PhD. Defense

Theory of Sequences Tailored for Program Verification

Théorie des séquences adaptée à la vérification des programmes

by **Hichem Rami AIT EL HARA**[1,2]

Under the supervision of **François BOBOT**[2] and **Guillaume BURY**[1]

[1]OCamlPro
[2]Université Paris-Saclay, CEA, LIST

Software is everywhere

Personal information

Energy

Aerospace

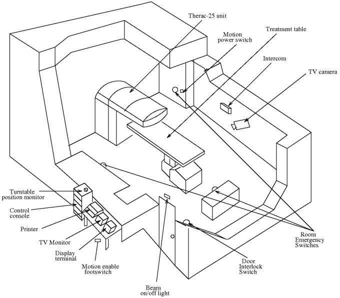
Healthcare

And it can be faulty (bugged)

Ariane 5 *

(Arithmetic overflow)

Loss of over US$370 million



Therac-25 **

(Race condition)

Deaths and injuries of patients

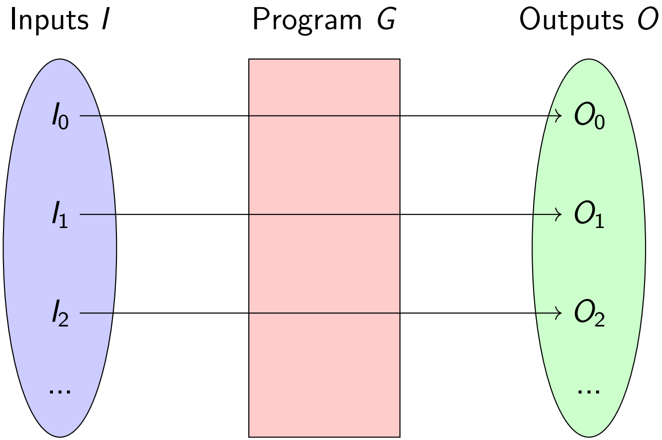
More examples:
`https://en.wikipedia.org/wiki/List_of_software_bugs`

---

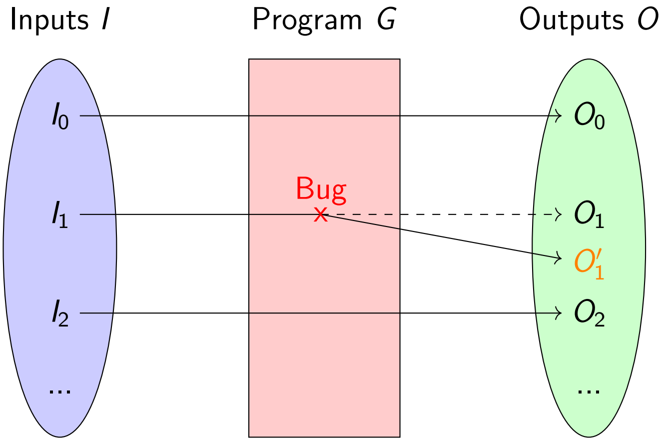
*Photo: ©ESA

**Figure from: "Medical Devices: The Therac-25" by Nancy G. Leveson

Behind the software, there are programs.

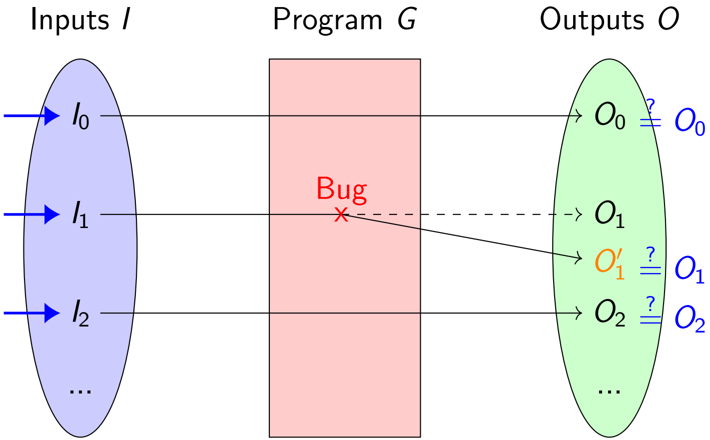


Behind the software, there are programs.

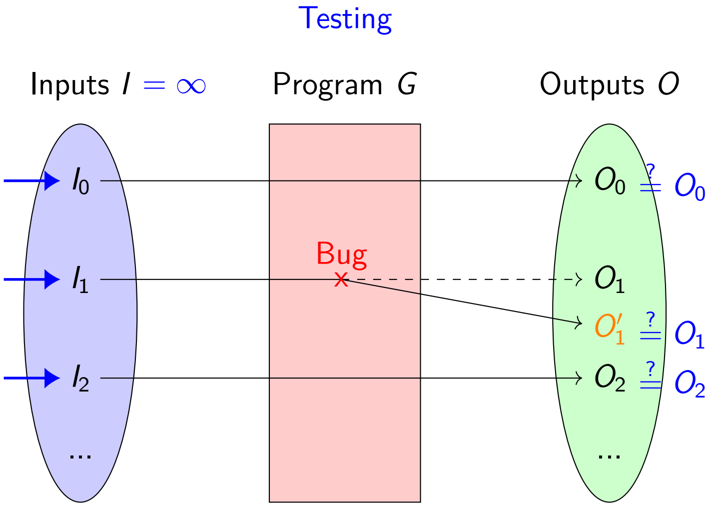


Behind the software, there are programs.

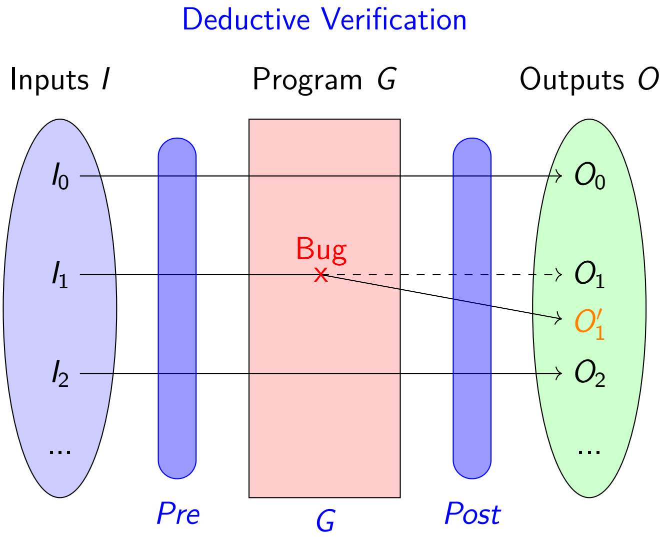
Testing



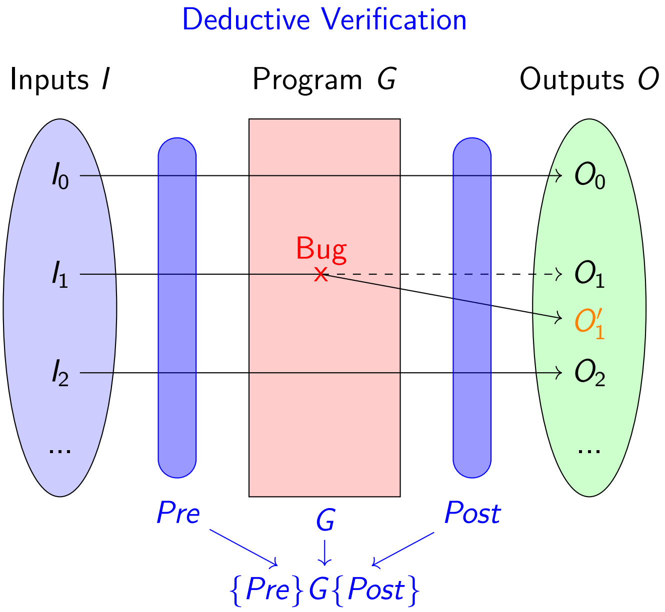
Behind the software, there are programs.



Behind the software, there are programs.

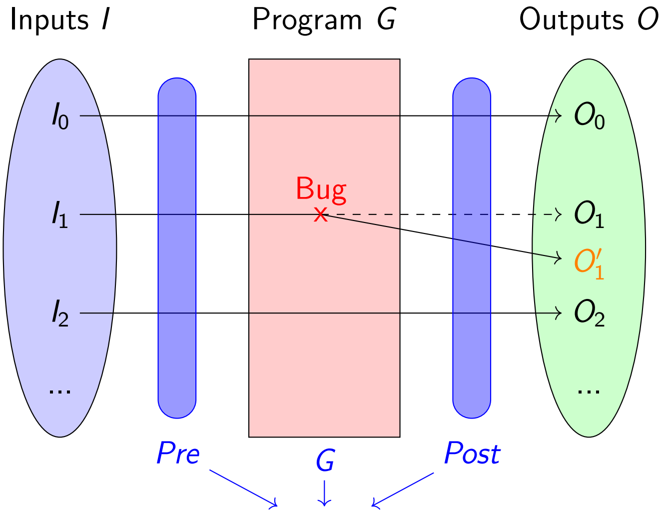


Behind the software, there are programs.



Behind the software, there are programs.

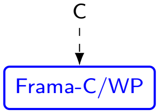
Deductive Verification



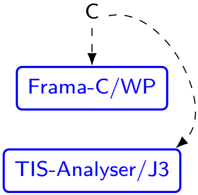
$\{Pre\}G\{Post\} \Rightarrow \Psi$ (proven?)

