Reasoning About Vectors Using an SMT Theory of Sequences

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

\textsuperscript{1}Stanford University
\textsuperscript{2}The University of Iowa
\textsuperscript{3}Bar-Ilan University
\textsuperscript{4}Meta Novi

SMT 2023
Reasoning About Vectors
Using an SMT Theory of Sequences
The SMT Cycle

Some Examples

- Arithmetic
- Bit-vectors
- Floating Points
- Arrays
- Sets
- Lists
- Algebraic Datatypes
This Time: Sequences

Smart Contract Verification

Sequences

Arrays + Axioms

Imp: This Talk

Future: Standardize

App.

Theory

Standard

Axioms

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

July 2020: Move Prover paper published

December 2020: name change to Diem

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: Quantifiers can be used for axiomatization
- ✓ Generality: Can capture general element 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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Arity</th>
<th>SMT-LIB</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>Int</td>
<td>n</td>
</tr>
<tr>
<td>$+$</td>
<td>Int × Int → Int</td>
<td>+</td>
</tr>
<tr>
<td>$-$</td>
<td>Int → Int</td>
<td>−</td>
</tr>
<tr>
<td>$\leq$</td>
<td>Int × Int → Bool</td>
<td>&lt;=</td>
</tr>
</tbody>
</table>

### Sequence Operators

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Arity</th>
<th>SMT-LIB</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>Seq → Seq</td>
<td>seq.empty</td>
</tr>
<tr>
<td>unit</td>
<td>Elem → Seq</td>
<td>seq.unit</td>
</tr>
<tr>
<td>$</td>
<td>\cdot</td>
<td>\text{ }$</td>
</tr>
<tr>
<td>nth</td>
<td>Seq × Int → Elem</td>
<td>seq.nth</td>
</tr>
<tr>
<td>update</td>
<td>Seq × Int × Elem → Seq</td>
<td>seq.update</td>
</tr>
<tr>
<td>extract</td>
<td>Seq × Int × Int → Seq</td>
<td>seq.extract</td>
</tr>
<tr>
<td></td>
<td>Seq × ⋯ × Seq → Seq</td>
<td>seq.concat</td>
</tr>
</tbody>
</table>
Comparison Example

```cpp
// @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 : Int`
- `s, s_out : V`

Auxiliary Variables

- `\ell : V \rightarrow Int`
- `c : V \rightarrow \text{Arr}`
- `\approx_A : V \times V \rightarrow \text{Bool}`
- `nth_A : V \times \text{Int} \rightarrow \text{Int}`
- `update_A : V \times \text{Int} \times \text{Int} \rightarrow V`

Axioms

- `\forall x, y. x \approx_A y \leftrightarrow (\ell(x) \approx \ell(y) \land \forall 0 \leq i < \ell(x). c(x)[i] \approx c(y)[i])`
- `\forall x, y, i. a.y \approx_A update_A(x, i, a) \leftrightarrow (\ell(x) \approx \ell(y) \land (0 \leq i < \ell(x) \rightarrow c(y) \approx c(x)[i \leftarrow a]))`

Program

- `a \approx nth_A(s, i) \land b \approx nth_A(s, j)`
- `s_out \approx_A update_A(update_A(s, i, b), j, a)`

Spec.

- `0 \leq i, j < \ell(s) \land nth_A(s, i) \approx nth_A(s, j)`
- `\neg s_out \approx_A s`
Comparison Example

```cpp
// @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

<table>
<thead>
<tr>
<th>Problem Variables</th>
<th>a, b, i, j : Int</th>
<th>s, s_out : Seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auxiliary Variables</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Axioms

<table>
<thead>
<tr>
<th>Program</th>
<th>a \approx \text{nth}(s, i) \land b \approx \text{nth}(s, j)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>s_{out} \approx \text{update}(\text{update}(s, i, b), j, a)</td>
</tr>
</tbody>
</table>

| Spec. | 0 \leq i, j < |s| \land \text{nth}(s, i) \approx \text{nth}(s, j) |
|-------|----------------------------------------------------------|
|       | \neg s_{out} \approx s |
$ 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

