z)=1$$

$$\mathrm{substr}(x,1,\mathrm{len}(x)-1)=y$$

$$x \in \Sigma \cdot \mathrm{to\_re}(y)$$

… `y` is the result of removing the first character from `x`

- **Idea:** *reuse variables* in extended functions and regular expression reductions

**Reynolds et al FMCAD20**

# Revisiting Reductions for Strings

- Reduction for $\texttt{substr(x,1,n)}$:

  ⇒ $(\texttt{len}(x)>0 \land n>0) \Rightarrow (x=\mathbf{z_1}\cdot z_2\cdot z_3 \land \texttt{len}(\mathbf{z_1})=1 \land \texttt{len}(z_2)\leq n \land \ldots)$

- Map $W$ from variables to "witness form"
  - E.g. $W(\mathbf{z_1})=\texttt{substr}(x,0,1)$, $W(z_2)=\texttt{substr}(x,1,n)$, $W(z_3)=$ …

- Reduction for $x\in\Sigma\cdot\texttt{to\_re}(y)$:

  ⊗ $x=z_4\cdot z_5 \land z_4\in\Sigma \land z_5\in\texttt{to\_re}(y)$

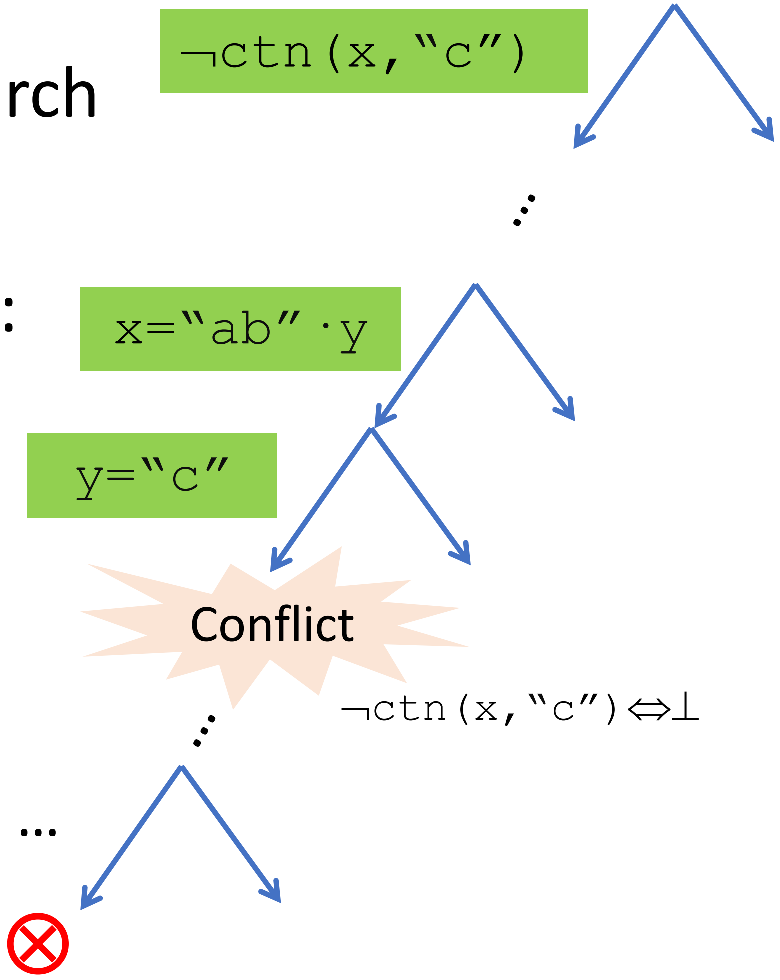
  ⇒ $x=\mathbf{z_1}\cdot z_5 \land \mathbf{z_1}\in\Sigma \land z_5\in\texttt{to\_re}(y)$

  … since first component $z_4$ also corresponds to $\texttt{substr}(x,0,1)$

- Witness forms can leverage rewriting ↓, share variables $z_i$ and $z_j$ when $W(z_i)\downarrow=W(z_j)\downarrow$

# Even *Faster Conflicts* and Lazier Reductions

- **Idea:** apply simplifications eagerly during CDCL(T) search
- Instrument congruence closure to detect conflicts via:
  - Rewriting
  - Inferred properties of equivalence classes
    - Upper/lower bounds for integer equivalence classes
    - Prefix and suffix approximations for string equivalence classes
- Report conflicts as soon as they arise
  - Avoids redundant search space




