Entscheidungsverfahren mit Anwendungen in der Softwareverifikation

IX: Logik der Arrays

Carsten Sinz
Institut für Theoretische Informatik

07.01.2020

Motivation

- Beispiel 1: (Initialisierung)

```
int a[MAX];
for (int i=0; i < MAX; i++)
  a[i] = 0;
assert(∀j : 0≤j<MAX ⇒ a[j]=0);
```

- Beispiel 2: (Suche in Array)

```
int a[3], res = 0;
for (int i=0; i < 3; i++)
  if (a[i] == 42)
    res = 1;
assert(res=1 ⇔ ∃j : 0≤j<3 ∧ a[j] = 42);
```

- Beispiel 3: (Einfache Array-Eigenschaft)

$(i = j \wedge a[j] = \text{'z'}) \Rightarrow a[i] = \text{'z'}$

Motivation

- Schleifen mit einer festen Anzahl von Iterationen können abgerollt werden:

```
1:  int a[3], res = 0;
2:  if (a[0] == 42) res = 1;
3:  if (a[1] == 42) res = 1;
4:  if (a[2] == 42) res = 1;
5:  assert(res=1 ⇔ a[0]=42 ∨ a[1]=42 ∨ a[2]=42);
```

- Umsetzung in Logik (Formel ist unerfüllbar, wenn Assertion gilt):

$$
\begin{aligned}
&res_1=0 \wedge \\
&(a[0]=42 \Rightarrow res_2=1) \wedge (a[0]\neq 42 \Rightarrow res_2=res_1) \wedge \\
&(a[1]=42 \Rightarrow res_3=1) \wedge (a[1]\neq 42 \Rightarrow res_3=res_2) \wedge \\
&(a[2]=42 \Rightarrow res_4=1) \wedge (a[2]\neq 42 \Rightarrow res_4=res_3) \wedge \\
&\neg(res_4=1 \Leftrightarrow (a[0]=42 \vee a[1]=42 \vee a[2]=42))
\end{aligned}
$$

- **Anmerkung:** Prüfung von Array-Grenzen benötigt keine Array-Theorie

Motivation

- **Arrays sind wichtige Datenstrukturen:**
  - In (fast) allen Programmiersprachen vorhanden
  - Von fast allen Mikroprozessoren werden spezielle Adressierungsarten für Arrays bereitgestellt, z.B. x86:

```
movl   $17, 20(%eax)
movl   $42, (%eax,%ecx,4)
```

  - O(1)-Zugriff auf Elemente
  - Hauptspeicher kann als Array modelliert werden

Theorie der Arrays

- Arrays sind Abbildungen eines *Indextyps* auf einen *Elementtyp:*
  - T_I : Indextyp
  - T_E : Elementtyp
  - $T_A = (T_I \rightarrow T_E)$: Arraytyp
- Wir nehmen implizit an, dass Gleichheit sowohl auf T_I als auch auf T_E definiert ist:

  $=_I \subseteq (T_I \times T_I)$ und $=_E \subseteq (T_E \times T_E)$
- Die entsprechenden Theorien (Indextheorie, Elementtheorie) bezeichnen wir ebenfalls mit T_I und T_E.

Theorie der Arrays

- Auf Arrays sind zwei Operationen definiert:

  `read`: $T_A \times T_I \rightarrow T_E$

  `write`: $T_A \times T_I \times T_E \rightarrow T_A$

- Informelle Semantik:
  - `read(a,i)` ist der Wert von Array an Index i.
  - `write(a,i,e)` ist das Array, in dem das Element an Index i durch e ersetzt wurde.
- Formale Semantik definiert durch *read-over-write*-Axiom (McCarthy, 1962):

$$\texttt{read}(\texttt{write}(a, i, e), j) = \begin{cases} e & \text{falls } i = j \\ \texttt{read}(a, j) & \text{sonst} \end{cases}$$