+, −, ≤,
ε, unit, | |, extract, ++

Reduction Rules
nth, update

Specialized Rules
nth, update
Calculi and Rules

Rules

Core Rules
\(+, -, \leq,\)
\(\epsilon, \text{unit}, | \cdot |, \text{extract}, ++\)

Reduction Rules
nth, update

Specialized Rules
nth, update

Core Rules
- Adaptation of Strings [CAV’14]
- Simplification
Atithmetic and Equality

- **S**: Seq-constraints (≈, ∉)
- **A**: Arith-constraints (Arith terms)
- ℓₓ ≈ |x|

**Conflicts**

- **A-Conf**: \( A \models_{\text{LIA}} \perp \)
- **S-Conf**: \( S \models \perp \)

**Propagation**

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

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

- **S-A**: \( x, y \in T(S) \cap T(A) \quad x, y : \text{Int} \)
  \( A := A, x \approx y \quad \parallel \quad A := A, x \not\approx y \)
Reduced Form

- **S**: Seq-constraints \((\approx, \not\approx)\)
- **A**: Arith-constraints (Arith terms)
- \(\ell_x \approx |x|\)
- \(\ell\) x \(\subseteq\) \(|x|\)
- \(\mathcal{T}(S)\) – terms in \(S\)
- \(\models_{\text{LIA}}\): LIA-entailed
- \(\models\): FOL-entailed

**Rule**

\[
\text{L-Intro} \quad \frac{s \in \mathcal{T}(S) \quad s : \text{Seq}}{S := S, |s| \approx (|s|)↓}
\]

**Rewrites**

\[
|\epsilon| \rightarrow 0 \quad |\text{unit}(t)| \rightarrow 1
\]

\[
|\text{update}(s, i, t)| \rightarrow |s| \quad |s_1 ++ \ldots ++ s_n| \rightarrow |s_1| + \ldots + |s_n|
\]

\[
\bar{u} ++ \epsilon ++ \bar{v} \rightarrow \bar{u} ++ \bar{v} \quad \bar{u} ++ (s_1 ++ \ldots ++ s_n) ++ \bar{v} \rightarrow \bar{u} ++ s_1 ++ \ldots ++ s_n ++ \bar{v}
\]

**REWRITE**
Empty Sequence, Unit Sequence, Extentionality

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

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

**Emptyness**

\[
\frac{x \in T(S) \quad x: \text{Seq}}{S := S, x \approx \epsilon \quad \| \quad A := A, \ell_x > 0}
\]

**Unit**

\[
\frac{S \models \text{unit}(x) \approx \text{unit}(y)}{S := S, x \approx y}
\]

**Extentionionality**

\[
\frac{x \not\approx y \in S \quad x, y: \text{Seq}}{A := A, \ell_x \not\approx \ell_y \quad \| \quad A := A, \ell_x \approx \ell_y, 0 \leq i < \ell_x \quad S := S, w_1 \approx \text{nth}(x, i), w_2 \approx \text{nth}(y, i), w_1 \not\approx w_2}
\]
Extraction

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

\( T(S) \) – terms in \( S \)

- \( \models_{\text{LIA}} \): LIA-entailed
- \( \models_{\text{FOL}} \): FOL-entailed

Extraction Elimination

**R-Extract**

\[
x \approx \text{extract}(y, i, j) \in S
\]

\[
\frac{A := A, i < 0 \lor i \geq \ell_y \land j \leq 0}{S := S, x \approx \epsilon}
\]

\[
A := A, 0 \leq i < \ell_y, j > 0, \ell_k \approx i, \ell_x \approx \min(j, \ell_y - i), \ell_{k'} \approx \ell_y - \ell_x - i
\]

\[
S := S, y \approx k' + x + k'
\]

Intuition

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

- **S**: Seq-constraints ($\approx, \not\approx$)
- **A**: Arith-constraints (Arith terms)
- $\ell_x \approx |x|$
- \(T(S)\) – terms in \(S\)
- $\models_{\text{LIA}}$: LIA-entailed
- $\models$: FOL-entailed

### Unifying

C-Eq

\[
\begin{align*}
S & \models^* x \approx z \\
S & \models^* y \approx z
\end{align*}
\]

\[
S := S, x \approx y
\]