Noetzli et al CAV22

# Even Faster Conflicts and ***Lazier Reductions***

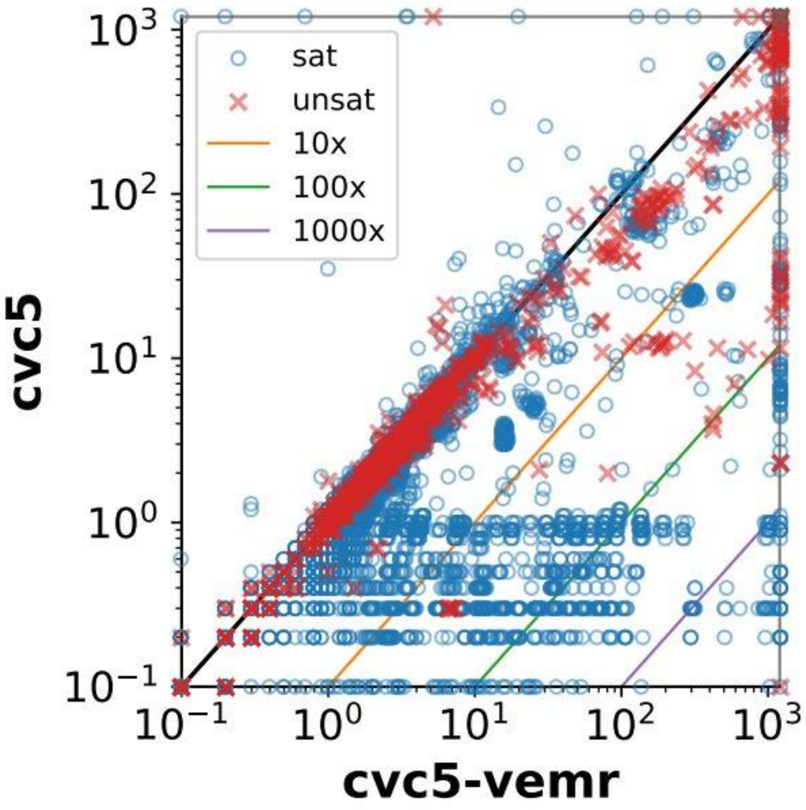
- Avoid reasoning about *unnecessary* reduction lemmas

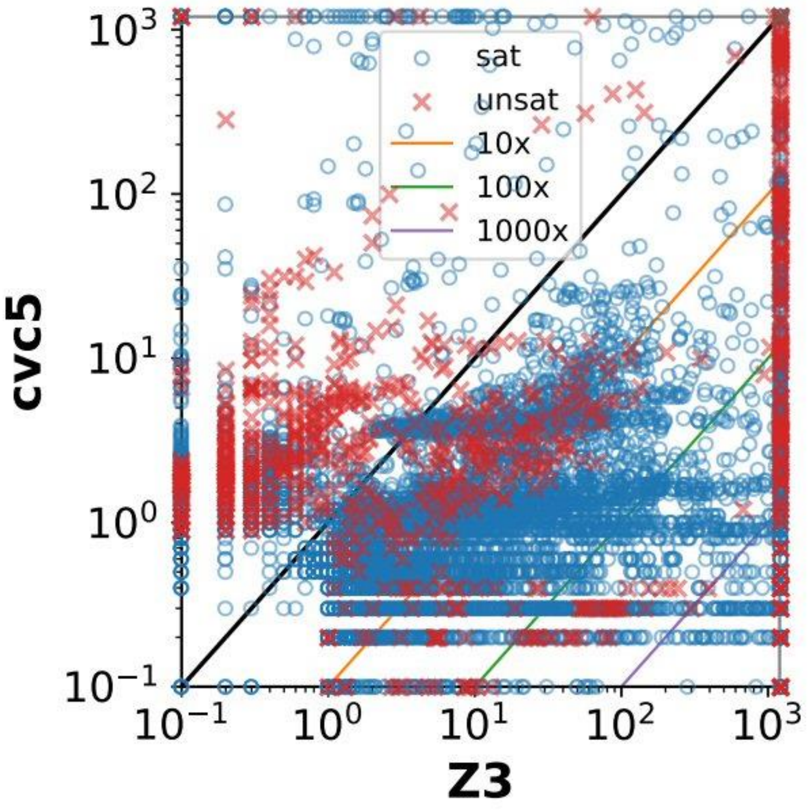
- Regular expression inclusion tests
  - ⊗ E.g. do not reduce $x \in \Sigma* \cdot a \cdot \Sigma*$ if already reduced $x \in \Sigma* \cdot a \cdot \Sigma* \cdot b \cdot \Sigma*$
    - Since $L(\Sigma* \cdot a \cdot \Sigma* \cdot b \cdot \Sigma*) \subseteq L(\Sigma* \cdot a \cdot \Sigma*)$
  - Fast incomplete procedure for language inclusion
  - Can also be used for finding conflicts

