Deciding Boolean Separation Logic via Small Models

Alpine Verification Meeting 2024

Tomáš Dacík[1,*] Adam Rogalewicz[1] Tomáš Vojnar[1] Florian Zuleger[2]

[1] Brno University of Technology, Faculty of Information Technology
[2] TU Wien, Faculty of Informatics

* Supported by Brno Ph.D. Talent scholarship

Separation Logic

- Frequently used for reasoning about heap-manipulating programs
  - Separating conjunction $\varphi * \psi$ – the heap can be split into disjoint parts satisfying φ and ψ
  - Inductive predicates describing data structures (lists, trees, ...)

Separation Logic

- Frequently used for reasoning about heap-manipulating programs
  - Separating conjunction $\varphi * \psi$ – the heap can be split into disjoint parts satisfying φ and ψ
  - Inductive predicates describing data structures (lists, trees, ...)

- A majority of automated decision procedures handles the symbolic heap fragment that forbids a boolean structure of spatial assertions:

$$\underbrace{(x = y + 1 \land w \neq \mathrm{nil})}_{\text{pure part}} \quad * \quad \underbrace{x \mapsto y * \mathrm{sls}(y, z) * z \mapsto w}_{\text{spatial part}}$$

- **Pure part**: a boolean structure is sometimes allowed
- **Spatial part**: no boolean structure allowed

Decision Procedures for SL

- Many approaches for complex user-defined inductive predicates:
  - Tree automata (Slide, Spen), heap automata (Harrsh)
  - Cyclic proofs (Cyclist, S2S), induction (Songbird)

Decision Procedures for SL

- Many approaches for complex user-defined inductive predicates:
  - Tree automata (SLIDE, SPEN), heap automata (HARRSH)
  - Cyclic proofs (CYCLIST, S2S), induction (SONGBIRD)
- SMT-translation-based approaches:
  - Limited boolean structures + built-in inductive predicates
  - GRASSHOPPER – intermediate theory of reachability
  - SLOTH – direct translation based on small-model property

Decision Procedures for SL

- Many approaches for complex user-defined inductive predicates:
  - Tree automata (SLIDE, SPEN), heap automata (HARRSH)
  - Cyclic proofs (CYCLIST, S2S), induction (SONGBIRD)

- SMT-translation-based approaches:
  - Limited boolean structures + built-in inductive predicates
  - GRASSHOPPER – intermediate theory of reachability
  - SLOTH – direct translation based on small-model property

- Decision procedure in cvc5
  - Fine-grained combination with other SMT theories
  - Arbitrary nesting of boolean and spatial connectives
  - But no inductive predicates

Decision Procedures for SL

- Many approaches for complex user-defined inductive predicates:
  - Tree automata (SLIDE, SPEN), heap automata (HARRSH)
  - Cyclic proofs (CYCLIST, S2S), induction (SONGBIRD)
- SMT-translation-based approaches:
  - Limited boolean structures + built-in inductive predicates
  - GRASSHOPPER – intermediate theory of reachability
  - SLOTH – direct translation based on small-model property
- Decision procedure in cvc5
  - Fine-grained combination with other SMT theories
  - Arbitrary nesting of boolean and spatial connectives
  - But no inductive predicates

$*$ *Decidability proofs for more complex fragments*

Boolean Separation Logic (BSL)

Syntax:

$$\varphi_{atom} ::= x = y \mid x \neq y \mid x \mapsto (y_1, \ldots, y_n) \mid \mathsf{sls}(x, y) \mid \mathsf{dls}(x, y, x', y') \mid \mathsf{nls}(x, y, z)$$

$$\varphi ::= \varphi_{atom} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \wedge \neg\varphi \mid \varphi * \varphi$$

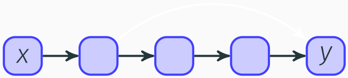
Boolean Separation Logic (BSL)

Syntax:

$$\varphi_{atom} ::= x = y \mid x \neq y \mid x \mapsto (y_1, \ldots, y_n) \mid \mathsf{sls}(x, y) \mid \mathsf{dls}(x, y, x', y') \mid \mathsf{nls}(x, y, z)$$

$$\varphi ::= \varphi_{atom} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \wedge \neg\varphi \mid \varphi * \varphi$$