### Splitting

C-Split

\[
\begin{align*}
S & \models^* x \approx (w + y + z) \\
S & \models^* x \approx (w + y' + z')
\end{align*}
\]

\[
\begin{align*}
A & := A, \ell_y > \ell_{y'} \\
S & := S, y \approx y' + k \\
A & := A, \ell_y < \ell_{y'} \\
S & := S, y' \approx y + k \\
A & := A, \ell_y \approx \ell_{y'} \\
S & := S, y \approx y'
\end{align*}
\]
Example of Splitting

\[
\frac{S \models^* x \approx (\overline{w} ++ y ++ z)\downarrow}{A := A, \ell_y > \ell_{y'} \quad S := S, y \approx y' ++ k}
\]

\[
\frac{A := A, \ell_y < \ell_{y'} := S, y' \approx y ++ k}{A := A, \ell_y \approx \ell_{y'} \quad S := S, y \approx y'}
\]

\[
x = \begin{array}{c}
\text{y} \\
\text{z}
\end{array}
\]

\[
x = \begin{array}{c}
\text{y'} \\
\text{z'}
\end{array}
\]

\[
x = \begin{array}{c}
\text{y} \\
\text{k} \\
\text{z'}
\end{array}
\]
Calculi and Rules

Rules

Core Rules

\[ +, -, \leq, \epsilon, \text{unit}, |\cdot|, \text{extract}, ++ \]

Reduction Rules

nth, update

Specialized Rules

nth, update

RULES

1.
2.
3.
nth, update

- S: Seq-constraints (≈, ∉)
- A: Arith-constraints (Arith terms)
- ℓx ≈ |x|

T(S) – terms in S

∪LIA: LIA-entailed
∪L: FOL-entailed

**Reduction Rules**

R-Nth

\[
\begin{align*}
 x & \approx \text{nth}(y, i) \in S \\
 A & := A, i < 0 \lor i \geq \ell_y \\
 A & := A, 0 \leq i < \ell_y, \ell_k \approx i \\
 S & := S, y \approx k \mathit{++} \text{unit}(x) \mathit{++} k'
\end{align*}
\]

**Intuition**

nth and update are eliminated using fresh variables and ++

Main case: e ≈ nth(s, i) iff s ≈ s ++ unite ++ s'

OOB cases
nth, update

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

Reduction Rules

<table>
<thead>
<tr>
<th>Reduction Rule</th>
<th>$x \approx \text{update}(y, i, z) \in S$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>R-Update</strong></td>
<td>$A := A, i &lt; 0 \lor i \geq \ell_y$ \quad $S := S, x \approx y$ \quad $\parallel$ \quad $A := A, 0 \leq i &lt; \ell_y, \ell_k \approx i, \ell_{k'} \approx 1$ \quad $S := S, y \approx k ++ k' ++ k'', x \approx k ++ \text{unit}(z) ++ k''$</td>
</tr>
</tbody>
</table>

Intuition

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

$\mathcal{T}(S)$ – terms in $S$

$\models_{\text{LIA}}$: LIA-entailed

$\models$: FOL-entailed
Calculi and Rules

Core Rules

\[ +, -, \leq, \epsilon, \text{unit}, | \cdot |, \text{extract}, ++ \]

Reduction Rules

nth, update

Specialized Rules

nth, update
Array Reasoning

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

$\mathcal{T}(S)$ – terms in S

$\models_{\text{LIA}}$: LIA-entailed

$\models$: FOL-entailed

Read Over Write

Nth-Update: $\text{nth}(x, j) \in \mathcal{T}(S)$  $y \approx \text{update}(z, i, v) \in S$  $S \models x \approx y$ or $S \models x \approx z$

\[
\begin{align*}
A & := A, j < 0 \lor j \geq \ell_x & || \\
A & := A, i \approx j, 0 \leq j < \ell_x & S := S, \text{nth}(y, j) \approx v & || \\
A & := A, i \not\approx j, 0 \leq j < \ell_x & S := S, \text{nth}(y, j) \approx \text{nth}(z, j)
\end{align*}
\]

