Reasoning About Vectors Using an SMT Theory of Sequences

Ying Sheng[1], Andres Noetzli[1], Andrew Reynolds[2], Yoni Zohar[3], David Dill[4], Wolfgang Grieskamp[4], Junkil Park[4], Shaz Qadeer[4], Clark Barrett[1], Cesare Tinelli[2]

[1]Stanford University

[2]The University of Iowa

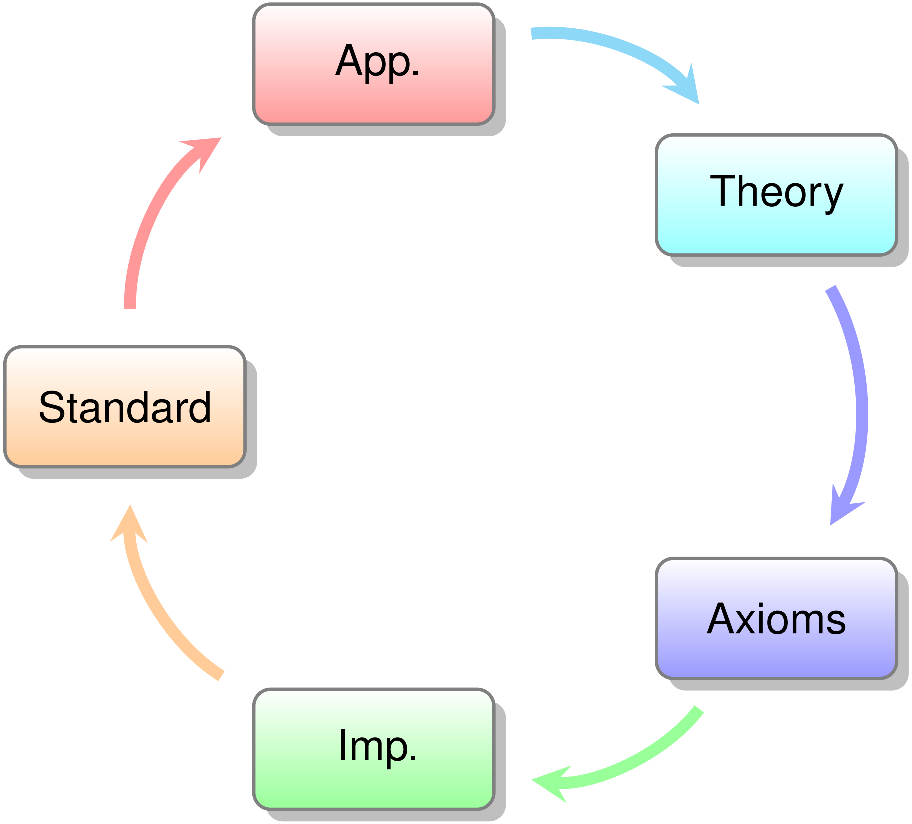
[3]Bar-Ilan University

[4]Meta Novi

SMT 2023

Reasoning About Vectors
Using an SMT Theory of Sequences

The SMT Cycle



App.

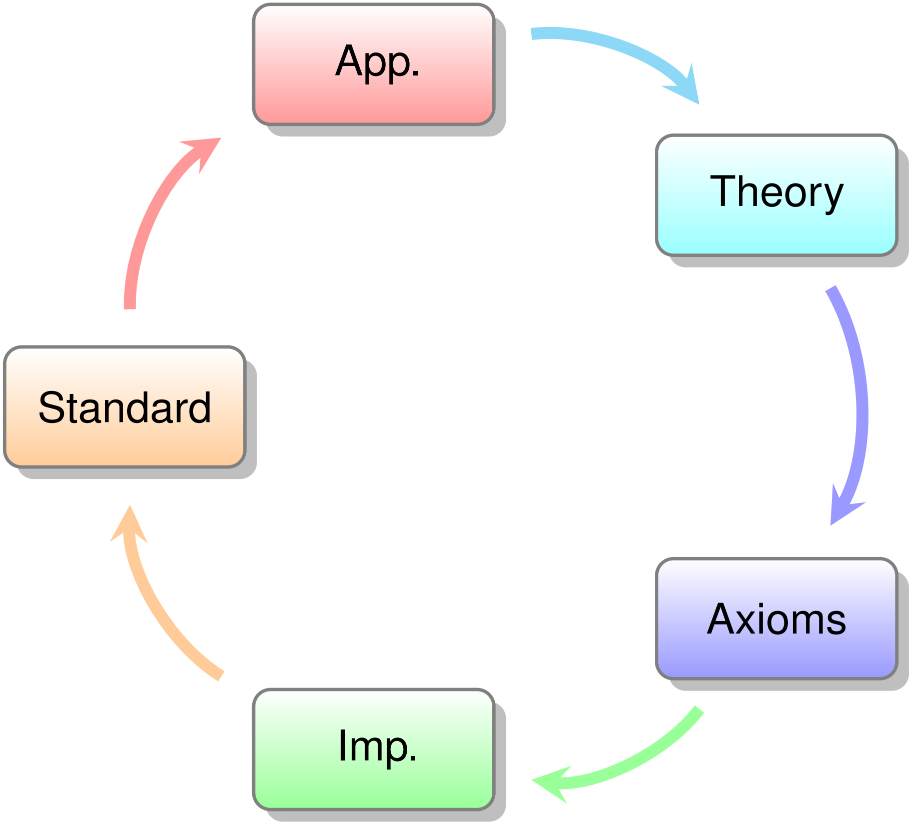
Theory

Axioms

Imp.

Standard

~~The~~ An SMT Cycle



App.

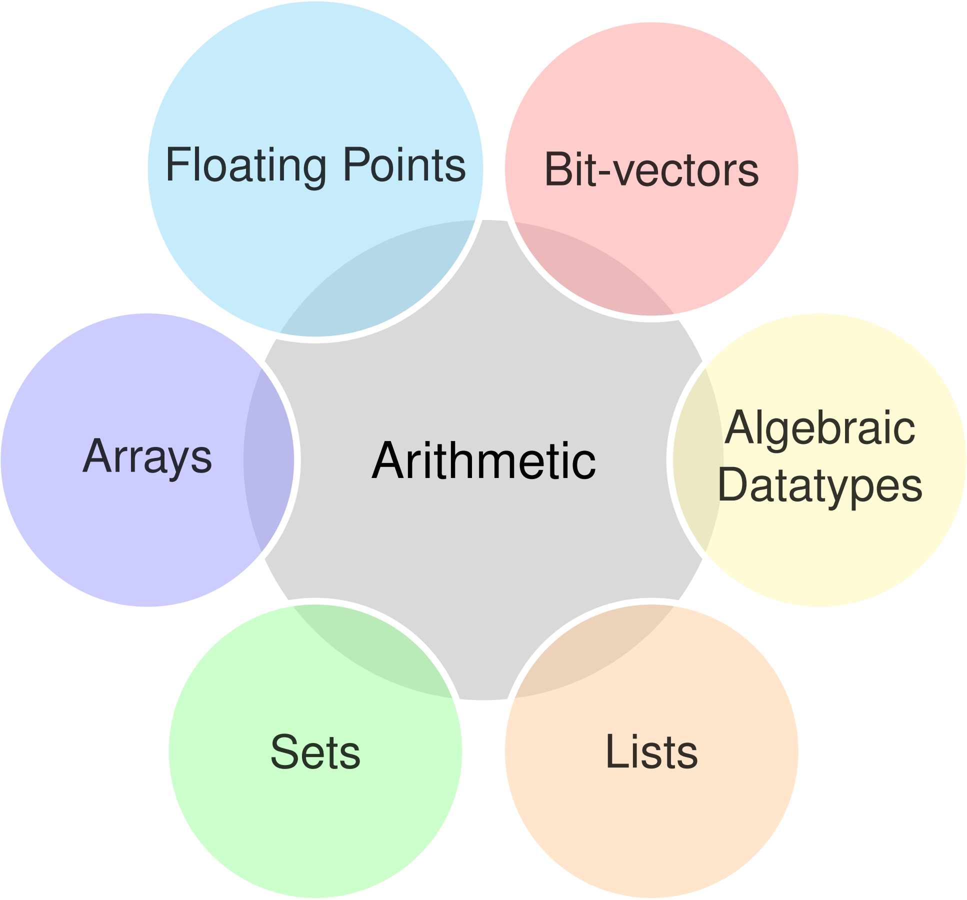
Theory

Axioms

Imp.

Standard

Some Examples



Floating Points

Bit-vectors

Arrays

Arithmetic

Algebraic Datatypes

Sets

Lists

This Time: Sequences

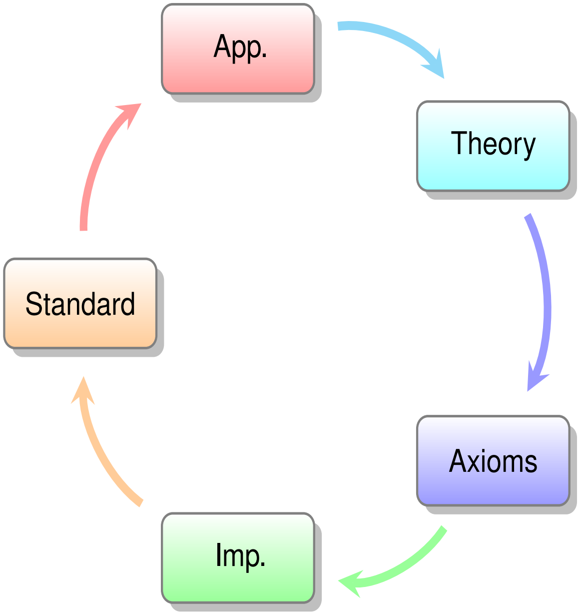
Smart Contract Verification

Sequences

Arrays + Axioms

Imp: This Talk

Future: Standardize



App.

Theory

Axioms

Imp.