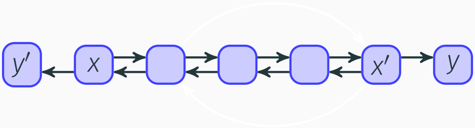
Singly-linked list:



Syntax:

$$\varphi_{atom} ::= x = y \mid x \neq y \mid x \mapsto (y_1, \ldots, y_n) \mid \mathsf{sls}(x, y) \mid \mathsf{dls}(x, y, x', y') \mid \mathsf{nls}(x, y, z)$$
$$\varphi ::= \varphi_{atom} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \wedge \neg\varphi \mid \varphi * \varphi$$

Doubly-linked list:



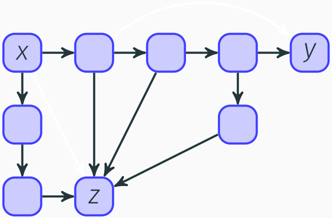
Boolean Separation Logic (BSL)

Syntax:

$$\varphi_{atom} ::= x = y \mid x \neq y \mid x \mapsto (y_1, \ldots, y_n) \mid \mathsf{sls}(x, y) \mid \mathsf{dls}(x, y, x', y') \mid \mathsf{nls}(x, y, z)$$

$$\varphi ::= \varphi_{atom} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \wedge \neg\varphi \mid \varphi * \varphi$$

Nested singly-linked list:



Boolean Separation Logic (BSL)

Syntax:

$$\varphi_{atom} ::= x = y \mid x \neq y \mid x \mapsto (y_1, \ldots, y_n) \mid \mathsf{sls}(x, y) \mid \mathsf{dls}(x, y, x', y') \mid \mathsf{nls}(x, y, z)$$

$$\varphi ::= \varphi_{atom} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \wedge \neg\varphi \mid \varphi * \varphi$$

Guarded negation $\varphi \wedge \neg\psi$:

- Ensures that models are ”garbage-free”
- Makes logic closed w.r.t. entailment checking
- Can be used for model enumeration

- *Note:* true *cannot be expressed in BSL because we use the so-called precise semantics in which* $x = x \vee x \neq x$ *is equivalent to* emp

Motivation I

- Disjunctions naturally appear in program verification:

$$\{\mathrm{emp}\}\ \texttt{x = malloc()}\ \{x \mapsto f \vee (x = \mathrm{nil} \wedge \mathrm{emp})\}$$

- Boolean connectives can be introduced by translation from more complex flavours of SL (e.g., quantitative SL or by unfolding of inductive definitions)

Motivation II: Membership in Data Structures

Property 1: List containing elements $\ell_1, \ldots, \ell_n$ (in arbitrary order):

- Symbolic heaps: not straightforward (needs enumeration of permutations or existential variables)
- BSL: $\mathsf{sls}(x, y) \;\wedge\; \bigwedge_{\ell_i} \Big(\mathsf{sls}(x, \ell_i) * \mathsf{sls}(\ell_i, y)\Big)$.

Motivation II: Membership in Data Structures

Property 1: List containing elements $\ell_1, \ldots, \ell_n$ (in arbitrary order):

- Symbolic heaps: not straightforward (needs enumeration of permutations or existential variables)
- BSL: $\mathsf{sls}(x, y) \;\wedge\; \bigwedge_{\ell_i} \Big(\mathsf{sls}(x, \ell_i) * \mathsf{sls}(\ell_i, y)\Big)$.

Property 2: List not containing element ℓ:

- Symbolic heaps: needs a dedicated inductive predicate
- BSL: $\mathsf{sls}(x, y) \wedge \neg\Big(\mathsf{sls}(x, \ell) * \mathsf{sls}(\ell, y)\Big)$

Both properties can be nested inside more complex formulae which would lead to an alternation of boolean and spatial connectives.

Small-Model Property

Small-Model Property

Theorem

A satisfiable formula φ has a model of linear size (w.r.t. number of vars.)