Avoiding Bugs in the Real World

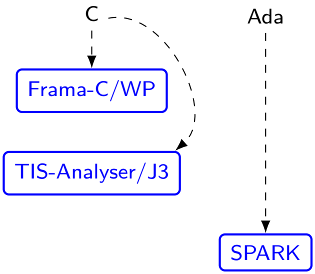
Avoiding Bugs in the Real World



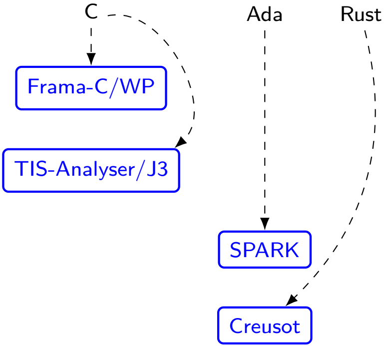
Avoiding Bugs in the Real World



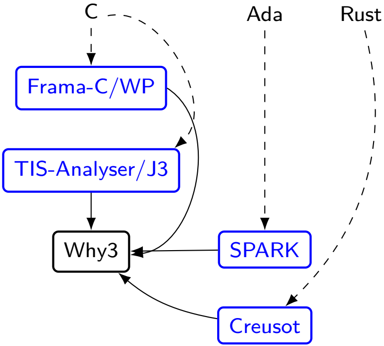
Avoiding Bugs in the Real World



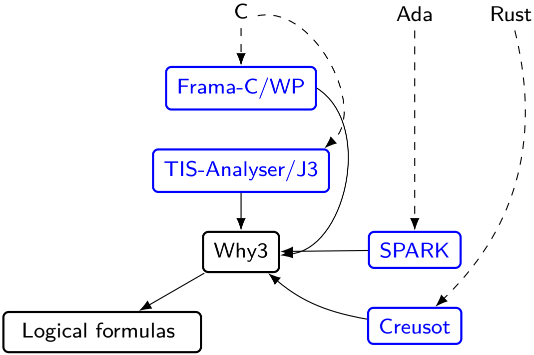
Avoiding Bugs in the Real World



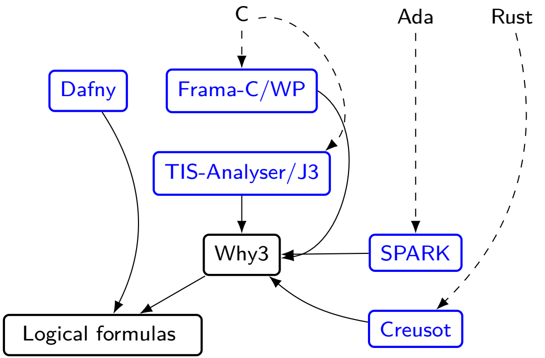
Avoiding Bugs in the Real World



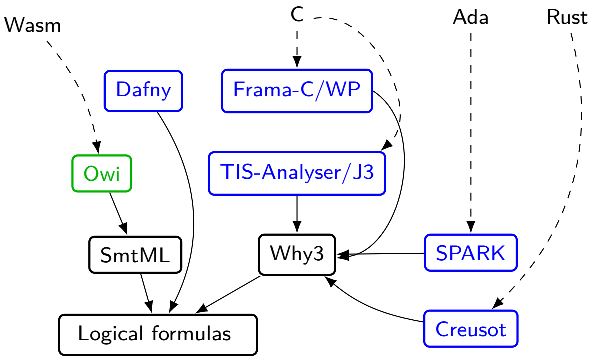
Avoiding Bugs in the Real World



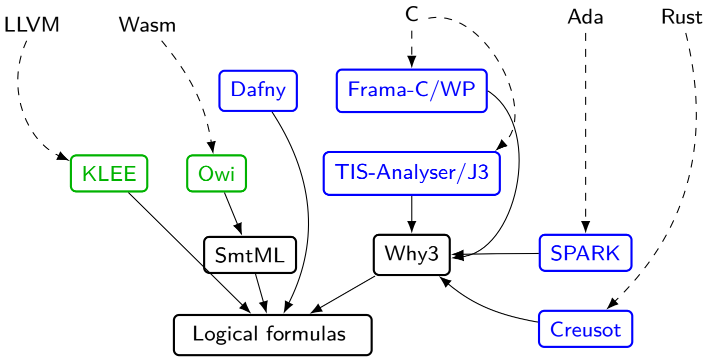
Avoiding Bugs in the Real World



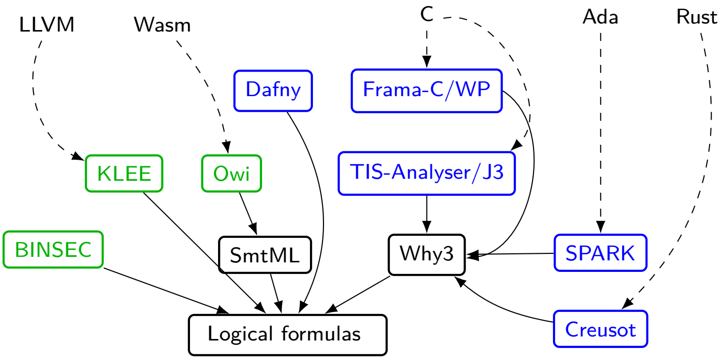
Avoiding Bugs in the Real World



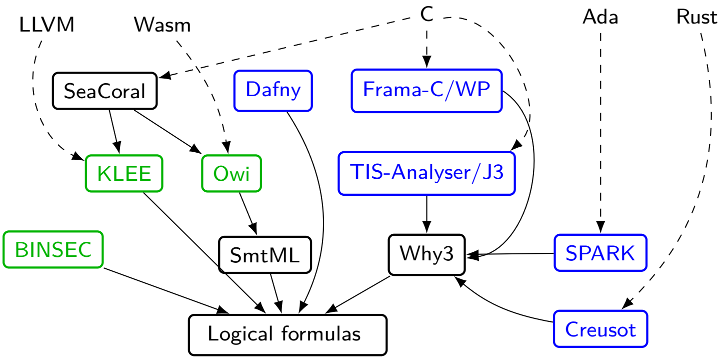
Avoiding Bugs in the Real World



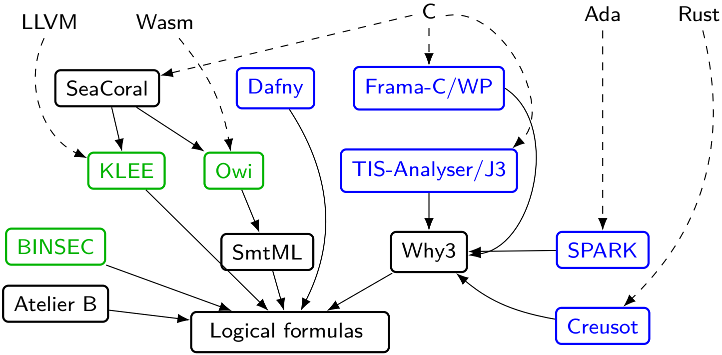
Avoiding Bugs in the Real World



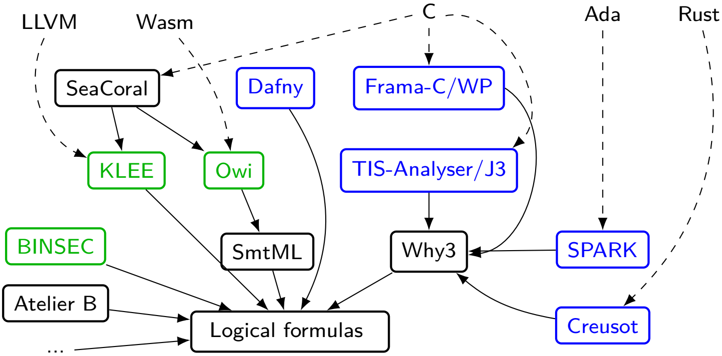
Avoiding Bugs in the Real World



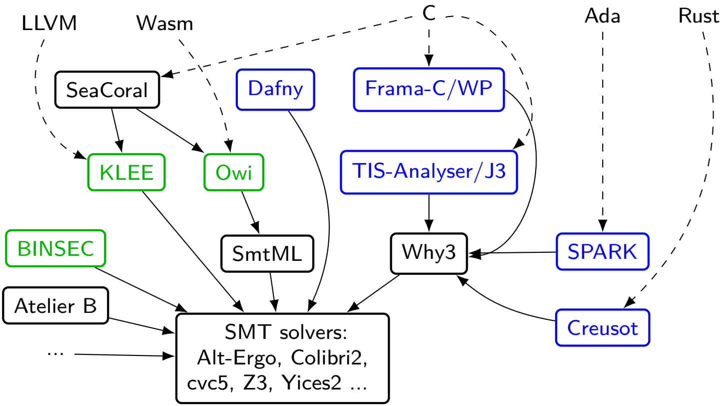
Avoiding Bugs in the Real World



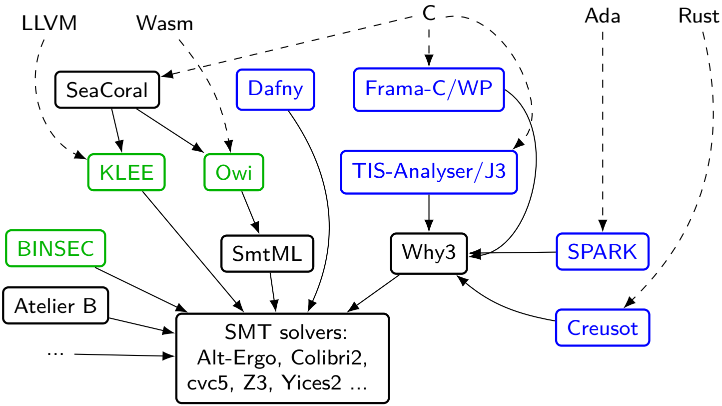
Avoiding Bugs in the Real World



Avoiding Bugs in the Real World

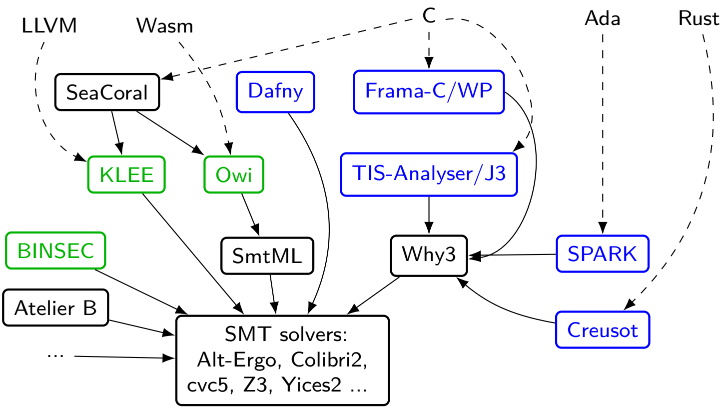


Avoiding Bugs in the Real World



In the industry:

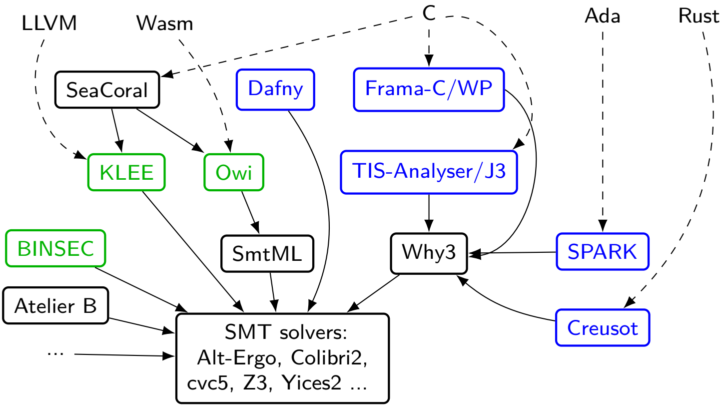
Avoiding Bugs in the Real World



In the industry:

- AirBus (Frama-C/WP)

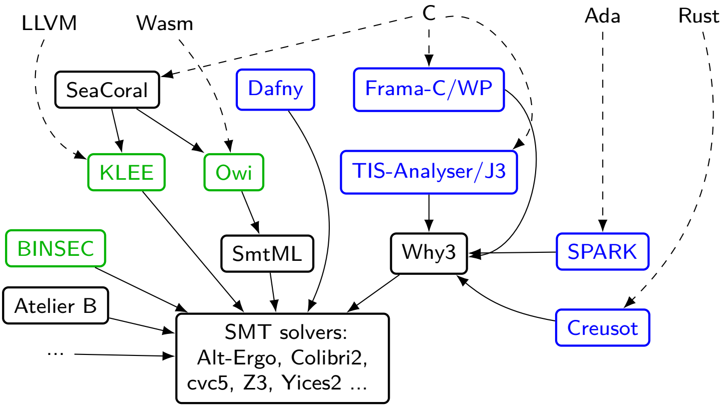
Avoiding Bugs in the Real World



In the industry:

- AirBus (Frama-C/WP)
- NVIDIA (SPARK)

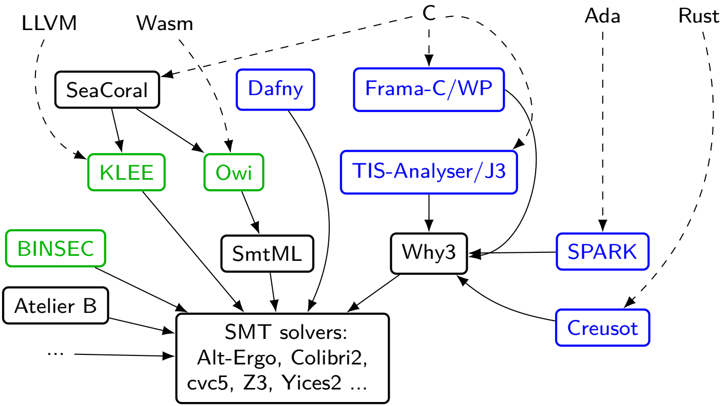
Avoiding Bugs in the Real World



In the industry:

- AirBus (Frama-C/WP)
- NVIDIA (SPARK)
- Mitsubishi Electric (TIS-Analyzer)

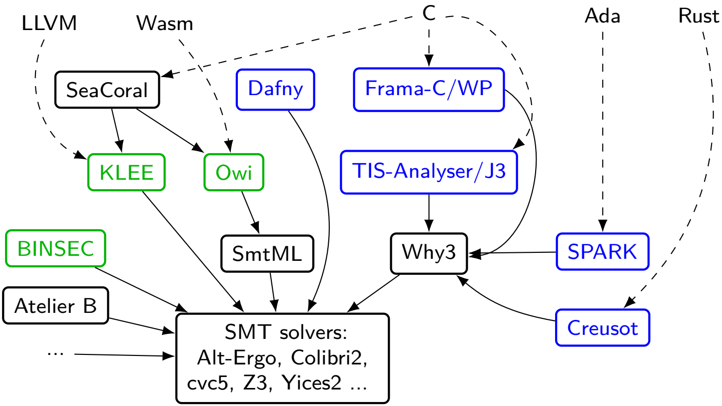
Avoiding Bugs in the Real World



In the industry:

- AirBus (Frama-C/WP)
- NVIDIA (SPARK)
- Mitsubishi Electric (TIS-Analyzer)
- Thales (Frama-C/WP & SeaCoral)

Avoiding Bugs in the Real World



In the industry:

- AirBus (Frama-C/WP)
- NVIDIA (SPARK)
- Mitsubishi Electric (TIS-Analyzer)
- Thales (Frama-C/WP & SeaCoral)
- Alstom (Atelier B)

What is SMT (Satisfiability Modulo Theories)?

Definition

Boolean Satisfiability (Propositional Logic)

$+$

Built-in First-Order Logic Theories

What is SMT (Satisfiability Modulo Theories)?

Definition

Boolean Satisfiability (Propositional Logic)

+

Built-in First-Order Logic Theories

Example

$F(p, q, x, y) = (p \Rightarrow x > 5) \land (q \Rightarrow y \geq 4) \land (p \lor q) \land (x + y < 10)$

What is SMT (Satisfiability Modulo Theories)?

Definition

Boolean Satisfiability (Propositional Logic)

+

Built-in First-Order Logic Theories

Example

$F(p, q, x, y) = (p \Rightarrow x > 5) \land (q \Rightarrow y \geq 4) \land (p \lor q) \land (x + y < 10)$

What is SMT (Satisfiability Modulo Theories)?

Definition

Boolean Satisfiability (Propositional Logic)

+

Built-in First-Order Logic Theories

Example

$F(p, q, x, y) = (p \Rightarrow x > 5) \wedge (q \Rightarrow y \geq 4) \wedge (p \vee q) \wedge (x + y < 10)$

What is SMT (Satisfiability Modulo Theories)?

Definition

Boolean Satisfiability (Propositional Logic)

$+$

Built-in First-Order Logic Theories

Example

$$F(p, q, x, y) = (p \Rightarrow x > 5) \wedge (q \Rightarrow y \geq 4) \wedge (p \vee q) \wedge (x + y < 10)$$

Question: is there a **satisfying** interpretation?

What is SMT (Satisfiability Modulo Theories)?

Definition

Boolean Satisfiability (Propositional Logic)

$+$

Built-in First-Order Logic Theories

Example

$$F(p, q, x, y) = (p \Rightarrow x > 5) \land (q \Rightarrow y \geq 4) \land (p \lor q) \land (x + y < 10)$$

Question: is there a **satisfying** interpretation?

$$F(\top, \bot, 6, 3) = (\top \Rightarrow 6 > 5) \land (\bot \Rightarrow 3 \geq 4) \land (\top \lor \bot) \land (6 + 3 < 10)$$

What is SMT (Satisfiability Modulo Theories)?

Definition

Boolean Satisfiability (Propositional Logic)

+

Built-in First-Order Logic Theories

Example

$$F(p, q, x, y) = (p \Rightarrow x > 5) \wedge (q \Rightarrow y \geq 4) \wedge (p \vee q) \wedge (x + y < 10)$$

Question: is there a **satisfying** interpretation?

$$\begin{aligned} F(\top, \bot, 6, 3) &= (\top \Rightarrow 6 > 5) \wedge (\bot \Rightarrow 3 \geq 4) \wedge (\top \vee \bot) \wedge (6 + 3 < 10) \\ &= \top \wedge \top \wedge \top \wedge \top \end{aligned}$$

What is SMT (Satisfiability Modulo Theories)?

Definition

Boolean Satisfiability (Propositional Logic)

$+$

Built-in First-Order Logic Theories

Example

$$F(p, q, x, y) = (p \Rightarrow x > 5) \wedge (q \Rightarrow y \geq 4) \wedge (p \vee q) \wedge (x + y < 10)$$

Question: is there a **satisfying** interpretation?

$$\begin{aligned} F(\top, \bot, 6, 3) &= (\top \Rightarrow 6 > 5) \wedge (\bot \Rightarrow 3 \geq 4) \wedge (\top \vee \bot) \wedge (6 + 3 < 10) \\ &= \top \wedge \top \wedge \top \wedge \top \\ &= \top \end{aligned}$$

What is SMT (Satisfiability Modulo Theories)?

Definition

Boolean Satisfiability (Propositional Logic)

+

Built-in First-Order Logic Theories

Example

$$F(p, q, x, y) = (p \Rightarrow x > 5) \wedge (q \Rightarrow y \geq 4) \wedge (p \vee q) \wedge (x + y < 10)$$

Question: is there a **satisfying** interpretation?

$$\begin{aligned} F(\top, \bot, 6, 3) &= (\top \Rightarrow 6 > 5) \wedge (\bot \Rightarrow 3 \geq 4) \wedge (\top \vee \bot) \wedge (6 + 3 < 10) \\ &= \top \wedge \top \wedge \top \wedge \top \\ &= \top \end{aligned}$$

Yes, therefore F is **satisfiable**.

What is SMT (Satisfiability Modulo Theories)?

Definition

Boolean Satisfiability (Propositional Logic)

+

Built-in First-Order Logic Theories

Example

$$F(p, q, x, y) = (p \Rightarrow x > 5) \land (q \Rightarrow y \geq 4) \land (p \lor q) \land (x + y < 10) \land (p \Leftrightarrow q)$$

Question: is there a **satisfying** interpretation?

$$F(p, q, x, y) = (p \Rightarrow x > 5) \land (q \Rightarrow y \geq 4) \land (p \lor q) \land (x + y < 10) \land (p \Leftrightarrow q)$$

What is SMT (Satisfiability Modulo Theories)?

Definition

Boolean Satisfiability (Propositional Logic)

+

Built-in First-Order Logic Theories

Example

$$F(p, q, x, y) = (p \Rightarrow x > 5) \wedge (q \Rightarrow y \geq 4) \wedge (p \vee q) \wedge (x + y < 10) \wedge (p \Leftrightarrow q)$$

Question: is there a **satisfying** interpretation?

$$\begin{aligned} F(p, q, x, y) &= (p \Rightarrow x > 5) \wedge (q \Rightarrow y \geq 4) \wedge (p \vee q) \wedge (x + y < 10) \wedge (p \Leftrightarrow q) \\ &= (\top \Rightarrow x > 5) \wedge (\top \Rightarrow y \geq 4) \wedge \top \wedge (x + y < 10) \wedge \top \end{aligned}$$

What is SMT (Satisfiability Modulo Theories)?

Definition

Boolean Satisfiability (Propositional Logic)

+

Built-in First-Order Logic Theories

Example

$$F(p, q, x, y) = (p \Rightarrow x > 5) \wedge (q \Rightarrow y \geq 4) \wedge (p \vee q) \wedge (x + y < 10) \wedge (p \Leftrightarrow q)$$

Question: is there a **satisfying** interpretation?

$$\begin{aligned} F(p, q, x, y) &= (p \Rightarrow x > 5) \wedge (q \Rightarrow y \geq 4) \wedge (p \vee q) \wedge (x + y < 10) \wedge (p \Leftrightarrow q) \\ &= (\top \Rightarrow x > 5) \wedge (\top \Rightarrow y \geq 4) \wedge \top \wedge (x + y < 10) \wedge \top \\ &= (x > 5) \wedge (y \geq 4) \wedge (x + y < 10) \end{aligned}$$

What is SMT (Satisfiability Modulo Theories)?

Definition

Boolean Satisfiability (Propositional Logic)

$+$

Built-in First-Order Logic Theories

Example

$$F(p, q, x, y) = (p \Rightarrow x > 5) \wedge (q \Rightarrow y \geq 4) \wedge (p \vee q) \wedge (x + y < 10) \wedge (p \Leftrightarrow q)$$

Question: is there a **satisfying** interpretation?

$$\begin{aligned}
F(p, q, x, y) &= (p \Rightarrow x > 5) \wedge (q \Rightarrow y \geq 4) \wedge (p \vee q) \wedge (x + y < 10) \wedge (p \Leftrightarrow q) \\
&= (\top \Rightarrow x > 5) \wedge (\top \Rightarrow y \geq 4) \wedge \top \wedge (x + y < 10) \wedge \top \\
&= (x > 5) \wedge (y \geq 4) \wedge (x + y < 10) \\
&= \bot
\end{aligned}$$

What is SMT (Satisfiability Modulo Theories)?

Definition

Boolean Satisfiability (Propositional Logic)

+

Built-in First-Order Logic Theories

Example

$$F(p, q, x, y) = (p \Rightarrow x > 5) \wedge (q \Rightarrow y \geq 4) \wedge (p \vee q) \wedge (x + y < 10) \wedge (p \Leftrightarrow q)$$

Question: is there a **satisfying** interpretation?

$$\begin{aligned} F(p, q, x, y) &= (p \Rightarrow x > 5) \wedge (q \Rightarrow y \geq 4) \wedge (p \vee q) \wedge (x + y < 10) \wedge (p \Leftrightarrow q) \\ &= (\top \Rightarrow x > 5) \wedge (\top \Rightarrow y \geq 4) \wedge \top \wedge (x + y < 10) \wedge \top \\ &= (x > 5) \wedge (y \geq 4) \wedge (x + y < 10) \\ &= \bot \end{aligned}$$

No, therefore F is **unsatisfiable**.

What is SMT (Satisfiability Modulo Theories)?

Definition

Boolean Satisfiability (Propositional Logic)

$+$

Built-in First-Order Logic Theories

Example

$$F(p, q, x, y) = (p \Rightarrow x > 5) \wedge (q \Rightarrow y \geq 4) \wedge (p \vee q) \wedge (x + y < 10) \wedge (p \Leftrightarrow q)$$

Question: is there a **satisfying** interpretation?

$$\begin{aligned} F(p, q, x, y) &= (p \Rightarrow x > 5) \wedge (q \Rightarrow y \geq 4) \wedge (p \vee q) \wedge (x + y < 10) \wedge (p \Leftrightarrow q) \\ &= (\top \Rightarrow x > 5) \wedge (\top \Rightarrow y \geq 4) \wedge \top \wedge (x + y < 10) \wedge \top \\ &= (x > 5) \wedge (y \geq 4) \wedge (x + y < 10) \\ &= \bot \end{aligned}$$

No, therefore F is **unsatisfiable**. Inversely $\neg F$ is **valid**.