Standard

More Background

June 2019: Libra is announced
Move is the Smart Contract Language

July 2020: Move Prover paper published

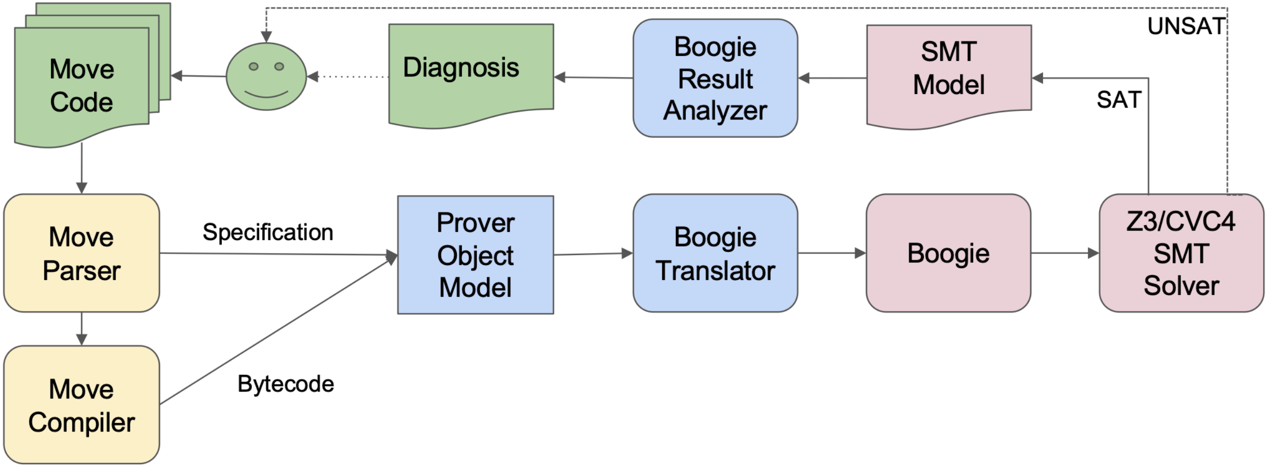
Decmber 2020: name change to Diem

This Work ⟶

January 2022: Libra/Diem shuts down

Today: Move (and Move Prover) lives on

The Move Prover



The Move Prover & Sequences

Applications of Sequences

1. Encoding of the state (tree-shaped)
2. Modeling Move built-in `vector` data structure
3. Used in many programming languages

Desired Properties

1. Expressiveness: Supporting common operations
2. Generality: Generic vectors
3. Efficiency: Fast and efficient reasoning tool

Arrays vs. Sequences

Arrays

- ✗ Expressiveness: Only read and write
- ✓ Generality: The theory of arrays is generic
- ✓ Efficiency: There are efficient procedures

Arrays + Quantifiers

- ✓ Expressiveness: Quantifieres can be used for axiomatization
- ✓ Generality: Can capture general elelemnt sorts
- ✗ Efficiency: Quantifiers reduce efficiency and stability

Sequences (this work)

- ✓ Expressiveness: Designed to support common operators
- ✓ Generality: Completely generic
- ✓ Efficiency: Leverages strings and arrays

Reasoning About Vectors Using an SMT Theory of Sequences

The Theory of Sequences

Arithmetic Operators

Symbol	**Arity**	**SMT-LIB**
n	Int	n
$+$	Int × Int → Int	+
$-$	Int → Int	-
$\leqslant$	Int × Int → Bool	<=

Sequence Operators

Symbol	**Arity**	**SMT-LIB**
ϵ	Seq	`seq.empty`
unit	Elem → Seq	`seq.unit`
$\lvert\_\rvert$	Seq → Int	`seq.len`
nth	Seq × Int → Elem	`seq.nth`
update	Seq × Int × Elem → Seq	`seq.update`
extract	Seq × Int × Int → Seq	`seq.extract`
$\_ \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} \_$	Seq × ⋯ × Seq → Seq	`seq.concat`

Comparison Example

```
// @pre: 0 <= i,j < s.size() and s[i] == s[j]
// @post: s_out == s
void swap(std::vector<int>& s, int i, int j) {
  int a = s[i];
  int b = s[j];
  s[i] = b;
  s[j] = a;
}
```

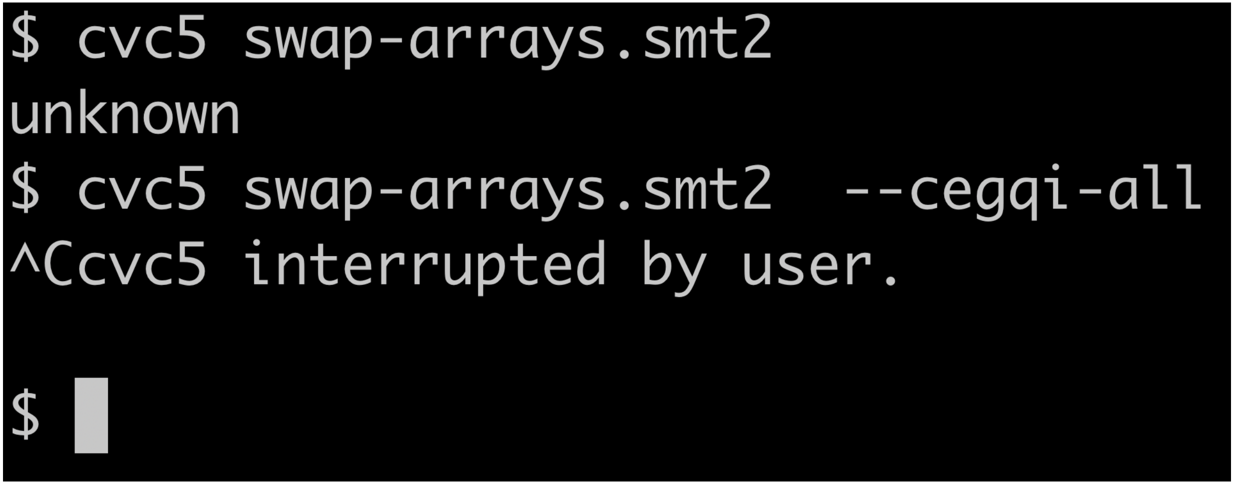
	Arrays
Problem Variables	$a, b, i, j : \text{Int} \qquad s, s_{out} : V$
Auxiliary Variables	$\ell : V \longrightarrow \text{Int} \qquad \mathbf{c} : V \longrightarrow \text{Arr}$ $\approx_{\mathbf{A}} : V \times V \longrightarrow \text{Bool}$ $\text{nth}_{\mathbf{A}} : V \times \text{Int} \longrightarrow \text{Int}$ $\text{update}_{\mathbf{A}} : V \times \text{Int} \times \text{Int} \longrightarrow V$
Axioms	$\forall x, y. x \approx_{\mathbf{A}} y \leftrightarrow (\ell(x) \approx \ell(y) \wedge \forall\, 0 \leqslant i < \ell(x). \mathbf{c}(x)[i] \approx \mathbf{c}(y)[i])$ $\forall x, y, i, a. y \approx_{\mathbf{A}} \text{update}_{\mathbf{A}}(x, i, a) \leftrightarrow (\ell(x) \approx \ell(y) \wedge (0 \leqslant i < \ell(x) \rightarrow \mathbf{c}(y) \approx \mathbf{c}(x)[i \leftarrow a]))$
Program	$a \approx \text{nth}_{\mathbf{A}}(s, i) \wedge b \approx \text{nth}_{\mathbf{A}}(s, j)$ $s_{out} \approx_{\mathbf{A}} \text{update}_{\mathbf{A}}(\text{update}_{\mathbf{A}}(s, i, b), j, a)$
Spec.	$0 \leqslant i, j < \ell(s) \wedge \text{nth}_{\mathbf{A}}(s, i) \approx \text{nth}_{\mathbf{A}}(s, j)$ $\neg s_{out} \approx_{\mathbf{A}} s$

Comparison Example

```
// @pre: 0 <= i,j < s.size() and s[i] == s[j]
// @post: s_out == s
void swap(std::vector<int>& s, int i, int j) {
  int a = s[i];
  int b = s[j];
  s[i] = b;
  s[j] = a;
}
```

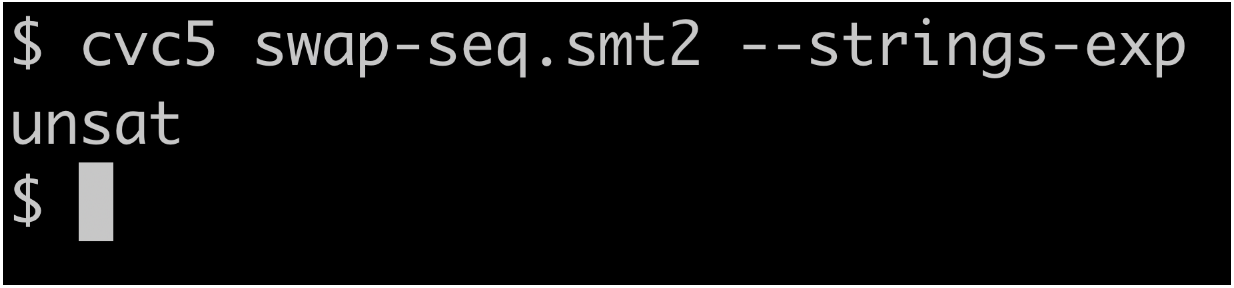
	Sequences	
Problem Variables	a, b, i, j : Int	s, s_{out} : Seq
Auxiliary Variables		
Axioms		
Program	$a \approx \mathrm{nth}(s, i) \wedge b \approx \mathrm{nth}(s, j)$ $s_{out} \approx \mathrm{update}(\mathrm{update}(s, i, b), j, a)$	
Spec.	$0 \leqslant i, j < \|s\| \wedge \mathrm{nth}(s, i) \approx \mathrm{nth}(s, j)$ $\neg s_{out} \approx s$	

Demo



```
$ cvc5 swap-arrays.smt2
unknown
$ cvc5 swap-arrays.smt2  --cegqi-all
^Ccvc5 interrupted by user.

$ 
```



```
$ cvc5 swap-seq.smt2 --strings-exp
unsat
$ 
```