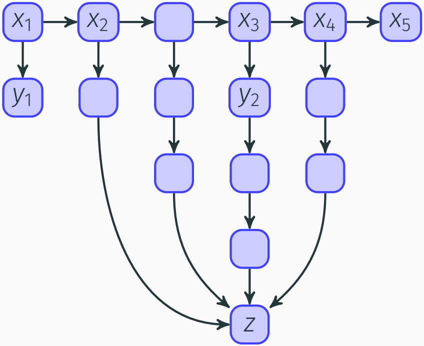
Small-Model Property

Theorem

A satisfiable formula φ has a model of linear size (w.r.t. number of vars.)

Proof idea:

- Take an arbitrary model of φ



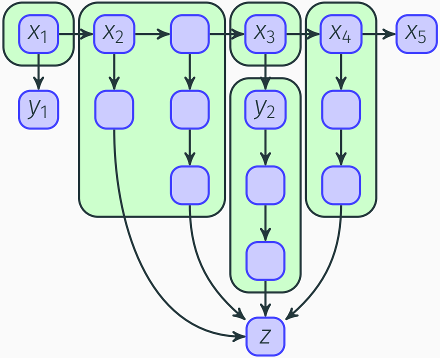
Small-Model Property

Theorem

A satisfiable formula φ has a model of linear size (w.r.t. number of vars.)

Proof idea:

- Take an arbitrary model of φ
- Split it into atomic parts (cannot be split further to non-empty models)



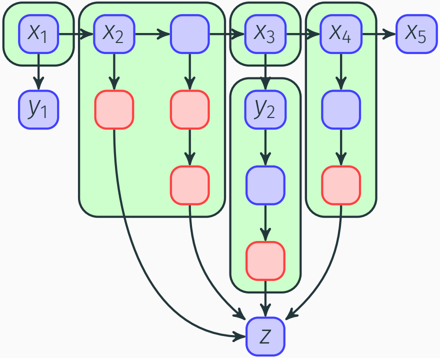
Small-Model Property

Theorem

A satisfiable formula φ has a model of linear size (w.r.t. number of vars.)

Proof idea:

- Take an arbitrary model of φ
- Split it into atomic parts (cannot be split further to non-empty models)
- Reduce those parts



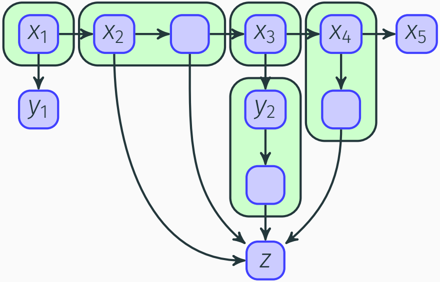
Small-Model Property

Theorem

A satisfiable formula φ has a model of linear size (w.r.t. number of vars.)

Proof idea:

- Take an arbitrary model of φ
- Split it into atomic parts (cannot be split further to non-empty models)
- Reduce those parts
- Composition of reduced parts is a model of φ



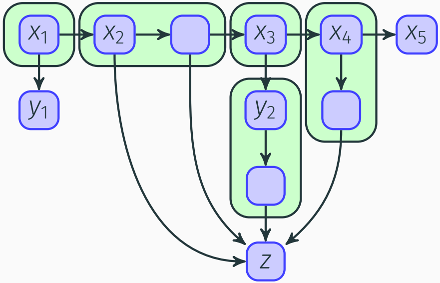
Small-Model Property

Theorem

A satisfiable formula φ has a model of linear size (w.r.t. number of vars.)

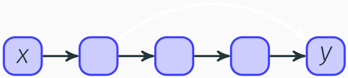
Proof idea:

- Take an arbitrary model of φ
- Split it into atomic parts (cannot be split further to non-empty models)
- Reduce those parts
- Composition of reduced parts is a model of φ
- Size of reduced model is lesser than $2n$



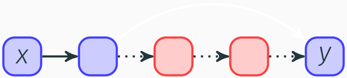
Reduction of Atomic Sub-heaps

- **Singly-linked list** $\mathsf{sls}(x, y)$ – reduction to size at most 2:



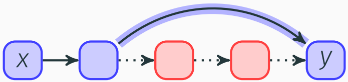
Reduction of Atomic Sub-heaps

- **Singly-linked list** $\mathrm{sls}(x, y)$ – reduction to size at most 2:



