A Decision Procedure for an Extensional Theory of Arrays

Aaron Stump[1], Clark W. Barrett[1], David L. Dill[1], Jeremy Levitt[2]
[1]Computer Systems Laboratory
Stanford University

[2]0–in Design Automation
San Jose, California

Why reason about arrays

- verification and analysis:
  - software: arrays are ubiquitous
  - hardware: model memories
  - protocols: model message queues
- theorem provers:
  - PVS has undocumented DP for arrays
  - HOL has some automatic support (fpf library)

Example: symbolic simulation

- states s of microprocessor modeled as records
- memory modeled as an array a
- initial state s_0 is a given record
- transition function is record term t(s) with subterms
  - read(a, i)
  - write(a, i, v)

Example: prove an invariant $\forall s \,.\, I(s)$ by proving

$$I(s_0) \;\wedge\; (I(s) \;\Rightarrow\; I(t(s)))$$

Overview

- The language
- The theory
- Related work
- The procedure
- Correctness

Signature

Sorts τ

$$\tau ::= I \mid \mathrm{array}_\tau$$

Variables and constant symbols

a,b,...	: array_τ	(arrays)
i,j,...	: I	(indices)
u,v,...	: I, array_τ	(values)

Function symbols

read_τ : $\mathrm{array}_\tau, I \rightarrow \tau$

write_τ : $\mathrm{array}_\tau, I, \tau \rightarrow \mathrm{array}_\tau$

Dimension

- dimension of a sort
  - $\dim(I) = 0$
  - $\dim(\mathrm{array}_\tau) = \dim(\tau) + 1$
- dimension of a term $t : \tau$
  - $\dim(t) = \dim(\tau)$

array term, index term, multi–dimensional

Language

Atomic formulas

$t = t'$ $\qquad\qquad$ $t, t' : \tau$

Formulas φ

$$\varphi \ ::= \ t = t' \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \Rightarrow \varphi_2 \mid \neg\varphi$$
$$\mid \ \forall x{:}\tau \,.\, \varphi$$

literal, free variables, sentence, theory

Theory Arr

Arr := set of quantifier−free consequences of

(read−over−write):

$$\forall a{:}array_\tau \,.\, \forall i{:}I \,.\, \forall j{:}I \,.\, \forall v{:}\tau \,.$$
$$(i = j \;\Rightarrow\; read(write(a, i, v), j) \;=\; v) \quad \wedge$$
$$(i \neq j \;\Rightarrow\; read(write(a, i, v), j) \;=\; read(a, j))$$

(extensionality):

$$\forall a{:}array_\tau \,.\, \forall b{:}array_\tau \,.$$
$$(\forall i{:}I \,.\, read(a, i) = read(b, i)) \;\Rightarrow\; a = b$$

Related work

- McCarthy (1962)
- Kaplan (1968)
- Suzuki and Jefferson (1977)
- Downey and Sethi (1978)
- Nelson and Oppen (1979)
- Collins and Syme (1995)
- Levitt (1999)

Variations:

- maximal q.f. theory vs. equational theory only
- 1–dimensional vs. multi–dimensional arrays
- extensional vs. non–extensional theory

$\boxed{\text{All but Levitt decide weaker theory}}$

Refutation procedure

$\text{Valid}(\varphi) \iff \neg\text{Sat}(\neg\varphi)$

Restrict attention to conjunctions of literals (goals), since

$\text{Sat}(\varphi) \iff \text{DNF}(\varphi)$ has satisfiable disjunct $\varphi_1 \wedge \ldots \wedge \varphi_n$

- Goals $\varphi_1 \wedge \ldots \wedge \varphi_n$ written $\varphi_1, \ldots, \varphi_n$
- Γ used for goals
- Γ_1 and Γ_2 are equisatisfiable iff $\text{Sat}(\Gamma_1) \iff \text{Sat}(\Gamma_2)$
- Transform goals to equisatisfiable goals

Analysis of literals

Without equations between writes

- Eliminate disequations between array terms:

$$\frac{\Gamma,\ \mathrm{read}(a, k) \neq \mathrm{read}(b, k)}{\Gamma,\ a \neq b}$$