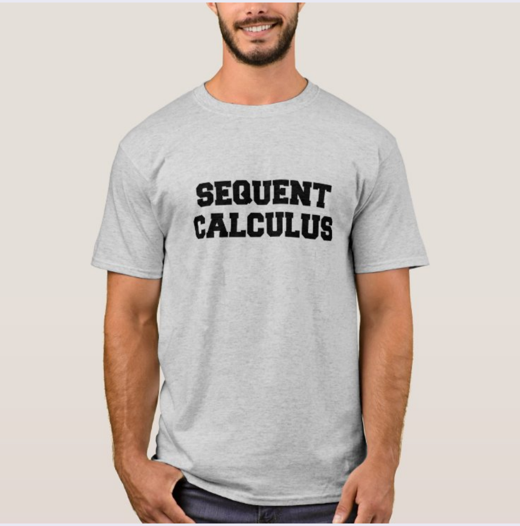
SMT-based Calculi for Sequences

Framework

- Procedures are presented as calculi
- cvc5 implements a strategy
- Not a decision procedure

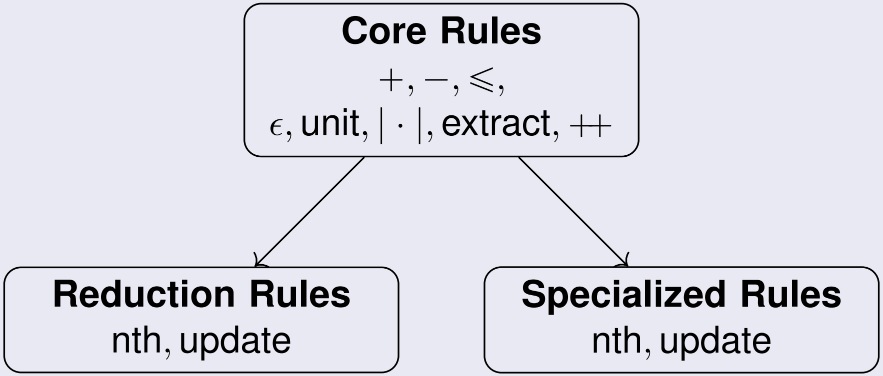
Calculi

- Basic calculus
  - Strings-based Reasoning
  - Lifting characters to arbitrary elements
  - Eliminates nth and update
- Extended calculus
  - Same core rules
  - Array-like reasoning for nth and update



Calculi and Rules

Rules



Core Rules

$+, -, \leqslant,$

ϵ, unit, $|\cdot|$, extract, $+\!\!+$

Reduction Rules

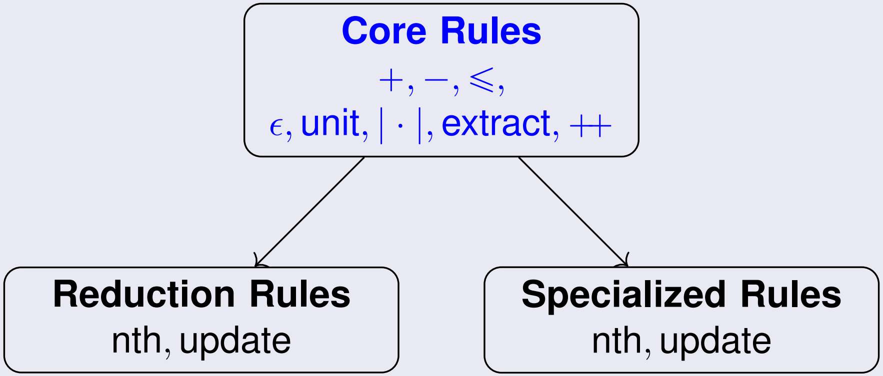
nth, update

Specialized Rules

nth, update

Calculi and Rules

Rules



Core Rules

$+, -, \leqslant,$

ϵ, unit, $|\cdot|$, extract, $+\!\!+$

Reduction Rules

nth, update

Specialized Rules

nth, update

Core Rules

- Adaptation of Strings [CAV'14]
- Simplification

Atithmetic and Equality

- S: Seq-constraints ($\approx, \not\approx$)
- A: Arith-constraints (Arith terms)
- $\ell_x \approx |x|$

- $\mathcal{T}(\mathsf{S})$ – terms in S
- $\models_{\mathsf{LIA}}$: LIA-entailed
- $\models$: FOL-entailed

Conflicts

$$\text{A-Conf}\ \frac{\mathsf{A} \models_{\mathsf{LIA}} \bot}{\mathsf{unsat}} \qquad\qquad \text{S-Conf}\ \frac{\mathsf{S} \models \bot}{\mathsf{unsat}}$$

Propagation

$$\text{A-Prop}\ \frac{\mathsf{A} \models_{\mathsf{LIA}} s \approx t \quad s, t \in \mathcal{T}(\mathsf{S})}{\mathsf{S} := \mathsf{S}, s \approx t}$$

$$\text{S-Prop}\ \frac{\mathsf{S} \models s \approx t \quad s, t \in \mathcal{T}(\mathsf{S}) \quad s, t \text{ are } \Sigma_{\mathsf{LIA}}\text{-terms}}{\mathsf{A} := \mathsf{A}, s \approx t}$$

$$\text{S-A}\ \frac{x, y \in \mathcal{T}(\mathsf{S}) \cap \mathcal{T}(\mathsf{A}) \quad x, y : \mathsf{Int}}{\mathsf{A} := \mathsf{A}, x \approx y \quad \| \quad \mathsf{A} := \mathsf{A}, x \not\approx y}$$

Reduced Form

- S: Seq-constraints ($\approx, \not\approx$)
- A: Arith-constraints (Arith terms)
- $\ell_x \approx |x|$

- $\mathcal{T}(\mathsf{S})$ – terms in S
- $\models_{\mathsf{LIA}}$: LIA-entailed
- $\models$: FOL-entailed

Rule

$$\text{L-Intro}\ \frac{s \in \mathcal{T}(\mathsf{S}) \qquad s : \mathsf{Seq}}{\mathsf{S} := \mathsf{S}, |s| \approx (|s|){\downarrow}}$$

Rewrites

$|\epsilon| \rightarrow 0$

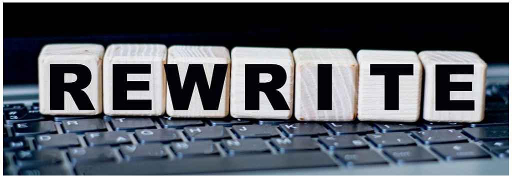
$|\mathsf{unit}(t)| \rightarrow 1$

$|\mathsf{update}(s, i, t)| \rightarrow |s|$

$|s_1 \mathbin{+\!\!+} \ldots \mathbin{+\!\!+} s_n| \rightarrow |s_1| + \ldots + |s_n|$

$\overline{\boldsymbol{u}} \mathbin{+\!\!+} \epsilon \mathbin{+\!\!+} \overline{\boldsymbol{v}} \rightarrow \overline{\boldsymbol{u}} \mathbin{+\!\!+} \overline{\boldsymbol{v}}$

$\overline{\boldsymbol{u}} \mathbin{+\!\!+} (s_1 \mathbin{+\!\!+} \ldots \mathbin{+\!\!+} s_n) \mathbin{+\!\!+} \overline{\boldsymbol{v}} \rightarrow \overline{\boldsymbol{u}} \mathbin{+\!\!+} s_1 \mathbin{+\!\!+} \ldots \mathbin{+\!\!+} s_n \mathbin{+\!\!+} \overline{\boldsymbol{v}}$



Empty Sequence, Unit Sequence, Extentionality

- S: Seq-constraints ($\approx, \not\approx$)
- A: Arith-constraints (Arith terms)
- $\ell_x \approx |x|$

- $\mathcal{T}(\mathsf{S})$ – terms in S
- $\models_{\mathsf{LIA}}$: LIA-entailed
- $\models$: FOL-entailed

Emptyness

$$\text{L-Valid}\ \frac{x \in \mathcal{T}(\mathsf{S}) \quad x : \mathsf{Seq}}{\mathsf{S} := \mathsf{S}, x \approx \epsilon \quad \| \quad \mathsf{A} := \mathsf{A}, \ell_x > 0}$$

Unit

$$\text{U-Eq}\ \frac{\mathsf{S} \models \mathsf{unit}(x) \approx \mathsf{unit}(y)}{\mathsf{S} := \mathsf{S}, x \approx y}$$

Extentionality

$$\text{Deq-Ext}\ \frac{x \not\approx y \in \mathsf{S} \quad x, y : \mathsf{Seq}}{\begin{array}{c}\mathsf{A} := \mathsf{A}, \ell_x \not\approx \ell_y \quad \| \\ \mathsf{A} := \mathsf{A}, \ell_x \approx \ell_y, 0 \leqslant i < \ell_x \quad \mathsf{S} := \mathsf{S}, w_1 \approx \mathsf{nth}(x, i), w_2 \approx \mathsf{nth}(y, i), w_1 \not\approx w_2\end{array}}$$

Extraction

- S: Seq-constraints ($\approx, \not\approx$)
- A: Arith-constraints (Arith terms)
- $\ell_x \approx |x|$

- $\mathcal{T}(\mathsf{S})$ – terms in S
- $\models_{\mathsf{LIA}}$: LIA-entailed
- $\models$: FOL-entailed

Extraction Elimination

R-Extract

$$\frac{x \approx \mathsf{extract}(y, i, j) \in \mathsf{S}}{\begin{array}{c}\mathsf{A} := \mathsf{A}, i < 0 \vee i \geqslant \ell_y \vee j \leqslant 0 \qquad \mathsf{S} := \mathsf{S}, x \approx \epsilon \qquad \| \\ \mathsf{A} := \mathsf{A}, 0 \leqslant i < \ell_y, j > 0, \ell_k \approx i, \ell_x \approx \min(j, \ell_y - i), \ell_{k'} \approx \ell_y - \ell_x - i \\ \mathsf{S} := \mathsf{S}, y \approx k \mathbin{+\!\!+} x \mathbin{+\!\!+} k'\end{array}}$$