Theorie der Arrays

- **Welche Logiken kommen für Index- und Elementtheorie in Frage?**
  - Indextheorie sollte existentielle und universelle Quantifikation erlauben
    - Siehe einführende Beispiele:
      „für alle j ist a[j]=0“, „es gibt ein j mit a[j]=42“
  - Indextheorie sollte (lineare) Arithmetik umfassen
  - Elementtheorie relativ beliebig

- **Problem:**
  - Die Entscheidbarkeit der Arraytheorie (als kombinierte Theorie) hängt von Index- und Elementtheorie ab!
- **Generelle Annahme im Weiteren:**
  - T_I und T_E seien entscheidbar

Syntax der vollen Arraytheorie

- **Signatur:**

$$\begin{aligned} \Sigma &= \Sigma_I \cup \Sigma_E \cup \Sigma_A \\ \Sigma_A &= \{\texttt{read}, \texttt{write}, =_A\} \\ =_I &\in \Sigma_I \\ =_E &\in \Sigma_E \end{aligned}$$

- Quantoren über Variablen aus T_A, T_I und T_E
- **Axiome:**

  $Ax_I \cup Ax_E \cup \{$ *read-over-write*-Axiom $\} \cup \{$ Extensionalitäts-Axiom $\}$

- **Extensionalitätsaxiom:**

  $a = b \quad \Leftrightarrow \quad \forall i : a[i] = b[i]$

- **Notation:**
  - Anstelle von `read`(*a*, *i*) schreiben wir manchmal auch *a*[*i*]

Varianten der Arraytheorie

- **Quantorenfreies Fragment von T_A (T_A^{QFF}):**
  - keinerlei Quantoren erlaubt (weder in T_I, noch in T_E, noch über Array-Variablen); kein Gleichheitsprädikat auf Array-Termen
  - nicht sehr ausdrucksstark: nur Aussagen über einzelne Arrayelemente möglich
- **Extensionelle Theorie der Arrays:**
  - Zusätzlich zu `read` und `write` auch Gleichheit auf Array-Termen
- **Array-Property-Fragment:**
  - Universelle Quantoren für Index-Variablen mit Beschränkungen erlaubt
- **T_A^{QFF} mit Lambda-Termen:**
  - „Konstruktoren“ für Terme: $a = \lambda i \,.\, t(i)$ für einen Term t
- **Volle Array-Logik:**
  - Quantoren erlaubt, auch über Arrayelemente

Quantorenfreies Fragment der Array-Theorie

- Beispiele: Sind die folgenden Formeln gültig?

  1. $(i = j \wedge \texttt{read}(a,j) = \text{'z'}) \Rightarrow \texttt{read}(a,i) = \text{'z'}$

  2. $\texttt{read}(\texttt{write}(a,i,e),i) \geq e$

- **Entscheidungsverfahren zur Erfüllbarkeit einer Formel F** $\in T_A^{QFF}$**:**

  3. Falls F keine `write`-Terme enthält:

     - Führe für jede Array-Variable *a* ein neues 1-stelliges Funktionssymbol f_a ein und ersetze jeden Term der From `read`(*a*,*i*) durch $f_a(i)$.

     - Die entstandene Formel ist in $T_I \cup T_E \cup T_{EUF}$ und kann mit bekannten Verfahren gelöst werden (Kongruenzabschluss, Ackermann-Reduktion)

  4. Ansonsten: Wähle einen read-over-write-Term `read`(`write`(*a*,*i*,*e*),*j*) aus und rufe das Entscheidungsverfahren zwei Mal rekursiv auf:

     - Ersetze F[`read`(`write`(*a*,*i*,*e*),*j*)] durch F' := (F[e] $\wedge$ i=j) und rufe das Entscheidungsverfahren mit F' auf. Falls F' erfüllbar, gib „erfüllbar“ aus.

