Karlsruhe Institute of Technology

A Theory of Arrays with `set` and `copy` Operations

Stephan Falke, **Carsten Sinz, and Florian Merz**

INSTITUTE FOR THEORETICAL COMPUTER SCIENCE (ITI)

```
struct list_node {
    int data;
    struct list_node *tail;
};
typedef struct list_node list;

list *reverse(list *l) {
  list *r = l, *p = NULL;
  while (r != NULL) {
    list *q = r;
    r = r->tail;
    q->tail = p;
    p = q;
  }
  return p;
}
```

Motivation

- SMT-solvers are routinely used in program analysis:
  - Deductive program verification
  - Symbolic execution
  - Bounded model checking
  - ...

Motivation

- SMT-solvers are routinely used in program analysis:
  - Deductive program verification
  - Symbolic execution
  - Bounded model checking
  - ...
- Prominent theory: $\mathcal{T}_{\mathcal{A}}$ (theory of arrays)
  - Model arrays/structures/objects in the program
  - Model main memory

Motivation

What to do with standard library functions such as `memset` and `memcpy`/`memmove`?

```
void *memset(void *s, int c, size_t n);

void *memcpy(void *dest, const void *src, size_t n);

void *memmove(void *dest, const void *src, size_t n);
```

Motivation

What to do with standard library functions such as `memset` and `memcpy`/`memmove`?

might not be constant-sized!

```
void *memset(void *s, int c, size_t n);
```

might not be constant-sized!

```
void *memcpy(void *dest, const void *src, size_t n);
```

might not be constant-sized!

```
void *memmove(void *dest, const void *src, size_t n);
```

Motivation

What to do with standard library functions such as `memset` and `memcpy`/`memmove`?

might not be constant-sized!

```
void *memset(void *s, int c, size_t n);
```

might not be constant-sized!

```
void *memcpy(void *dest, const void *src, size_t n);
```

might not be constant-sized!

```
void *memmove(void *dest, const void *src, size_t n);
```

$\Longrightarrow$ Extend $\mathcal{T}_{\mathcal{A}}$ to allow for dynamically-sized read/writes!

Contributions

1. $\mathcal{T}_{\mathcal{ASC}}$: an extension of $\mathcal{T}_{\mathcal{A}}$ with `set` and `copy` operations
   - `set`: sets ranges in an array at once
   - `copy`: copies ranges between arrays

Contributions

1. $\mathcal{T}_{\mathcal{ASC}}$: an extension of $\mathcal{T}_{\mathcal{A}}$ with `set` and `copy` operations
   - `set`: sets ranges in an array at once
   - `copy`: copies ranges between arrays
2. Satisfiability checking for $\mathcal{T}_{\mathcal{ASC}}$ by reducing to $\mathcal{T}_{\mathcal{A}}$ + index arithmetic

Applications of $\mathcal{T}_{\mathcal{ASC}}$

- Precisely model `memset` and `memcpy`/`memmove`

Applications of $\mathcal{T}_{\mathcal{ASC}}$

- Precisely model `memset` and `memcpy`/`memmove`
- Zero initialization of global variables

Applications of $\mathcal{T}_{\mathcal{ASC}}$

- Precisely model `memset` and `memcpy`/`memmove`
- Zero initialization of global variables
- Zero initialization of fresh memory pages

Applications of $\mathcal{T}_{\mathcal{ASC}}$

- Precisely model `memset` and `memcpy`/`memmove`
- Zero initialization of global variables
- Zero initialization of fresh memory pages
- “Havoc” memory regions (volatile variables)

Applications of $\mathcal{T}_{\mathcal{ASC}}$

- Precisely model `memset` and `memcpy`/`memmove`
- Zero initialization of global variables
- Zero initialization of fresh memory pages
- “Havoc” memory regions (volatile variables)
- Model memory mapped I/O

Applications of $\mathcal{T}_{\mathcal{ASC}}$

- Precisely model `memset` and `memcpy`/`memmove`
- Zero initialization of global variables
- Zero initialization of fresh memory pages
- "Havoc" memory regions (volatile variables)
- Model memory mapped I/O
- Attaching metadata to memory regions (allocated, de-allocated, . . .)