Intuition

- extract is eliminated using $\mathbin{+\!\!+}$ and fresh variables
- Main case: $x \approx \mathsf{extract}(y, i, j)$ iff $y \approx k \mathbin{+\!\!+} x \mathbin{+\!\!+} k'$
- Other corner cases are handled

Concatenation: Rules

- S: Seq-constraints ($\approx, \not\approx$)
- A: Arith-constraints (Arith terms)
- $\ell_x \approx |x|$

- $\mathcal{T}(\mathsf{S})$ – terms in S
- $\models_{\mathsf{LIA}}$: LIA-entailed
- $\models$: FOL-entailed

Unifying

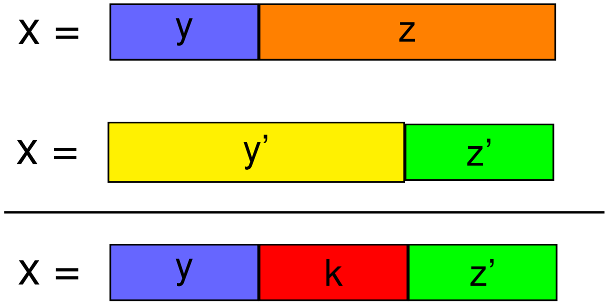
$$\text{C-Eq}\ \frac{\mathsf{S} \models^*_{++} x \approx \overline{\boldsymbol{z}} \qquad \mathsf{S} \models^*_{++} y \approx \overline{\boldsymbol{z}}}{\mathsf{S} := \mathsf{S}, x \approx y}$$

Splitting

$$\text{C-Split}\ \frac{\mathsf{S} \models^*_{++} x \approx (\overline{\boldsymbol{w}} ++ y ++ \overline{\boldsymbol{z}})\!\downarrow \qquad \mathsf{S} \models^*_{++} x \approx (\overline{\boldsymbol{w}} ++ y' ++ \overline{\boldsymbol{z}'})\!\downarrow}{\begin{array}{c} \mathsf{A} := \mathsf{A}, \ell_y > \ell_{y'} \quad \mathsf{S} := \mathsf{S}, y \approx y' ++ k \quad \| \\ \mathsf{A} := \mathsf{A}, \ell_y < \ell_{y'} \quad \mathsf{S} := \mathsf{S}, y' \approx y ++ k \quad \| \\ \mathsf{A} := \mathsf{A}, \ell_y \approx \ell_{y'} \quad \mathsf{S} := \mathsf{S}, y \approx y' \end{array}}$$

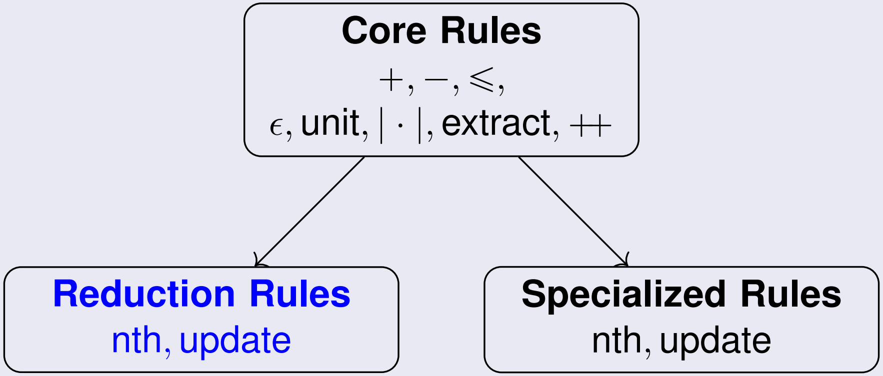
Example of Splitting

$$\text{C-Split}\quad \frac{\mathsf{S} \models^{*}_{+\!+} x \approx (\overline{\boldsymbol{w}} \mathbin{+\!+} y \mathbin{+\!+} \overline{\boldsymbol{z}})\!\downarrow \qquad \mathsf{S} \models^{*}_{+\!+} x \approx (\overline{\boldsymbol{w}} \mathbin{+\!+} y' \mathbin{+\!+} \overline{\boldsymbol{z}'})\!\downarrow}{\begin{array}{c} \mathsf{A} := \mathsf{A}, \ell_y > \ell_{y'} \qquad \mathsf{S} := \mathsf{S}, y \approx y' \mathbin{+\!+} k \quad \| \\ \mathsf{A} := \mathsf{A}, \ell_y < \ell_{y'} := \mathsf{S}, y' \approx y \mathbin{+\!+} k \quad \| \\ \mathsf{A} := \mathsf{A}, \ell_y \approx \ell_{y'} \qquad \mathsf{S} := \mathsf{S}, y \approx y' \end{array}}$$



Calculi and Rules

Rules



Core Rules

$+, -, \leqslant,$

ϵ, unit, $|\cdot|$, extract, $+\!\!+$

Reduction Rules

nth, update

Specialized Rules

nth, update

nth, update

- S: Seq-constraints ($\approx, \not\approx$)
- A: Arith-constraints (Arith terms)
- $\ell_x \approx |x|$

- $\mathcal{T}(\mathsf{S})$ – terms in S
- $\models_{\mathsf{LIA}}$: LIA-entailed
- $\models$: FOL-entailed

Reduction Rules

R-Nth $$\frac{x \approx \mathsf{nth}(y, i) \in \mathsf{S}}{\mathsf{A} := \mathsf{A}, i < 0 \vee i \geqslant \ell_y \quad \| \quad \mathsf{A} := \mathsf{A}, 0 \leqslant i < \ell_y, \ell_k \approx i \qquad \mathsf{S} := \mathsf{S}, y \approx k +\!\!+ \mathsf{unit}(x) +\!\!+ k'}$$

Intuition

- nth and update are eliminated using fresh variables and $+\!\!+$
- Main case: $e \approx \mathsf{nth}(s, i)$ iff $s \approx s +\!\!+ \mathsf{unit} e +\!\!+ s'$
- OOB cases

nth, update

- S: Seq-constraints ($\approx, \not\approx$)
- A: Arith-constraints (Arith terms)
- $\ell_x \approx |x|$

- $\mathcal{T}(\mathsf{S})$ – terms in S
- $\models_{\mathsf{LIA}}$: LIA-entailed
- $\models$: FOL-entailed

Reduction Rules

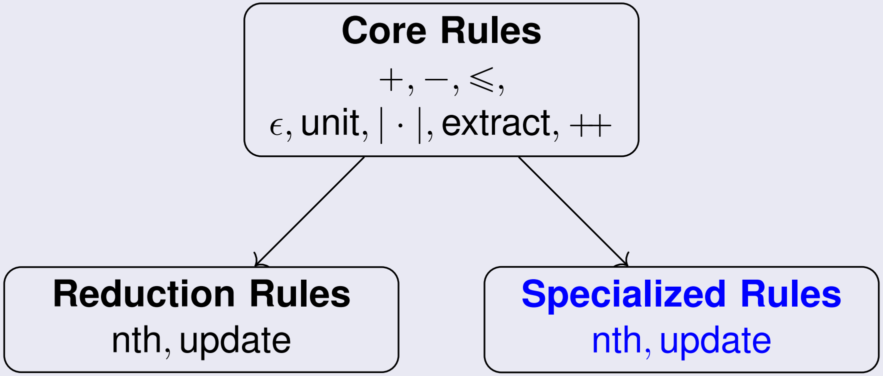
$$\text{R-Update}\ \frac{x \approx \mathsf{update}(y, i, z) \in \mathsf{S}}{\begin{array}{c}\mathsf{A} := \mathsf{A}, i < 0 \vee i \geqslant \ell_y \qquad \mathsf{S} := \mathsf{S}, x \approx y \qquad \| \\ \mathsf{A} := \mathsf{A}, 0 \leqslant i < \ell_y, \ell_k \approx i, \ell_{k'} \approx 1 \quad \mathsf{S} := \mathsf{S}, y \approx k \mathbin{++} k' \mathbin{++} k'', x \approx k \mathbin{++} \mathsf{unit}(z) \mathbin{++} k''\end{array}}$$

Intuition

- nth and update are eliminated using fresh variables and $++$
- Main case: $e \approx \mathsf{nth}(s, i)$ iff $s \approx s \mathbin{++} \mathsf{unit} e \mathbin{++} s'$
- OOB cases

Calculi and Rules

Rules



Core Rules

$+, -, \leqslant,$

ϵ, unit, $|\cdot|$, extract, $+\!\!+$

Reduction Rules

nth, update

Specialized Rules

nth, update

Array Reasoning

- S: Seq-constraints ($\approx, \not\approx$)
- A: Arith-constraints (Arith terms)
- $\ell_x \approx |x|$

- $\mathcal{T}(\mathsf{S})$ – terms in S
- $\models_{\mathsf{LIA}}$: LIA-entailed
- $\models$: FOL-entailed

Read Over Write

Nth-Update

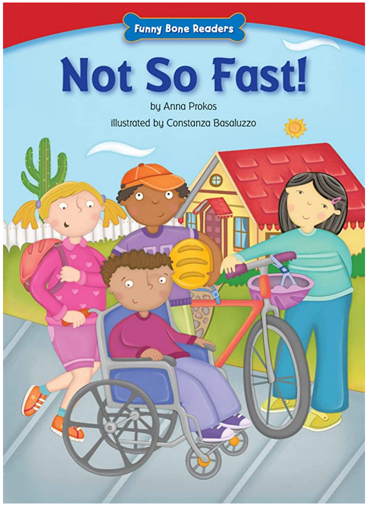
$$\frac{\mathsf{nth}(x, j) \in \mathcal{T}(\mathsf{S}) \qquad y \approx \mathsf{update}(z, i, v) \in \mathsf{S} \qquad \mathsf{S} \models x \approx y \text{ or } \mathsf{S} \models x \approx z}{\begin{array}{c} \mathsf{A} := \mathsf{A}, j < 0 \vee j \geqslant \ell_x \quad \| \\ \mathsf{A} := \mathsf{A}, i \approx j, 0 \leqslant j < \ell_x \qquad \mathsf{S} := \mathsf{S}, \mathsf{nth}(y, j) \approx v \quad \| \\ \mathsf{A} := \mathsf{A}, i \not\approx j, 0 \leqslant j < \ell_x \qquad \mathsf{S} := \mathsf{S}, \mathsf{nth}(y, j) \approx \mathsf{nth}(z, j) \end{array}}$$

Intuition

- Variant of read-over-write axioms
- Takes into account OOB
- Within bounds: updated element changes, the rest remain the same
- Out of bounds: no effect

Not So Fast...