Why are SMT solvers used?

Why are SMT solvers used?

- Expressiveness (Quantifies and various theories)

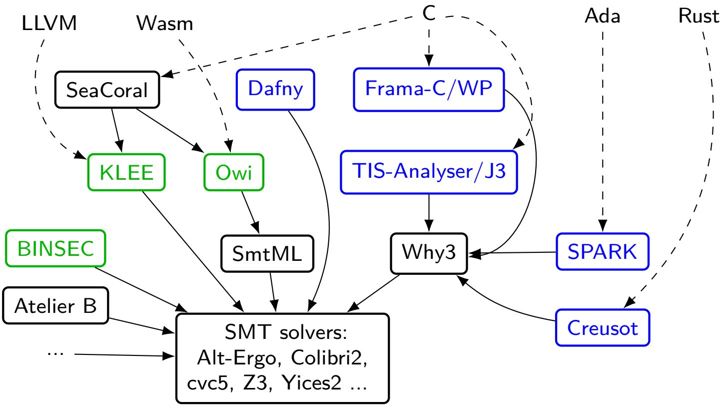
Why are SMT solvers used?

- Expressiveness (Quantifies and various theories)
- Efficiency (Combination of powerful decision procedures)

Why are SMT solvers used?

- Expressiveness (Quantifies and various theories)
- Efficiency (Combination of powerful decision procedures)
- The SMT-LIB (Standard language and theories)

Avoiding Bugs in the Real World



Why are SMT solvers used?

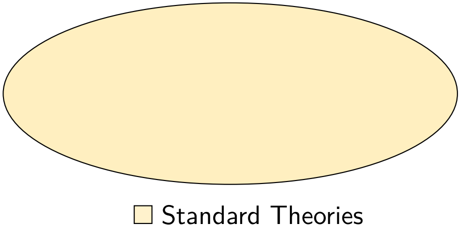
- Expressiveness (Quantifies and various theories)
- Efficiency (Combination of powerful decision procedures)
- The SMT-LIB (Standard language and theories)

Why are SMT solvers used?

- Expressiveness (Quantifies and various theories)
- Efficiency (Combination of powerful decision procedures)
- The SMT-LIB (Standard language and theories)

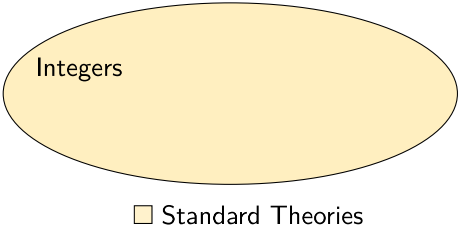
Why are SMT solvers used?

- Expressiveness (Quantifies and various theories)
- Efficiency (Combination of powerful decision procedures)
- The SMT-LIB (Standard language and theories)



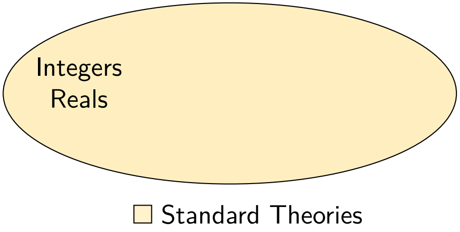
Why are SMT solvers used?

- Expressiveness (Quantifies and various theories)
- Efficiency (Combination of powerful decision procedures)
- The SMT-LIB (Standard language and theories)



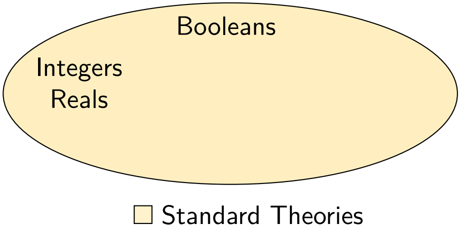
Why are SMT solvers used?

- Expressiveness (Quantifies and various theories)
- Efficiency (Combination of powerful decision procedures)
- The SMT-LIB (Standard language and theories)



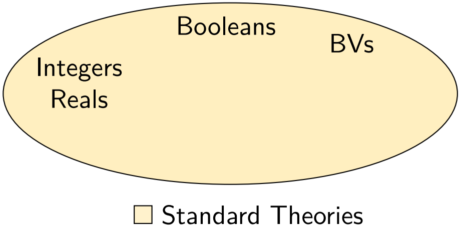
Why are SMT solvers used?

- Expressiveness (Quantifies and various theories)
- Efficiency (Combination of powerful decision procedures)
- The SMT-LIB (Standard language and theories)



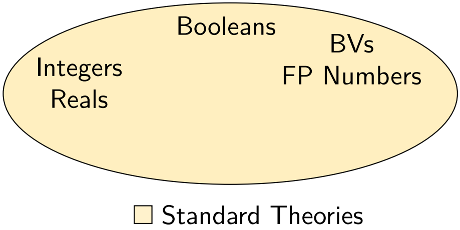
Why are SMT solvers used?

- Expressiveness (Quantifies and various theories)
- Efficiency (Combination of powerful decision procedures)
- The SMT-LIB (Standard language and theories)



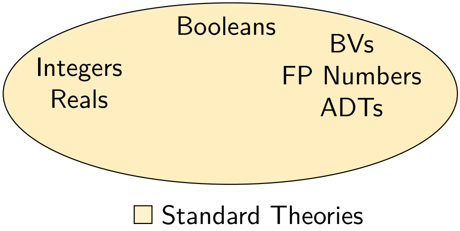
Why are SMT solvers used?

- Expressiveness (Quantifies and various theories)
- Efficiency (Combination of powerful decision procedures)
- The SMT-LIB (Standard language and theories)



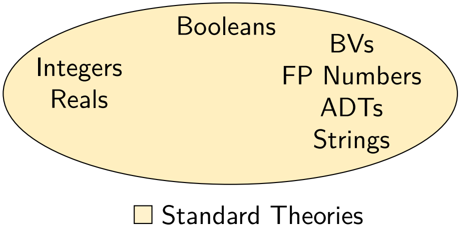
Why are SMT solvers used?

- Expressiveness (Quantifies and various theories)
- Efficiency (Combination of powerful decision procedures)
- The SMT-LIB (Standard language and theories)



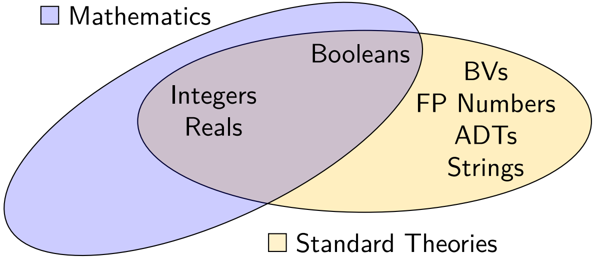
Why are SMT solvers used?

- Expressiveness (Quantifies and various theories)
- Efficiency (Combination of powerful decision procedures)
- The SMT-LIB (Standard language and theories)



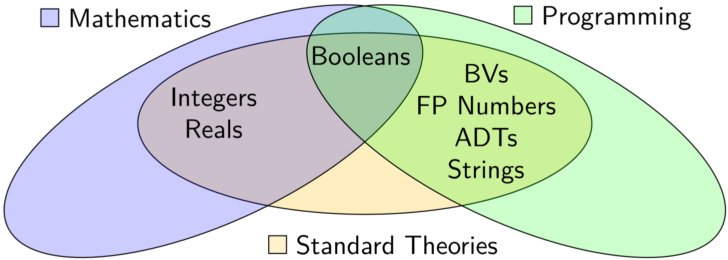
Why are SMT solvers used?

- Expressiveness (Quantifies and various theories)
- Efficiency (Combination of powerful decision procedures)
- The SMT-LIB (Standard language and theories)



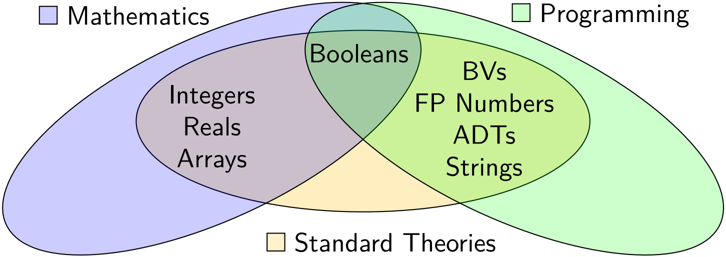
Why are SMT solvers used?

- Expressiveness (Quantifies and various theories)
- Efficiency (Combination of powerful decision procedures)
- The SMT-LIB (Standard language and theories)



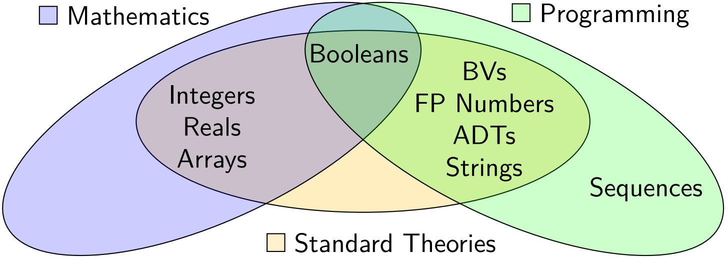
Why are SMT solvers used?

- Expressiveness (Quantifies and various theories)
- Efficiency (Combination of powerful decision procedures)
- The SMT-LIB (Standard language and theories)



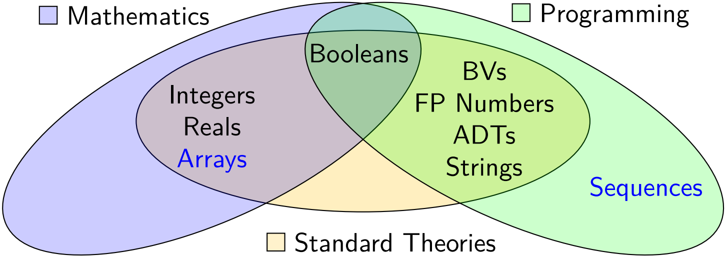
Why are SMT solvers used?

- Expressiveness (Quantifies and various theories)
- Efficiency (Combination of powerful decision procedures)
- The SMT-LIB (Standard language and theories)



Why are SMT solvers used?

- Expressiveness (Quantifies and various theories)
- Efficiency (Combination of powerful decision procedures)
- The SMT-LIB (Standard language and theories)



Arrays and Sequences in SMT

	Arrays [McC62]	Sequences [Bjø+12]

Arrays and Sequences in SMT

	Arrays [McC62]	Sequences [Bjø+12]
Sort	Array I E	Seq E

Arrays and Sequences in SMT

	Arrays [McC62]	Sequences [Bjø+12]
Sort	Array I E	Seq E
Structure	$\left\{ \begin{array}{c} \ldots \\ i_n \mapsto e_n \\ i_{n+1} \mapsto e_{n+1} \\ i_{n+2} \mapsto e_{n+2} \\ \ldots \end{array} \right\}$	$[e_0; e_1; e_2; \ldots; e_{l-1}]$

Arrays and Sequences in SMT

	Arrays [McC62]	Sequences [Bjø+12]
Sort	Array I E	Seq E
Structure	$\left\{ \begin{array}{c} \dots \\ i_n \mapsto e_n \\ i_{n+1} \mapsto e_{n+1} \\ i_{n+2} \mapsto e_{n+2} \\ \dots \end{array} \right\}$	$[e_0; e_1; e_2; \dots; e_{l-1}]$
Operations	select and store.	len, nth, extract, concat, update and others.

Arrays and Sequences in SMT

	Arrays [McC62]	Sequences [Bjø+12]
Sort	Array I E	Seq E
Structure	$\left\{ \begin{array}{c} \dots \\ i_n \mapsto e_n \\ i_{n+1} \mapsto e_{n+1} \\ i_{n+2} \mapsto e_{n+2} \\ \dots \end{array} \right\}$	$[e_0; e_1; e_2; \dots; e_{l-1}]$
Operations	select and store.	len, nth, extract, concat, update and others.
Pros	Widely explored [CH15; MB09]	Expressiveness

Arrays and Sequences in SMT

	Arrays [McC62]	Sequences [Bjø+12]
Sort	Array I E	Seq E
Structure	$\left\{ \begin{array}{c} \ldots \\ i_n \mapsto e_n \\ i_{n+1} \mapsto e_{n+1} \\ i_{n+2} \mapsto e_{n+2} \\ \ldots \end{array} \right\}$	$[e_0; e_1; e_2; \ldots; e_{l-1}]$
Operations	select and store.	len, nth, extract, concat, update and others.
Pros	Widely explored [CH15; MB09]	Expressiveness
Cons	– Lack of expressiveness – Fixed size	– Scarce literature – Few solvers support it

Arrays and Sequences in SMT

	Arrays [McC62]	Sequences [Bjø+12]
Sort	Array I E	Seq E
Structure	$\left\{ \begin{array}{c} \dots \\ i_n \mapsto e_n \\ i_{n+1} \mapsto e_{n+1} \\ i_{n+2} \mapsto e_{n+2} \\ \dots \end{array} \right\}$	$[e_0; e_1; e_2; \dots; e_{l-1}]$
Operations	select and store.	len, nth, extract, concat, update and others.
Pros	Widely explored [CH15; MB09]	Expressiveness
Cons	– Lack of expressiveness – Fixed size	– Scarce literature – Few solvers support it

For the following data structures from programming languages:

Arrays and Sequences in SMT

	Arrays [McC62]	Sequences [Bjø+12]
Sort	Array I E	Seq E
Structure	$\left\{ \begin{array}{c} \ldots \\ i_n \mapsto e_n \\ i_{n+1} \mapsto e_{n+1} \\ i_{n+2} \mapsto e_{n+2} \\ \ldots \end{array} \right\}$	$[e_0; e_1; e_2; \ldots; e_{l-1}]$
Operations	select and store.	len, nth, extract, concat, update and others.
Pros	Widely explored [CH15; MB09]	Expressiveness
Cons	– Lack of expressiveness – Fixed size	– Scarce literature – Few solvers support it

For the following data structures from programming languages:

- Arrays (OCaml, C)
- Vectors (Rust, C++)
- ArrayLists (Java)
- Lists (Python)

Arrays and Sequences in SMT

	Arrays [McC62]	Sequences [Bjø+12]
Sort	Array I E	Seq E
Structure	$\left\{ \begin{array}{c} \ldots \\ i_n \mapsto e_n \\ i_{n+1} \mapsto e_{n+1} \\ i_{n+2} \mapsto e_{n+2} \\ \ldots \end{array} \right\}$	$[e_0; e_1; e_2; \ldots; e_{l-1}]$
Operations	select and store.	len, nth, extract, concat, update and others.
Pros	Widely explored [CH15; MB09]	Expressiveness
Cons	– Lack of expressiveness – Fixed size	– Scarce literature – Few solvers support it

For the following data structures from programming languages:

- Arrays (OCaml, C)
- Vectors (Rust, C++)
- ArrayLists (Java)
- Lists (Python)

Sequences are more suitable as they are semantically closer.

In this thesis: A different theory of Sequences

Context:

In this thesis: A different theory of Sequences

Context:

- In Ada/Spark: sequences can be defined over an arbitrary range of integers.

In this thesis: A different theory of Sequences

Context:

- In Ada/Spark: sequences can be defined over an arbitrary range of integers.
- Encoding them in SMT is cumbersome and inefficient.

In this thesis: A different theory of Sequences

Context:

- In Ada/Spark: sequences can be defined over an arbitrary range of integers.
- Encoding them in SMT is cumbersome and inefficient.

n-Indexed Sequences

An n-indexed sequence (or n-sequence) s is a sequence that is indexed from a first index f_s to a last index l_s.

In this thesis: A different theory of Sequences

Context:

- In Ada/Spark: sequences can be defined over an arbitrary range of integers.
- Encoding them in SMT is cumbersome and inefficient.

n-Indexed Sequences

An n-indexed sequence (or n-sequence) s is a sequence that is indexed from a first index f_s to a last index l_s.

Motivation

In this thesis: A different theory of Sequences

Context:

- In Ada/Spark: sequences can be defined over an arbitrary range of integers.
- Encoding them in SMT is cumbersome and inefficient.

n-Indexed Sequences

An n-indexed sequence (or n-sequence) s is a sequence that is indexed from a first index f_s to a last index l_s.

Motivation

- Conveniently represent and efficiently reason over n-indexed sequences.

In this thesis: A different theory of Sequences

Context:

- In Ada/Spark: sequences can be defined over an arbitrary range of integers.
- Encoding them in SMT is cumbersome and inefficient.

n-Indexed Sequences

An n-indexed sequence (or n-sequence) s is a sequence that is indexed from a first index f_s to a last index l_s.

Motivation

- Conveniently represent and efficiently reason over n-indexed sequences.
- A generalization of the theory of sequences.

Outline

1. The SMT theory of n-Indexed Sequences
2. Reasoning over n-Indexed Sequences
3. Implementation
4. Experimental Evaluation
5. Conclusion

Outline

Semantics of the theory of n-Indexed Sequences I

- f_s and l_s: the first and last index of s.