$\mathcal{T}_{\mathcal{A}}$: The Theory of Arrays

Sorts	E : elements I : indices A : arrays
Functions	$\texttt{read} : \mathsf{A} \times \mathsf{I} \to \mathsf{E}$ $\texttt{write} : \mathsf{A} \times \mathsf{I} \times \mathsf{E} \to \mathsf{A}$

$$p = r \implies \texttt{read}(\texttt{write}(a, p, v), r) = v$$

$$\neg(p = r) \implies \texttt{read}(\texttt{write}(a, p, v), r) = \texttt{read}(a, r)$$

$\mathcal{T}_{\mathcal{A}}$: The Theory of Arrays

Sorts	E : elements I : indices A : arrays
Functions	$\texttt{read} : \mathsf{A} \times \mathsf{I} \to \mathsf{E}$ $\texttt{write} : \mathsf{A} \times \mathsf{I} \times \mathsf{E} \to \mathsf{A}$

a write modifies the position written to ...

$$p = r \implies \texttt{read}(\texttt{write}(a, p, v), r) = v$$

$$\neg(p = r) \implies \texttt{read}(\texttt{write}(a, p, v), r) = \texttt{read}(a, r)$$

$\mathcal{T}_{\mathcal{A}}$: The Theory of Arrays

Sorts	E : elements I : indices A : arrays
Functions	$\texttt{read} : \mathsf{A} \times \mathsf{I} \to \mathsf{E}$ $\texttt{write} : \mathsf{A} \times \mathsf{I} \times \mathsf{E} \to \mathsf{A}$

a write modifies the position written to ...

$$p = r \implies \texttt{read}(\texttt{write}(a, p, v), r) = v$$

$$\neg(p = r) \implies \texttt{read}(\texttt{write}(a, p, v), r) = \texttt{read}(a, r)$$

... and nothing else.

$\mathcal{T}_\mathcal{A}$: The Theory of Arrays

Sorts	E : elements I : indices A : arrays
Functions	$\texttt{read} : A \times I \to E$ $\texttt{write} : A \times I \times E \to A$

a write modifies the position written to ...

$$p = r \implies \texttt{read}(\texttt{write}(a, p, v), r) = v$$

$$\neg(p = r) \implies \texttt{read}(\texttt{write}(a, p, v), r) = \texttt{read}(a, r)$$

... and nothing else.

replace read (write (a, p, v), r) by ITE (p = r, v, read (a, r))
treat remaining arrays as uninterpreted functions

The Theory of Arrays with `set` and `copy`

Sorts	$S = I$
Functions	$\texttt{set} : A \times I \times E \times S \to A$ $\texttt{copy} : A \times I \times A \times I \times S \to A$

$$p \leq r < p + s \implies \texttt{read}(\texttt{set}(a, p, v, s), r) = v$$

$$\neg(p \leq r < p + s) \implies \texttt{read}(\texttt{set}(a, p, v, s), r) = \texttt{read}(a, r)$$

The Theory of Arrays with `set` and `copy`

Sorts	$S = I$
Functions	$\texttt{set} : A \times I \times E \times S \to A$ $\texttt{copy} : A \times I \times A \times I \times S \to A$

$S = I = \mathbb{N}$

index type needs arithmetic

$$p \leq r < p + s \implies \texttt{read}(\texttt{set}(a, p, v, s), r) = v$$

$$\neg(p \leq r < p + s) \implies \texttt{read}(\texttt{set}(a, p, v, s), r) = \texttt{read}(a, r)$$

The Theory of Arrays with `set` and `copy`

Sorts	$S = I$
Functions	$\texttt{set} : A \times I \times E \times S \to A$ $\texttt{copy} : A \times I \times A \times I \times S \to A$