- Eliminate reads over writes by case−splitting:

$$\frac{\Gamma\ [v],\ i = j \qquad\qquad \Gamma\ [\mathrm{read}(a, j)],\ i \neq j}{\Gamma\ [\mathrm{read}(\mathrm{write}(a, i, v), j)]}$$

- Use congruence closure algorithm on result
  - no writes left
  - read behaves like function application

Partial equations

Definition:

$$a =_{\mathbb{I}} b \quad \Leftrightarrow \quad \forall i \notin \mathbb{I} \,.\, \mathrm{read}(a, i) = \mathrm{read}(b, i)$$

Analysis of equations between writes based on

$$\mathrm{write}(a, i, v) = b \quad \Leftrightarrow \quad a =_{\{i\}} b \ \wedge\ \mathrm{read}(b, i) = v$$

Manipulating partial equations

- Eliminate remaining writes:

$$\frac{\Gamma, a =_{\{i\} \cup \mathbb{I}} b,\ \ \mathrm{read}(b, i) = v,\ i \notin \mathbb{I} \qquad\qquad \Gamma, a =_{\mathbb{I}} b,\ \ i \in \mathbb{I}}{\Gamma, \mathrm{write}(a, i, v) =_{\mathbb{I}} b}$$

- Use modified congruence closure:

$$\frac{\Gamma\,[\mathrm{read}(b, i)], a =_{\mathbb{I}} b,\ \ i \notin \mathbb{I} \qquad\qquad \Gamma\,[\mathrm{read}(a, i)], a =_{\mathbb{I}} b,\ \ i \in \mathbb{I}}{\Gamma\,[\mathrm{read}(a,i)], a =_{\mathbb{I}} b}$$

Example

$$
\begin{array}{c}
\hline
k = i,\ a =_{\{i\}} b,\ \mathrm{read}(b, i) = v,\ \mathrm{read}(a, i) = v,\ v \neq v \\
\hline
k = i,\ a =_{\{i\}} b,\ \mathrm{read}(b, i) = v,\ \mathrm{read}(a, i) = v,\ \mathrm{read}(a, i) \neq v \\
\hline
k = i,\ a =_{\{i\}} b,\ \mathrm{read}(b, i) = v,\ \mathrm{read}(a, i) = v,\ \mathrm{read}(a, i) \neq \mathrm{read}(b, i) \\
\hline
k = i,\ a =_{\{i\}} b,\ \mathrm{read}(b, i) = v,\ \mathrm{read}(a, i) = v,\ \mathrm{read}(a, k) \neq \mathrm{read}(b, k)
\end{array}
$$

$$
\begin{array}{c}
\hline
k \neq i,\ a =_{\{i\}} b,\ \mathrm{read}(b, k) \neq \mathrm{read}(b, k) \\
\hline
a =_{\{i\}} b,\ \mathrm{read}(b, i) = v,\ \mathrm{read}(a, i) = v,\ \mathrm{read}(a, k) \neq \mathrm{read}(b, k) \\
\hline
a =_{\{i\}} b,\ \mathrm{read}(b, i) = v,\ \mathrm{read}(a, i) = v,\ a \neq b \\
\hline
\mathrm{write}(a, i, v) = b,\ \mathrm{read}(a, i) = v,\ a \neq b
\end{array}
$$

Correctness

- termination
  - problem is NP–complete
  - algorithm runs in $O(2^{n \log n})$ time
- soundness of rules
- invertibility of rules
- construct model for normal goal

Cooperation

- to be useful, DP must cooperate with other DPs
  - e.g., Presburger arithmetic, ADTs
- two frameworks commonly used to allow cooperation
  - Nelson–Oppen
  - Shostak
- algorithm will not fit into Shostak's framework
- extensions required for Nelson–Oppen (in paper)

Implementation

- C++ implementation in SVC
  - incremental addition of literals to goal
  - includes extensions needed for Nelson−Oppen
  - 2200 lines of C++ in 16 classes
    - 1200 deal with partial equations
  - algorithm not hard to code
- earlier prototype in XSB Prolog took 300 lines

Conclusion

- first decision procedure for Arr
- suitable for Nelson–Oppen framework

http://verify.stanford.edu