- The core nth and update rule captures their behavior
- Is that enough?
- nth and update interact with other operators as well: unit and $\_ \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} \_$
- These interactions must be modeled as well



Unit + nth + update

- S: Seq-constraints ($\approx, \not\approx$)
- A: Arith-constraints (Arith terms)
- $\ell_x \approx |x|$

- $\mathcal{T}(\mathsf{S})$ – terms in S
- $\models_{\mathsf{LIA}}$: LIA-entailed
- $\models$: FOL-entailed

unit + nth

$$\textsf{Nth-Unit}\ \frac{x \approx \mathsf{nth}(y, i) \in \mathsf{S} \quad \mathsf{S} \models y \approx \mathsf{unit}(u)}{\mathsf{A} := \mathsf{A}, i < 0 \vee i > 0 \quad \| \quad \mathsf{A} := \mathsf{A}, i \approx 0 \quad \mathsf{S} := \mathsf{S}, x \approx u}$$

unit + update

$$\textsf{Update-Unit}\ \frac{x \approx \mathsf{update}(y, i, v) \in \mathsf{S} \quad \mathsf{S} \models y \approx \mathsf{unit}(u)}{\begin{array}{c}\mathsf{A} := \mathsf{A}, i < 0 \vee i > 0 \quad \mathsf{S} := \mathsf{S}, x \approx \mathsf{unit}(u) \quad \| \\ \mathsf{A} := \mathsf{A}, i \approx 0 \quad \mathsf{S} := \mathsf{S}, x \approx \mathsf{unit}(v)\end{array}}$$

$+\!\!+$ + nth + update

- S: Seq-constraints ($\approx, \not\approx$)
- A: Arith-constraints (Arith terms)
- $\ell_x \approx |x|$

- $\mathcal{T}(\mathrm{S})$ – terms in S
- $\models_{\mathrm{LIA}}$: LIA-entailed
- $\models$: FOL-entailed

$+\!\!+$ + nth

Nth-Concat

$$\frac{x \approx \mathrm{nth}(y, i) \in \mathrm{S} \quad \mathrm{S} \models^{*}_{+\!\!+} y \approx w_1 \mathbin{+\!\!+} \dots \mathbin{+\!\!+} w_n}{\begin{array}{c} \mathrm{A} := \mathrm{A}, i < 0 \lor i \geqslant \ell_y \quad \| \quad \mathrm{A} := \mathrm{A}, 0 \leqslant i < \ell_{w_1} \quad \mathrm{S} := \mathrm{S}, x \approx \mathrm{nth}(w_1, i) \quad \| \quad \dots \quad \| \\ \mathrm{A} := \mathrm{A}, \sum\limits_{j=1}^{n-1} \ell_{w_j} \leqslant i < \sum\limits_{j=1}^{n} \ell_{w_j} \quad \mathrm{S} := \mathrm{S}, x \approx \mathrm{nth}(w_n, i - \sum\limits_{j=1}^{n-1} \ell_{w_j}) \end{array}}$$

$+\!\!+$ + nth + update

- S: Seq-constraints ($\approx, \not\approx$)
- A: Arith-constraints (Arith terms)
- $\ell_x \approx |x|$

- $\mathcal{T}(S)$ – terms in S
- $\models_{\text{LIA}}$: LIA-entailed
- $\models$: FOL-entailed

$+\!\!+$ + update

Update-Concat

$$\frac{x \approx \text{update}(y, i, v) \in S \quad S \models^{*}_{+\!+} y \approx w_1 +\!\!+ \dots +\!\!+ w_n}{S := S, x \approx z_1 +\!\!+ \dots +\!\!+ z_n, z_1 \approx \text{update}(w_1, i, v), \dots, z_n \approx \text{update}(w_n, i - \sum_{j=1}^{n-1} \ell_{w_j}, v)}$$

$+\!\!+$ + nth + update

- S: Seq-constraints ($\approx, \not\approx$)
- A: Arith-constraints (Arith terms)
- $\ell_x \approx |x|$

- $\mathcal{T}(\mathsf{S})$ – terms in S
- $\models_{\mathsf{LIA}}$: LIA-entailed
- $\models$: FOL-entailed

$+\!\!+$ + update

Update-Concat-Inv

$$\frac{x \approx \mathsf{update}(y, i, v) \in \mathsf{S} \quad \mathsf{S} \models^{*}_{+\!+} x \approx w_1 +\!\!+ \ldots +\!\!+ w_n}{\mathsf{S} := \mathsf{S}, y \approx z_1 +\!\!+ \ldots +\!\!+ z_n, \qquad w_1 \approx \mathsf{update}(z_1, i, v), \ldots, w_n \approx \mathsf{update}(z_n, i - \sum_{j=1}^{n-1} \ell_{w_j}, v)}$$

Additional Rules

- S: Seq-constraints ($\approx, \not\approx$)
- A: Arith-constraints (Arith terms)
- $\ell_x \approx |x|$

- $\mathcal{T}(\mathrm{S})$ – terms in S
- $\models_{\mathrm{LIA}}$: LIA-entailed
- $\models$: FOL-entailed

Additional Rules

$$\textsf{Nth-Intro}\ \frac{s' \approx \mathrm{update}(s, i, t) \in \mathrm{S}}{\mathrm{S} := \mathrm{S}, e \approx \mathrm{nth}(s, i), e' \approx \mathrm{nth}(s', i)}$$

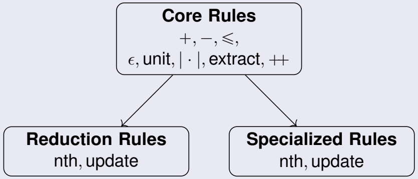
$$\textsf{Update-Bound}\ \frac{x \approx \mathrm{update}(y, i, v) \in \mathrm{S}}{\mathrm{A} := \mathrm{A}, 0 \leqslant i < \ell_y \qquad \mathrm{S} := \mathrm{S}, \mathrm{nth}(y, i) \not\approx v \quad \| \quad \mathrm{S} := \mathrm{S}, x \approx y}$$

$$\textsf{Nth-Split}\ \frac{\mathrm{nth}(x, i), \mathrm{nth}(x', i') \in \mathcal{T}(\mathrm{S}) \quad i \approx i' \in \mathrm{A}}{\mathrm{S} := \mathrm{S}, x \approx x' \quad \| \quad \mathrm{S} := \mathrm{S}, x \not\approx x'}$$

Correctness

Algorithm

1. Partition to Seq and Arith constraints
2. Apply all rules until no rule applies
3. If got 'unsat', return 'unsat'. Else return 'sat'



Termination

The above algorithm does not always terminate (e.g., $x +\!\!+ y \approx y +\!\!+ x$).

Theorem

When the above algorithm terminates, its result is correct.

proof

- unsat: simple local soundness check.
- sat: model construction (weak equivalence + normal forms).

Model Construction – Basics

Sorts

- Elem is some infinite set
- Seq and Int are pre-determined

Theory Symbols

- All but OOB nth are pre-determined
- OOB nth is handled at a later stage

Integer Variables

- There must be some integer model
- Use that model

$$\textcolor{blue}{\text{A-Conf}}\ \frac{A \models_{\text{LIA}} \bot}{\text{unsat}}$$

Element Variables

- Chosen arbitrarily by equality reasoning
- Possible since the domain is infinite

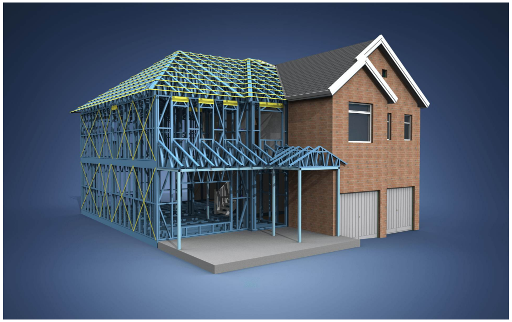
Model Construction – Sequences

Atomic Sequence Variables

- length: According to ℓ_x variables (assigned by the arithmetic model)
- unit variables x: $x \approx \text{unit}(e)$ is entailed and e was already assigned
- non-units: weakly equivalent arrays [Christ & Hoenicke 2015]

Non-atomic Sequence Variables

- Transformation to normal form
- Concatenation of atomic variables (already assigned)



Reasoning About Vectors

Using an SMT Theory of Sequences

Sequences in cvc5

Implementation

- Both calculi are Implemented in cvc5
- Extension of an existing string solver

Benchmarks

- Smart Contracts Verification (558)
  - Real-world
  - Generated by the Move Prover
- Array benchmarks (551)
  - Translated from SMT-LIB QF_AX
  - Only update and nth

Tools

- *cvc5*: Basic calculus + Extended calculus
- Z3

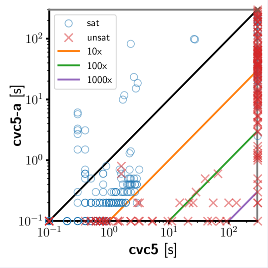
Results

Overall Resutls

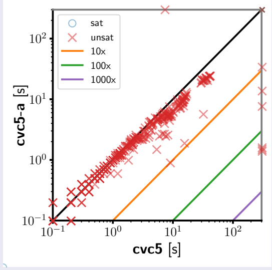
		w/ update		
Set		**cvc5**	**cvc5-a**	**z3**
Arrays (551)	Slvd	242	**390**	170
	Time	162	303	4329
Diem (558)	Slvd	542	**547**	443
	Time	518	440	639

Commonly Solved

Arrays



SC Verification



- cvc5: core calculus + reduction rules
- cvc5-a: core calculus + specialized rules
- Z3: updates were eagerly eliminated
- 300s timeout