Semantics of the theory of n-Indexed Sequences I

- f_s and l_s: the first and last index of s.

Empty n-indexed sequence

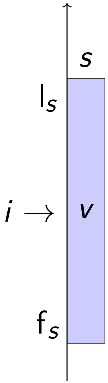
An n-indexed sequence s is said to be empty if $\mathsf{l}_s < \mathsf{f}_s$. Two empty n-indexed sequences a and b are equal only if $\mathsf{f}_a = \mathsf{f}_b$ and $\mathsf{l}_a = \mathsf{l}_b$.

Semantics of the theory of n-Indexed Sequences I

- f_s and l_s: the first and last index of s.
- $\mathsf{get}(s, i)$: if $\mathsf{f}_s \leq i \leq \mathsf{l}_s$ returns the ith element of s, otherwise it is uninterpreted.

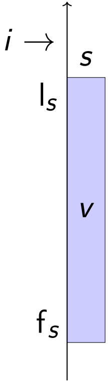
Semantics of the theory of n-Indexed Sequences I

- f_s and l_s: the first and last index of s.
- get(s, i): if $\mathsf{f}_s \leq i \leq \mathsf{l}_s$ returns the ith element of s, otherwise it is uninterpreted.



Semantics of the theory of n-Indexed Sequences I

- f_s and l_s: the first and last index of s.
- get(s, i): if $\mathsf{f}_s \leq i \leq \mathsf{l}_s$ returns the ith element of s, otherwise it is uninterpreted.



Semantics of the theory of n-Indexed Sequences I

- f_s and l_s: the first and last index of s.
- $\mathsf{get}(s, i)$: if $\mathsf{f}_s \leq i \leq \mathsf{l}_s$ returns the ith element of s, otherwise it is uninterpreted.
- $\mathsf{set}(s, i, v)$: if $\mathsf{f}_s \leq i \leq \mathsf{l}_s$ returns a copy of s in which i is associated to v, otherwise returns s.

Semantics of the theory of n-Indexed Sequences I

- f_s and l_s: the first and last index of s.
- get(s, i): if $\mathsf{f}_s \leq i \leq \mathsf{l}_s$ returns the ith element of s, otherwise it is uninterpreted.
- set(s, i, v): if $\mathsf{f}_s \leq i \leq \mathsf{l}_s$ returns a copy of s in which i is associated to v, otherwise returns s.

Semantics of the theory of n-Indexed Sequences I

- f_s and l_s: the first and last index of s.
- $\mathsf{get}(s, i)$: if $\mathsf{f}_s \leq i \leq \mathsf{l}_s$ returns the ith element of s, otherwise it is uninterpreted.
- $\mathsf{set}(s, i, v)$: if $\mathsf{f}_s \leq i \leq \mathsf{l}_s$ returns a copy of s in which i is associated to v, otherwise returns s.

Semantics of the theory of n-Indexed Sequences I

- f_s and l_s: the first and last index of s.
- $\mathrm{get}(s, i)$: if $\mathrm{f}_s \leq i \leq \mathrm{l}_s$ returns the ith element of s, otherwise it is uninterpreted.
- $\mathrm{set}(s, i, v)$: if $\mathrm{f}_s \leq i \leq \mathrm{l}_s$ returns a copy of s in which i is associated to v, otherwise returns s.
- $\mathrm{const}(f, l, v)$: an n-indexed sequence with f as a first index, l as a last index and all its elements are v.

Semantics of the theory of n-Indexed Sequences I

- f_s and l_s: the first and last index of s.
- get(s, i): if $\mathsf{f}_s \leq i \leq \mathsf{l}_s$ returns the ith element of s, otherwise it is uninterpreted.
- set(s, i, v): if $\mathsf{f}_s \leq i \leq \mathsf{l}_s$ returns a copy of s in which i is associated to v, otherwise returns s.
- const(f, l, v): an n-indexed sequence with f as a first index, l as a last index and all its elements are v.
- relocate(s, f): a copy of s relocated to the index f.

Semantics of the theory of n-Indexed Sequences II

Semantics of the theory of n-Indexed Sequences II

- concat(a, b): Concatenates two non-empty n-sequences if they follow one another, otherwise returns a.

Semantics of the theory of n-Indexed Sequences II

- concat(a, b): Concatenates two non-empty n-sequences if they follow one another, otherwise returns a.
- update(a, b): if $\mathsf{f}_a \leq \mathsf{f}_b \leq \mathsf{l}_b \leq \mathsf{l}_a$ returns a new n-indexed sequence that has the same elements as b within the bounds of b and the same elements as a within the bounds of a and outside those of b, otherwise returns a.

Semantics of the theory of n-Indexed Sequences II

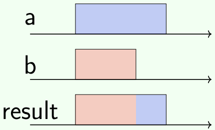
- concat(a, b): Concatenates two non-empty n-sequences if they follow one another, otherwise returns a.
- update(a, b): if $\mathsf{f}_a \leq \mathsf{f}_b \leq \mathsf{l}_b \leq \mathsf{l}_a$ returns a new n-indexed sequence that has the same elements as b within the bounds of b and the same elements as a within the bounds of a and outside those of b, otherwise returns a.

Example

Semantics of the theory of n-Indexed Sequences II

- concat(a, b): Concatenates two non-empty n-sequences if they follow one another, otherwise returns a.
- update(a, b): if $\mathsf{f}_a \leq \mathsf{f}_b \leq \mathsf{l}_b \leq \mathsf{l}_a$ returns a new n-indexed sequence that has the same elements as b within the bounds of b and the same elements as a within the bounds of a and outside those of b, otherwise returns a.

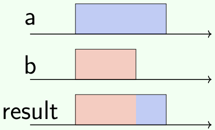
Example

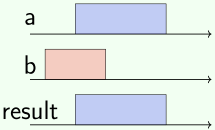


Semantics of the theory of n-Indexed Sequences II

- concat(a, b): Concatenates two non-empty n-sequences if they follow one another, otherwise returns a.
- update(a, b): if $\mathsf{f}_a \leq \mathsf{f}_b \leq \mathsf{l}_b \leq \mathsf{l}_a$ returns a new n-indexed sequence that has the same elements as b within the bounds of b and the same elements as a within the bounds of a and outside those of b, otherwise returns a.

Example

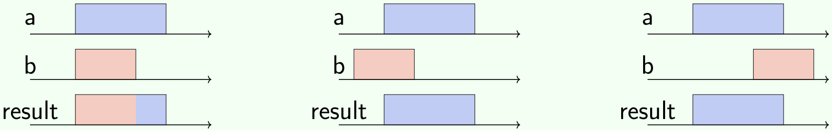




Semantics of the theory of n-Indexed Sequences II

- concat(a, b): Concatenates two non-empty n-sequences if they follow one another, otherwise returns a.
- update(a, b): if $\mathsf{f}_a \leq \mathsf{f}_b \leq \mathsf{l}_b \leq \mathsf{l}_a$ returns a new n-indexed sequence that has the same elements as b within the bounds of b and the same elements as a within the bounds of a and outside those of b, otherwise returns a.

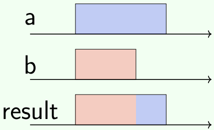
Example

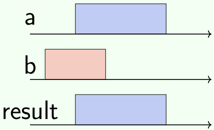


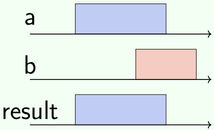
Semantics of the theory of n-Indexed Sequences II

- concat(a, b): Concatenates two non-empty n-sequences if they follow one another, otherwise returns a.
- update(a, b): if $\mathsf{f}_a \leq \mathsf{f}_b \leq \mathsf{l}_b \leq \mathsf{l}_a$ returns a new n-indexed sequence that has the same elements as b within the bounds of b and the same elements as a within the bounds of a and outside those of b, otherwise returns a.

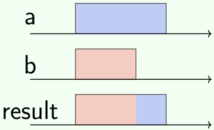
Example

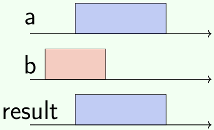


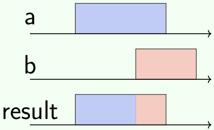




In the theory of sequences (in cvc5) update(a, i, b):







Semantics of the theory of n-Indexed Sequences II

- concat(a, b): Concatenates two non-empty n-sequences if they follow one another, otherwise returns a.
- update(a, b): if $\mathsf{f}_a \leq \mathsf{f}_b \leq \mathsf{l}_b \leq \mathsf{l}_a$ returns a new n-indexed sequence that has the same elements as b within the bounds of b and the same elements as a within the bounds of a and outside those of b, otherwise returns a.

To read more on SMT theory design and semantic choices:

- **"On SMT Theory Design: The Case of Sequences"**
  Hichem Rami Ait-El-Hara, François Bobot and Guillaume Bury. LPAR 2024

Semantics of the theory of n-Indexed Sequences II

- concat(a, b): Concatenates two non-empty n-sequences if they follow one another, otherwise returns a.
- update(a, b): if $\mathsf{f}_a \leq \mathsf{f}_b \leq \mathsf{l}_b \leq \mathsf{l}_a$ returns a new n-indexed sequence that has the same elements as b within the bounds of b and the same elements as a within the bounds of a and outside those of b, otherwise returns a.

Semantics of the theory of n-Indexed Sequences II

- concat(a, b): Concatenates two non-empty n-sequences if they follow one another, otherwise returns a.
- update(a, b): if $\mathrm{f}_a \leq \mathrm{f}_b \leq \mathrm{l}_b \leq \mathrm{l}_a$ returns a new n-indexed sequence that has the same elements as b within the bounds of b and the same elements as a within the bounds of a and outside those of b, otherwise returns a.
- slice(a, f, l): if $\mathrm{f}_a \leq f \leq l \leq \mathrm{l}_a$ returns a new n-indexed sequence that has the same elements as a within the bounds f and l, otherwise returns a.

Semantics of the theory of n-Indexed Sequences II

- concat(a, b): Concatenates two non-empty n-sequences if they follow one another, otherwise returns a.
- update(a, b): if $\mathsf{f}_a \leq \mathsf{f}_b \leq \mathsf{l}_b \leq \mathsf{l}_a$ returns a new n-indexed sequence that has the same elements as b within the bounds of b and the same elements as a within the bounds of a and outside those of b, otherwise returns a.
- slice(a, f, l): if $\mathsf{f}_a \leq f \leq l \leq \mathsf{l}_a$ returns a new n-indexed sequence that has the same elements as a within the bounds f and l, otherwise returns a.

Extensionality

The theory of n-indexed sequences is extensional. Therefore, given two n-indexed sequences a and b:

$$\begin{gathered}(a = b) \equiv \\ (\mathsf{f}_a = \mathsf{f}_b \wedge \mathsf{l}_a = \mathsf{l}_b \wedge \\ \forall i : \mathsf{Int}, \mathsf{f}_a \leq i \leq \mathsf{l}_a \rightarrow \mathsf{get}(a, i) = \mathsf{get}(b, i))\end{gathered}$$

Outline

Axiomatization/Encoding of n-Sequences

Axiomatization (with arrays):

- Most operations need to be axiomatized.
- Introduces too many quantified formulas.

Encoding using Sequences and Algebraic Data Types:

- Avoids using as many quantifiers.
- Depends on two other theories (Sequences and ADTs).
- Differences in the semantics make the definitions complex.

Reasoning with Sequences and Algebraic Data Types I

n-Indexed sequences are defined as a record:

$$\mathrm{NSeq}(a) = \{seq : \mathrm{Seq}(a); fst : \mathrm{Int}\}$$

Reasoning with Sequences and Algebraic Data Types I

n-Indexed sequences are defined as a record:

$$\mathsf{NSeq}(a) = \{seq : \mathsf{Seq}(a); fst : \mathsf{Int}\}$$

Other symbols of the theory are defined over it:

Reasoning with Sequences and Algebraic Data Types I

n-Indexed sequences are defined as a record:

$$\mathsf{NSeq}(a) = \{seq : \mathsf{Seq}(a); fst : \mathsf{Int}\}$$

Other symbols of the theory are defined over it:

- $\mathsf{f}_n = n.fst$

Reasoning with Sequences and Algebraic Data Types I

n-Indexed sequences are defined as a record:

$$\mathsf{NSeq}(a) = \{seq : \mathsf{Seq}(a); fst : \mathsf{Int}\}$$

Other symbols of the theory are defined over it:

- $\mathsf{f}_n = n.fst$ and $\mathsf{l}_n = n.fst + \mathsf{len}(n.seq) - 1$

Reasoning with Sequences and Algebraic Data Types I

n-Indexed sequences are defined as a record:

$$\mathsf{NSeq}(a) = \{seq : \mathsf{Seq}(a); fst : \mathsf{Int}\}$$

Other symbols of the theory are defined over it:

- $\mathsf{f}_n = n.fst$ and $\mathsf{l}_n = n.fst + \mathsf{len}(n.seq) - 1$
- $\mathsf{get}(n, i) = \mathsf{nth}(n.seq, i - n.fst)$

Reasoning with Sequences and Algebraic Data Types I

n-Indexed sequences are defined as a record:

$$\mathsf{NSeq}(a) = \{seq : \mathsf{Seq}(a); fst : \mathsf{Int}\}$$

Other symbols of the theory are defined over it:

- $\mathsf{f}_n = n.fst$ and $\mathsf{l}_n = n.fst + \mathsf{len}(n.seq) - 1$
- $\mathsf{get}(n, i) = \mathsf{nth}(n.seq, i - n.fst)$

Except $\mathsf{const}(f, l, v)$, which has no counterpart in the theory of sequences:

Reasoning with Sequences and Algebraic Data Types I

n-Indexed sequences are defined as a record:

$$\mathsf{NSeq}(a) = \{seq : \mathsf{Seq}(a); fst : \mathsf{Int}\}$$

Other symbols of the theory are defined over it:

- $\mathsf{f}_n = n.fst$ and $\mathsf{l}_n = n.fst + \mathsf{len}(n.seq) - 1$
- $\mathsf{get}(n, i) = \mathsf{nth}(n.seq, i - n.fst)$

Except $\mathsf{const}(f, l, v)$, which has no counterpart in the theory of sequences:

- It can be axiomatized:

$$n = \mathsf{const}(f, l, v) \iff \mathsf{f}_n = f \wedge \mathsf{l}_n = l \wedge \\ \forall i.\ f \leq i \leq l \implies \mathsf{get}(n, i) = v$$

Outline

Porting calculi on Sequences to n-Sequences

Porting calculi on Sequences to n-Sequences

Based on the paper "Reasoning About Vectors: Satisfiability Modulo a Theory of Sequences" by Sheng et al. [She+23],

Porting calculi on Sequences to n-Sequences

Based on the paper "Reasoning About Vectors: Satisfiability Modulo a Theory of Sequences" by Sheng et al. [She+23], which presents two calculi:

Porting calculi on Sequences to n-Sequences

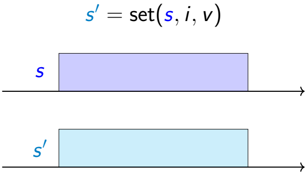
Based on the paper "Reasoning About Vectors: Satisfiability Modulo a Theory of Sequences" by Sheng et al. [She+23], which presents two calculi:

- BASE: based on string reasoning, works by reducing to concatenations.

Porting calculi on Sequences to n-Sequences

Based on the paper "Reasoning About Vectors: Satisfiability Modulo a Theory of Sequences" by Sheng et al. [She+23], which presents two calculi:

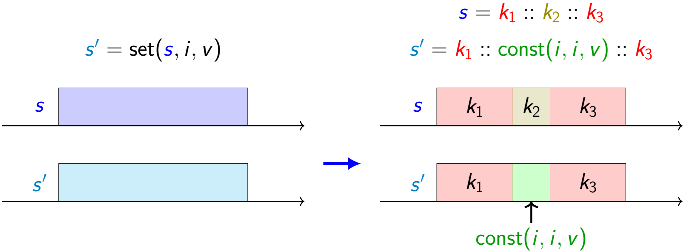
- BASE: based on string reasoning, works by reducing to concatenations.



Porting calculi on Sequences to n-Sequences

Based on the paper "Reasoning About Vectors: Satisfiability Modulo a Theory of Sequences" by Sheng et al. [She+23], which presents two calculi:

- BASE: based on string reasoning, works by reducing to concatenations.



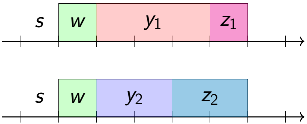
Porting calculi on Sequences to n-Sequences

Based on the paper "Reasoning About Vectors: Satisfiability Modulo a Theory of Sequences" by Sheng et al. [She+23], which presents two calculi:

- BASE: based on string reasoning, works by reducing to concatenations.

$$s = w :: y_1 :: z_1$$

$$s = w :: y_2 :: z_2$$



Porting calculi on Sequences to n-Sequences

Based on the paper "Reasoning About Vectors: Satisfiability Modulo a Theory of Sequences" by Sheng et al. [She+23], which presents two calculi:

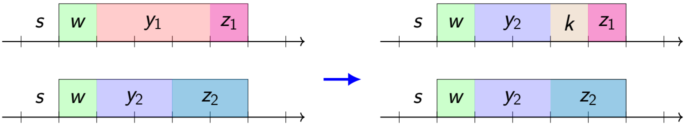
- BASE: based on string reasoning, works by reducing to concatenations.