- Model-based reductions
  - Construct candidate model $M$
  - ⊗ Do not reduce e.g. string predicates $p$ that are already satisfied by $M$
  - Often, *negative* reg exp memberships are satisfied by current model

**Noetzli et al CAV22**

# Even Faster Conflicts and Lazier Reductions



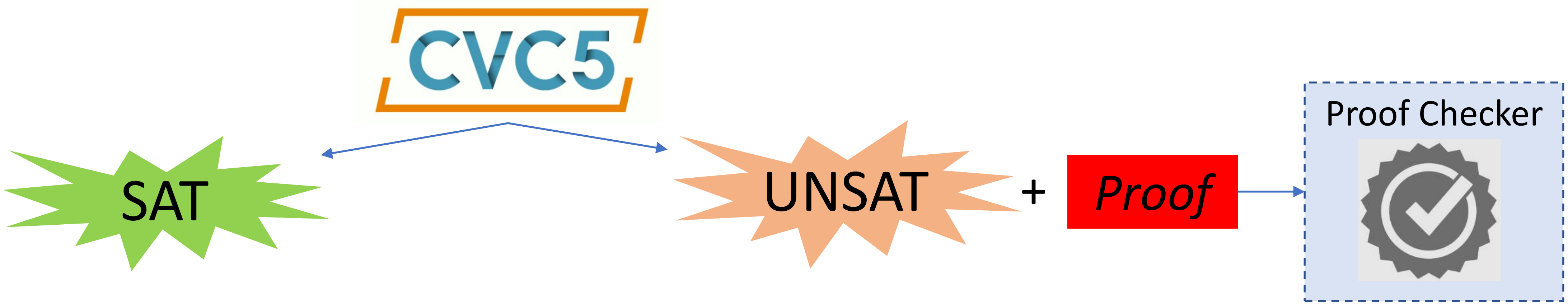

- Results on 10857 SMT-LIB string benchmarks, 1200 second timeout
  - cvc5 solves 10347, z3 solves 8863

Noetzli et al CAV22

# Designing a **Trustworthy** String Solver

# The Need for SMT Proofs

- Correctness of cvc5 is highly critical to applications
  - In particular, refutational soundness
    
    $\Rightarrow$ An incorrect UNSAT response may tell a user a system is safe when it is not!
- cvc5 is a highly complex code base
  - 150k+ LOC, constantly changing with new algorithmic advancements
    
    $\Rightarrow$ Infeasible to verify statically
- Solution: Instrument cvc5 to generate *externally checkable* proofs



# Proofs in cvc5

- Covers many parts of the system
- Evaluated on many SMT-LIB theories

  **Barbosa et al IJCAR22**

**Flexible Proof Production in an Industrial-Strength SMT Solver***

Haniel Barbosa[1], Andrew Reynolds[2], Gereon Kremer[3], Hanna Lachnitt[3], Aina Niemetz[3], Andres Nötzli[3], Alex Ozdemir[3], Mathias Preiner[3], Arjun Viswanathan[2], Scott Viteri[3], Yoni Zohar[4], Cesare Tinelli[2], Clark Barrett[3]

[1] Universidade Federal de Minas Gerais, Belo Horizonte, Brasil
[2] The University of Iowa, Iowa City, USA
[3] Stanford University, Stanford, USA
[4] Bar-Ilan University, Ramat Gan, Israel

- Highly detailed and complete
- Fine-grained proofs for rewrites, for strings

  **Noetzli et al FMCAD22**

**Reconstructing Fine-Grained Proofs of Complex Rewrites Using a Domain-Specific Language**

Andres Nötzli[*], Haniel Barbosa[‡], Aina Niemetz[*], Mathias Preiner[*], Andrew Reynolds[†], Clark Barrett[*], and Cesare Tinelli[†]