- cvc5-a solves the highest number of benchmarks
- Commonly solved: cvc5 5 is usually faster

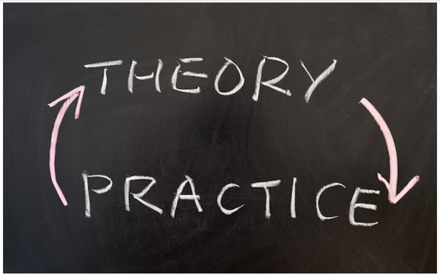
Conclusion – 1

From Theory to Practice

- Got stuck with a case in the proof
- Created a benchmark that represents the case
- Found a wrong model in the implementation

And Back

- Got stuck in another case in the proof
- Checked the implementation
- Found out that it splits more equalities than we have in the calculus
- fixed the calculus and the proof



Conclusion – 2

We Have Seen

- New theory for reasoning about vectors
  - ✓ Expressive
  - ✓ General
  - ✓ Efficient
- Two Calculi
- Evaluation: efficient on real-world + crafted benchmarks

Future Work

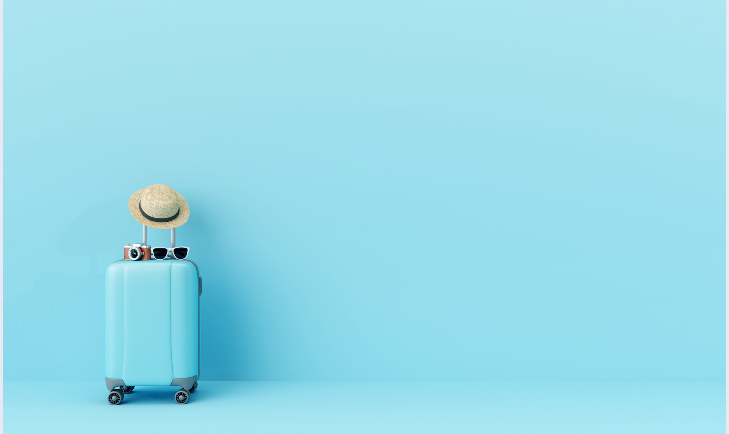
- Standardize
- More array-like reasoning
- Decidability of fragments
- Polite theory combination

Conclusion – 2

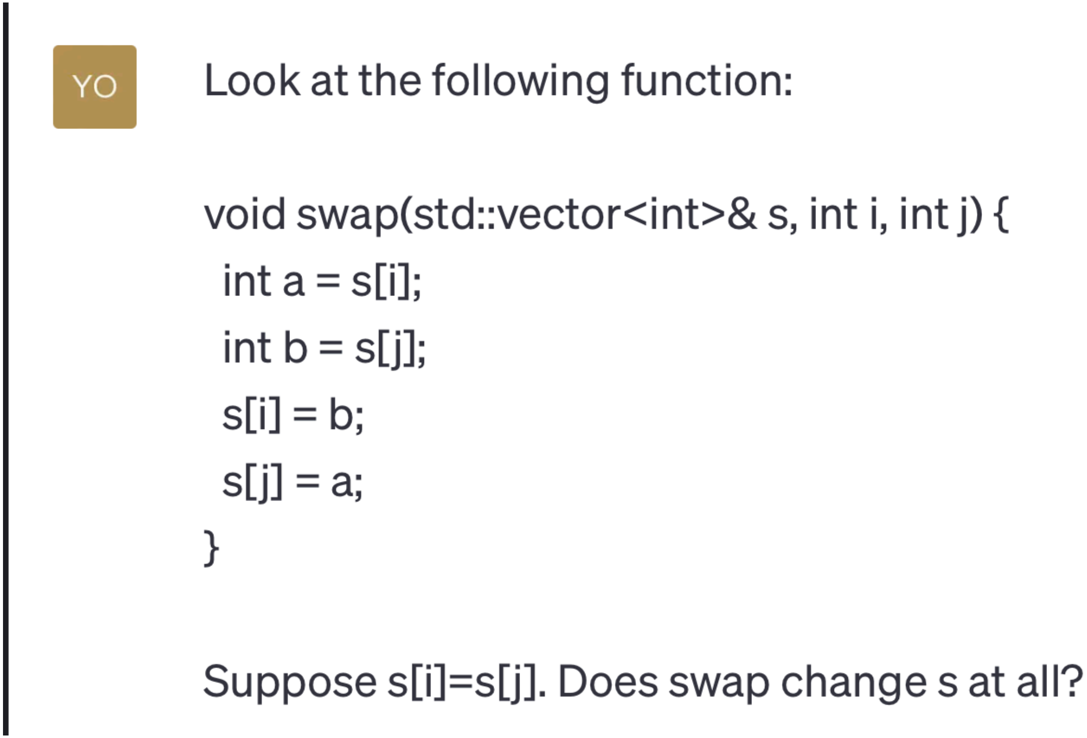
We Have Seen

- New theory for reasoning about vectors
  - ✓ Expressive
  - ✓ General
  - ✓ Efficient
- Two Calculi
- Evaluation: efficient on real-world + crafted benchmarks

Future Work



Relax

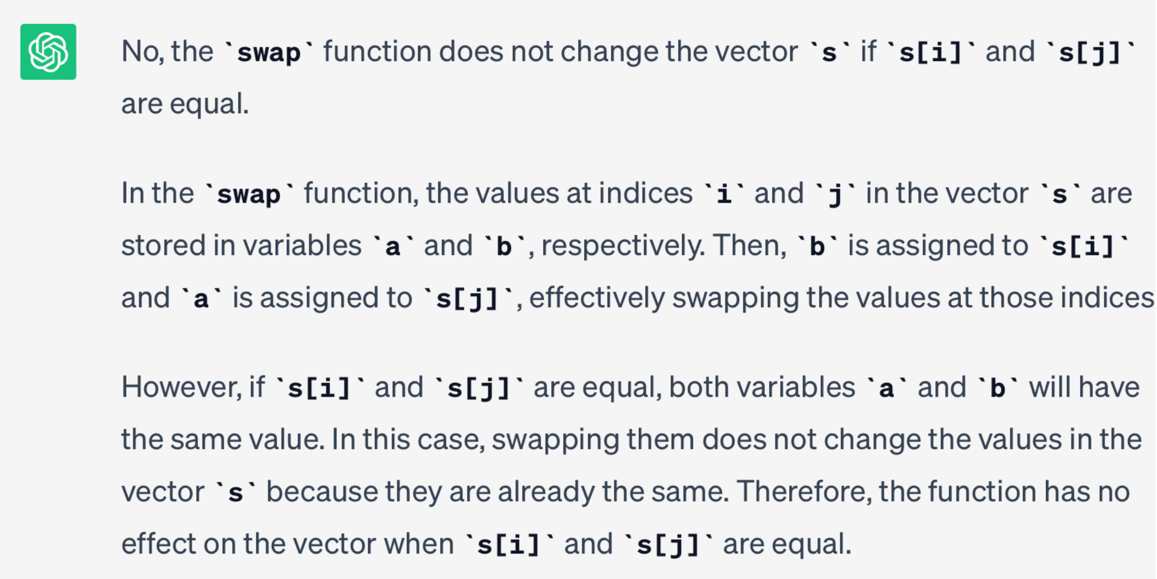


Look at the following function:

```
void swap(std::vector<int>& s, int i, int j) {
  int a = s[i];
  int b = s[j];
  s[i] = b;
  s[j] = a;
}
```

Suppose s[i]=s[j]. Does swap change s at all?

Relax

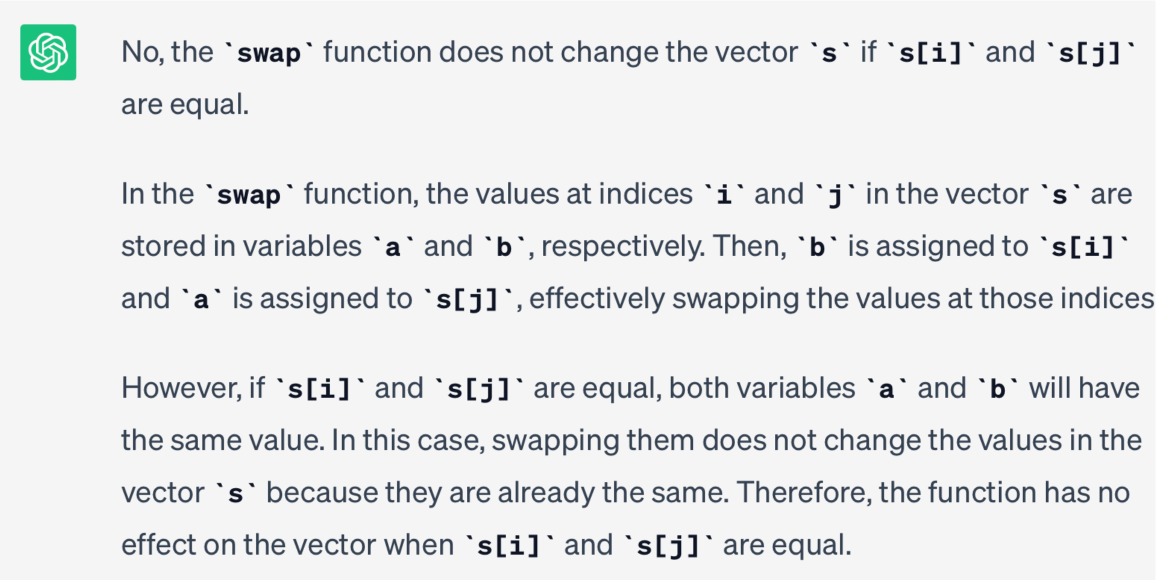


No, the `swap` function does not change the vector `s` if `s[i]` and `s[j]` are equal.

In the `swap` function, the values at indices `i` and `j` in the vector `s` are stored in variables `a` and `b`, respectively. Then, `b` is assigned to `s[i]` and `a` is assigned to `s[j]`, effectively swapping the values at those indices

However, if `s[i]` and `s[j]` are equal, both variables `a` and `b` will have the same value. In this case, swapping them does not change the values in the vector `s` because they are already the same. Therefore, the function has no effect on the vector when `s[i]` and `s[j]` are equal.