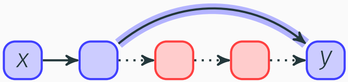
Reduction of Atomic Sub-heaps

- **Singly-linked list** $\mathrm{sls}(x, y)$ – reduction to size at most 2:

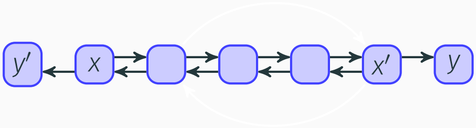


Reduction of Atomic Sub-heaps

- **Singly-linked list** $\mathrm{sls}(x, y)$ – reduction to size at most 2:

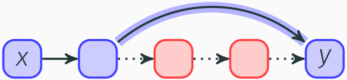


- **Doubly-linked list** $\mathrm{dls}(x, y, x', y')$ – reduction to size at most 3:

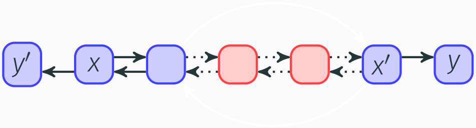


Reduction of Atomic Sub-heaps

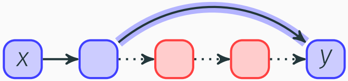
- **Singly-linked list** $\mathrm{sls}(x, y)$ – reduction to size at most 2:



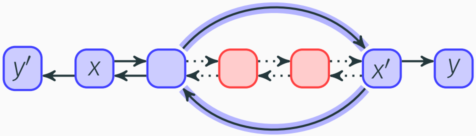
- **Doubly-linked list** $\mathrm{dls}(x, y, x', y')$ – reduction to size at most 3:



- **Singly-linked list** $\mathrm{sls}(x, y)$ – reduction to size at most 2:

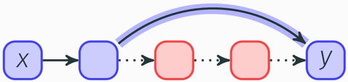


- **Doubly-linked list** $\mathrm{dls}(x, y, x', y')$ – reduction to size at most 3:

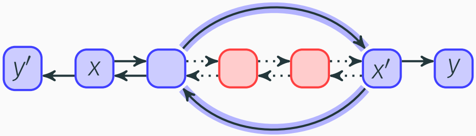


Reduction of Atomic Sub-heaps

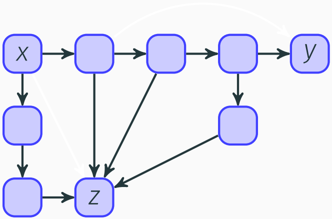
- **Singly-linked list** $\mathrm{sls}(x, y)$ – reduction to size at most 2:



- **Doubly-linked list** $\mathrm{dls}(x, y, x', y')$ – reduction to size at most 3:

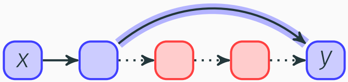


- **Nested singly-linked list** $\mathrm{nls}(x, y, z)$ – reduction to size at most 2:

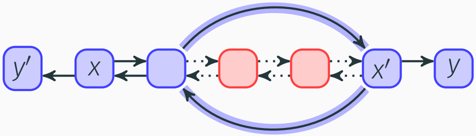


Reduction of Atomic Sub-heaps

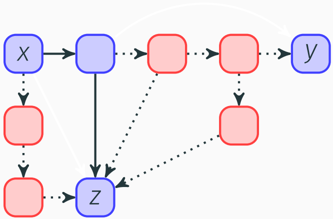
- **Singly-linked list** $\mathrm{sls}(x, y)$ – reduction to size at most 2:



- **Doubly-linked list** $\mathrm{dls}(x, y, x', y')$ – reduction to size at most 3:

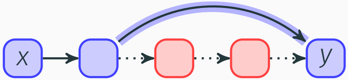


- **Nested singly-linked list** $\mathrm{nls}(x, y, z)$ – reduction to size at most 2:

