Decision Procedures and Verification

Martin Blicha

Charles University

7.5.2018

Arrays

Introduction

- Arrays are fundamental nonrecursive data type in programming language
  - Also used for modelling memory in hardware.
- Analysis of software requires the ability to decide formulas containing arrays.
- The array theory permits expressions over arrays, which are formalized as maps from an *index type* to an *element type*.
  - Index type is denoted by T_I, element type by T_E, arrays by T_A which is a short-hand for $T_I \to T_E$.
- Two basic operations on arrays:
  1. *Reading* an element with index i from array a. Value denoted by $a[i]$.
     - *array index* operator
  2. *Writing* a value to array. Writing a value e to the array a at index i is denoted by $a\{i \leftarrow e\}$.
     - *array update* or *array store* operator

Syntax

- Theories used to reason about indices and elements are called *index* theory and *element* theory, respectively.
  - Index theory is usually the theory of linear integer arithmetic.
- Array theory is parametrized by the index theory and the element theory.
- Syntax is an extension of a combination of index and element theory.
  - We add the following rules for valid terms:

$$
\begin{aligned}
term_A &: \text{array-identifier} \mid term_A\{term_I \leftarrow term_E\} \\
term_E &: term_A[term_I]
\end{aligned}
$$

- Equality between array terms is not possible.
  - Will be added later.

Semantics

- The meaning of the new symbols is captured by the following axioms
- $\forall a \in T_A, e \in T_E, i, j \in T_I$

$$
\begin{aligned}
i = j &\Rightarrow a[i] = a[j] && \text{(array congurence)} \\
i = j &\Rightarrow a\{i \leftarrow e\}[j] = e && \text{(read-over-write 1)} \\
i \neq j &\Rightarrow a\{i \leftarrow e\}[j] = a[j] && \text{(read-over-write 2)}
\end{aligned}
$$

Quantifier-free fragment

- Can express properties of elements of arrays, but not properties of arrays.
- Has a decision procedure for satisfiability.
  - It is enough to consider only conjunctive fragment.
- Intuitively:
  - Only read terms: Read terms can be viewed as interpreted function terms
  - Write terms only in the context of a read (equality between arrays not allowed here). read-over-write axiom can be used to deconstruct the read-over-write terms.

Decision procedure for quantifier-free conjunctive fragment

- Let φ be a conjunction of literals in theory of arrays.
- Assumption: there is a decision procedure for quantifier-free fragment of combination of index theory, element theory and uninterpreted functions.

Algorithm QFA-DP

---

1. If φ does not contain any write terms, associate with each array variable a a fresh function symbol f_a and replace each read $a[i]$ with $f_a(i)$. Decide the resulting formula using the assumed decision procedure.
2. Select some read-over-write term $a\{i \leftarrow e\}[j]$ and split on two cases:
   2.1 Replace $\varphi[a\{i \leftarrow e\}[j]]$ with $\varphi[e] \wedge i = j$ and recurse. If answer is SAT, return SAT.
   2.2 Replace $\varphi[a\{i \leftarrow e\}[j]]$ with $\varphi[a[j]] \wedge i \neq j$ and recurse. If answer is SAT, return SAT.
   2.3 If both cases were UNSAT, return UNSAT.

Array property fragment

- Full theory of arrays (with quantifiers) is undecidable in general.
- There is a large, useful fragment that is decidable: *array property* fragment
  - Allows universal quantification over array indices, with some restrictions.

Definition (Array property)

Array property is a formula of the form $\forall i_1, \ldots, i_k . \varphi(i_1, \ldots, i_k) \rightarrow \psi(i_1, \ldots, i_k)$, where $i_1, \ldots, i_k$ is a list of variables and φ, ψ are the *index guard* and *value constraint*, respectively.

Array property fragment

Assumption: index theory is linear integer arithmetic.

Definition

Index guard is a formula syntactically constructed according to the following grammar:

$$\begin{aligned} \mathit{iguard} &: \mathit{iguard} \wedge \mathit{iguard} \mid \mathit{iguard} \vee \mathit{iguard} \mid \mathit{iterm} \leq \mathit{iterm} \mid \mathit{iterm} = \mathit{iterm} \\ \mathit{iterm} &: i_1 \mid \dots \mid i_k \mid \mathit{term} \\ \mathit{term} &: \mathit{integer\text{-}constant} \mid \mathit{integer\text{-}constant} \cdot \mathit{index\text{-}identifier} \mid \\ &\quad \mathit{term} + \mathit{term} \end{aligned}$$

where index-identifier used in term cannot be one of $i_1, \dots, i_k$.