$s = w :: y_1 :: z_1$

$s = w :: y_2 :: z_2$

$y_1 = y_2 :: k \;\&\; z_2 = k :: z_1$

$s = w :: y_2 :: k :: z_1$



Porting calculi on Sequences to n-Sequences

Based on the paper "Reasoning About Vectors: Satisfiability Modulo a Theory of Sequences" by Sheng et al. [She+23], which presents two calculi:

- BASE: based on string reasoning, works by reducing to concatenations.
- EXT: combines array-like reasoning (for get and set) with string like reasoning for other operations.

Porting calculi on Sequences to n-Sequences

Based on the paper "Reasoning About Vectors: Satisfiability Modulo a Theory of Sequences" by Sheng et al. [She+23], which presents two calculi:

- BASE: based on string reasoning, works by reducing to concatenations.
- EXT: combines array-like reasoning (for get and set) with string like reasoning for other operations.
  - Adapts array axioms to sequences (idx,select-over-store)

Porting calculi on Sequences to n-Sequences

Based on the paper "Reasoning About Vectors: Satisfiability Modulo a Theory of Sequences" by Sheng et al. [She+23], which presents two calculi:

- BASE: based on string reasoning, works by reducing to concatenations.
- EXT: combines array-like reasoning (for get and set) with string like reasoning for other operations.
  - Adapts array axioms to sequences (idx,select-over-store)
  - Propagates get and set operations to normal forms.

Porting calculi on Sequences to n-Sequences

Based on the paper "Reasoning About Vectors: Satisfiability Modulo a Theory of Sequences" by Sheng et al. [She+23], which presents two calculi:

- BASE: based on string reasoning, works by reducing to concatenations.
- EXT: combines array-like reasoning (for get and set) with string like reasoning for other operations.
  - Adapts array axioms to sequences (idx,select-over-store)
  - Propagates get and set operations to normal forms.

Contribution:

The adapted versions are called NS-BASE and NS-EXT, with the changes:

Porting calculi on Sequences to n-Sequences

Based on the paper "Reasoning About Vectors: Satisfiability Modulo a Theory of Sequences" by Sheng et al. [She+23], which presents two calculi:

- BASE: based on string reasoning, works by reducing to concatenations.
- EXT: combines array-like reasoning (for get and set) with string like reasoning for other operations.
  - Adapts array axioms to sequences (idx,select-over-store)
  - Propagates get and set operations to normal forms.

Contribution:

The adapted versions are called NS-BASE and NS-EXT, with the changes:

- The bounds of n-sequences are between the first and the last index

Porting calculi on Sequences to n-Sequences

Based on the paper "Reasoning About Vectors: Satisfiability Modulo a Theory of Sequences" by Sheng et al. [She+23], which presents two calculi:

- BASE: based on string reasoning, works by reducing to concatenations.
- EXT: combines array-like reasoning (for get and set) with string like reasoning for other operations.
  - Adapts array axioms to sequences (idx,select-over-store)
  - Propagates get and set operations to normal forms.

Contribution:

The adapted versions are called NS-BASE and NS-EXT, with the changes:

- The bounds of n-sequences are between the first and the last index
- Reasoning over the relocation of n-indexed sequences

Porting calculi on Sequences to n-Sequences

Based on the paper "Reasoning About Vectors: Satisfiability Modulo a Theory of Sequences" by Sheng et al. [She+23], which presents two calculi:

- BASE: based on string reasoning, works by reducing to concatenations.
- EXT: combines array-like reasoning (for get and set) with string like reasoning for other operations.
  - Adapts array axioms to sequences (idx,select-over-store)
  - Propagates get and set operations to normal forms.

Contribution:

The adapted versions are called NS-BASE and NS-EXT, with the changes:

- The bounds of n-sequences are between the first and the last index
- Reasoning over the relocation of n-indexed sequences

To read more on NS-BASE and NS-EXT:

- **"An SMT Theory for n-Indexed Sequences"**
  Hichem Rami Ait-El-Hara, François Bobot, and Guillaume Bury. SMT 2024
- **"Reasoning over n-indexed sequences in SMT"**
  Hichem Rami Ait-El-Hara, François Bobot, and Guillaume Bury. Acta Informatica 62.3 (Aug. 2025)

Calculi Summary: NS-BASE and NS-EXT

Operations	NS-BASE	NS-EXT
get set	String reasoning	Array reasoning
concat slice update ...	String reasoning	String reasoning

Limitations:

- Eager normalization is often costly and sometimes unnecessary.

Alternative:

- A new calculus that lazily reasons over slices.

Outline

The Shared-Slices (NS-ShS) Calculus I

Consists of representing operations over n-sequences as relations:

The Shared-Slices (NS-ShS) Calculus I

Consists of representing operations over n-sequences as relations:

- Weak-equivalence relations [CH15].

The Shared-Slices (NS-ShS) Calculus I

Consists of representing operations over n-sequences as relations:

- Weak-equivalence relations [CH15].

$$\text{Set-Bound-WEq}\ \frac{s_2 = \mathsf{set}(s_1, i, v)}{}$$

The Shared-Slices (NS-ShS) Calculus I

Consists of representing operations over n-sequences as relations:

- Weak-equivalence relations [CH15].

$$\text{Set-Bound-WEq}\ \frac{s_2 = \mathsf{set}(s_1, i, v)}{(i < \mathsf{f}_{s_1} \vee i > \mathsf{l}_{s_1}) \wedge s_1 = s_2 \qquad \|}$$

The Shared-Slices (NS-ShS) Calculus I

Consists of representing operations over n-sequences as relations:

- Weak-equivalence relations [CH15].

$$\text{Set-Bound-WEq}\;\frac{s_2 = \mathsf{set}(s_1, i, v)}{\begin{array}{c}(i < \mathsf{f}_{s_1} \vee i > \mathsf{l}_{s_1}) \wedge s_1 = s_2 \qquad \| \\ \mathsf{f}_{s_1} \leq i \leq \mathsf{l}_{s_1} \wedge \mathsf{f}_{s_1} = \mathsf{f}_{s_2} \wedge \mathsf{l}_{s_1} = \mathsf{l}_{s_2} \wedge\end{array}}$$

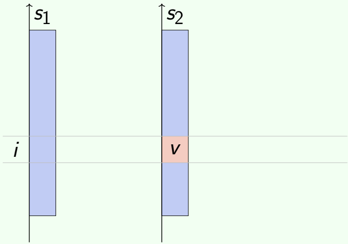
The Shared-Slices (NS-ShS) Calculus I

Consists of representing operations over n-sequences as relations:

- Weak-equivalence relations [CH15].

$$\text{Set-Bound-WEq}\ \frac{s_2 = \mathrm{set}(s_1, i, v)}{\begin{array}{c}(i < \mathrm{f}_{s_1} \vee i > \mathrm{l}_{s_1}) \wedge s_1 = s_2 \qquad \| \\ \mathrm{f}_{s_1} \leq i \leq \mathrm{l}_{s_1} \wedge \mathrm{f}_{s_1} = \mathrm{f}_{s_2} \wedge \mathrm{l}_{s_1} = \mathrm{l}_{s_2} \wedge \\ \mathrm{get}(s_2, i) = v \wedge s_1 \overset{\{i\}}{\leftrightarrow} s_2\end{array}}$$

Illustration



The Shared-Slices (NS-ShS) Calculus I

Consists of representing operations over n-sequences as relations:

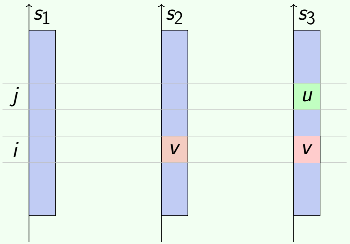
- Weak-equivalence relations [CH15].

$$\text{Set-Bound-WEq}\ \frac{s_2 = \mathrm{set}(s_1, i, v)}{\begin{array}{c}(i < \mathrm{f}_{s_1} \vee i > \mathrm{l}_{s_1}) \wedge s_1 = s_2 \qquad \| \\ \mathrm{f}_{s_1} \leq i \leq \mathrm{l}_{s_1} \wedge \mathrm{f}_{s_1} = \mathrm{f}_{s_2} \wedge \mathrm{l}_{s_1} = \mathrm{l}_{s_2} \wedge \\ \mathrm{get}(s_2, i) = v \wedge s_1 \overset{\{i\}}{\leftrightarrow} s_2\end{array}}$$

Illustration

Given $s_3 = \mathrm{set}(s_2, j, u)$,
With $\mathrm{f}_{s_2} \leq j \leq \mathrm{l}_{s_2}$:

- $\mathrm{f}_{s_2} = \mathrm{f}_{s_3} \wedge \mathrm{l}_{s_2} = \mathrm{l}_{s_3}$
- $s_1 \overset{\{i,j\}}{\leftrightarrow} s_3$



The Shared-Slices (NS-ShS) Calculus II

- The shared-slice relation:

The Shared-Slices (NS-ShS) Calculus II

- The shared-slice relation:

$$s_1 =_{[f;l]} s_2 \implies \forall i.\ f \leq i \leq l \implies \mathrm{get}(s_1, i) = \mathrm{get}(s_2, i)$$

The Shared-Slices (NS-ShS) Calculus II

- The shared-slice relation:

$$s_1 =_{[f;l]} s_2 \implies \forall i.\ f \le i \le l \implies \mathrm{get}(s_1, i) = \mathrm{get}(s_2, i)$$

$$\text{Slice-ShS-Intro}\ \frac{s' = \mathrm{slice}(s, f, l)}{}$$

The Shared-Slices (NS-ShS) Calculus II

- The shared-slice relation:

$$s_1 =_{[f;l]} s_2 \implies \forall i.\ f \leq i \leq l \implies \text{get}(s_1, i) = \text{get}(s_2, i)$$

$$\text{Slice-ShS-Intro}\ \frac{s' = \text{slice}(s, f, l)}{(\mathsf{f}_s > \mathsf{l}_s \vee f > l \vee \mathsf{f}_s > f \vee l > \mathsf{l}_s) \wedge s = s' \qquad ||}$$

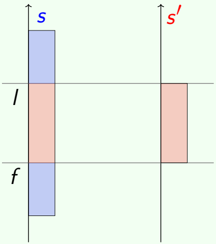
The Shared-Slices (NS-ShS) Calculus II

- The shared-slice relation:

$$s_1 =_{[f;l]} s_2 \implies \forall i.\ f \leq i \leq l \implies \mathrm{get}(s_1, i) = \mathrm{get}(s_2, i)$$

$$\text{Slice-ShS-Intro}\ \frac{s' = \mathrm{slice}(s, f, l)}{\begin{array}{c}(\mathsf{f}_s > \mathsf{l}_s \vee f > l \vee \mathsf{f}_s > f \vee l > \mathsf{l}_s) \wedge s = s' \qquad \| \\ \mathsf{f}_s \leq f \leq l \leq \mathsf{l}_s \wedge \mathsf{f}_{s'} = f \wedge \mathsf{l}_{s'} = l \wedge s =_{[f;l]} s'\end{array}}$$

Illustration



The Shared-Slices (NS-ShS) Calculus II

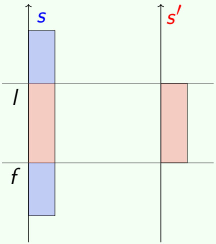
- The shared-slice relation:

$$s_1 =_{[f;l]} s_2 \implies \forall i.\ f \leq i \leq l \implies \text{get}(s_1, i) = \text{get}(s_2, i)$$

$$\text{Slice-ShS-Intro}\ \frac{s' = \text{slice}(s, f, l)}{\begin{array}{c}(\mathsf{f}_s > \mathsf{l}_s \vee f > l \vee \mathsf{f}_s > f \vee l > \mathsf{l}_s) \wedge s = s' \quad \| \\ \mathsf{f}_s \leq f \leq l \leq \mathsf{l}_s \wedge \mathsf{f}_{s'} = f \wedge \mathsf{l}_{s'} = l \wedge s =_{[f;l]} s'\end{array}}$$

Illustration

if $\mathsf{f}_s = f \wedge \mathsf{l}_s = l$ then:



The Shared-Slices (NS-ShS) Calculus II

- The shared-slice relation:

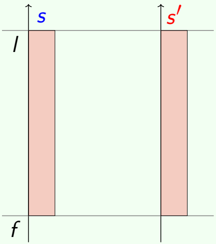
$$s_1 =_{[f;l]} s_2 \implies \forall i.\ f \le i \le l \implies \mathrm{get}(s_1, i) = \mathrm{get}(s_2, i)$$

$$\textsf{Slice-ShS-Intro}\ \frac{s' = \mathrm{slice}(s, f, l)}{\begin{array}{c}(\mathsf{f}_s > \mathsf{l}_s \vee f > l \vee \mathsf{f}_s > f \vee l > \mathsf{l}_s) \wedge s = s' \quad \| \\ \mathsf{f}_s \le f \le l \le \mathsf{l}_s \wedge \mathsf{f}_{s'} = f \wedge \mathsf{l}_{s'} = l \wedge s =_{[f;l]} s'\end{array}}$$

Illustration

if $\mathsf{f}_s = f \wedge \mathsf{l}_s = l$ then:

- $s = s'$.



Reasoning over weak-equivalency and shared-slices

$$\text{Get-Over-WEq}\ \frac{\text{get}(s_1, i) = v \qquad s_1 \overset{K}{\leftrightarrow} s_2}{}$$

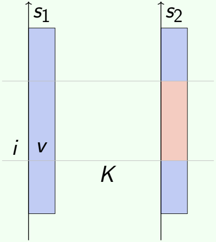
Reasoning over weak-equivalency and shared-slices

$$\text{Get-Over-WEq}\ \frac{\text{get}(s_1, i) = v \qquad s_1 \overset{K}{\leftrightarrow} s_2}{i < \mathsf{f}_{s_1} \vee i > \mathsf{l}_{s_1} \qquad\qquad ||}$$

Reasoning over weak-equivalency and shared-slices

$$\text{Get-Over-WEq}\ \frac{\mathrm{get}(s_1, i) = v \qquad s_1 \overset{K}{\leftrightarrow} s_2}{\begin{array}{ll} i < \mathrm{f}_{s_1} \vee i > \mathrm{l}_{s_1} & \| \\ \exists j \in K.\ i = j & \| \end{array}}$$

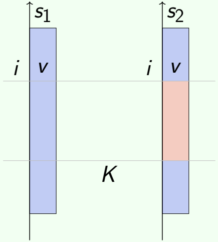
Illustration



Reasoning over weak-equivalency and shared-slices

$$\text{Get-Over-WEq}\ \frac{\mathrm{get}(s_1, i) = v \qquad s_1 \overset{K}{\leftrightarrow} s_2}{\begin{array}{c} i < \mathrm{f}_{s_1} \vee i > \mathrm{l}_{s_1} \quad || \\ \exists j \in K.\ i = j \quad || \\ \mathrm{f}_{s_1} \leq i \leq \mathrm{l}_{s_1} \wedge (\forall j \in K.\ i \neq j) \wedge \mathrm{get}(s_2, i) = v \end{array}}$$

Illustration

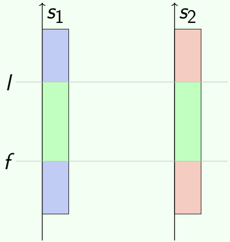


Reasoning over weak-equivalency and shared-slices

$$
\textsc{Get-Over-WEq}\ \frac{\mathrm{get}(s_1, i) = v \qquad s_1 \overset{K}{\leftrightarrow} s_2}{\begin{array}{c} i < \mathrm{f}_{s_1} \lor i > \mathrm{l}_{s_1} \quad \| \\ \exists j \in K.\ i = j \quad \| \\ \mathrm{f}_{s_1} \leq i \leq \mathrm{l}_{s_1} \land (\forall j \in K.\ i \neq j) \land \mathrm{get}(s_2, i) = v \end{array}}
$$

$$
\textsc{Get-Over-ShS}\ \frac{v = \mathrm{get}(s_1, i) \qquad s_1 =_{[f;l]} s_2}{}
$$

Illustration

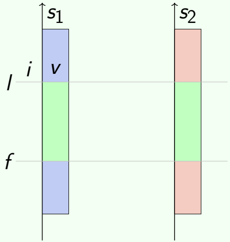


Reasoning over weak-equivalency and shared-slices

$$\text{Get-Over-WEq}\ \frac{\text{get}(s_1, i) = v \qquad s_1 \overset{K}{\leftrightarrow} s_2}{\begin{array}{c} i < \mathsf{f}_{s_1} \vee i > \mathsf{l}_{s_1} \qquad || \\ \exists j \in K.\ i = j \qquad || \\ \mathsf{f}_{s_1} \leq i \leq \mathsf{l}_{s_1} \wedge (\forall j \in K.\ i \neq j) \wedge \text{get}(s_2, i) = v \end{array}}$$

$$\text{Get-Over-ShS}\ \frac{v = \text{get}(s_1, i) \qquad s_1 =_{[f;l]} s_2}{i < f \vee i > l \qquad ||}$$