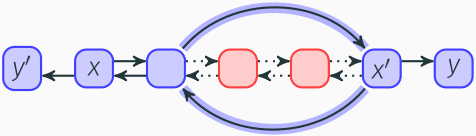


Reduction of Atomic Sub-heaps

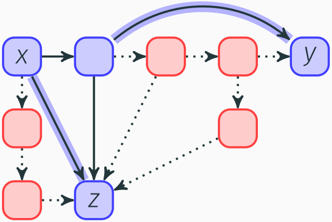
- **Singly-linked list** sls(x, y) – reduction to size at most 2:



- **Doubly-linked list** dls(x, y, x', y') – reduction to size at most 3:



- **Nested singly-linked list** nls(x, y, z) – reduction to size at most 2:



Translation-Based Decision Procedure

Translation-Based Decision Procedure

- Method inspired by existing approaches (GRASSHOPPER and SLOTH)
- Key improvements:
  - More expressive fragment (boolean connectives under $*$)
    - $\rightsquigarrow$ Needs generalisation of unique footprint property used for efficient translation of separating conjunctions
  - Bounds on sizes of predicate instances and predicate encoding which can leverage them
    - We already have location bound,
    - but we want to compute smaller bounds on individual predicate instances
    - Improved scalability: often independent of location bound

Predicate Bounds

Goal

Bound sizes of predicate instances to decrease size of their encoding.

- Based on SL-graphs which capture must-relations in all models of φ

Predicate Bounds

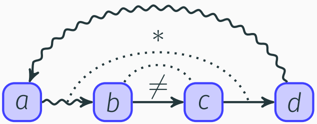
Goal

Bound sizes of predicate instances to decrease size of their encoding.

- Based on SL-graphs which capture must-relations in all models of φ

Example formula

$$\varphi \triangleq \mathrm{sls}(a, b) * b \mapsto c * c \mapsto d * \mathrm{sls}(d, a) \wedge \neg(\mathrm{sls}(a, c) * \mathrm{sls}(c, a))$$



Fragment of SL-graph of φ
(some $*$-edges are missing)

Predicate Bounds

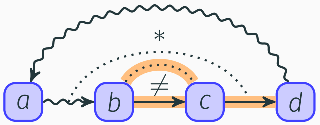
Goal

Bound sizes of predicate instances to decrease size of their encoding.

- Based on SL-graphs which capture must-relations in all models of φ

Example formula

$$\varphi \triangleq \mathrm{sls}(a, b) * b \mapsto c * c \mapsto d * \mathrm{sls}(d, a) \land \neg(\mathrm{sls}(a, c) * \mathrm{sls}(c, a))$$



Fragment of SL-graph of φ
(some $*$-edges are missing)

Predicate Bounds

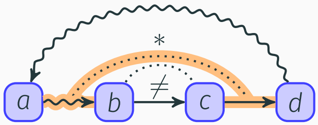
Goal

Bound sizes of predicate instances to decrease size of their encoding.

- Based on SL-graphs which capture must-relations in all models of φ

Example formula

$$\varphi \triangleq \mathsf{sls}(a, b) * b \mapsto c * c \mapsto d * \mathsf{sls}(d, a) \land \neg(\mathsf{sls}(a, c) * \mathsf{sls}(c, a))$$



Fragment of SL-graph of φ
(some $*$-edges are missing)

Predicate Bounds

Goal

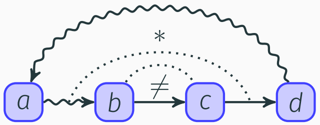
Bound sizes of predicate instances to decrease size of their encoding.

- Based on SL-graphs which capture must-relations in all models of φ

Example formula

$$\varphi \triangleq \mathsf{sls}(a, b) * b \mapsto c * c \mapsto d * \mathsf{sls}(d, a) \wedge \neg(\mathsf{sls}(a, c) * \mathsf{sls}(c, a))$$

- Improved location bound: 6



Fragment of SL-graph of φ
(some $*$-edges are missing)

Predicate Bounds

Goal

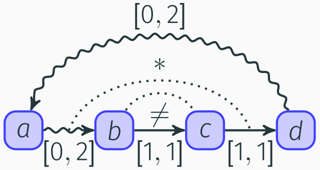
Bound sizes of predicate instances to decrease size of their encoding.