$S = I = \mathbb{N}$

index type needs arithmetic

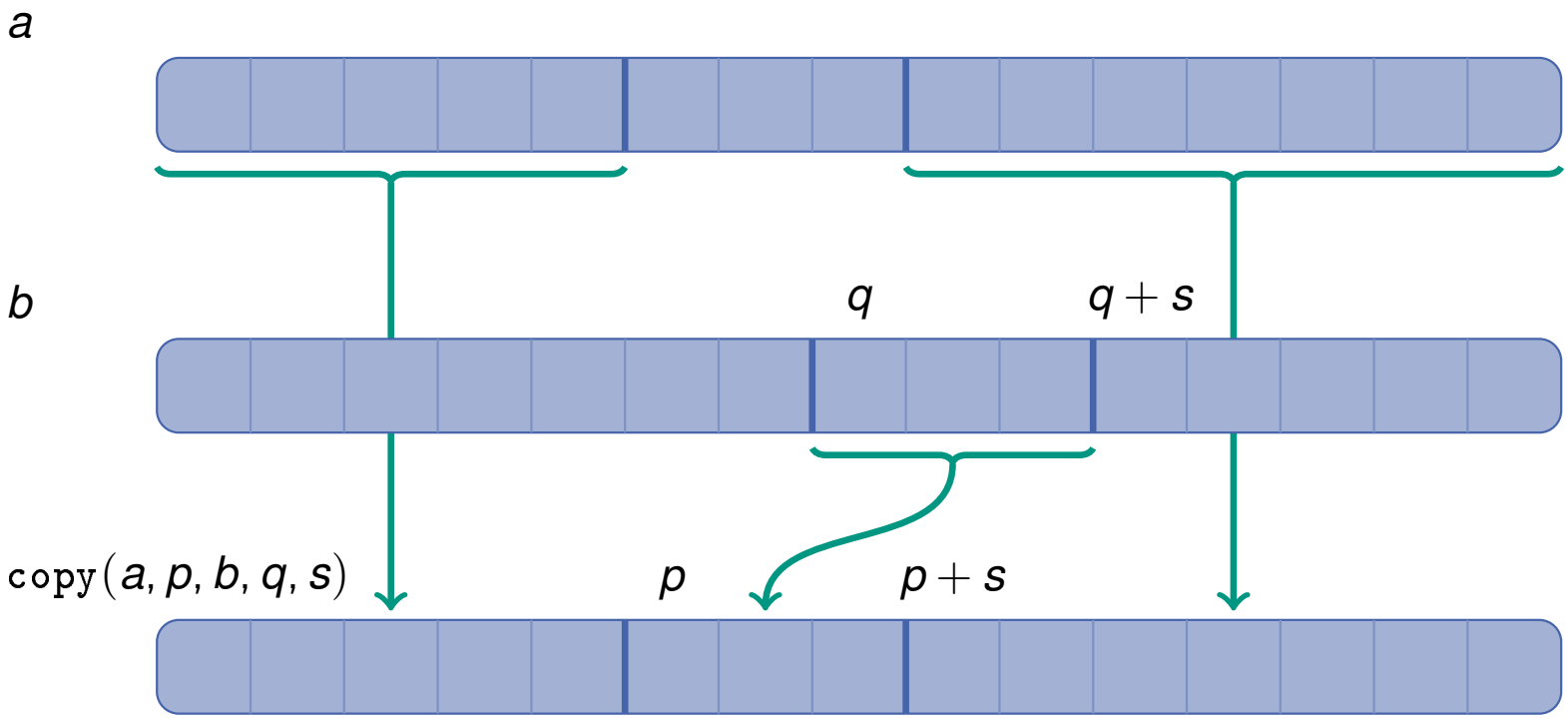
$$p \leq r < p + s \implies \texttt{read}(\texttt{set}(a, p, v, s), r) = v$$

$$\neg(p \leq r < p + s) \implies \texttt{read}(\texttt{set}(a, p, v, s), r) = \texttt{read}(a, r)$$

$$p \leq r < p + s \implies \texttt{read}(\texttt{copy}(a, p, b, q, s), r) = \texttt{read}(b, q + r - p)$$

$$\neg(p \leq r < p + s) \implies \texttt{read}(\texttt{copy}(a, p, b, q, s), r) = \texttt{read}(a, r)$$

The `copy` Operation: Illustration



The Theory of Arrays with $\mathtt{set}_\infty$ and $\mathtt{copy}_\infty$

Sorts	S = I
Functions	$\mathtt{set}_\infty : A \times I \times E \to A$ $\mathtt{copy}_\infty : A \times I \times A \times I \to A$ $\mathtt{set} : A \times I \times E \times S \to A$ $\mathtt{copy} : A \times I \times A \times I \times S \to A$

$$p \leq r \implies \mathtt{read}(\mathtt{set}_\infty(a, p, v), r) = v$$

$$\neg(p \leq r) \implies \mathtt{read}(\mathtt{set}_\infty(a, p, v), r) = \mathtt{read}(a, r)$$

$$p \leq r \implies \mathtt{read}(\mathtt{copy}_\infty(a, p, b, q), r) = \mathtt{read}(b, q + r - p)$$

$$\neg(p \leq r) \implies \mathtt{read}(\mathtt{copy}_\infty(a, p, b, q), r) = \mathtt{read}(a, r)$$