Relax



No, the `swap` function does not change the vector `s` if `s[i]` and `s[j]` are equal.

In the `swap` function, the values at indices `i` and `j` in the vector `s` are stored in variables `a` and `b`, respectively. Then, `b` is assigned to `s[i]` and `a` is assigned to `s[j]`, effectively swapping the values at those indices

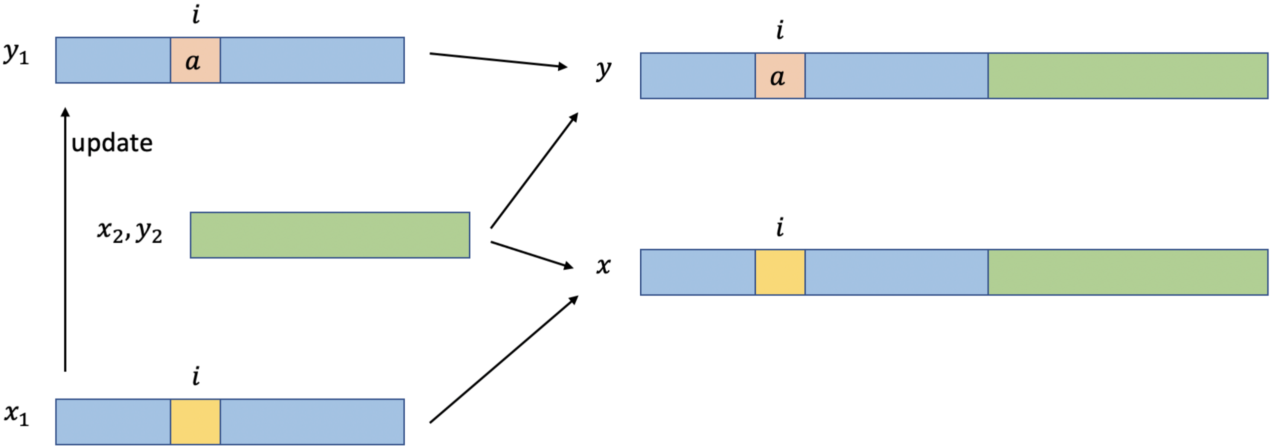
However, if `s[i]` and `s[j]` are equal, both variables `a` and `b` will have the same value. In this case, swapping them does not change the values in the vector `s` because they are already the same. Therefore, the function has no effect on the vector when `s[i]` and `s[j]` are equal.

Thank You!

Back Up Slides

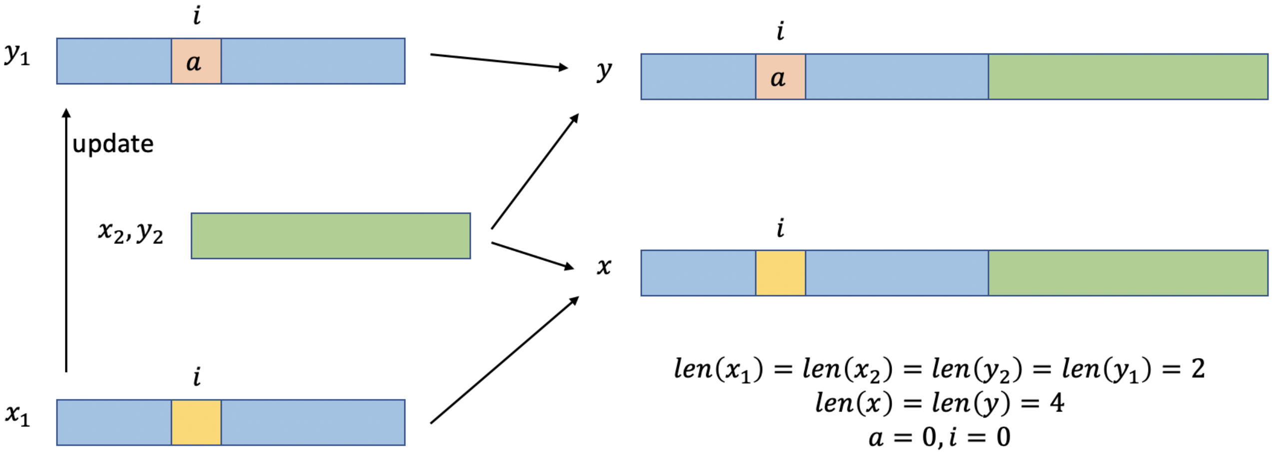
Model Construction - a very simple example

- Initial Configuration:
  $y = \text{update}(x, i, a), y = y_1 +\!\!+\, y_2 \in \mathrm{S}_0$
  $0 \leqslant i < |y_1|, |y_1| > 0, |y_2| > 0 \in \mathrm{A}_0.$
- Saturated Configuration: $y \approx y_1 +\!\!+\, y_2$, $x \approx x_1 +\!\!+\, x_2$, $y_2 \approx x_2$, $y_1 \approx \text{update}(x_1, i, a)$, $|y_1| = |x_1|$, $|y_2| = |x_2|$, $\text{nth}(y, i) \approx a$, $\text{nth}(y_1, i) \approx a$, plus the original constraints.



Model Construction - a very simple example

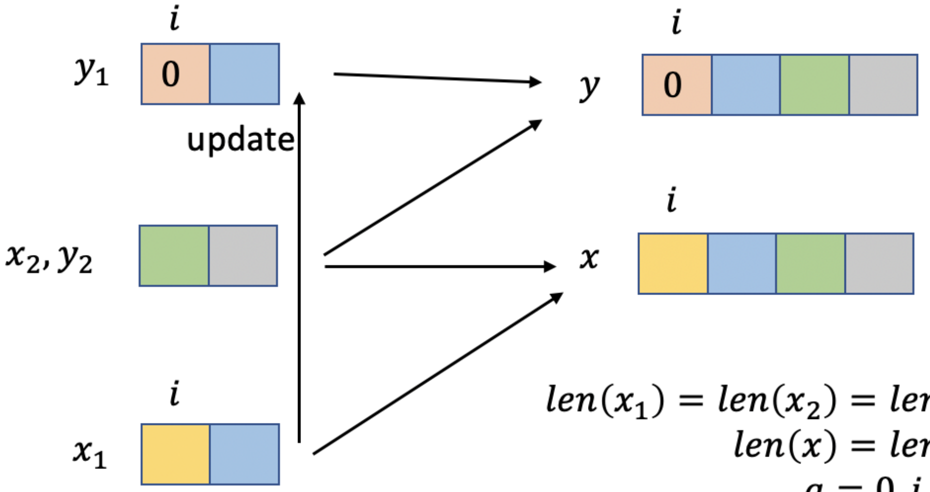
- Initial Configuration:
  $y = \text{update}(x, i, a), y = y_1 ++ y_2 \in S_0$
  $0 \leqslant i < |y_1|, |y_1| > 0, |y_2| > 0 \in A_0$.
- Saturated Configuration: $y \approx y_1 ++ y_2$, $x \approx x_1 ++ x_2$, $y_2 \approx x_2$, $y_1 \approx \text{update}(x_1, i, a)$, $|y_1| = |x_1|$, $|y_2| = |x_2|$, $\text{nth}(y, i) \approx a$, $\text{nth}(y_1, i) \approx a$, plus the original constraints.



$$len(x_1) = len(x_2) = len(y_2) = len(y_1) = 2$$
$$len(x) = len(y) = 4$$
$$a = 0, i = 0$$

Model Construction - a very simple example

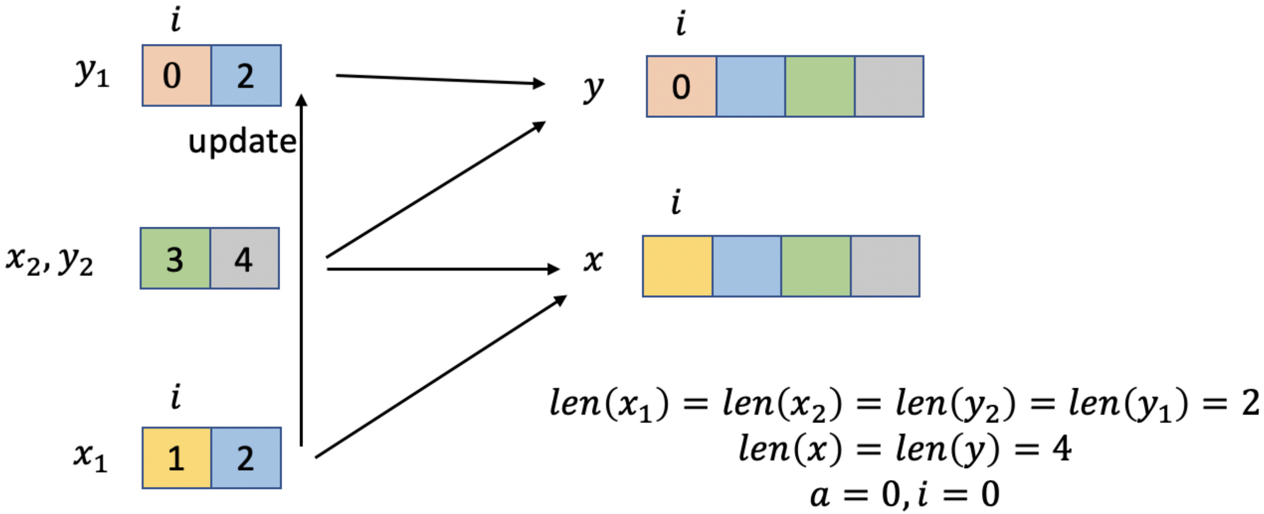
- Initial Configuration: $y = \text{update}(x, i, a), y = y_1 \mathbin{+\!\!+} y_2 \in S_0$ $0 \leqslant i < |y_1|, |y_1| > 0, |y_2| > 0 \in A_0$.
- Saturated Configuration: $y \approx y_1 \mathbin{+\!\!+} y_2$, $x \approx x_1 \mathbin{+\!\!+} x_2$, $y_2 \approx x_2$, $y_1 \approx \text{update}(x_1, i, a)$, $|y_1| = |x_1|$, $|y_2| = |x_2|$, $\text{nth}(y, i) \approx a$, $\text{nth}(y_1, i) \approx a$, plus the original constraints.