Additionaly, a universally quantified index variable can occur in value constraint ψ only in an array read.
Array property fragment consists of Boolean combinations of quantifier free array formulas and array properties.

Array properties

Example

- *Extensionality* is an array property.
  - Two arrays are equal if all their elements are equal.
  - $a = b$ iff $\forall i . a[i] = b[i]$
- Bounded and unbounded *sorted array* is an array property.
  - $\forall i, j . l \leq i \leq j \leq u \Rightarrow a[i] \leq a[j]$
- *Partitioned array* is an array property.
  - $\forall i, j . l_1 \leq i \leq u_1 < l_2 \leq j \leq u_2 \Rightarrow a[i] \leq a[j]$

Write rule

- Deconstructs write terms
  - Encoding the read-over-write axiom into the formula.

$$\frac{\varphi[a\{i \leftarrow e\}]}{\varphi[a'] \wedge a'[i] = e \wedge \forall j. j \neq i \Rightarrow a[j] = a'[j]} \text{ for fresh } a' \quad \text{(write)}$$

- After application, the resulting formula contians at least one fewer write terms.
- To meet the syntactic constraint rewrite the inequality as $j \leq i - 1 \vee i + 1 \leq j$.

Exists rule

- Removes existential quantifiers by introducing fresh variables.
  - Which are *implicitly* existentially quantified when deciding *satisfiability*.

$$\frac{\varphi[\exists \bar{i}. \psi[\bar{i}]]}{\varphi[\psi[\bar{j}]]} \text{ for fresh } \bar{j} \qquad \text{(exists)}$$

- Existential quantifiers can occur in the formula when it contains a negated array property.

From universal quantification to finite conjunction

- The main idea is to select a set of symbolic index terms on which to instantiate all universal quantifiers.
- Construct an *index set* $\mathcal{I}$ for input formula φ
  1. Add all expressions used as an array index in φ that are not quantified variables.
  2. Add all expressions used inside index guards in φ that are not quantified variables.
  3. If φ contains none of the above, $\mathcal{I}$ is $\{0\}$ in order to obtain a nonempty set of index expressions.
- Replace universal quantification $\forall i.P(i)$ with $\bigwedge_{i \in \mathcal{I}} P(i)$.

Decision procedure for array property fragment

Algorithm APF-DP

---

1. Convert φ to NNF
2. Remove write terms using write rule.
3. Remove existential quantifiers using exists rule.
4. Reduce universal quantification to finite conjunction, instantiating symbolic index terms from corresponding index set.
5. Replace array read terms by uninterpreted functions.
6. Decide the resulting (quantifier-free) formula in index and element theories with uninterpreted functions.

Pointer logic

Simple Pointer Logic

Syntax

$$
\begin{aligned}
\mathit{fla} &: \mathit{fla} \wedge \mathit{fla} \mid \mathit{fla} \vee \mathit{fla} \mid \neg \mathit{fla} \mid \mathit{atom} \\
\mathit{atom} &: \mathit{pointer} = \mathit{pointer} \mid \mathit{term} = \mathit{term} \mid \mathit{pointer} < \mathit{pointer} \mid \\
&\quad \mathit{term} < \mathit{term} \\
\mathit{pointer} &: \mathit{pointer\text{-}identifier} \mid \mathit{pointer} + \mathit{term} \mid \&\mathit{identifier} \mid \& * \mathit{pointer} \mid \\
&\quad * \mathit{pointer} \mid \mathit{NULL} \\
\mathit{term} &: \mathit{identifier} \mid * \mathit{pointer} \mid \mathit{term}\ \mathrm{op}\ \mathit{term} \mid \mathit{integer\text{-}constant} \mid \\
&\quad \mathit{identifier}[\mathit{term}] \\
\mathit{op} &: + \mid -
\end{aligned}
$$

- Assumes variables of pointer type and variables of type integer or array of intefer.
- Allows pointer arithmetic, does not allow conversion between pointers and integers.

Pointer logic formulas - examples

- The following expressions are well-formed according to the grammar:
  - $*(p + i) = 1$
  - $*(p + *p) = 0$
  - $p = q \wedge *p = 5$
  - $* * * * * p = 1$
  - $p < q$
- The following expressions are *not* well-formed according to the grammar:
  - $p + i$
  - $p = i$
  - $*(p + q)$
  - $*1 = 1$
  - $p < i$

Memory model

Definition (Memory model)

Memory model is an address space A corresponding to a subinterval of $\{0, 1, \ldots, N-1\}$. Each address identifies a memory cell that can store a single *data word*. The set of data words is denoted by D. A *memory valuation* $M : A \longrightarrow D$ is a mapping from a set of adresses A into domain D of data words.