$\mathcal{T}_{\mathcal{ASC}}$ with Bitvectors (`set`)

$$\mathsf{bvule}(p, r) \wedge \mathsf{bvult}(r, \mathsf{bvadd}(p, s)) \implies \mathtt{read}(\mathtt{set}(a, p, v, s), r) = v$$

$$\neg(\mathsf{bvule}(p, r) \wedge \mathsf{bvult}(r, \mathsf{bvadd}(p, s))) \implies \mathtt{read}(\mathtt{set}(a, p, v, s), r) = \mathtt{read}(a, r)$$

$\mathcal{T}_{\mathcal{ASC}}$ with Bitvectors (`set`)

might overflow

$$\mathsf{bvule}(p, r) \wedge \mathsf{bvult}(r, \mathsf{bvadd}(p, s)) \implies \mathtt{read}(\mathtt{set}(a, p, v, s), r) = v$$

$$\neg(\mathsf{bvule}(p, r) \wedge \mathsf{bvult}(r, \mathsf{bvadd}(p, s))) \implies \mathtt{read}(\mathtt{set}(a, p, v, s), r) = \mathtt{read}(a, r)$$

might overflow

$\mathcal{T}_{\mathcal{ASC}}$ with Bitvectors (`set`)

might overflow

$$\underbrace{\mathsf{bvule}(p, r) \wedge \mathsf{bvult}(r, \mathsf{bvadd}(p, s))}_{\text{always false on overflow}} \implies \mathtt{read}(\mathtt{set}(a, p, v, s), r) = v$$

$$\neg(\mathsf{bvule}(p, r) \wedge \mathsf{bvult}(r, \underbrace{\mathsf{bvadd}(p, s)}_{\text{might overflow}})) \implies \mathtt{read}(\mathtt{set}(a, p, v, s), r) = \mathtt{read}(a, r)$$

$\mathcal{T}_{\mathcal{ASC}}$ with Bitvectors (`set`)

$$\underbrace{\mathsf{bvule}(p, r) \wedge \mathsf{bvult}(r, \overbrace{\mathsf{bvadd}(p, s)}^{\text{might overflow}})}_{\text{always false on overflow}} \implies \mathtt{read}(\mathtt{set}(a, p, v, s), r) = v$$

$$\overbrace{\neg(\mathsf{bvule}(p, r) \wedge \mathsf{bvult}(r, \underbrace{\mathsf{bvadd}(p, s)}_{\text{might overflow}}))}^{\text{always true on overflow}} \implies \mathtt{read}(\mathtt{set}(a, p, v, s), r) = \mathtt{read}(a, r)$$

$\mathcal{T}_{\mathcal{ASC}}$ with Bitvectors (`set`)

$$\underbrace{\mathsf{bvule}(p, r) \wedge \mathsf{bvult}(r, \overbrace{\mathsf{bvadd}(p, s)}^{\text{might overflow}})}_{\text{always false on overflow}} \implies \mathtt{read}(\mathtt{set}(a, p, v, s), r) = v$$

$$\overbrace{\neg(\mathsf{bvule}(p, r) \wedge \mathsf{bvult}(r, \underbrace{\mathsf{bvadd}(p, s)}_{\text{might overflow}}))}^{\text{always true on overflow}} \implies \mathtt{read}(\mathtt{set}(a, p, v, s), r) = \mathtt{read}(a, r)$$

$\implies$ If $\mathsf{bvadd}(p, s)$ overflows, then $\mathtt{set}(a, p, v, s)$ is a no-op

$\mathcal{T}_{\mathcal{ASC}}$ with Bitvectors (copy)

$$\mathsf{bvule}(p, r) \wedge \mathsf{bvult}(r, \mathsf{bvadd}(p, s)) \implies \mathtt{read}(\mathtt{copy}(a, p, b, q, s), r) = \mathtt{read}(b, \mathsf{bvadd}(q, \mathsf{bvsub}(r, p)))$$

$$\neg(\mathsf{bvule}(p, r) \wedge \mathsf{bvult}(r, \mathsf{bvadd}(p, s))) \implies \mathtt{read}(\mathtt{copy}(a, p, b, q, s), r) = \mathtt{read}(a, r)$$