- Based on SL-graphs which capture must-relations in all models of φ

Example formula

$$\varphi \triangleq \mathsf{sls}(a, b) * b \mapsto c * c \mapsto d * \mathsf{sls}(d, a) \land \neg(\mathsf{sls}(a, c) * \mathsf{sls}(c, a))$$

- Improved location bound: 6
- First phase: bounds on paths which appear in SL-graph



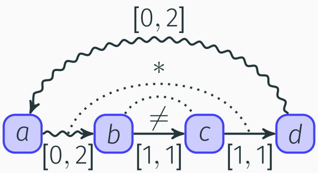
Fragment of SL-graph of φ (some $*$-edges are missing)

Predicate Bounds: Example

Example formula

$$\varphi \triangleq \mathrm{sls}(a, b) * b \mapsto c * c \mapsto d * \mathrm{sls}(d, a) \wedge \neg(\mathrm{sls}(a, c) * \mathrm{sls}(c, a))$$

Computation for $\mathrm{sls}(a, c)$ is based on two projections of SL-graph:



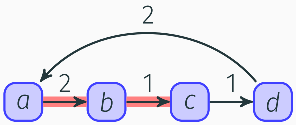
Example formula

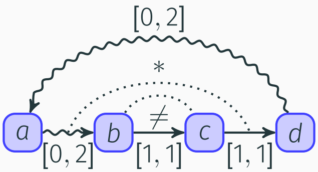
$$\varphi \triangleq \mathrm{sls}(a,b) * b \mapsto c * c \mapsto d * \mathrm{sls}(d,a) \wedge \neg(\mathrm{sls}(a,c) * \mathrm{sls}(c,a))$$

Computation for $\mathrm{sls}(a,c)$ is based on two projections of SL-graph:

Upper bound:

- All directed edges
- Bound is given as length of the shortest path from a to c



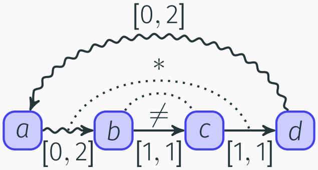


Predicate Bounds: Example

Example formula

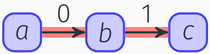
$$\varphi \triangleq \mathsf{sls}(a, b) * b \mapsto c * c \mapsto d * \mathsf{sls}(d, a) \wedge \neg(\mathsf{sls}(a, c) * \mathsf{sls}(c, a))$$

Computation for $\mathsf{sls}(a, c)$ is based on two projections of SL-graph:



Lower bound:

- Edges which surely do not contain end of path (c)
- Bound is given as length of the longest path starting from a not containing c



Predicate Bounds: Example

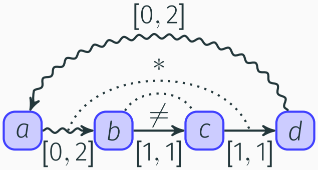
Example formula

$$\varphi \triangleq \mathsf{sls}(a, b) * b \mapsto c * c \mapsto d * \mathsf{sls}(d, a) \land \neg(\mathsf{sls}(a, c) * \mathsf{sls}(c, a))$$

Computation for $\mathsf{sls}(a, c)$ is based on two projections of SL-graph:

Result:

- Bound $[1, 3]$ instead of default $[0, 6]$
- Bound is stable when LHS and RHS grows
- Method is easily generalised for DLLs and NLLs



Experimental Evaluation

Implementation

- New solver Astral[1]
- Support for:
  - Subset of the standard format based on SMT-LIB
  - Multiple strategies of encoding (e.g., direct x bitvector set encoding)
  - Multiple SMT backends – Z3, cvc5, Bitwuzla
  - Model generation

[1] `https://github.com/TDacik/Astral`

Comparison on SL-COMP Benchmarks: SLLs

Symbolic heaps from the SL-COMP competition:

- Significantly simpler than full BSL, main goal is to compare with other translation-based decision procedures (GRASSHOPPER, SLOTH)

Comparison on SL-COMP Benchmarks: SLLs

Symbolic heaps from the SL-COMP competition:

- Significantly simpler than full BSL, main goal is to compare with other translation-based decision procedures (GRASSHOPPER, SLOTH)