Illustration

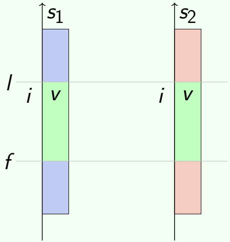


Reasoning over weak-equivalency and shared-slices

$$\text{Get-Over-WEq}\ \frac{\text{get}(s_1, i) = v \qquad s_1 \overset{K}{\leftrightarrow} s_2}{\begin{array}{cr} i < \text{f}_{s_1} \vee i > \text{l}_{s_1} & \| \\ \exists j \in K.\ i = j & \| \\ \text{f}_{s_1} \leq i \leq \text{l}_{s_1} \wedge (\forall j \in K.\ i \neq j) \wedge \text{get}(s_2, i) = v & \end{array}}$$

$$\text{Get-Over-ShS}\ \frac{v = \text{get}(s_1, i) \qquad s_1 =_{[f;l]} s_2}{\begin{array}{cr} i < f \vee i > l & \| \\ f \leq i \leq l \wedge \text{get}(s_2, i) = v & \end{array}}$$

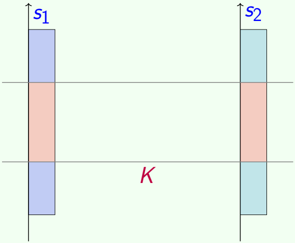
Illustration



Extensionality with NS-ShS

Ext-ShS $s_1 \overset{K}{\leftrightarrow} s_2$

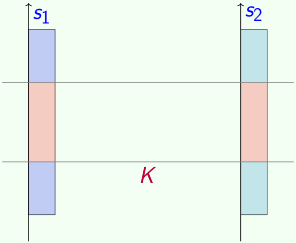
Illustration



Extensionality with NS-ShS

$$\text{Ext-ShS}\ \frac{s_1 \overset{K}{\leftrightarrow} s_2}{s_1 = s_2}\ \|$$

Illustration

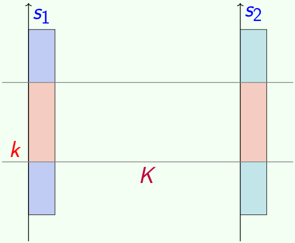


Extensionality with NS-ShS

$$\text{Ext-ShS}\ \frac{s_1 \overset{K}{\leftrightarrow} s_2}{s_1 = s_2} \quad ||$$

$\exists k \in K.\ \mathsf{f}_{s_1} \leq k \leq \mathsf{l}_{s_1} \wedge$

Illustration

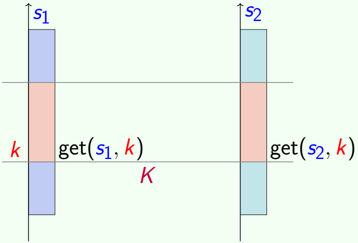


Extensionality with NS-ShS

$$\text{Ext-ShS}\ \frac{s_1 \overset{K}{\leftrightarrow} s_2}{s_1 = s_2} \quad ||$$

$$\exists k \in K.\ \mathsf{f}_{s_1} \leq k \leq \mathsf{l}_{s_1} \wedge \mathrm{get}(s_1, k) \neq \mathrm{get}(s_2, k) \wedge$$

Illustration

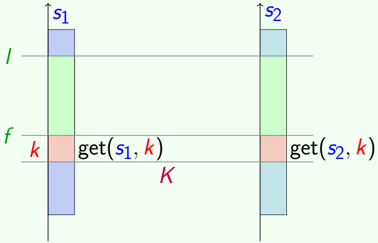


Extensionality with NS-ShS

$$\text{Ext-ShS}\ \frac{s_1 \overset{K}{\leftrightarrow} s_2}{s_1 = s_2} \quad \|$$

$$\exists k \in K.\ \mathsf{f}_{s_1} \leq k \leq \mathsf{l}_{s_1} \wedge \text{get}(s_1, k) \neq \text{get}(s_2, k) \wedge$$
$$\forall f, l.\ s_1 =_{[f;l]} s_2 \implies k < f \vee k > l \wedge$$

Illustration

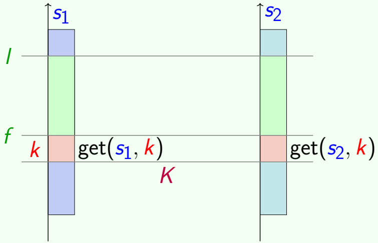


Extensionality with NS-ShS

$$\text{Ext-ShS}\ \frac{s_1 \overset{K}{\leftrightarrow} s_2}{s_1 = s_2} \quad ||$$

$$\exists k \in K.\ \mathrm{f}_{s_1} \leq k \leq \mathrm{l}_{s_1} \wedge \mathrm{get}(s_1, k) \neq \mathrm{get}(s_2, k) \wedge$$
$$\forall f, l.\ s_1 =_{[f;l]} s_2 \implies k < f \vee k > l \wedge$$
$$s_1 \neq s_2$$

Illustration



Calculi Summary: NS-BASE, NS-EXT and NS-ShS

Operations	NS-BASE	NS-EXT	NS-ShS
get set	String reasoning	Array reasoning	Array reasoning
concat slice update ...	String reasoning	String reasoning	Lazy (Shared-slices) reasoning

Outline

Reasoning over relocation 1

Definition (Equivalence modulo relocation)

Given s_1 and s_2 two n-indexed sequences, equivalence modulo relocation is denoted with the relation $s_1 =_{reloc} s_2$, such that:

$$s_1 =_{reloc} s_2 \equiv$$
$$\mathsf{l}_{s_2} = \mathsf{l}_{s_1} - \mathsf{f}_{s_1} + \mathsf{f}_{s_2} \wedge$$
$$\forall i : Int, \mathsf{f}_{s_1} \leq i \leq \mathsf{l}_{s_1} \Rightarrow \mathrm{get}(s_1, i) = \mathrm{get}(s_2, i - \mathsf{f}_{s_1} + \mathsf{f}_{s_2})$$

Reasoning over relocation 1

Definition (Equivalence modulo relocation)

Given s_1 and s_2 two n-indexed sequences, equivalence modulo relocation is denoted with the relation $s_1 =_{reloc} s_2$, such that:

$$s_1 =_{reloc} s_2 \equiv$$
$$\mathsf{l}_{s_2} = \mathsf{l}_{s_1} - \mathsf{f}_{s_1} + \mathsf{f}_{s_2} \wedge$$
$$\forall i : Int, \mathsf{f}_{s_1} \leq i \leq \mathsf{l}_{s_1} \Rightarrow \mathsf{get}(s_1, i) = \mathsf{get}(s_2, i - \mathsf{f}_{s_1} + \mathsf{f}_{s_2})$$

$$\text{Reloc-Bounds}\ \frac{s' = \mathsf{relocate}(s, i)}{}$$

Reasoning over relocation 1

Definition (Equivalence modulo relocation)

Given s_1 and s_2 two n-indexed sequences, equivalence modulo relocation is denoted with the relation $s_1 =_{reloc} s_2$, such that:

$$s_1 =_{reloc} s_2 \equiv$$
$$\mathsf{l}_{s_2} = \mathsf{l}_{s_1} - \mathsf{f}_{s_1} + \mathsf{f}_{s_2} \land$$
$$\forall i : Int, \mathsf{f}_{s_1} \leq i \leq \mathsf{l}_{s_1} \Rightarrow \mathrm{get}(s_1, i) = \mathrm{get}(s_2, i - \mathsf{f}_{s_1} + \mathsf{f}_{s_2})$$

$$\text{Reloc-Bounds}\ \frac{s' = \mathrm{relocate}(s, i)}{i = \mathsf{f}_s \land s' = s \qquad ||}$$

Reasoning over relocation 1

Definition (Equivalence modulo relocation)

Given s_1 and s_2 two n-indexed sequences, equivalence modulo relocation is denoted with the relation $s_1 =_{reloc} s_2$, such that:

$$s_1 =_{reloc} s_2 \equiv$$
$$\mathsf{l}_{s_2} = \mathsf{l}_{s_1} - \mathsf{f}_{s_1} + \mathsf{f}_{s_2} \land$$
$$\forall i : Int, \mathsf{f}_{s_1} \leq i \leq \mathsf{l}_{s_1} \Rightarrow \mathrm{get}(s_1, i) = \mathrm{get}(s_2, i - \mathsf{f}_{s_1} + \mathsf{f}_{s_2})$$

$$\text{Reloc-Bounds}\ \frac{s' = \mathrm{relocate}(s, i)}{\begin{array}{c} i = \mathsf{f}_s \land s' = s \qquad || \\ i \neq \mathsf{f}_s \land \mathsf{f}_{s'} = i \land \mathsf{l}_{s'} = i + \mathsf{l}_s - \mathsf{f}_s \land \end{array}}$$

Reasoning over relocation 1

Definition (Equivalence modulo relocation)

Given s_1 and s_2 two n-indexed sequences, equivalence modulo relocation is denoted with the relation $s_1 =_{reloc} s_2$, such that:

$$s_1 =_{reloc} s_2 \equiv$$
$$\mathsf{l}_{s_2} = \mathsf{l}_{s_1} - \mathsf{f}_{s_1} + \mathsf{f}_{s_2} \land$$
$$\forall i : Int, \mathsf{f}_{s_1} \leq i \leq \mathsf{l}_{s_1} \Rightarrow \mathrm{get}(s_1, i) = \mathrm{get}(s_2, i - \mathsf{f}_{s_1} + \mathsf{f}_{s_2})$$

$$\text{Reloc-Bounds}\ \frac{s' = \mathrm{relocate}(s, i)}{\begin{array}{c} i = \mathsf{f}_s \land s' = s \qquad\qquad || \\ i \neq \mathsf{f}_s \land \mathsf{f}_{s'} = i \land \mathsf{l}_{s'} = i + \mathsf{l}_s - \mathsf{f}_s \land \\ s' =_{reloc} s \end{array}}$$

Reasoning over relocation 1

Definition (Equivalence modulo relocation)

Given s_1 and s_2 two n-indexed sequences, equivalence modulo relocation is denoted with the relation $s_1 =_{reloc} s_2$, such that:

$$s_1 =_{reloc} s_2 \equiv$$
$$\mathsf{l}_{s_2} = \mathsf{l}_{s_1} - \mathsf{f}_{s_1} + \mathsf{f}_{s_2} \land$$
$$\forall i : Int, \mathsf{f}_{s_1} \leq i \leq \mathsf{l}_{s_1} \Rightarrow \mathrm{get}(s_1, i) = \mathrm{get}(s_2, i - \mathsf{f}_{s_1} + \mathsf{f}_{s_2})$$

$$\text{Reloc-Bounds}\ \frac{s' = \mathrm{relocate}(s, i)}{\begin{array}{c} i = \mathsf{f}_s \land s' = s \qquad\qquad || \\ i \neq \mathsf{f}_s \land \mathsf{f}_{s'} = i \land \mathsf{l}_{s'} = i + \mathsf{l}_s - \mathsf{f}_s \land \\ s' =_{reloc} s \end{array}}$$

This reasoning is used for all the three calculi: NS-BASE, NS-EXT and NS-ShS.

Reasoning over relocation 2

$$\text{NS-Comp-Reloc}\ \frac{s = k_1 :: k_2 :: ... :: k_n \qquad s =_{reloc} r}{}$$

Reasoning over relocation 2

$$\text{NS-Comp-Reloc}\ \frac{s = k_1 :: k_2 :: ... :: k_n \qquad s =_{reloc} r}{\begin{array}{c} r = \mathrm{relocate}(k_1, \mathrm{f}_r) :: \\ \mathrm{relocate}(k_2, \mathrm{f}_{k_2} - \mathrm{f}_s + \mathrm{f}_r) :: ... :: \\ \mathrm{relocate}(k_n, \mathrm{f}_{k_n} - \mathrm{f}_s + \mathrm{f}_r) \end{array}}$$

Reasoning over relocation 2

$$\text{NS-Comp-Reloc}\ \frac{s = k_1 :: k_2 :: ... :: k_n \qquad s =_{reloc} r}{\begin{array}{c} r = \text{relocate}(k_1, \text{f}_r) :: \\ \text{relocate}(k_2, \text{f}_{k_2} - \text{f}_s + \text{f}_r) :: ... :: \\ \text{relocate}(k_n, \text{f}_{k_n} - \text{f}_s + \text{f}_r) \end{array}}$$

$$\text{Get-Reloc}\ \frac{v = \text{get}(s, i) \qquad s =_{reloc} r}{}$$

Reasoning over relocation 2

$$\text{NS-Comp-Reloc}\ \frac{s = k_1 :: k_2 :: ... :: k_n \qquad s =_{reloc} r}{\begin{array}{c} r = \text{relocate}(k_1, \mathrm{f}_r) :: \\ \text{relocate}(k_2, \mathrm{f}_{k_2} - \mathrm{f}_s + \mathrm{f}_r) :: ... :: \\ \text{relocate}(k_n, \mathrm{f}_{k_n} - \mathrm{f}_s + \mathrm{f}_r) \end{array}}$$

$$\text{Get-Reloc}\ \frac{v = \text{get}(s, i) \qquad s =_{reloc} r}{i < f_s \vee l_s < i \quad ||}$$

Reasoning over relocation 2

$$\text{NS-Comp-Reloc}\ \frac{s = k_1 :: k_2 :: \ldots :: k_n \qquad s =_{reloc} r}{\begin{array}{c} r = \mathrm{relocate}(k_1, \mathrm{f}_r) :: \\ \mathrm{relocate}(k_2, \mathrm{f}_{k_2} - \mathrm{f}_s + \mathrm{f}_r) :: \ldots :: \\ \mathrm{relocate}(k_n, \mathrm{f}_{k_n} - \mathrm{f}_s + \mathrm{f}_r) \end{array}}$$

$$\text{Get-Reloc}\ \frac{v = \mathrm{get}(s, i) \qquad s =_{reloc} r}{i < f_s \vee l_s < i \quad || \quad f_s \leq i \leq l_s \wedge v = \mathrm{get}(r, i - \mathrm{f}_s + \mathrm{f}_r)}$$

Reasoning over relocation 2

$$\text{NS-Comp-Reloc}\,\frac{s = k_1 :: k_2 :: \ldots :: k_n \qquad s =_{reloc} r}{\begin{array}{c} r = \text{relocate}(k_1, \mathrm{f}_r) :: \\ \text{relocate}(k_2, \mathrm{f}_{k_2} - \mathrm{f}_s + \mathrm{f}_r) :: \ldots :: \\ \text{relocate}(k_n, \mathrm{f}_{k_n} - \mathrm{f}_s + \mathrm{f}_r) \end{array}}$$

$$\text{Get-Reloc}\,\frac{v = \text{get}(s, i) \qquad s =_{reloc} r}{i < f_s \vee l_s < i \quad || \quad f_s \leq i \leq l_s \wedge v = \text{get}(r, i - \mathrm{f}_s + \mathrm{f}_r)}$$

- Applying NS-Comp-Reloc and Get-Reloc eagerly can be **costly**.

Reasoning over relocation 2

$$\text{NS-Comp-Reloc}\ \frac{s = k_1 :: k_2 :: \ldots :: k_n \qquad s =_{reloc} r}{\begin{array}{c} r = \text{relocate}(k_1, \mathrm{f}_r) :: \\ \text{relocate}(k_2, \mathrm{f}_{k_2} - \mathrm{f}_s + \mathrm{f}_r) :: \ldots :: \\ \text{relocate}(k_n, \mathrm{f}_{k_n} - \mathrm{f}_s + \mathrm{f}_r) \end{array}}$$

$$\text{Get-Reloc}\ \frac{v = \text{get}(s, i) \qquad s =_{reloc} r}{i < f_s \lor l_s < i \quad || \quad f_s \leq i \leq l_s \land v = \text{get}(r, i - \mathrm{f}_s + \mathrm{f}_r)}$$

- Applying NS-Comp-Reloc and Get-Reloc eagerly can be **costly**.
- Our extension to the union-find data structure helps mitigate that.

Outline

Implementation context

NS-Base, NS-Ext and NS-ShS were implemented in Colibri2:

- A reimplementation in OCaml of the COLIBRI CP solver.
- A CP solver used to reason over SMT problems.
- That does not use a SAT solver or clause learning.
- Compensates with (abstract) domains, propagations and scheduling.

Outline

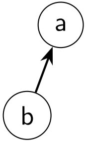
Equivalence modulo relocation I

The equivalence modulo relocation relation is represented using a **labeled union-find**.

Equivalence modulo relocation I

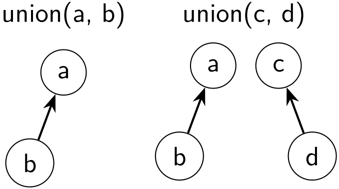
The equivalence modulo relocation relation is represented using a **labeled union-find**.

union(a, b)



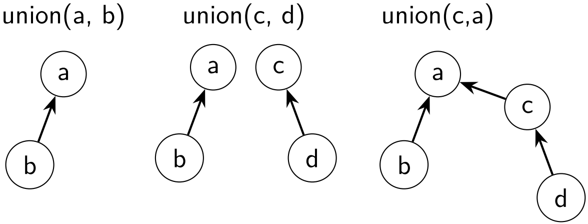
Equivalence modulo relocation I

The equivalence modulo relocation relation is represented using a **labeled union-find**.