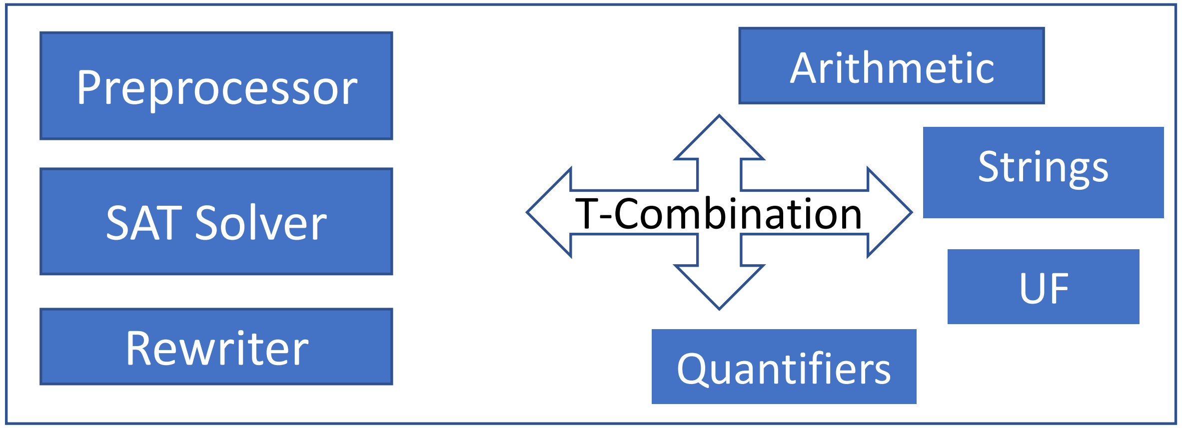
[*]Stanford University, [†]The University of Iowa, [‡]Universidade Federal de Minas Gerais

# Proofs in cvc5: Design Principles

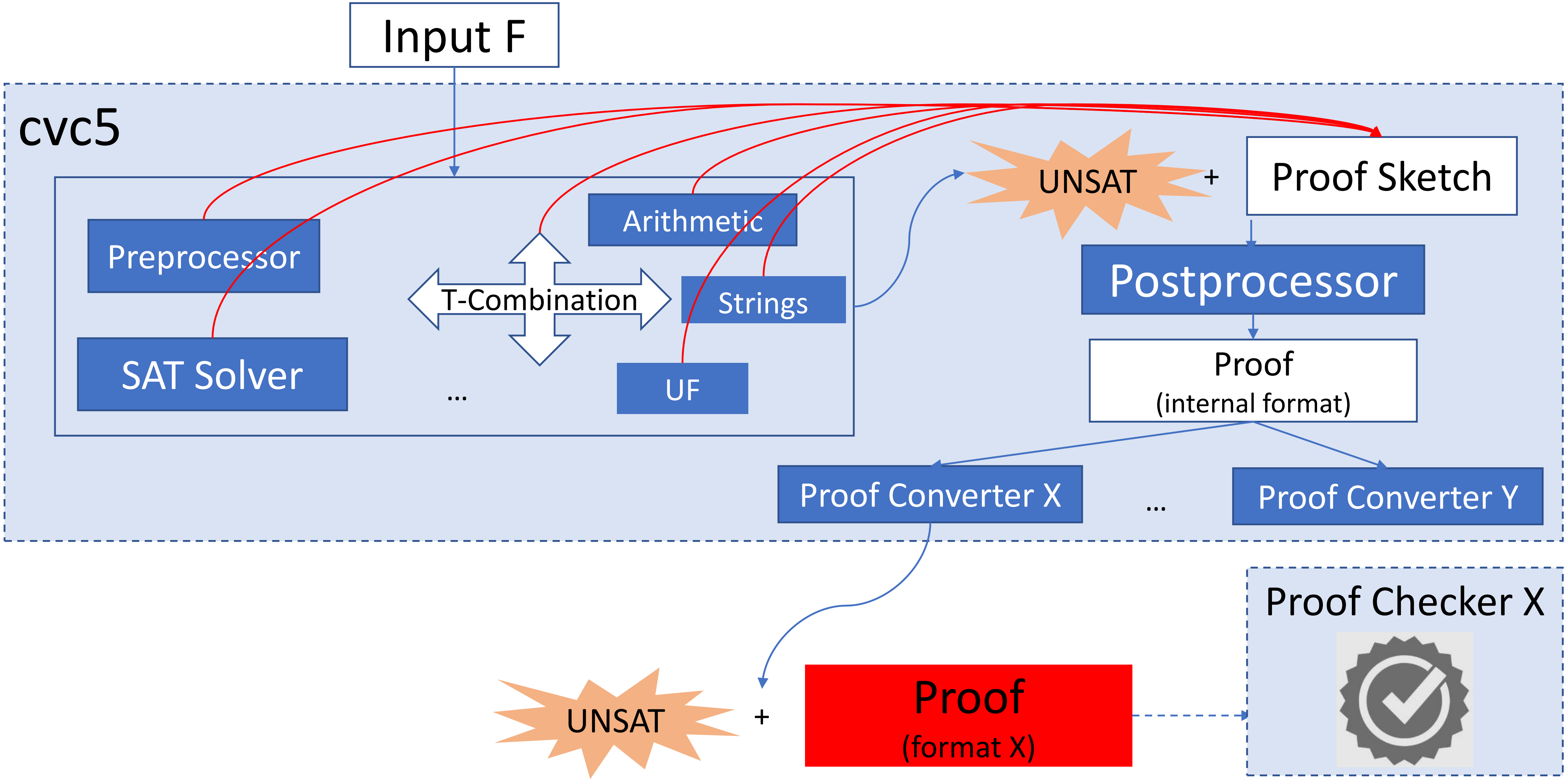
- Flexible
  - Target several backend formats: LFSC, Lean, Alethe, visualization formats
- Also *internally* checkable
  - Use of native proof checker in cvc5 for the purposes of catching errors early
- Provide proofs for all components required for fast solving
  - User should not have to disable features when asking for proofs
- Acceptable performance overhead (~50% performance overhead)
  - Make all optimizations capable of tracking proofs
  - Lazy proof generation