	Verification conditions (86)					
Solver	OK	RO	WIN	<0.1 s	≤ 1 s	Total time [s]
ASTRAL	86	0	-	84	86	4.62
GRASSHOPPER	86	0	70	52	86	8.65
S2S	86	0	5	86	86	2.08
SLOTH	64	3	86	0	28	235.28

OK – Correctly solved, RO – Out of time/memory, WIN – ASTRAL is faster

Comparison on SL-COMP Benchmarks: SLLs

Symbolic heaps from the SL-COMP competition:

- Significantly simpler than full BSL, main goal is to compare with other translation-based decision procedures (GRASSHOPPER, SLOTH)

	bolognesa+clones (210)					
Solver	OK	RO	WIN	<0.1 s	≤ 1 s	Total time [s]
ASTRAL	210	0	-	68	169	202.91
GRASSHOPPER	203	7	148	60	87	1229.35
S2S	210	0	3	203	210	8.18
SLOTH	70	140	210	0	50	149.42

OK – Correctly solved, RO – Out of time/memory, WIN – ASTRAL is faster

Comparison on SL-COMP Benchmarks: NLLs

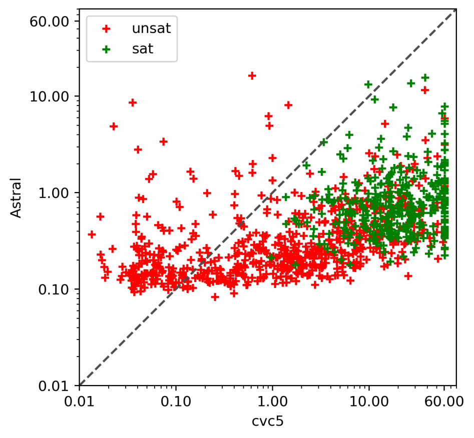
- NLL formulae selected from SL-COMP category of linear IDs
- S2S, SONGBIRD and HARRSH (three best solvers in the category)

				Nested singly-linked lists (19)		
Solver	OK	RO	WIN	<0.1 *s*	≤1 *s*	Total time [s]
ASTRAL	19	0	-	3	9	86.93
HARRSH	14	5	18	0	0	183.01
S2S	19	0	0	19	19	0.43
SONGBIRD	11	5	8	4	11	1.38

OK – Correctly solved, RO – Out of time/memory, WIN – ASTRAL is faster

Comparison with cvc5

- Randomly generated formulae of depth 8 with 8 variables
- BSL formulae without inductive predicates
- Astral run with cvc5 backend to provide better comparison of translation method



Summary

- New translation-based decision procedure for a rich fragment of SL
- Outperforms existing translation-based decision procedures and extends fragment which can be translated

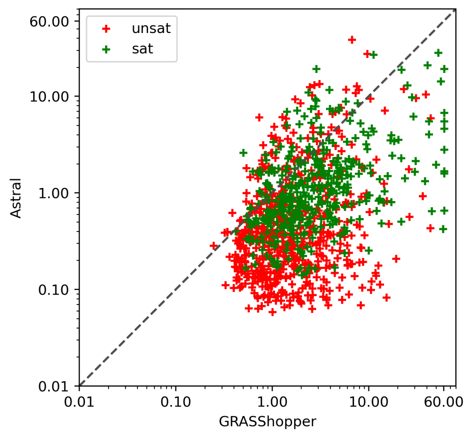
Future work:

- User-defined inductive predicates
- Fine-grained combination with SMT (arbitrary location sorts)
- Interactive/lazy translation

Appendices

Comparison with GRASSHOPPER

- Entailments of positive boolean combinations of lists
- Formulae of depth 6 with 6 variables



Intuition behind Reduction

- Such reduction is possible because considered fragment cannot speak about arbitrary sizes of predicates
- For example, for DLLs, we may express the following:

Intuition behind Reduction

- Such reduction is possible because considered fragment cannot speak about arbitrary sizes of predicates
- For example, for DLLs, we may express the following:

$$\boxed{x\ ,y} \quad \boxed{x', y'} \quad \models \mathrm{dls}(x, y, x', y')$$

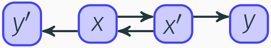
Intuition behind Reduction

- Such reduction is possible because considered fragment cannot speak about arbitrary sizes of predicates
- For example, for DLLs, we may express the following:

$$\models \underbrace{\mathrm{dls}(x, y, x', y') * x \neq y}_{\text{DLL of size greater than 0}}$$

