Acceleration-based Safety Decision Procedure for Programs with Arrays

F. Alberti$^1$, S. Ghilardi$^2$, N. Sharygina$^1$

$^1$University of Lugano, Switzerland
$^2$University of Milan, Italy

LPAR-19
December 15, 2013

Talk based on results previously published at FroCoS 2013.
procedure Find( a[L] , e ) {
  \( l_I \)
  \( i = 0; \)
  \( l_L \)
  while ( \( i < L \land a[i] \neq e \) ) {
    \( i = i + 1; \)
  }
  \( l_F \)
  assert ( \( \forall x.(0 \leq x < i) \rightarrow a[x] \neq e \) );
}
procedure Find( a[L] , e ) {
  l_I  i = 0;
  l_L  while ( i < L ∧ a[i] ≠ e ) {
           i = i + 1;
  }
  l_F  assert ( ∀x.(0 ≤ x < i) → a[x] ≠ e );
  }

Is this program safe?
procedure Find( a[L] , e ) {

  l_I    i = 0;

  l_L    while ( i < L ∧ a[i] ≠ e ) {
           i = i + 1;
         }

  l_F    assert ( ∀x.(0 ≤ x < i) → a[x] ≠ e );
}

■ Is this program safe?

■ Can we decide its safety automatically?
Problem:

- Infinitely many paths to analyze because of loops bounded by symbolic constants (e.g., $L$, the length of the array)
Our solution

Problem:
- Infinitely many paths to analyze because of loops bounded by symbolic constants (e.g., $L$, the length of the array)

Idea:
Formal framework

\[ S_T = (v, I(v), \tau(v, v')) \]

1In all the formulæ we admit the term \( a(t) \) only if \( t \) is a variable or a constant.
Formal framework

\[ S_T = (v, I(v), \tau(v, v')) \]

- \( T \) is Presburger arithmetic enriched with free function symbols
- Satisfiability and validity with respect to structures having the standard structure of natural numbers as reduct
- \( v \) contains free unary function symbols (\( a \)) and free constants (\( c \))

\(^1\)In all the formulæ we admit the term \( a(t) \) only if \( t \) is a variable or a constant.
Formal framework

\[ S_T = ( \mathbf{v} , I(\mathbf{v}) , \tau(\mathbf{v}, \mathbf{v}') ) \]

- \( T \) is Presburger arithmetic enriched with free function symbols
  - satisfiability and validity with respect to structures having the standard structure of natural numbers as reduct
  - \( \mathbf{v} \) contains free unary function symbols (a) and free constants (c)

Classification of formulæ\(^1\):

- \textit{ground} – formulas of the kind \( \phi(\mathbf{v}) \)

\(^1\)In all the formulæ we admit the term \( a(t) \) only if \( t \) is a variable or a constant.
Formal framework

\[ S_T = (\, v, I(v), \tau(v, v') \, ) \]

- \( T \) is Presburger arithmetic enriched with free function symbols
  - satisfiability and validity with respect to structures having the standard structure of natural numbers as reduct
  - \( v \) contains free unary function symbols \( (a) \) and free constants \( (c) \)

Classification of formulæ\(^1\):

- \textit{ground} – formulas of the kind \( \phi(v) \)
- \( \Sigma^0_1 \) – formulas of the kind \( \exists_i.\phi(i, v) \)

---

\(^1\)In all the formulæ we admit the term \( a(t) \) only if \( t \) is a variable or a constant.
Formal framework

\[ S_T = (\, v \, , \, I(v) \, , \, \tau(v, v') \, ) \]

- \( T \) is Presburger arithmetic enriched with free function symbols
- satisfiability and validity with respect to structures having the standard structure of natural numbers as reduct
- \( v \) contains free unary function symbols (\( a \)) and free constants (\( c \))

Classification of formulæ\(^1\):

- \textit{ground} – formulas of the kind \( \phi(v) \)
- \( \Sigma_1^0 \) – formulas of the kind \( \exists i.\phi(i, v) \)
- \( \Sigma_2^0 \) – formulas of the kind \( \exists i \forall j.\phi(i, j, v) \)

\(^1\)In all the formulæ we admit the term \( a(t) \) only if \( t \) is a variable or a constant.
Acceleration: Transitive closure $\tau^+$ of transitions $\tau$ encoding cyclic actions
Acceleration: Transitive closure $\tau^+$ of transitions $\tau$ encoding cyclic actions

Challenges:

In general transitive closure cannot be expressed in FOL
Acceleration
State of the art

Acceleration: Transitive closure $\tau^+$ of transitions $\tau$ encoding cyclic actions

Challenges:

- In general transitive closure cannot be expressed in FOL
- Only some (important) classes of $\tau$’s allow the definability of $\tau^+$
  - Polling-based systems [BBD+02]
  - Imperative programs over integers [BIK10]
Acceleration: Transitive closure $\tau^+$ of transitions $\tau$ encoding cyclic actions

Challenges:

- In general transitive closure cannot be expressed in FOL
- Only some (important) classes of $\tau$’s allow the definability of $\tau^+$
  - Polling-based systems [BBD$^+$02]
  - Imperative programs over integers [BIK10]

- What about arrays?
  - Acceleration of local ground assignment [AGS13] can be expressed in the theory $T$ as $\Sigma^0_2$-assignments
$\tau_1 := pc = l_L \land i < L \land a[i] \neq e \land i' = i + 1$
Example

\( \tau_1 := pc = l_L \land i < L \land a[i] \neq e \land i' = i + 1 \)