# Instrumenting cvc5 for Producing Proofs for Strings

- String solving involves *many parts of the system*:
  - Preprocessing
  - SAT solver (resolution)
  - CNF conversion
  - Theory Combination
  - UF / Congruence closure
  - Linear Arithmetic Solver
  - Rewriting
  - Quantifier instantiation (for reductions)
  - Strings Theory Solver
    - Core calculus (Liang et al CAV 2014)
    - Extended function reductions
    - Regular expression unfolding



# Proof Architecture



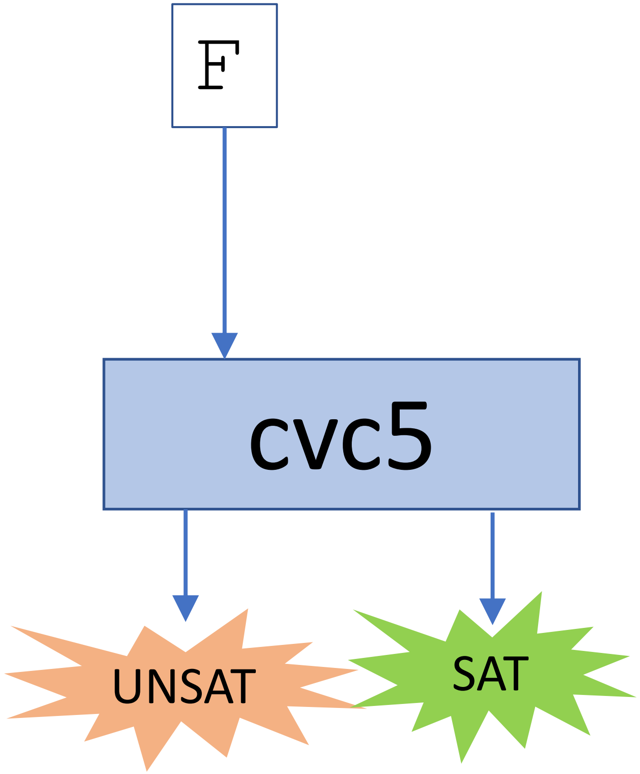
# Future Directions

# String Solving: Better, Faster

- **Better proofs:**
  - Fine-grained proofs for string rewrites
    - User control over granularity
  - Better integration with external proof checkers
  - Modular extraction of parts of the proof (e.g. SAT skeleton, theory lemmas)
- **Faster solver:**
  - Techniques specialized to constraints of interest to applications
  - More advanced solving architectures