$\mathcal{T}_{\mathcal{ASC}}$ with Bitvectors (copy)

$$\text{bvule}(p, r) \wedge \text{bvult}(r, \text{bvadd}(p, s)) \implies \texttt{read}(\texttt{copy}(a, p, b, q, s), r) = \texttt{read}(b, \text{bvadd}(q, \text{bvsub}(r, p)))$$

$$\neg(\text{bvule}(p, r) \wedge \text{bvult}(r, \text{bvadd}(p, s))) \implies \texttt{read}(\texttt{copy}(a, p, b, q, s), r) = \texttt{read}(a, r)$$

$\implies$ If $\text{bvadd}(p, s)$ overflows, then $\texttt{copy}(a, p, b, q, s)$ is a no-op

$\mathcal{T}_{\mathcal{ASC}}$ with Bitvectors (copy)

$$\text{bvule}(p, r) \wedge \text{bvult}(r, \text{bvadd}(p, s)) \implies \texttt{read}(\texttt{copy}(a, p, b, q, s), r) = \texttt{read}(b, \text{bvadd}(q, \underbrace{\text{bvsub}(r, p)}_{\text{underflow guarded}}))$$

$$\neg(\text{bvule}(p, r) \wedge \text{bvult}(r, \text{bvadd}(p, s))) \implies \texttt{read}(\texttt{copy}(a, p, b, q, s), r) = \texttt{read}(a, r)$$

$\implies$ If $\text{bvadd}(p, s)$ overflows, then $\texttt{copy}(a, p, b, q, s)$ is a no-op

$\mathcal{T}_{\mathcal{ASC}}$ with Bitvectors (copy)

$$\text{bvule}(p, r) \wedge \text{bvult}(r, \text{bvadd}(p, s)) \implies \texttt{read}(\texttt{copy}(a, p, b, q, s), r) = \texttt{read}(b, \text{bvadd}(q, \underbrace{\text{bvsub}(r, p)}_{\text{underflow guarded}}))$$

range wraps around on overflow

$$\neg(\text{bvule}(p, r) \wedge \text{bvult}(r, \text{bvadd}(p, s))) \implies \texttt{read}(\texttt{copy}(a, p, b, q, s), r) = \texttt{read}(a, r)$$

$\implies$ If $\text{bvadd}(p, s)$ overflows, then $\texttt{copy}(a, p, b, q, s)$ is a no-op

Satisfiability Checking

- Based on reductions to $\mathcal{T}_{\mathcal{A}}$ + index arithmetic

Satisfiability Checking

- Based on reductions to $\mathcal{T}_A$ + index arithmetic
- 3 quantifier-based approaches

Satisfiability Checking

- Based on reductions to $\mathcal{T}_A$ + index arithmetic
- 3 quantifier-based approaches
- 2 quantifier-free approaches
  - Eager encoding
  - Instantiation-based approach

Quantified Approaches

- Fully Quantified Axioms

$$\forall a, p, v, s, r.\ \texttt{read}(\texttt{set}(a, p, v, s), r) = \\ \text{ITE}(p \leq r < p + s,\ v,\ \texttt{read}(a, r))$$

Quantified Approaches

- Fully Quantified Axioms

$$\forall a, p, v, s, r.\ \mathtt{read}(\mathtt{set}(a, p, v, s), r) = \mathrm{ITE}(p \leq r < p + s,\ v,\ \mathtt{read}(a, r))$$

- Quantified Axioms Instantiated for Arrays

$$\forall p, v, s, r.\ \mathtt{read}(\mathtt{set}_a(p, v, s), r) = \mathrm{ITE}(p \leq r < p + s,\ v,\ \mathtt{read}(a, r))$$

Quantified Approaches

- Fully Quantified Axioms

$$\forall a, p, v, s, r.\ \mathtt{read}(\mathtt{set}(a, p, v, s), r) = \mathrm{ITE}(p \leq r < p + s,\ v,\ \mathtt{read}(a, r))$$

- Quantified Axioms Instantiated for Arrays

$$\forall p, v, s, r.\ \mathtt{read}(\mathtt{set}_a(p, v, s), r) = \mathrm{ITE}(p \leq r < p + s,\ v,\ \mathtt{read}(a, r))$$

- Quantified Axioms Instantiated for all Arguments

$$\forall r.\ \mathtt{read}(\mathtt{set}_{a,p,v,s}, r) = \mathrm{ITE}(p' \leq r < p + s,\ v,\ \mathtt{read}(a, r))$$

Quantified Approaches

- Fully Quantified Axioms

$$\forall a, p, v, s, r.\ \mathtt{read}(\mathtt{set}(a, p, v, s), r) = \mathrm{ITE}(p \leq r < p + s,\ v,\ \mathtt{read}(a, r))$$