$$len(x_1) = len(x_2) = len(y_2) = len(y_1) = 2$$
$$len(x) = len(y) = 4$$
$$a = 0, i = 0$$

Model Construction - a very simple example

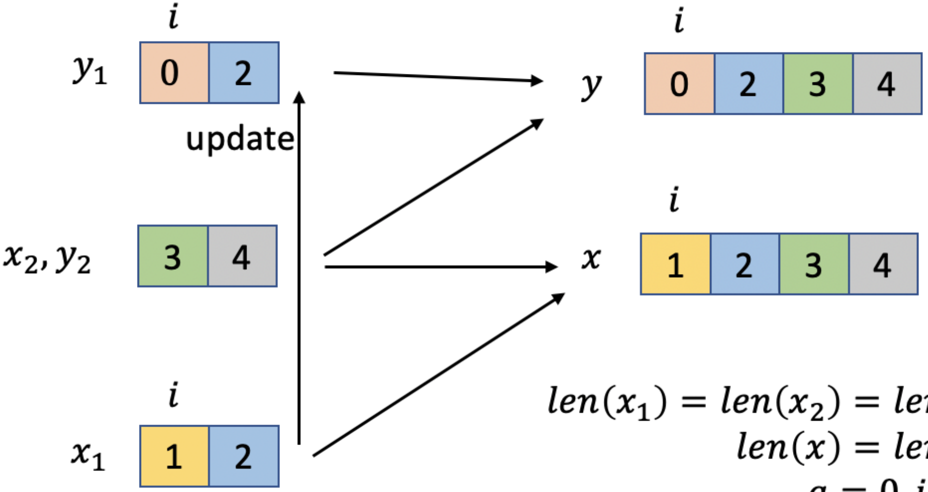
- Initial Configuration:
  $y = \text{update}(x, i, a), y = y_1 +\!\!+\ y_2 \in \mathrm{S}_0$
  $0 \leqslant i < |y_1|, |y_1| > 0, |y_2| > 0 \in \mathrm{A}_0.$
- Saturated Configuration: $y \approx y_1 +\!\!+\ y_2$, $x \approx x_1 +\!\!+\ x_2$, $y_2 \approx x_2$, $y_1 \approx \text{update}(x_1, i, a)$, $|y_1| = |x_1|$, $|y_2| = |x_2|$, $\text{nth}(y, i) \approx a$, $\text{nth}(y_1, i) \approx a$, plus the original constraints.



$$len(x_1) = len(x_2) = len(y_2) = len(y_1) = 2$$
$$len(x) = len(y) = 4$$
$$a = 0, i = 0$$

Model Construction - a very simple example

- Initial Configuration:
  $y = \mathrm{update}(x, i, a), y = y_1 \mathbin{++} y_2 \in \mathrm{S}_0$
  $0 \leqslant i < |y_1|, |y_1| > 0, |y_2| > 0 \in \mathrm{A}_0.$
- Saturated Configuration: $y \approx y_1 \mathbin{++} y_2$, $x \approx x_1 \mathbin{++} x_2$, $y_2 \approx x_2$, $y_1 \approx \mathrm{update}(x_1, i, a)$, $|y_1| = |x_1|$, $|y_2| = |x_2|$, $\mathrm{nth}(y, i) \approx a$, $\mathrm{nth}(y_1, i) \approx a$, plus the original constraints.

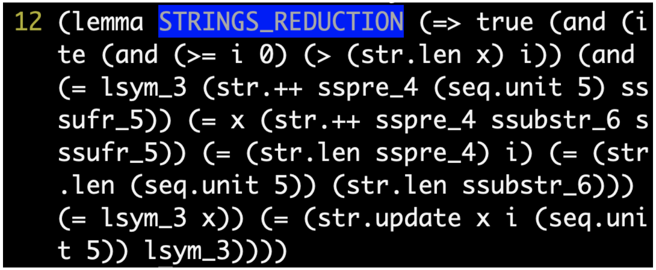


$$len(x_1) = len(x_2) = len(y_2) = len(y_1) = 2$$
$$len(x) = len(y) = 4$$
$$a = 0, i = 0$$

Details

1. Every step relies on the previous steps
2. No inconsistencies thanks to normal forms, weak equivalence graph, equivalence class
3. All of these are consistent with one another
4. In the paper: well-definedness

An Example of Lemma Logs



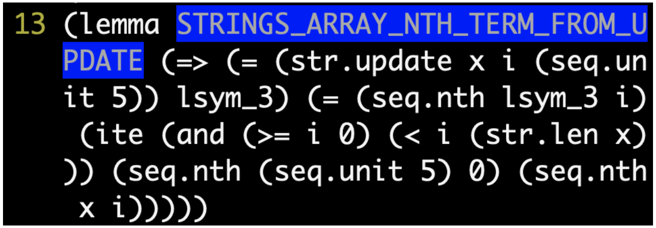
```
(lemma STRINGS_REDUCTION (=> true (and (i
te (and (>= i 0) (> (str.len x) i)) (and
(= lsym_3 (str.++ sspre_4 (seq.unit 5) ss
sufr_5)) (= x (str.++ sspre_4 ssubstr_6 s
ssufr_5)) (= (str.len sspre_4) i) (= (str
.len (seq.unit 5)) (str.len ssubstr_6)))
(= lsym_3 x)) (= (str.update x i (seq.uni
t 5)) lsym_3))))
```

BASE: **36** lemmas generated

$lsym\_3 = update(x, i, unit(5)) \wedge$
(ITE $0 \leqslant i < |x|$,
$lsym\_3 = sspre\_4 ++ unit(5) ++ ssufr\_5 \wedge$
$x = sspre\_4 ++ ssubstr\_6 ++ sssufr\_5 \wedge$
$|sspre\_4| = i \wedge$
$|unit(5)| = |ssubstr\_6|$,
$lsym\_3 = x$)

R-Update

$$\dfrac{y \approx \text{update}(x, i, z) \in \text{S}}{\begin{array}{c} \text{A} := \text{A}, i < 0 \vee i \geqslant \ell_y \qquad \text{S} := \text{S}, x \approx y \quad \| \\ \text{A} := \text{A}, 0 \leqslant i < \ell_y, \ell_k \approx i, \ell_{k'} \approx 1 \\ \text{S} := \text{S}, y \approx k ++ k' ++ k'', x \approx k ++ \text{unit}(z) ++ k'' \end{array}}$$



```
(lemma STRINGS_ARRAY_NTH_TERM_FROM_U
PDATE (=> (= (str.update x i (seq.un
it 5)) lsym_3) (= (seq.nth lsym_3 i)
 (ite (and (>= i 0) (< i (str.len x)
)) (seq.nth (seq.unit 5) 0) (seq.nth
 x i)))))
```

EXT: **22** lemmas generated

$lsym\_3 = update(x, i, unit(5)) \rightarrow$
$nth(lysm\_3, i) =$
(ITE $0 \leqslant i < |x|$, nth(unit(5), 0), nth(x, i))

Nth-Update

$$\dfrac{\begin{array}{c} \text{nth}(x, j) \in \mathcal{T}(\text{S}) \\ y \approx \text{update}(z, i, v) \in \text{S} \qquad \text{S} \models x \approx y \text{ or } \text{S} \models x \approx z \end{array}}{\begin{array}{c} \text{A} := \text{A}, j < 0 \vee j \geqslant \ell_x \quad \| \\ \text{A} := \text{A}, i \approx j, 0 \leqslant j < \ell_x \qquad \text{S} := \text{S}, \text{nth}(y, j) \approx v \quad \| \\ \text{A} := \text{A}, i \not\approx j, 0 \leqslant j < \ell_x \qquad \text{S} := \text{S}, \text{nth}(y, j) \approx \text{nth}(z, j) \end{array}}$$

Concatenation: Normal Form

Definition

$x_1 \mathbin{+\!\!+} \ldots \mathbin{+\!\!+} x_n$ is singular in S if S $\models x_i \approx \epsilon$ for all, except at most one, variables x_i, $i \in [1, n]$.

Definition

1. S $\models_{++} x \approx x$ for all sequence variables $x \in \mathcal{T}(\mathrm{S})$.
2. S $\models_{++} x \approx t$ for all sequence variables $x \in \mathcal{T}(\mathrm{S})$ and variable concatenation terms t, where $x \approx t \in$ S.
3. If S $\models_{++} x \approx (\overline{\boldsymbol{w}} \mathbin{+\!\!+} y \mathbin{+\!\!+} \overline{\boldsymbol{z}})\!\downarrow$ and S $\models y \approx t$ and t is ϵ or a variable concatenation term in S that is not singular in S, then S $\models_{++} x \approx (\overline{\boldsymbol{w}} \mathbin{+\!\!+} t \mathbin{+\!\!+} \overline{\boldsymbol{z}})\!\downarrow$.

Intuition

- S $\models_{++} x \approx t$ if x can be expanded to t via equalities in S.
- Each component of t is singular.

Concatanation: Atomic Representatives

Definition

x is atomic in S if S $\not\models x \approx \epsilon$ and for all variable concatenation terms $s \in \mathcal{T}(\mathsf{S})$ such that S $\models x \approx s$, s is singular in S.

Definition

$x \equiv_{\mathsf{S}} y$ iff S $\models x \approx y$. From each eq. class, we choose a representative.

Definition

S $\models^{*}_{++} x \approx \overline{\boldsymbol{y}}$ $\overline{\boldsymbol{y}}$ consists of atomic representatives and there exists $\overline{\boldsymbol{z}}$ of such that S $\models_{++} x \approx \overline{\boldsymbol{z}}$ and S $\models y_i \approx z_i$.

Intuition

S $\models^{*}_{++} x \approx t$ holds when t is a concatenation of atomic representatives.

Normal Form

$\overline{\boldsymbol{t}}$ is the normal form of x.