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
The core nth and update rule captures their behavior

Is that enough?
nth and update interact with other operators as well: unit and

These interactions must be modeled as well
Unit + nth + update

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

$T(S)$ – terms in S

$\models_{\text{LIA}}$: LIA-entailed

$\models$: FOL-entailed

unit + nth

Nth-Unit

\[
\frac{x \approx \text{nth}(y, i) \in S \quad S \models y \approx \text{unit}(u)}{A := A, i < 0 \lor i > 0 \quad \mid \quad A := A, i \approx 0 \quad S := S, x \approx u}
\]

unit + update

Update-Unit

\[
\frac{x \approx \text{update}(y, i, v) \in S \quad S \models y \approx \text{unit}(u)}{A := A, i < 0 \lor i > 0 \quad \mid \quad A := A, i \approx 0 \quad S := S, x \approx \text{unit}(v)}
\]
++ + nth + update

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

++ + nth

**Nth-Concat**

\[
\begin{align*}
  x \approx \text{nth}(y, i) \in S &\quad S \models^* y \approx w_1 ++ \ldots ++ w_n \\
  A := A, i < 0 \lor i \geq \ell_y &\quad A := A, 0 \leq i < \ell_{w_1} \quad S := S, x \approx \text{nth}(w_1, i) \quad \ldots \quad \parallel \\
  A := A, \sum_{j=1}^{n-1} \ell_{w_j} \leq i < \sum_{j=1}^{n} \ell_{w_j} &\quad S := S, x \approx \text{nth}(w_n, i - \sum_{j=1}^{n-1} \ell_{w_j})
\end{align*}
\]
++ nth + update

- S: Seq-constraints (≈, ⊈)
- A: Arith-constraints (Arith terms)
- ℓ_x ≈ |x|

++ update

Update-Concat

\[ x \approx \text{update}(y, i, v) \in S \quad S \models_{++}^* y \approx w_1 ++ \ldots ++ w_n \]

\[ S := S, x \approx z_1 ++ \ldots ++ z_n, z_1 \approx \text{update}(w_1, i, v), \ldots, z_n \approx \text{update}(w_n, i - \sum_{j=1}^{n-1} \ell_{w_j}, v) \]
S: Seq-constraints ($\approx, \not\approx$)
A: Arith-constraints (Arith terms)
$\ell_x \approx |x|$

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

Update-Concat-Inv

$x \approx \text{update}(y, i, \nu) \in S \quad S \models^*_{++} x \approx w_1 + + \ldots + + w_n$

$S := S, y \approx z_1 + + \ldots + + z_n, \quad w_1 \approx \text{update}(z_1, i, \nu), \ldots, w_n \approx \text{update}(z_n, i - \sum_{j=1}^{n-1} \ell_{w_j}, \nu)$
Additional Rules

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

$\mathcal{T}(S)$ – terms in $S$

|\[\text{\textit{Nth-Intro}}\]
\[
\frac{s' \approx \text{update}(s, i, t) \in S}{S := S, e \approx \text{nth}(s, i), e' \approx \text{nth}(s', i)}
\]

$\text{\textit{Update-Bound}}$

\[
\frac{x \approx \text{update}(y, i, v) \in S}{A := A, 0 \leq i < \ell_y \quad S := S, \text{nth}(y, i) \not\approx v \quad | \quad S := S, x \approx y}
\]

$\text{\textit{Nth-Split}}$

\[
\frac{\text{nth}(x, i), \text{nth}(x', i') \in \mathcal{T}(S) \quad i \approx i' \in A}{S := S, x \approx x' \quad | \quad S := S, x \not\approx x'}
\]
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

**Element Variables**
- Chosen arbitrarily by equality reasoning
- Possible since the domain is infinite
Atomic Sequence Variables

- length: According to $\ell_x$ 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 Results