- Quantified Axioms Instantiated for Arrays

$$\forall p, v, s, r.\ \mathtt{read}(\mathtt{set}_a(p, v, s), r) = \mathrm{ITE}(p \leq r < p + s,\ v,\ \mathtt{read}(a, r))$$

- Quantified Axioms Instantiated for all Arguments

$$\forall r.\ \mathtt{read}(\mathtt{set}_{a,p,v,s}, r) = \mathrm{ITE}(p' \leq r < p + s,\ v,\ \mathtt{read}(a, r))$$

- Require a solver that supports quantifiers

Eager Encoding

- Replace $\texttt{read}(\texttt{set}(a, p, v, s), r)$ by $\text{ITE}(p \leq r < p + s, v, \texttt{read}(a, r))$

Eager Encoding

- Replace $\texttt{read}(\texttt{set}(a, p, v, s), r)$ by $\text{ITE}(p \leq r < p + s, v, \texttt{read}(a, r))$
- $\mathcal{T}_{\mathcal{ASC}}$ axioms are applied eagerly

Eager Encoding

- Replace $\texttt{read}(\texttt{set}(a, p, v, s), r)$ by $\text{ITE}(p \leq r < p + s, v, \texttt{read}(a, r))$
- $\mathcal{T}_{\mathcal{ASC}}$ axioms are applied eagerly
- Requires eager application of $\mathcal{T}_{\mathcal{A}}$'s read-over-write axioms

Eager Encoding

- Replace $\texttt{read}(\texttt{set}(a, p, v, s), r)$ by $\text{ITE}(p \leq r < p + s, v, \texttt{read}(a, r))$
- $\mathcal{T}_{\mathcal{ASC}}$ axioms are applied eagerly
- Requires eager application of $\mathcal{T}_{\mathcal{A}}$'s read-over-write axioms
- Can be implemented as a pre-processing step

Eager Encoding

- Replace $\texttt{read}(\texttt{set}(a, p, v, s), r)$ by $\text{ITE}(p \leq r < p + s, v, \texttt{read}(a, r))$
- $\mathcal{T}_{\mathcal{ASC}}$ axioms are applied eagerly
- Requires eager application of $\mathcal{T}_{\mathcal{A}}$'s read-over-write axioms
- Can be implemented as a pre-processing step
- Can be used in combination with any solver for $\mathcal{T}_{\mathcal{A}}$ + index arithmetic

Instantiation-Based Approach

- Replace $\texttt{set}(a, p, v, s)$ by a fresh constant symbol

Instantiation-Based Approach

- Replace $\texttt{set}(a, p, v, s)$ by a fresh constant symbol
- Needed instantiations of

$$\forall r.\ \texttt{read}(\texttt{set}_{a,p,v,s}, r) = \\ \text{ITE}(p' \leq r < p + s,\ v,\ \texttt{read}(a, r))$$

  are added eagerly

Instantiation-Based Approach

- Replace $\texttt{set}(a, p, v, s)$ by a fresh constant symbol
- Needed instantiations of

$$\forall r.\ \texttt{read}(\texttt{set}_{a,p,v,s}, r) = \\ \text{ITE}(p' \leq r < p + s,\ v,\ \texttt{read}(a, r))$$

  are added eagerly
- One step beyond Quantified Axioms Instantiated for all Arguments

Instantiation-Based Approach

- Replace $\texttt{set}(a, p, v, s)$ by a fresh constant symbol
- Needed instantiations of

$$\forall r.\ \texttt{read}(\texttt{set}_{a,p,v,s}, r) = \mathrm{ITE}(p' \leq r < p + s,\ v,\ \texttt{read}(a, r))$$

  are added eagerly
- One step beyond Quantified Axioms Instantiated for all Arguments
- Can be implemented as a pre-processing step

Instantiation-Based Approach

- Replace $\texttt{set}(a, p, v, s)$ by a fresh constant symbol
- Needed instantiations of

$$\forall r.\ \texttt{read}(\texttt{set}_{a,p,v,s}, r) = \\ \text{ITE}(p' \leq r < p + s,\ v,\ \texttt{read}(a, r))$$

  are added eagerly
- One step beyond Quantified Axioms Instantiated for all Arguments
- Can be implemented as a pre-processing step
- Can be used in combination with any solver for $\mathcal{T}_{\mathcal{A}}$ + index arithmetic