Definition (Memory layout)

Let V denote the set of variables. A *memory layout* $L : V \longrightarrow A$ is a mapping from each variable $v \in V$ to an address $a \in A$. The address of v is also called the memory location of v.

Semantics

- Example of a semantics with respect to a specific memory layout L and specific memory valuation M
- Reduction to integer arithmetic and array logic
  - M and L are treated as data types.

Definition (Semantics of simple pointer logic)

Let $\mathcal{L}_P$ denote the set of pointer logic expressions, and let $\mathcal{L}_D$ denote the set of expressions permitted by the logic for the data words. We define a meaning for $e \in \mathcal{L}_P$ using the function $[\![\cdot]\!] : \mathcal{L}_P \longrightarrow \mathcal{L}_D$. The function $[\![e]\!]$ is defined recursively. The expression $e \in \mathcal{L}_P$ is valid if and only if $[\![e]\!]$ is valid.

Semantic Translation

$$
\begin{aligned}
[\![f_1 \wedge f_2]\!] &\doteq [\![f_1]\!] \wedge [\![f_2]\!] && \\
[\![\neg f]\!] &\doteq \neg [\![f]\!] && \\
[\![p_1 = p_2]\!] &\doteq [\![p_1]\!] = [\![p_2]\!] && \text{where } p_1 \text{ and } p_2 \text{ are pointer expressions} \\
[\![p_1 < p_2]\!] &\doteq [\![p_1]\!] < [\![p_2]\!] && \text{where } p_1 \text{ and } p_2 \text{ are pointer expressions} \\
[\![t_1 = t_2]\!] &\doteq [\![t_1]\!] = [\![t_2]\!] && \text{where } t_1 \text{ and } t_2 \text{ are terms} \\
[\![t_1 < t_2]\!] &\doteq [\![t_1]\!] < [\![t_2]\!] && \text{where } t_1 \text{ and } t_2 \text{ are terms} \\
[\![p]\!] &\doteq M[L[p]] && \text{where } p \text{ is a pointer identifier} \\
[\![p + t]\!] &\doteq [\![p]\!] + [\![t]\!] && \text{where } p \text{ is a pointer identifier and } t \text{ is a term} \\
[\![\& v]\!] &\doteq L[v] && \text{where } v \text{ is a variable} \\
[\![\& * p]\!] &\doteq [\![p]\!] && \text{where } p \text{ is a pointer expression} \\
[\![NULL]\!] &\doteq 0 && \\
[\![v]\!] &\doteq M[L[v]] && \text{where } v \text{ is a variable} \\
[\![* p]\!] &\doteq M[[\![p]\!]] && \text{where } p \text{ is a pointer expression} \\
[\![t_1 \text{ op } t_2]\!] &\doteq [\![t_1]\!] \text{ op } [\![t_2]\!] && \text{where } t_1 \text{ and } t_2 \text{ are terms} \\
[\![c]\!] &\doteq c && \text{where } c \text{ is an integer constant} \\
[\![v[t]]\!] &\doteq M[L[v] + [\![t]\!]] && \text{where } v \text{ is an array identifier and } t \text{ is a term}
\end{aligned}
$$

Semantics - example

Example

Consider the following expression where a is an array identifier: $*(\&a+1) = a[1]$. Its semantic definition expands as follows:

$$
\begin{aligned}
[\![*(\&a+1) = a[1]]\!] &\iff [\![*(\&a+1)]\!] = [\![a[1]]\!] \\
&\iff M[[\![\&a+1]\!]] = M[L[a] + [\![1]\!]] \\
&\iff M[[\![\&a]\!] + [\![1]\!]] = M[L[a] + 1] \\
&\iff M[L[a] + 1] = M[L[a] + 1]
\end{aligned}
$$

The resulting formula is valid (TRUE for any M, L), thus so is the original one.

Decision procedure for simple pointer logic

- Formulas generated by this semantic translation contain array read operator and linear arithmetic over type of indices (e.g. integers).
- Decision procedure for pointer logic translates its input formula and calls a decision procedure for combined logic of linear arithmetic over integers and arrays of integers. The returned answer is also correct answer for the original formula.