Intuition behind Reduction

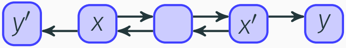
- Such reduction is possible because considered fragment cannot speak about arbitrary sizes of predicates
- For example, for DLLs, we may express the following:



$$\models \quad \underbrace{\mathrm{dls}(x, y, x', y') * x \neq y * x \neq x'}_{\text{DLL of size greater than 1}}$$

Intuition behind Reduction

- Such reduction is possible because considered fragment cannot speak about arbitrary sizes of predicates
- For example, for DLLs, we may express the following:

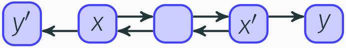


$$\models$$

$$\underbrace{\mathrm{dls}(x, y, x', y') * x \neq y * x \neq x' \wedge \neg\Big(x \mapsto (x', y') * x' \mapsto (y, x)\Big)}_{\text{DLL of size greater than 2}}$$

Intuition behind Reduction

- Such reduction is possible because considered fragment cannot speak about arbitrary sizes of predicates
- For example, for DLLs, we may express the following:



$$\models$$

$$\underbrace{\mathrm{dls}(x, y, x', y') * x \neq y * x \neq x' \wedge \neg\Big(x \mapsto (x', y') * x' \mapsto (y, x)\Big)}_{\text{DLL of size greater than 2}}$$

- DLLs of larger sizes cannot be expressed using BSL formulae without using additional variables

Unique Footprints

- Semantics of $*$ implicitly involves existential quantification:

There exists a split of heap such that …

Unique Footprints

- Semantics of $*$ implicitly involves existential quantification:

  There exists a split of heap such that …

- Unique footprints: quantification can be avoided when there is the unique relevant split:

$$\varphi \wedge \neg\Big(\underbrace{x \mapsto y}_{\substack{\text{Can be satisfied}\\ \text{only on } \{x\}}} \; * \; \underbrace{\mathsf{sls}(x, y)}_{\substack{\text{Can be satisfied only}\\ \text{on path from } x \text{ to } y\\ \text{(which is unique)}}} \Big)$$

Footprints: Generalisation

- Footprints are not unique in BSL because of disjunctions:

$$\varphi * \underbrace{\left(\mathrm{emp} \vee x \mapsto y\right)}_{\substack{\text{Can be satisfied} \\ \text{on } \emptyset \text{ or } \{x\}}}$$

- However, we can still use the principle of footprints:
  - For each operand of $*$, we compute sets of terms representing over-approximation of its footprints
  - Replace the “exists split” quantification underlying $*$
    - by its instantiation to footprint terms
    - provided they are small enough.

Bitvector Encoding

Direct Encoding:

- Datatypes (locations)
- Sets (heap domains)
- Arrays (heap mappings)

Bitvector Encoding

Direct Encoding:

- Datatypes (locations)
- Sets (heap domains) – non-standard theory
- Arrays (heap mappings)

Bitvector Encoding

Direct Encoding:

- Datatypes (locations) – standardised, not so commonly supported
- Sets (heap domains) – non-standard theory
- Arrays (heap mappings)

Bitvector Encoding

Direct Encoding:

- Datatypes (locations) – standardised, not so commonly supported
- Sets (heap domains) – non-standard theory
- Arrays (heap mappings)

Bitvector encoding

- Both locations and location sets are encoded as bitvectors
- Additional axioms: locations must fit into bitvector sets
- Better performance with quantifiers over (encoded) sets

Complexity

Theorem

Satisfiability problem for BSL is PSPACE-complete.

Proof idea:

- Problem is known to be PSPACE-complete for unbounded negations by reduction from QBF
- BSL can express the true atom in QBF encoding

$$\mathrm{true}[X] \triangleq \mathop{\ast}_{x \in X} x \mapsto \mathrm{nil} \vee \mathrm{emp}$$

When either guarded negation or disjunction is dropped, the problem is NP-complete