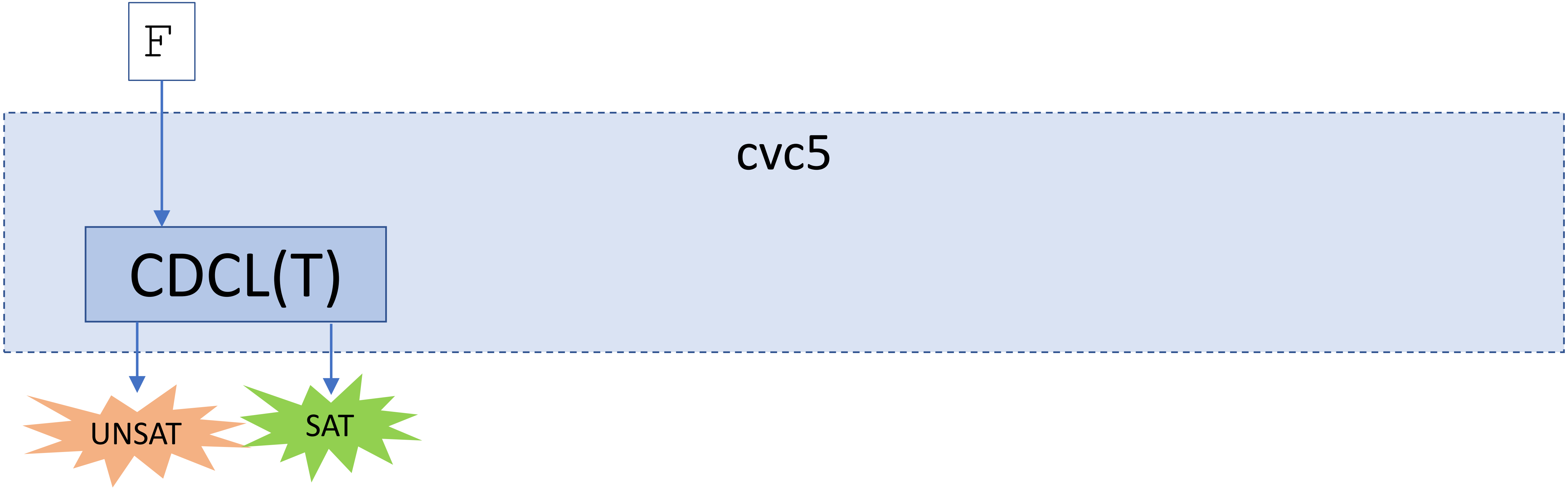
# Advanced Architectures in cvc5

- What if we used the CDCL(T) engine as a black box?