     - Ersetze F[`read`(`write`(*a*,*i*,*e*),*j*)] durch F'' := (F[`read`(a,j)] $\wedge$ i≠j) und rufe das Entscheidungsverfahren mit F' auf. Falls F'' erfüllbar, gib „erfüllbar“ aus.

     - Ist sowohl F' als auch F'' unerfüllbar, gib „unerfüllbar“ zurück.

Entscheidungsverfahren für T_A^{QFF}: Beispiele

1. $i = j \wedge \texttt{read}(a,j) = \text{'z'} \wedge \texttt{read}(a,i) \neq \text{'z'}$

   - Keine `write`-Terme, also neues Funktionssymbol f_a für a. Somit: $i = j \wedge f_a(j) = \text{'z'} \wedge f_a(i) \neq \text{'z'}$. Diese Formel ist T_{EUF}-unerfüllbar.

2. $\neg(\texttt{read}(\texttt{write}(a,i,e),i) \geq e)$

   - Fallunterscheidung:

     1. Ersetze `read(write(a,i,e),i)`: $F' = \neg(e \geq e) \wedge i = i$. Diese Formel ist $T_I \cup T_E \cup T_{EUF}$-unerfüllbar.
     2. Ersetze `read(write(a,i,e),i)`: $F'' = \neg(\text{read}(a,i) \geq e) \wedge i \neq i$. Diese Formel ist ebenfalls $T_I \cup T_E \cup T_{EUF}$-unerfüllbar.

   - Also ist $\texttt{read}(\texttt{write}(a,i,e),i) \geq e$ allgemeingültig.

Extensionelle Theorie der Arrays

- Nun sei auch **Gleichheit auf Array-Termen** erlaubt und definiert durch folgendes Axiom (für Array-Terme a, b):

  $a = b \quad \Leftrightarrow \quad \forall i : a[i] = b[i]$

- Wie sieht es mit der Entscheidbarkeit (der Erfüllbarkeit) dieser erweiterten Logik aus?
  - Für negierte Gleichheitsterme reicht unser bisheriges Entscheidungsverfahren aus.
    - Denn $\neg(a=b) = \neg(\forall i : a[i] = b[i]) = \exists i : a[i] \neq b[i]$.
      ➔ Ersetze $a \neq b$ durch $a[i^*] \neq b[i^*]$, wobei i^* eine frische Index-Variable ist.
  - Den allgemeinen Fall behandeln wir im Rahmen des Array-Property-Fragments.
  - Dieser wird auch von Stump, Barrett, Dill, Levitt (2001) behandelt.

Array-Property Fragment

- **Grundidee:** Erlaube Quantoren über Indexterme mit gewissen Beschränkungen
- **Grundbaustein:** Array-Properties: $\forall i_1,...,i_k: F[i_1,...,i_k] \Rightarrow G[i_1,...,i_k]$
  - F: *index guard*, G: *value constraint*
- Index-Guard muss nach folgender Grammatik aufgebaut sein:

iguard	::=	iguard $\wedge$ iguard $\mid$ iguard $\vee$ iguard $\mid$ atom
atom	::=	expr $\leq$ expr $\mid$ expr = expr
expr	::=	*uvar* $\mid$ pexpr
pexpr	::=	pexpr'
pexpr'	::=	$\mathbb{Z}$ $\mid$ $\mathbb{Z} \cdot$ *evar* $\mid$ pexpr' + pexpr'

  Dabei: *uvar*: universell quant. Variable; *evar*: exist. quant. oder freie Var.
- Value-Constraint unterliegt folgender Einschränkung:
  - Index-Variablen $i_1,...,i_k$ dürfen nur in der Form $a[i_j]$ auftreten
  - D.h., z.B. $a[b[i_j]]$ ist nicht erlaubt.
- Formeln des Array-Property-Fragments sind Boolesche Kombinationen von Array-Properties und quantorenfreien Array-Logik-Formeln