<table>
<thead>
<tr>
<th>Set</th>
<th>w/ update</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cvc5</td>
<td>cvc5-a</td>
<td>Z3</td>
<td></td>
</tr>
<tr>
<td>ARRAYS</td>
<td>Slvd</td>
<td>242</td>
<td>390</td>
<td>170</td>
</tr>
<tr>
<td>(551)</td>
<td>Time</td>
<td>162</td>
<td>303</td>
<td>4329</td>
</tr>
<tr>
<td>DIEM</td>
<td>Slvd</td>
<td>542</td>
<td>547</td>
<td>443</td>
</tr>
<tr>
<td>(558)</td>
<td>Time</td>
<td>518</td>
<td>440</td>
<td>639</td>
</tr>
</tbody>
</table>

Commonly Solved

- 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
Look at the following function:

```cpp
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?
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.
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:
  \[ y = \text{update}(x, i, a), \quad y = y_1 \sqcup y_2 \in S_0 \]
  \[ 0 \leq i < |y_1|, |y_1| > 0, |y_2| > 0 \in A_0. \]

- Saturated Configuration: \( y \approx y_1 \sqcup y_2, \ x \approx x_1 \sqcup x_2, \ y_2 \approx x_2, \ y_1 \approx \text{update}(x_1, i, a), \ |y_1| = |x_1|, |y_2| = |x_2|, \ \text{nth}(y, i) \approx a, \ \text{nth}(y_1, i) \approx a, \) plus the original constraints.
Model Construction - a very simple example

- Initial Configuration:
  \[ y = \text{update}(x, i, a), \ y = y_1 ++ y_2 \in S_0 \]
  \[ 0 \leq i < |y_1|, |y_1| > 0, |y_2| > 0 \in A_0. \]

- Saturated Configuration:
  \[ y \approx y_1 ++ y_2, \ x \approx x_1 ++ x_2, \ y_2 \approx x_2, \]
  \[ y_1 \approx \text{update}(x_1, i, a), \ |y_1| = |x_1|, |y_2| = |x_2|, \ \text{nth}(y, i) \approx a, \]
  \[ \text{nth}(y_1, i) \approx a, \text{ plus the original constraints}. \]

\[ \text{len}(x_1) = \text{len}(x_2) = \text{len}(y_2) = \text{len}(y_1) = 2 \]
\[ \text{len}(x) = \text{len}(y) = 4 \]
\[ a = 0, i = 0 \]
Model Construction - a very simple example

- **Initial Configuration:**
  \[
  y = \text{update}(x, i, a), \quad y = y_1 \uplus y_2 \in S_0
  \]
  \[0 \leq i < |y_1|, \quad |y_1| > 0, \quad |y_2| > 0 \in A_0.
  \]

- **Saturated Configuration:**
  \[y \approx y_1 \uplus y_2, \quad x \approx x_1 \uplus x_2, \quad y_2 \approx x_2,
  \]
  \[y_1 \approx \text{update}(x_1, i, a), \quad |y_1| = |x_1|, \quad |y_2| = |x_2|, \quad \text{nth}(y, i) \approx a,
  \]
  \[\text{nth}(y_1, i) \approx a, \text{ plus the original constraints}.
  \]

\[
\begin{align*}
\text{len}(x_1) &= \text{len}(x_2) = \text{len}(y_2) = \text{len}(y_1) = 2 \\
\text{len}(x) &= \text{len}(y) = 4 \\
a &= 0, \quad i = 0
\end{align*}
\]
Model Construction - a very simple example

- **Initial Configuration:**
  \[ y = \text{update}(x, i, a), \ y = y_1 ++ y_2 \in S_0 \]
  \[ 0 \leq i < |y_1|, \ |y_1| > 0, \ |y_2| > 0 \in A_0. \]

- **Saturated Configuration:**
  \[ y \approx y_1 ++ y_2, \ x \approx x_1 ++ x_2, \ y_2 \approx x_2, \]
  \[ y_1 \approx \text{update}(x_1, i, a), \ |y_1| = |x_1|, \ |y_2| = |x_2|, \ nth(y, i) \approx a, \]
  \[ nth(y_1, i) \approx a, \text{ plus the original constraints.} \]

\[
\begin{align*}
l\text{len}(x_1) &= l\text{len}(x_2) = l\text{len}(y_2) = l\text{len}(y_1) = 2 \\
l\text{len}(x) &= l\text{len}(y) = 4 \\
& a = 0, i = 0
\end{align*}
\]
Model Construction - a very simple example

- **Initial Configuration:**