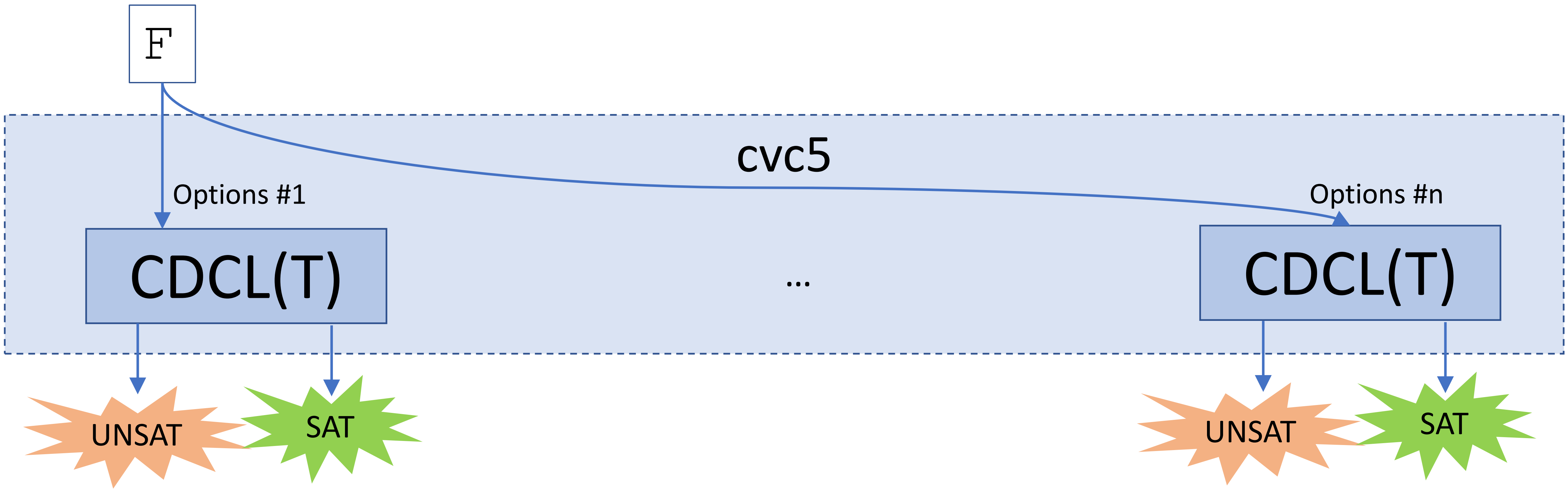


# Advanced Architectures in cvc5

- What if we used the CDCL(T) engine as a black box?

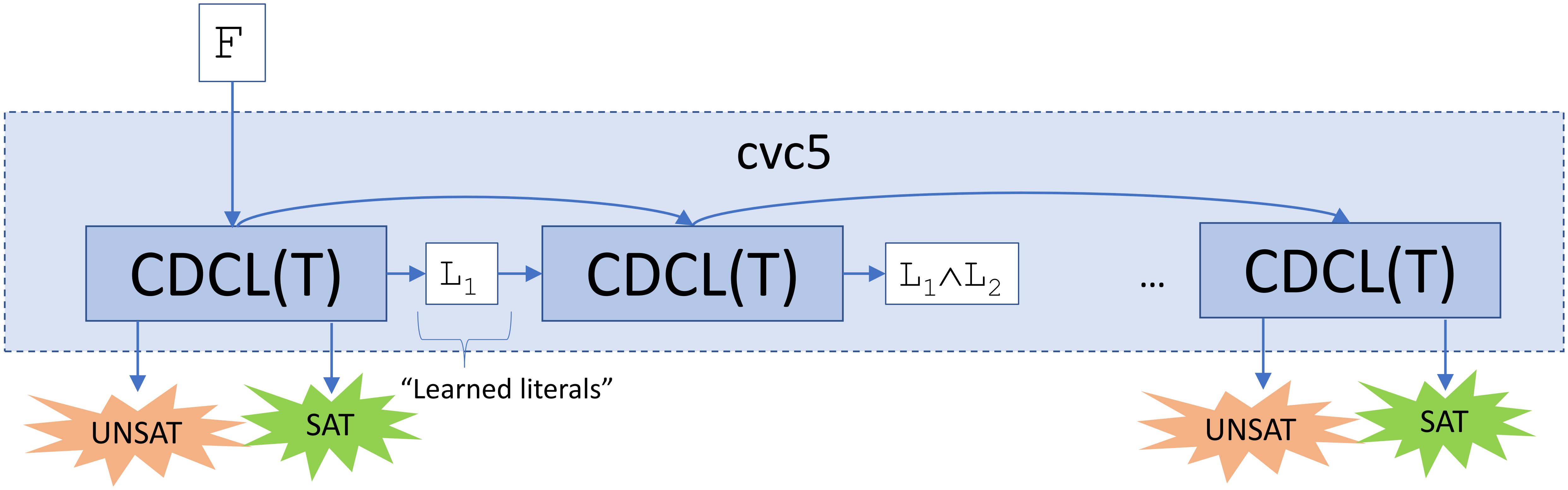


# Advanced Architecture: Portfolio



# Advanced Architecture: Deep Restarts

- Idea: Restart after learning a set of literals that are implied by F



# Deep Restarts

- Given input formula $F$, a learnable literal $l$ is:
  - Meets some syntactic criteria, e.g. $l$ is a literal from $F$
  - Is entailed by input, $F \models_T l$

- Strategy to apply deep restarts based on e.g. time threshold
  - Restart, preprocess, solve again

$\Rightarrow$ Preprocessing after learning may make problem significantly easier

# Deep Restarts: Possible Variants

- Restart while saving other learned formulas?
    - E.g. theory lemmas based on usefulness criteria
- Maintain SAT solver state on restart?
    - Dynamic mapping between SAT and theory literals
- Save state to disk and restart later?
- Only solve for part of the input formula at a time?

# Summary

- SMT solver cvc5 is efficient tool widely used in applications
  - Handles many problem domains
  - State-of-the-art for string solving

- Always looking for new features, faster techniques, increased trust

- Thanks for listening!