\( \downarrow \)
Acceleration for arrays

Example

\[ \tau_1 := pc = l_L \land i < L \land a[i] \neq e \land i' = i + 1 \]

\[ \tau_1^+ := \exists y. \left( y > 0 \land pc = l_L \land \forall j. (i \leq j < i + y \implies j < L \land a[j] \neq e) \land i' = i + y \right) \]
Example

\[ \tau_1 := pc = l_l \land i < L \land a[i] \neq e \land i' = i + 1 \]

\[ \Downarrow \]

\[ \tau_1^+ := \exists y. \left( y > 0 \land pc = l_l \land \forall j. (i \leq j < i + y \Rightarrow j < L \land a[j] \neq e) \land i' = i + y \right) \]
Acceleration for arrays

Example

\[ \tau_1 := pc = l_L \land i < L \land a[i] \neq e \land i' = i + 1 \]

Number of iterations

\[ \tau_1^+ := \exists y. \left( y > 0 \land pc = l_L \land \forall j. (i \leq j < i + y \rightarrow j < L \land a[j] \neq e) \land i' = i + y \right) \]
Acceleration for arrays

Example

$$\tau_1 := pc = l_L \land i < L \land a[i] \neq e \land i' = i + 1$$

↓

Number of iterations

$$\tau_1^+ := \exists y. \left( y > 0 \land pc = l_L \land \forall j. (i \leq j < i + y \rightarrow j < L \land a[j] \neq e) \land i' = i + y \right)$$

The guard is satisfied for all iterations
Acceleration for arrays

Example

\[ \tau_1 := pc = l_L \land i < L \land a[i] \neq e \land i' = i + 1 \]

Number of iterations

\[ \tau_1^+ := \exists y. \left( y > 0 \land pc = l_L \land \forall j. (i \leq j < i + y \rightarrow j < L \land a[j] \neq e) \land i' = i + y \right) \]

The guard is satisfied for all iterations

Do the “jump”
Contribution

- $\Sigma_2^0$-formulæ over $T$ may not admit decision procedures
Contribution

× $\Sigma^0_2$-formulæ over $T$ may not admit decision procedures

I. Notion of *basic-assignments*
   - Subclass of *local ground assignments* [AGS13]
   ✔ Acceleration of *basic assignments* is an Array Property formula [BMS06]
Contribution

✗ $\Sigma_2^0$-formulæ over $T$ may not admit decision procedures

I. Notion of basic-assignments

- Subclass of local ground assignments [AGS13]
- ✓ Acceleration of basic assignments is an Array Property formula [BMS06]

II. Notion of basic-flat-programs

- flat control flow graph
- every non-loop edge is labeled with a ground or $\Sigma_1^0$-assignment
- every loop edge is labeled with a basic-assignment.
Contribution

- $\Sigma^0_2$-formulæ over $T$ may not admit decision procedures

I. Notion of basic-assignments
   - Subclass of local ground assignments [AGS13]
   - Acceleration of basic assignments is an Array Property formula [BMS06]

II. Notion of basic-flat-programs
   - flat control flow graph
   - every non-loop edge is labeled with a ground or $\Sigma^0_1$-assignment
   - every loop edge is labeled with a basic-assignment.

III. The reachability problem for basic-flat-programs is **decidable**
   1. Accelerate all the loops (basic-assignments)
   2. Consider all (finitely many) paths from $l_{\text{init}}$ to $l_{\text{error}}$
      \[\Rightarrow\] Feasible iff the corresponding Array Property formula is satisfiable
Procedures handling arrays of unknown length like:

- Initialization of the array to a given value
- Searching in an array for a given value
- Swapping two different arrays
- Testing if two arrays are equal
Conclusion

1. Acceleration to reduce the number of possible error paths of a basic-flat-program from infinite to finite
1. Acceleration to reduce the number of possible error paths of a basic-flat-program from infinite to finite

2. Accelerations of basic-assignments are $\Sigma^0_2$-assignments belonging to the Array Property fragment [BMS06]
Conclusion

1. Acceleration to reduce the number of possible error paths of a basic-flat-program from infinite to finite

2. Accelerations of basic-assignments are \( \Sigma_2^0 \)-assignments belonging to the Array Property fragment [BMS06]

\( \Rightarrow \) The combination of the two above results allows to establish a full decidability result for basic-flat-program with arrays.

Future work:
- New decidability results for array programs based on New decidable (quantified) fragments of array theories
- New acceleration schemata for assignments modeling pieces of code

Thank you! Questions?
1. Acceleration to reduce the number of possible error paths of a basic-flat-program from infinite to finite

2. Accelerations of basic-assignments are $\Sigma^0_2$-assignments belonging to the Array Property fragment [BMS06]

$\Rightarrow$ The combination of the two above results allows to establish a full decidability result for basic-flat-program with arrays.

**Future work:** new decidability results for array programs based on

- New decidable (quantified) fragments of array theories
- New acceleration schemata for assignments modeling pieces of code
Conclusion

1. Acceleration to reduce the number of possible error paths of a basic-flat-program from infinite to finite

2. Accelerations of basic-assignments are $\Sigma_2^0$-assignments belonging to the Array Property fragment [BMS06]

⇒ The combination of the two above results allows to establish a full decidability result for basic-flat-program with arrays.

Future work: new decidability results for array programs based on
- New decidable (quantified) fragments of array theories
- New acceleration schemata for assignments modeling pieces of code

Thank you! Questions?
Francesco Alberti, Silvio Ghilardi, and Natasha Sharygina. Definability of accelerated relations in a theory of arrays and its applications.

Marius Bozga, Radu Iosif, and Filip Konecný.
Fast acceleration of ultimately periodic relations.

Aaron R. Bradley, Zohar Manna, and Henny B. Sipma.
What’s decidable about arrays?