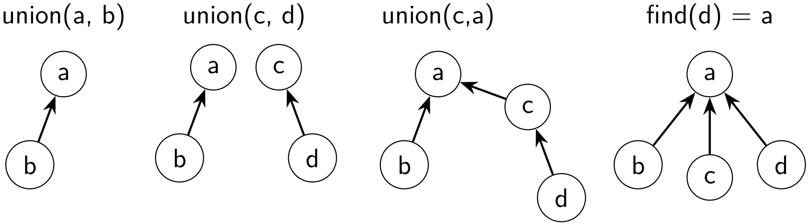
Equivalence modulo relocation I

The equivalence modulo relocation relation is represented using a **labeled union-find**.



Equivalence modulo relocation I

The equivalence modulo relocation relation is represented using a **labeled union-find**.



Equivalence modulo relocation I

The equivalence modulo relocation relation is represented using a **labeled union-find**.

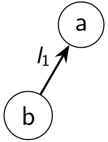
Definition

Labeled union-find is an extension of union-find in which the relation between elements is parametrized (labeled).

Equivalence modulo relocation I

The equivalence modulo relocation relation is represented using a **labeled union-find**.

add_relation(b, l_1, a)



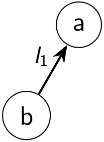
Definition

Labeled union-find is an extension of union-find in which the relation between elements is parametrized (labeled).

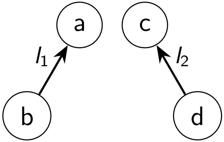
Equivalence modulo relocation I

The equivalence modulo relocation relation is represented using a **labeled union-find**.

add_relation(b, l_1, a)



add_relation(d, l_2, c)

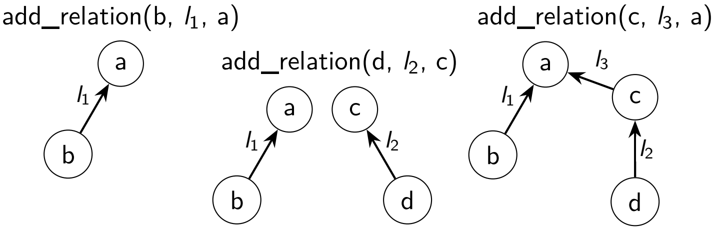


Definition

Labeled union-find is an extension of union-find in which the relation between elements is parametrized (labeled).

Equivalence modulo relocation I

The equivalence modulo relocation relation is represented using a **labeled union-find**.

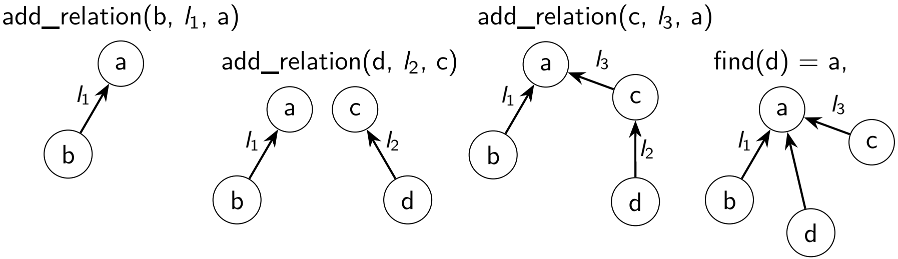


Definition

Labeled union-find is an extension of union-find in which the relation between elements is parametrized (labeled).

Equivalence modulo relocation I

The equivalence modulo relocation relation is represented using a **labeled union-find**.

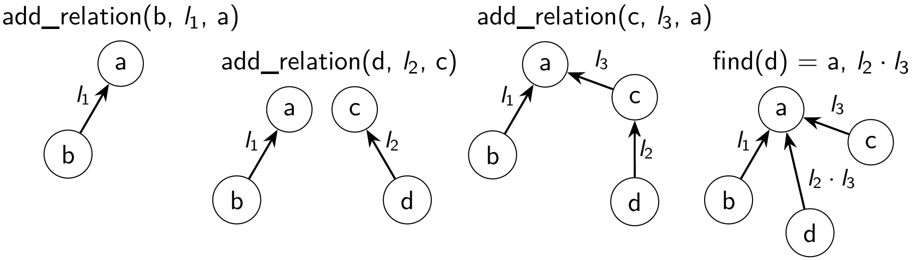


Definition

Labeled union-find is an extension of union-find in which the relation between elements is parametrized (labeled).

Equivalence modulo relocation I

The equivalence modulo relocation relation is represented using a **labeled union-find**.

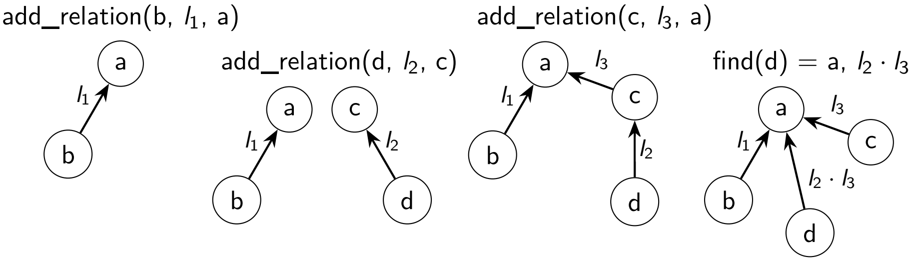


Definition

Labeled union-find is an extension of union-find in which the relation between elements is parametrized (labeled).
The labels have a **composition operation** that:

Equivalence modulo relocation I

The equivalence modulo relocation relation is represented using a **labeled union-find**.



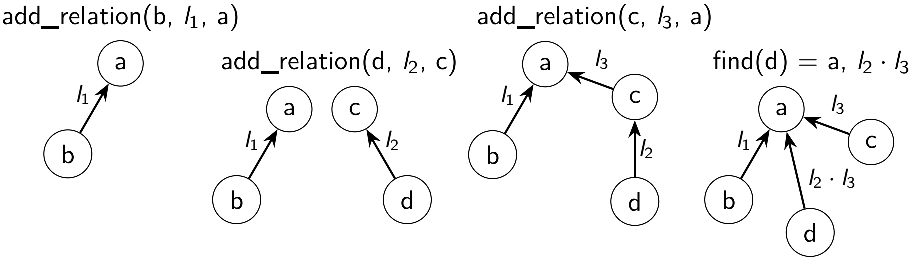
Definition

Labeled union-find is an extension of union-find in which the relation between elements is parametrized (labeled).
The labels have a **composition operation** that:

- Is invertible.

Equivalence modulo relocation I

The equivalence modulo relocation relation is represented using a **labeled union-find**.



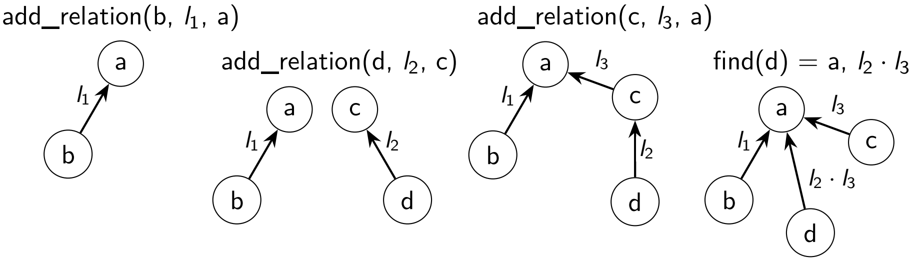
Definition

Labeled union-find is an extension of union-find in which the relation between elements is parametrized (labeled).
The labels have a **composition operation** that:

- Is invertible.
- Is associative.

Equivalence modulo relocation I

The equivalence modulo relocation relation is represented using a **labeled union-find**.



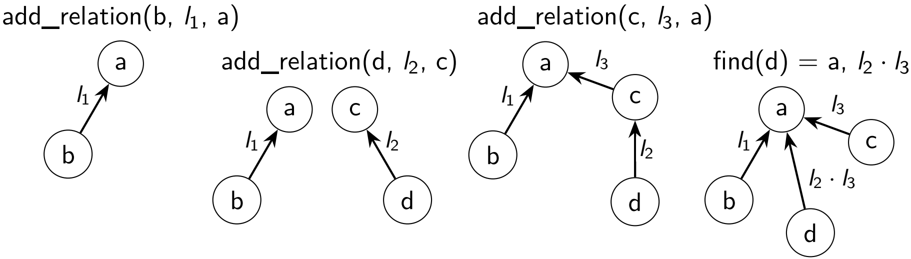
Definition

Labeled union-find is an extension of union-find in which the relation between elements is parametrized (labeled).
The labels have a **composition operation** that:

- Is invertible.
- Is associative.
- Has an identity element.

Equivalence modulo relocation I

The equivalence modulo relocation relation is represented using a **labeled union-find**.



Definition

Labeled union-find is an extension of union-find in which the relation between elements is parametrized (labeled).
The labels have a **composition operation** that:

- Is invertible.
- Is associative.
- Has an identity element.

Forming a **group** with the labels.

Equivalence modulo relocation II

In the **labeled union-find** used to represent equivalence modulo relocation:

Equivalence modulo relocation II

In the **labeled union-find** used to represent equivalence modulo relocation:

- Nodes: n-sequences.

Equivalence modulo relocation II

In the **labeled union-find** used to represent equivalence modulo relocation:

- Nodes: n-sequences.
- Labels: linear polynomials.

Equivalence modulo relocation II

In the **labeled union-find** used to represent equivalence modulo relocation:

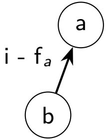
- Nodes: n-sequences.
- Labels: linear polynomials.
- Composition operation: integer addition.

Equivalence modulo relocation II

In the **labeled union-find** used to represent equivalence modulo relocation:

- Nodes: n-sequences.
- Labels: linear polynomials.
- Composition operation: integer addition.

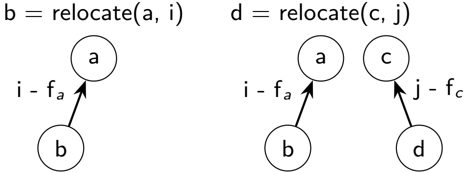
b = relocate(a, i)



Equivalence modulo relocation II

In the **labeled union-find** used to represent equivalence modulo relocation:

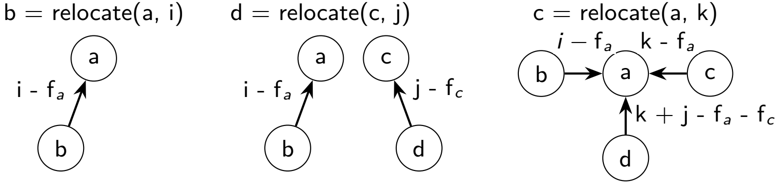
- Nodes: n-sequences.
- Labels: linear polynomials.
- Composition operation: integer addition.



Equivalence modulo relocation II

In the **labeled union-find** used to represent equivalence modulo relocation:

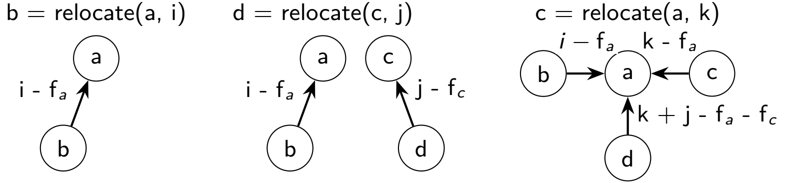
- Nodes: n-sequences.
- Labels: linear polynomials.
- Composition operation: integer addition.



Equivalence modulo relocation II

In the **labeled union-find** used to represent equivalence modulo relocation:

- Nodes: n-sequences.
- Labels: linear polynomials.
- Composition operation: integer addition.



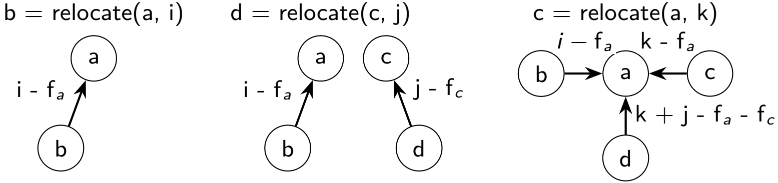
In the implementation, it also holds a domain:

$$r \mapsto M : \left\{ \begin{array}{c} 0 \mapsto r \\ \delta_1 \mapsto s_1 \\ \dots \\ \delta_n \mapsto s_n \end{array} \right\}$$

Equivalence modulo relocation II

In the **labeled union-find** used to represent equivalence modulo relocation:

- Nodes: n-sequences.
- Labels: linear polynomials.
- Composition operation: integer addition.



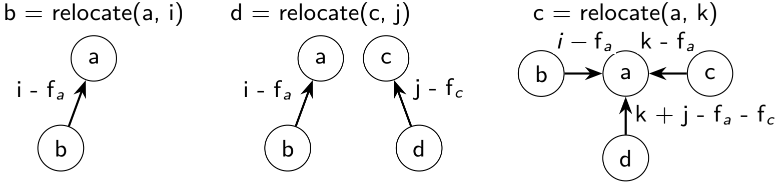
In the implementation, it also holds a domain:

$$r \mapsto M : \left\{ \begin{array}{c} 0 \mapsto r \\ \delta_1 \mapsto s_1 \\ \dots \\ \delta_n \mapsto s_n \end{array} \right\} \cup \{\delta_i \mapsto s'_i\}$$

Equivalence modulo relocation II

In the **labeled union-find** used to represent equivalence modulo relocation:

- Nodes: n-sequences.
- Labels: linear polynomials.
- Composition operation: integer addition.

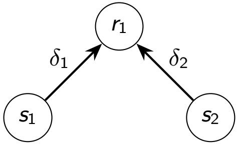


In the implementation, it also holds a domain:

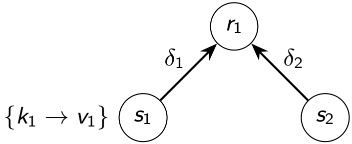
$$r \mapsto M : \left\{ \begin{array}{c} 0 \mapsto r \\ \delta_1 \mapsto s_1 \\ \dots \\ \delta_n \mapsto s_n \end{array} \right\} \cup \{\delta_i \mapsto s_i'\} \to \delta_i \in Dom(M) \implies s_i' = s_i$$

Outline

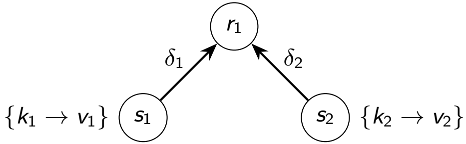
Constraint factorization



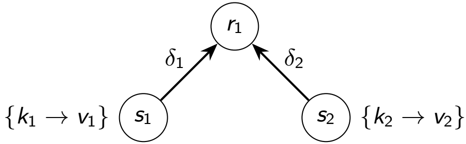
Constraint factorization



Constraint factorization

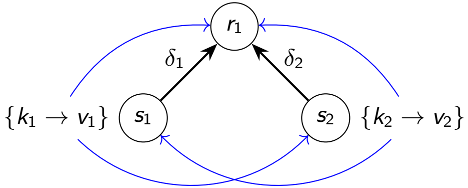


Constraint factorization



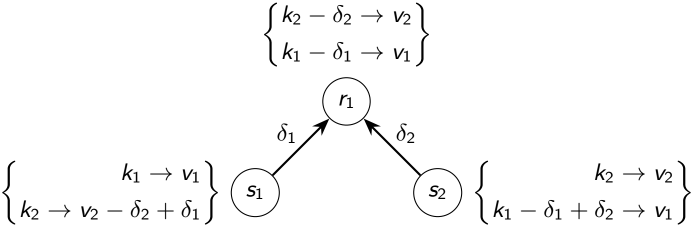
$$\text{Get-Reloc}\ \frac{v = \mathrm{get}(s, i) \qquad s =_{reloc} r}{i < f_s \vee l_s < i \quad || \quad f_s \leq i \leq l_s \wedge v = \mathrm{get}(r, i - \mathrm{f}_s + \mathrm{f}_r)}$$

Constraint factorization



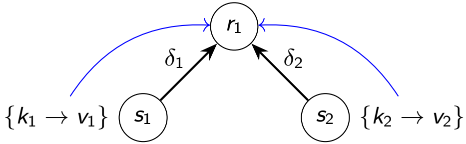
$$\text{Get-Reloc}\;\frac{v = \mathrm{get}(s, i) \qquad s =_{reloc} r}{i < f_s \vee l_s < i \quad || \quad f_s \leq i \leq l_s \wedge v = \mathrm{get}(r, i - \mathrm{f}_s + \mathrm{f}_r)}$$

Constraint factorization



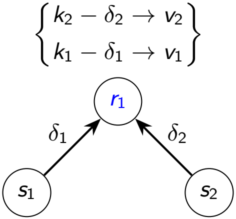
$$\text{Get-Reloc}\;\frac{v = \mathrm{get}(s, i) \qquad s =_{reloc} r}{i < f_s \vee l_s < i \quad || \quad f_s \leq i \leq l_s \wedge v = \mathrm{get}(r, i - \mathrm{f}_s + \mathrm{f}_r)}$$