Evaluation

- Done in the software bounded model checker `LLBMC`

Evaluation

- Done in the software bounded model checker `LLBMC`
- Uses bitvectors as index type

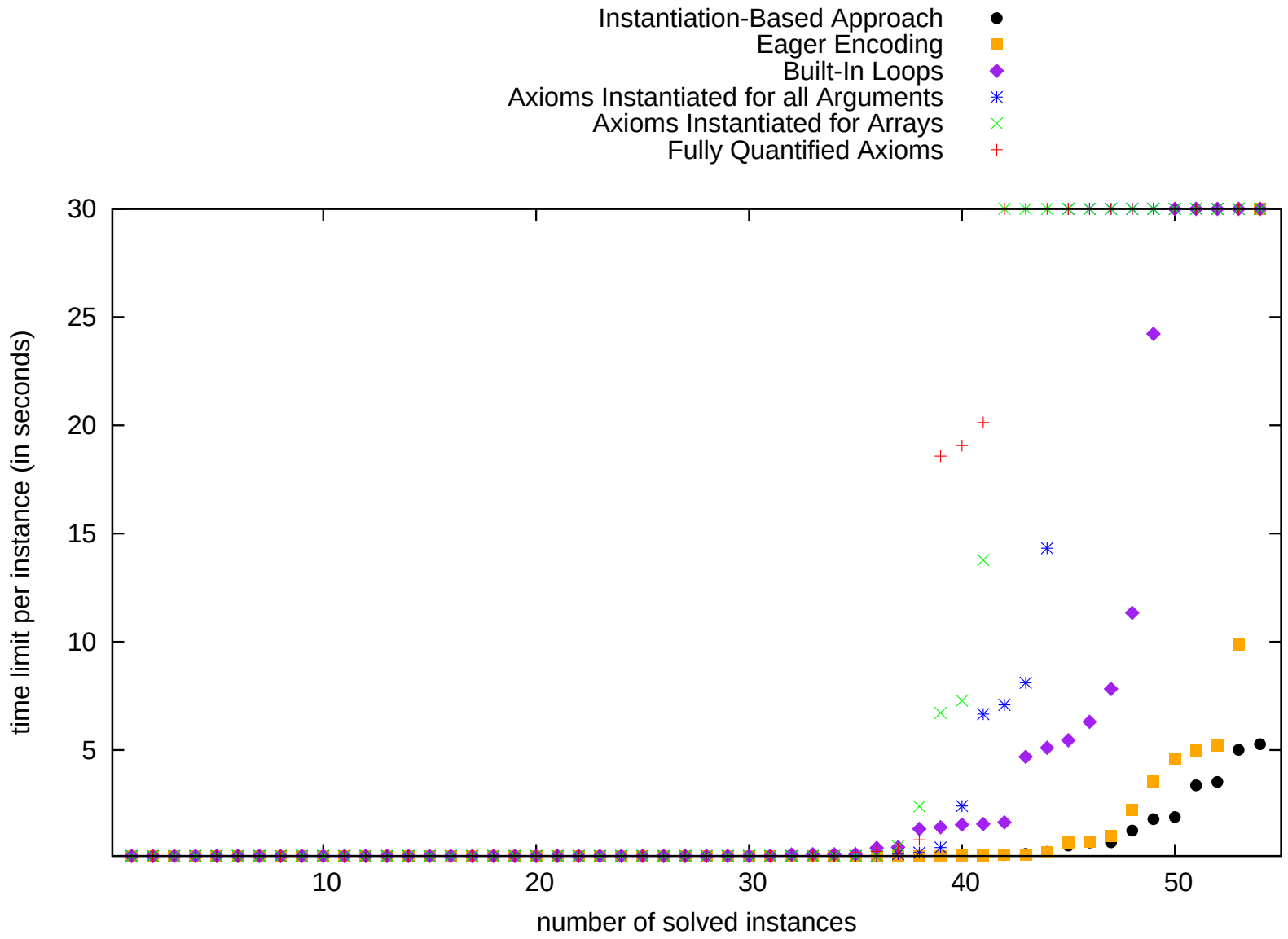
Evaluation

- Done in the software bounded model checker `LLBMC`
- Uses bitvectors as index type
- Applied on 55 examples
  - 19 programs from `LLBMC`'s own $\mathcal{T}_{\mathcal{ASC}}$ test suite
  - 19 programs from `LLBMC`'s own pre-$\mathcal{T}_{\mathcal{ASC}}$ test suite
  - 16 programs from the `SMT-CBMC` software bounded model checker
  - 1 program from the `KLEE` symbolic execution tool