  \[ y = \text{update}(x, i, a), \ y = y_1 \bowtie y_2 \in S_0 \]
  \[ 0 \leq i < |y_1|, |y_1| > 0, |y_2| > 0 \in A_0. \]

- **Saturated Configuration:** \[ y \approx y_1 \bowtie y_2, x \approx x_1 \bowtie x_2, y_2 \approx x_2, \]
  \[ y_1 \approx \text{update}(x_1, i, a), |y_1| = |x_1|, |y_2| = |x_2|, \text{nth}(y, i) \approx a, \]
  \[ \text{nth}(y_1, i) \approx a, \text{plus the original constraints}. \]

\[ \begin{align*}
\text{len}(x_1) &= \text{len}(x_2) = \text{len}(y_2) = \text{len}(y_1) = 2 \\
\text{len}(x) &= \text{len}(y) = 4 \\
a &= 0, i &= 0
\end{align*} \]
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

BASE: 36 lemmas generated

EXT: 22 lemmas generated

12 (lemma STRINGS_REDUCTION (=> true (and (ite (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 ss sufr_5)) (= (str.len sspre_4) i) (= (str len (seq.unit 5)) (str.len ssubstr_6))) (= lsym_3 x)) ( (str.update x i (seq.unit 5)) lsym_3))))

13 (lemma STRINGS_ARRAY_NTH_TERM_FROM_UPDATE (=> (= (str.update x i (seq.unit 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)))))

lsym_3 = update(x, i, unit(5)) \land
(ITE 0 \leq i < |x|,
lSYM_3 = sspre_4 \oplus unit(5) \oplus ssufr_5 \land
x = sspre_4 \oplus ssubstr_6 \oplus ssufr_5 \land
|sspre_4| = i \land
|unit(5)| = |sssubstr_6|,
lSYM_3 = x)

R-Update

y \approx update(x, i, z) \in S
A := A, i < 0 \lor i \geq \ell_y \quad S := S, x \approx y \quad ||
A := A, 0 \leq i < \ell_y, \ell_k \approx i, \ell_k' \approx 1
S := S, y \approx k ++ k' ++ k'', x \approx k ++ unit(z) ++ k''

lsym_3 = update(x, i, unit(5)) \rightarrow
nth(lSYM_3, i) =
(ITE 0 \leq i < |x|, nth(unit(5), 0), nth(x, i))

Nth-Update

nth(x, j) \in T(S)
y \approx update(z, i, v) \in S \quad S \models x \approx y \lor S \models x \approx z
A := A, j < 0 \lor j \geq \ell_x \quad ||
A := A, i \approx j, 0 \leq j < \ell_x \quad S := S, nth(y, j) \approx v \quad ||
A := A, i \not\approx j, 0 \leq j < \ell_x \quad S := S, nth(y, j) \approx nth(z, j)
## Concatenation: Normal Form

### Definition

$x_1 \ldots x_n$ 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 $x \in T(S)$.
2. $S \models \_x \approx t$ for all sequence variables $x \in T(S)$ and variable concatenation terms $t$, where $x \approx t \in S$.
3. If $S \models \_x \approx (\_w \_y \_z) \downarrow$ and $S \models y \approx t$ and $t$ is $\epsilon$ or a variable concatenation term in $S$ that is not singular in $S$, then $S \models \_x \approx (\_w \_t \_z) \downarrow$.

### Intuition

- $S \models \_x \approx t$ if $x$ can be expanded to $t$ via equalities in $S$.
- Each component of $t$ is singular.
Definition

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

Definition

\( x \equiv_S y \) iff \( S \models x \approx y \). From each eq. class, we choose a representative.

Definition

\( S \models^{\ast \ast} x \approx \overline{y} \overline{y} \) consists of atomic representatives and there exists \( \overline{z} \) of such that \( S \models^{\ast \ast} x \approx \overline{z} \) and \( S \models y_i \approx z_i \).

Intuition

\( S \models^{\ast \ast} x \approx t \) holds when \( t \) is a concatenation of atomic representatives.

Normal Form

\( \overline{t} \) is the normal form of \( x \).