Constraint factorization



$$\text{Get-Reloc}\ \frac{v = \text{get}(s, i) \qquad s =_{reloc} r}{i < f_s \vee l_s < i \quad || \quad f_s \leq i \leq l_s \wedge v = \text{get}(r, i - \mathrm{f}_s + \mathrm{f}_r)}$$

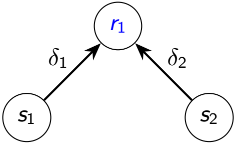
Constraint factorization



$$\text{Get-Reloc}\ \frac{v = \text{get}(s, i) \qquad s =_{reloc} r}{i < f_s \lor l_s < i \quad || \quad f_s \le i \le l_s \land v = \text{get}(r, i - \text{f}_s + \text{f}_r)}$$

Constraint factorization

$$\begin{Bmatrix} k_2 - \delta_2 \rightarrow v_2 \\ k_1 - \delta_1 \rightarrow v_1 \end{Bmatrix}$$



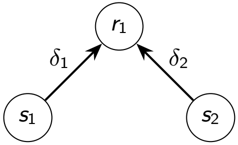
$$\text{Get-Reloc}\,\frac{v = \text{get}(s, i) \qquad s =_{reloc} r}{i < f_s \lor l_s < i \quad || \quad f_s \leq i \leq l_s \land v = \text{get}(r, i - \mathrm{f}_s + \mathrm{f}_r)}$$

Also applies to NS-Comp-Reloc:

$$\text{NS-Comp-Reloc}\,\frac{s = k_1 :: k_2 :: ... :: k_n \qquad s =_{reloc} r}{\begin{array}{c} r = \text{relocate}(k_1, \mathrm{f}_r) :: \\ \text{relocate}(k_2, \mathrm{f}_{k_2} - \mathrm{f}_s + \mathrm{f}_r) :: ... :: \\ \text{relocate}(k_n, \mathrm{f}_{k_n} - \mathrm{f}_s + \mathrm{f}_r) \end{array}}$$

Constraint factorization

$$\left\{ \begin{array}{l} k_2 - \delta_2 \rightarrow v_2 \\ k_1 - \delta_1 \rightarrow v_1 \end{array} \right\}$$



To read more on how it is used in arithmetic reasoning:

- **"Relational Abstractions Based on Labeled Union-Find"**
  Dorian Lesbre, Matthieu Lemerre, Hichem Rami Ait-El-Hara, and François Bobot. PLDI 2025

Outline

Encoding sequences over n-Indexed Sequences

Encoding sequences over n-Indexed Sequences

- Each sequences s: An n-sequence with $f_s = 0$ and $l_s \geq -1$

Encoding sequences over n-Indexed Sequences

- Each sequences s: An n-sequence with $\mathsf{f}_s = 0$ and $\mathsf{l}_s \geq -1$
- seq.empty: Represented by a special constant symbol ϵ, an empty n-sequence with $\mathsf{f}_\epsilon = 0$ and $\mathsf{l}_\epsilon = -1$.

Encoding sequences over n-Indexed Sequences

- Each sequences s: An n-sequence with $\mathsf{f}_s = 0$ and $\mathsf{l}_s \geq -1$
- seq.empty: Represented by a special constant symbol ϵ, an empty n-sequence with $\mathsf{f}_\epsilon = 0$ and $\mathsf{l}_\epsilon = -1$.
- seq.$++(s_1, s_2, s_3, \ldots, s_n)$:

$$\begin{gathered}\mathrm{let}\big(c_1, \mathrm{concat}(s_1, \mathrm{relocate}(s_2, \mathsf{l}_{s_1} + 1)),\\ \mathrm{let}(c_2, \mathrm{concat}(c_1, \mathrm{relocate}(s_3, \mathsf{l}_{c_1} + 1)),\\ \ldots\\ \mathrm{concat}(c_{n-2}, \mathrm{relocate}(s_n, \mathsf{l}_{c_{n-2}} + 1)))))\big)\end{gathered}$$

Outline

Experimental evaluation: context

- The experimentation was done on quantifier free sequence and n-sequence benchmarks, containing only sequence and n-sequence operations.
- The experimentation compares implementations of NS-BASE, NS-EXT and NS-ShS in Colibri2 with:
  - Sequence support in cvc5 and Z3.
  - Support for n-sequences encoded with ADTs and Sequences in cvc5 and Z3.

Experimental evaluation: UNSAT Seq

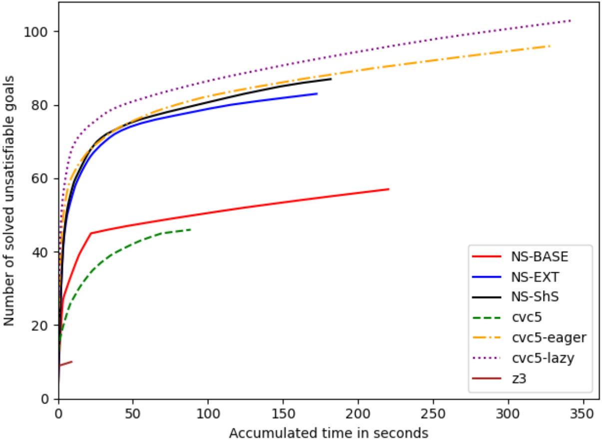


Figure: Number of solved goals by accumulated time (in seconds) on unsatisfiable quantifier-free Sequence benchmarks.

Experimental evaluation: SAT Seq

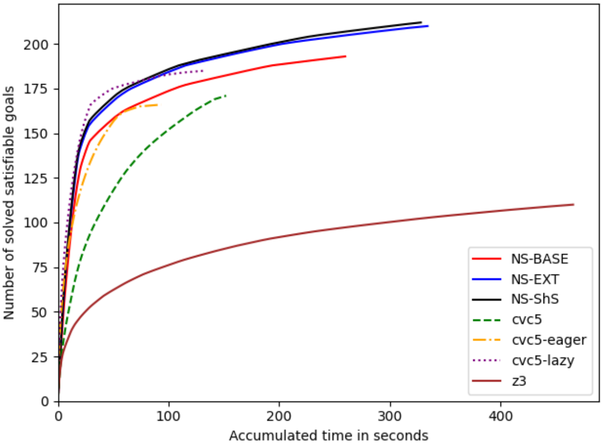


Figure: Number of solved goals by accumulated time (in seconds) on satisfiable quantifier-free Sequence benchmarks.

Experimental evaluation: UNSAT NSeq

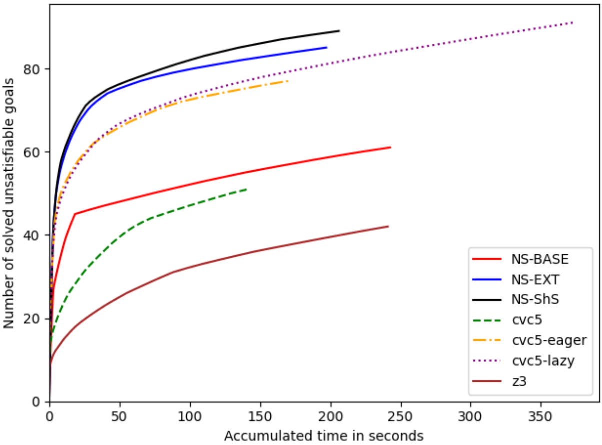


Figure: Number of solved goals by accumulated time (in seconds) on unsatisfiable quantifier-free *n*-Indexed Sequence benchmarks.

Experimental evaluation: SAT NSeq

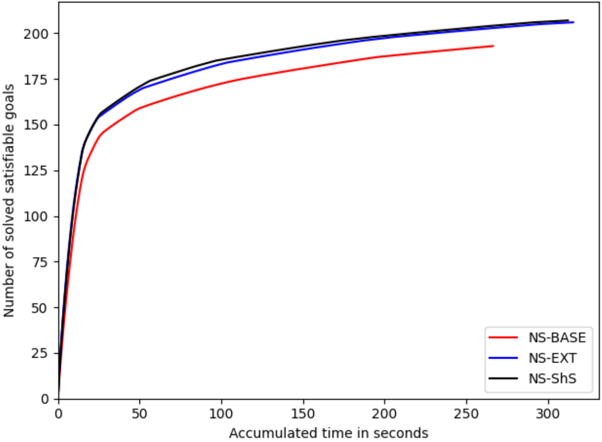


Figure: Number of solved goals by accumulated time (in seconds) on satisfiable quantifier-free *n*-Indexed Sequence benchmarks.

Outline

Conclusion

Conclusion

Contributions presented in this talk:

- The theory of n-Indexed Sequences.
- Various ways to reason over it.
- Experimental evaluation.

Conclusion

Contributions presented in this talk:

- The theory of n-Indexed Sequences.
- Various ways to reason over it.
- Experimental evaluation.

Additional contributions in the manuscript:

- Soundness proofs.
- Implementation details and formalizations.
- Work on real and integer arithmetic reasoning.
  (Labeled union-find for intervals and difference logic)

Conclusion

Contributions presented in this talk:

- The theory of n-Indexed Sequences.
- Various ways to reason over it.
- Experimental evaluation.

Additional contributions in the manuscript:

- Soundness proofs.
- Implementation details and formalizations.
- Work on real and integer arithmetic reasoning.
  (Labeled union-find for intervals and difference logic)

Future work:

- Acquire more benchmarks
- Add (n-)sequences to Alt-Ergo
- Improve reasoning over n-sequences with quantifiers.

Conclusion

Contributions presented in this talk:

- The theory of n-Indexed Sequences.
- Various ways to reason over it.
- Experimental evaluation.

Additional contributions in the manuscript:

- Soundness proofs.
- Implementation details and formalizations.
- Work on real and integer arithmetic reasoning. (Labeled union-find for intervals and difference logic)

Future work:

- Acquire more benchmarks
- Add (n-)sequences to Alt-Ergo
- Improve reasoning over n-sequences with quantifiers.

Contributions to software:

- Colibri2
- Alt-Ergo
- Smtml
- Dolmen
- SMT LSP

Conclusion

Contributions presented in this talk:

- The theory of n-Indexed Sequences.
- Various ways to reason over it.
- Experimental evaluation.

Additional contributions in the manuscript:

- Soundness proofs.
- Implementation details and formalizations.
- Work on real and integer arithmetic reasoning. (Labeled union-find for intervals and difference logic)

Future work:

- Acquire more benchmarks
- Add (n-)sequences to Alt-Ergo
- Improve reasoning over n-sequences with quantifiers.

Other:

- Co-supervized an intern for 6 months (Félix Loyau-Kahn, Master's student) on using AI for SMT solver selection.

Contributions to software:

- Colibri2
- Alt-Ergo
- Smtml
- Dolmen
- SMT LSP

Publications

- **"On SMT Theory Design: The Case of Sequences"**
  Hichem Rami Ait-El-Hara, François Bobot and Guillaume Bury. LPAR 2024
- **"An SMT Theory for n-Indexed Sequences"**
  Hichem Rami Ait-El-Hara, François Bobot, and Guillaume Bury. SMT 2024
- **"Reasoning over n-indexed sequences in SMT"**
  Hichem Rami Ait-El-Hara, François Bobot, and Guillaume Bury. Acta Informatica 62.3 (Aug. 2025)
- **"Relational Abstractions Based on Labeled Union-Find"**
  Dorian Lesbre, Matthieu Lemerre, Hichem Rami Ait-El-Hara, and François Bobot. PLDI 2025

- **"Constraint Propagation for Bit-Vectors in Alt-Ergo"**
  Hichem Rami Ait-El-Hara, Guillaume Bury, Basile Clément, and Pierre Villemot. SMT 2025

Preprints:

- **"Smt.ml: A Multi-Backend Frontend for SMT Solvers in OCaml"**
  João Madeira Pereira, Filipe Marques, Pedro Adão, Hichem Rami Ait-El-Hara, Léo Andrès, Arthur Carcano, Pierre Chambart, Nuno Santos, and José Fragoso Santos. To be submitted to TACAS 2026.

Appendix 1: bibliography I

[Bjø+12] N Bjørner et al. "An SMT-LIB Format for Sequences and Regular Expressions". In: *Strings* (Jan. 2012).

[CH15] Jürgen Christ and Jochen Hoenicke. "Weakly Equivalent Arrays". In: *Frontiers of Combining Systems*. Ed. by Carsten Lutz and Silvio Ranise. Cham: Springer International Publishing, 2015, pp. 119–134. ISBN: 978-3-319-24246-0. DOI: 10.1007/978-3-319-24246-0_8.

[MB09] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. "Generalized, efficient array decision procedures". In: *Proceedings of 9th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2009, 15-18 November 2009, Austin, Texas, USA*. IEEE, 2009, pp. 45–52. DOI: 10.1109/FMCAD.2009.5351142.

[McC62] John McCarthy. "Towards a Mathematical Science of Computation". In: *Information Processing, Proceedings of the 2nd IFIP Congress 1962, Munich, Germany, August 27 - September 1, 1962*. North-Holland, 1962, pp. 21–28.

[She+23] Ying Sheng et al. "Reasoning About Vectors: Satisfiability Modulo a Theory of Sequences". In: *Journal of Automated Reasoning* 67.3 (Sept. 2023), p. 32. ISSN: 1573-0670. DOI: 10.1007/s10817-023-09682-2.

Appendix

6. Labeled Union-Find for Arithmetic reasoning

7. NS-BASE and NS-EXT

Reduced Product

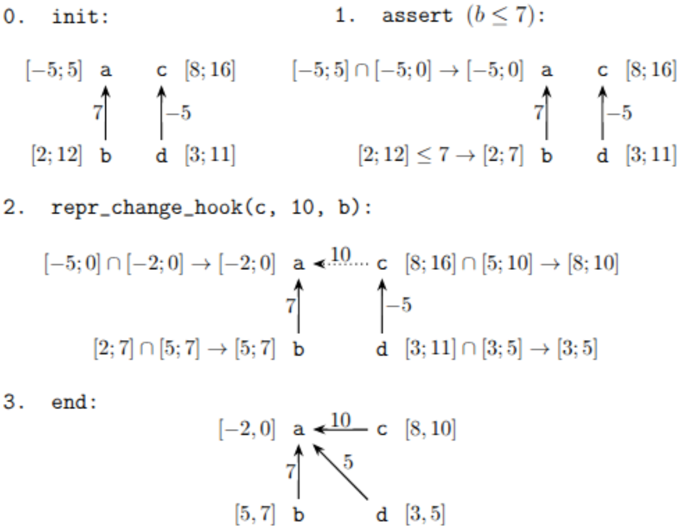


Figure 6.4: Example of the usage of the constant difference relation for constraint propagation over the domain of intervals.

Group Action

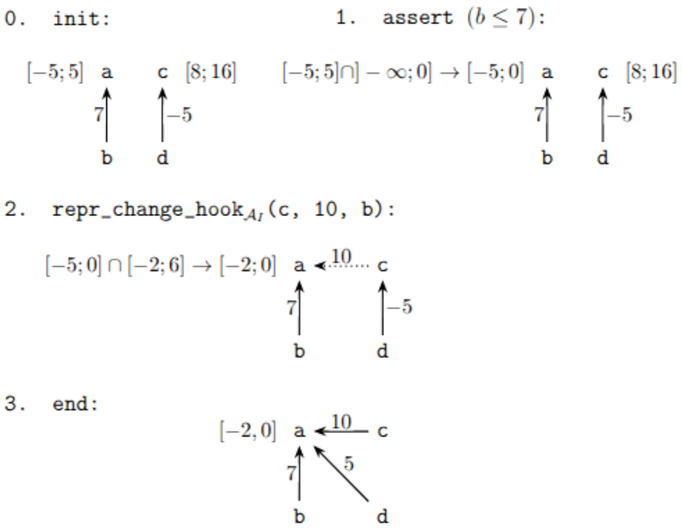


Figure 6.5: Example of the usage of the constant difference relation for constraint factorization over the domain of intervals.

Normal forms

Definition (NSeq term normal form)

For simplicity, we introduce the concatenation operator :: with the invariant:

$$s \mapsto s_1 :: s_2 \implies \mathsf{f}_s = \mathsf{f}_{s_1} \wedge \mathsf{l}_s = \mathsf{l}_{s_2} \wedge \mathsf{f}_{s_2} = \mathsf{l}_{s_1} + 1$$

Normalization

The following rewriting rules are applied whenever possible:

$$\begin{cases} s \mapsto [w_1 ::]x[:: w_2] \\ x \mapsto y :: z \end{cases} \longrightarrow \begin{cases} s \mapsto [w_1 ::]y :: z[:: w_2] \\ x \mapsto y :: z \end{cases}$$

And if $\mathsf{l}_y < \mathsf{f}_y$ is deduced:

$$\begin{cases} s \mapsto [w_1 ::]y :: z[:: w_2] \\ x \mapsto y :: z \end{cases} \longrightarrow \begin{cases} s \mapsto [w_1 ::]z[:: w_2] \\ x \mapsto z \end{cases}$$