Array-Property-Fragment: Beispiele

- Extensionalität: $\forall i: a[i] = b[i]$
- Beschränkte Extensionalität: $\forall i: l \leq i \leq u \implies a[i] = b[i]$
- Element-Eigenschaft: $\forall i: F[a[i]]$
- Eig. „sortierter Bereich“: $\forall i: l \leq i \leq j \leq u \implies a[i] \leq a[j]$

- Was ist mit $\forall i: i \leq a[k] \implies a[i] = a[k]$?
  - Nicht erlaubt, da in Index-Guard keine Array-Zugriffe vorkommen dürfen
  - Lässt sich aber umformen zu $v = a[k] \wedge (\forall i: i \leq v \implies a[i] = a[k])$, was dann ok ist.

Array-Property-Fragment: Entscheidungsverfahren

- **Eingabe:** Formel *F* des Array-Property-Fragments
- **Schritt 1:** Bringe *F* in NNF
- **Schritt 2:** Wende die folgende Ersetzungsregel an, um `write`-Ausdrücke zu eliminieren:

$$\frac{F[\texttt{write}(a, i, e)]}{F[a'] \ \wedge \ a'[i] = e \ \wedge (\forall j: \ j \neq i \Rightarrow a[j] = a'[j])} \quad \text{für neue Var. } a'$$

- **Schritt 3:** Wende die folgende Ersetzungsregel an, um exist. Quant. zu eliminieren:

$$\frac{F[\exists i_1, \ldots, i_k: \ G[i_1, \ldots, i_k]]}{F[G[j_1, \ldots, j_k]]} \quad \text{für neue Variablen } j_l$$

- **Schritt 4:** Berechne Index-Menge *I*:

$$\begin{aligned} I \ := \quad & \{t \ : \ \cdot[t] \text{ tritt in } F \text{ auf}, t \text{ keine univ. quant. Var}\} \\ & \cup \{t \ : \ t \text{ tritt innerhalb einer } \texttt{pexpr} \text{ im Index-Guard auf}\} \end{aligned}$$

  - Falls *I* = ∅, setze *I* = { 0 }.

AP-Fragment: Entscheidungsverfahren

- **Schritt 5:** Wende die folgende Ersetzungsregel an, um universelle Quantifizierung zu eliminieren:

$$\frac{H[\forall i_1, \ldots, i_k \ : \ F[i_1, \ldots, i_k] \Rightarrow G[i_1, \ldots, i_k]}{H[\bigwedge_{i_1, \ldots, i_k \in I^k} F[i_1, \ldots, i_k] \Rightarrow G[i_1, \ldots, i_k]]}$$

Schritt 6: *F* kann nun mit einem Entscheidungsverfahren für T_A^{QFF} gelöst werden

- **Beispiel:** Siehe Bradley/Manna, The Calculus of Computation

Erweiterungen des Array-Property-Fragments

- **Theorem (Bradley, Manna, Sipma; 2006):**

  Jede der folgenden Erweiterungen des Array-Property-Fragments resultiert in einem Fragment von T_A, für das die Erfüllbarkeit unentscheidbar ist:

  - Verschachtelte `reads` (z.B. *a*[*b*[*i*]], wobei *i* universell quantifiziert ist)
  - `reads` mit einer universell quantifizierten Variablen im Index-Guard
  - Presburger Arithmetik über universell quantifizierten Variablen im Index-Guard oder im Werte-Constraint (selbst Addition von 1, z.B. *i*+1, reicht dazu schon aus)

- Volle Array-Logik ist daher auch unentscheidbar

Zusammenfassung

Array-Theorie	Entscheidbar?
Quantorenfreies Fragment (T_A^{QFF})	ja
T_A^{QFF} mit Extensionalität	ja
Property-Fragment	ja
Erweiterungen des Property-Frag.	meist nein
Volle Arraylogik (mit Quantoren)	nein