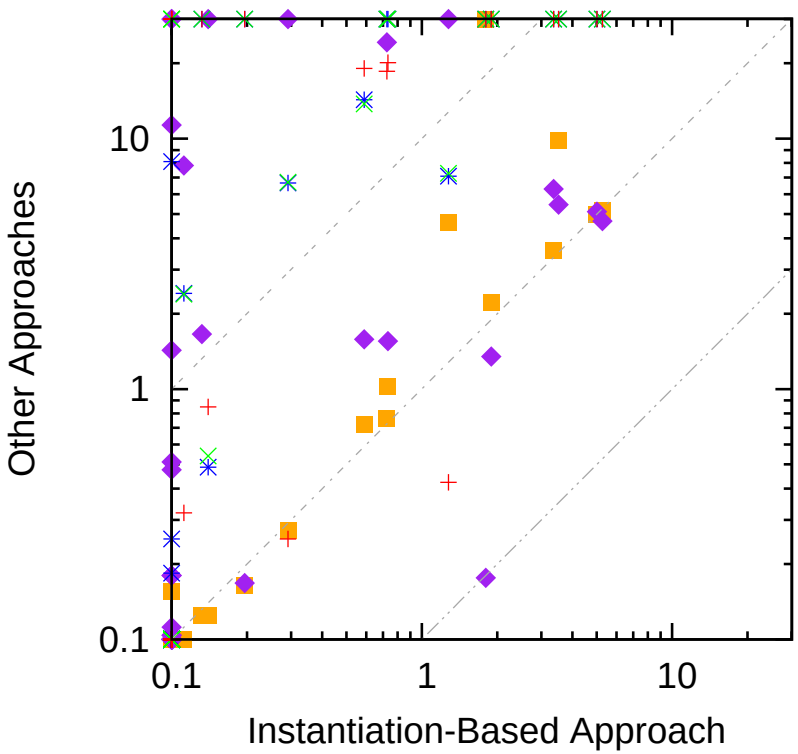
Evaluation

- Done in the software bounded model checker `LLBMC`
- Uses bitvectors as index type
- Applied on 55 examples
  - 19 programs from `LLBMC`'s own $\mathcal{T}_{\mathcal{ASC}}$ test suite
  - 19 programs from `LLBMC`'s own pre-$\mathcal{T}_{\mathcal{ASC}}$ test suite
  - 16 programs from the `SMT-CBMC` software bounded model checker
  - 1 program from the `KLEE` symbolic execution tool
- Evaluated all 5 encodings + built-in loops
  - Quantifier-based approaches used `Z3`
  - Eager- and instantiation-based approaches used `STP`
  - Built-in loops approach used `STP`

Results


Instantiation-Based Approach
Eager Encoding
Built-In Loops
Axioms Instantiated for all Arguments
Axioms Instantiated for Arrays
Fully Quantified Axioms
time limit per instance (in seconds)
number of solved instances
30
25
20
15
10
5
10
20
30
40
50

Results



Eager Encoding
Built-In Loops
Axioms Instantiated for all Arguments
Axioms Instantiated for Arrays
Fully Quantified Axioms

Conclusion and Future Work

- $\mathcal{T}_{\mathcal{ASC}}$ is an useful extension of $\mathcal{T}_{\mathcal{A}}$

Conclusion and Future Work

- $\mathcal{T}_{\mathcal{ASC}}$ is an useful extension of $\mathcal{T}_{\mathcal{A}}$
- Performs better than unrolling loops for `memset` and `memcpy`

Conclusion and Future Work

- $\mathcal{T}_{\mathcal{ASC}}$ is an useful extension of $\mathcal{T}_{\mathcal{A}}$
- Performs better than unrolling loops for `memset` and `memcpy`
- We can do better than `Z3`'s reasoning involving quantifiers

Conclusion and Future Work

- $\mathcal{T}_{\mathcal{ASC}}$ is an useful extension of $\mathcal{T}_{\mathcal{A}}$
- Performs better than unrolling loops for `memset` and `memcpy`
- We can do better than Z3's reasoning involving quantifiers
- "Lemmas-on-demand"/"layz instantiation" is the next step

Karlsruhe Institute of Technology

http://llbmc.org