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.

Context: decide the safety of programs with arrays

$$
\begin{array}{ll}
 & \textsf{procedure Find}(\ \mathtt{a}[\mathtt{L}]\ ,\ \mathtt{e}\ )\ \{ \\
l_I & \quad \mathtt{i} = 0; \\
l_L & \quad \textsf{while}\ (\ \mathtt{i} < \mathtt{L} \land \mathtt{a}[\mathtt{i}] \neq \mathtt{e}\ )\ \{ \\
 & \qquad \mathtt{i} = \mathtt{i} + 1; \\
 & \quad \} \\
l_F & \quad \textsf{assert}\ (\ \forall x.(0 \leq x < \mathtt{i}) \rightarrow \mathtt{a}[x] \neq \mathtt{e}\ ); \\
 & \}
\end{array}
$$

Context: decide the safety of programs with arrays

```
      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?

Context: decide the safety of programs with arrays

```
        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?

Our solution

Problem:

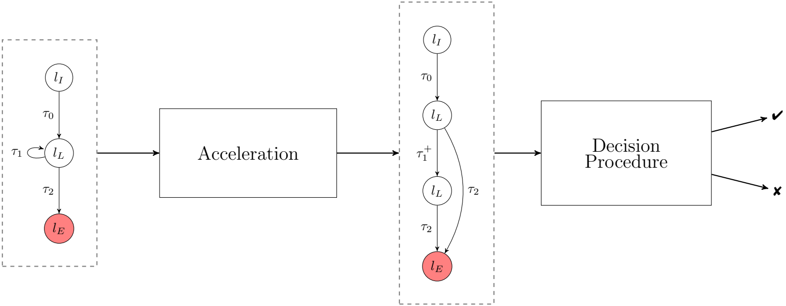
- Infinitely many paths to analyze because of loops bounded by symbolic constants (e.g., L, the length of the array)

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

$$\mathcal{S}_T \;=\; (\; \mathbf{v} \;,\; I(\mathbf{v}) \;,\; \tau(\mathbf{v}, \mathbf{v}') \;)$$

[1] In all the formulæ we admit the term $a(t)$ only if t is a variable or a constant.

Formal framework

$$\mathcal{S}_T \;=\; (\; \mathbf{v} \;,\; I(\mathbf{v}) \;,\; \tau(\mathbf{v}, \mathbf{v}') \;)$$

- 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 ($\mathbf{a}$) and free constants ($\mathbf{c}$)

[1] In all the formulæ we admit the term $a(t)$ only if t is a variable or a constant.

Formal framework

$$\mathcal{S}_T \;=\; (\;\mathbf{v}\;,\; I(\mathbf{v})\;,\; \tau(\mathbf{v},\mathbf{v}')\;)$$

- 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 ($\mathbf{a}$) and free constants ($\mathbf{c}$)

Classification of formulæ[1]:

- *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

$$\mathcal{S}_T \; = \; (\; \mathbf{v} \; , \; I(\mathbf{v}) \; , \; \tau(\mathbf{v}, \mathbf{v}') \;)$$

- 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 ($\mathbf{a}$) and free constants ($\mathbf{c}$)

Classification of formulæ[1]:

- *ground* – formulas of the kind $\phi(\mathbf{v})$
- Σ^0_1 – formulas of the kind $\exists \underline{i}.\phi(\underline{i}, \mathbf{v})$

[1] In all the formulæ we admit the term $a(t)$ only if t is a variable or a constant.

Formal framework

$$\mathcal{S}_T \;=\; (\;\mathbf{v}\;,\; I(\mathbf{v})\;,\; \tau(\mathbf{v},\mathbf{v}')\;)$$

- 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 ($\mathbf{a}$) and free constants ($\mathbf{c}$)

Classification of formulæ[1]:

- *ground* – formulas of the kind $\phi(\mathbf{v})$
- Σ_1^0 – formulas of the kind $\exists \underline{i}.\phi(\underline{i},\mathbf{v})$
- Σ_2^0 – formulas of the kind $\exists \underline{i} \forall \underline{j}.\phi(\underline{i},\underline{j},\mathbf{v})$

[1] In all the formulæ we admit the term $a(t)$ only if t is a variable or a constant.

Acceleration
State of the art

Acceleration: Transitive closure τ^+ of transitions τ encoding cyclic actions

Acceleration

State of the art

Acceleration: Transitive closure τ^+ of transitions τ encoding cyclic actions

Challenges:

- In general transitive closure cannot be expressed in FOL

Acceleration: Transitive closure τ^+ of transitions τ encoding cyclic actions

Challenges:

In general transitive closure cannot be expressed in FOL

Only some (important) classes of τ's allow the definability of τ^+

- Polling-based systems [BBD$^+$02]
- Imperative programs over integers [BIK10]

Acceleration
State of the art

Acceleration: Transitive closure τ^+ of transitions τ encoding cyclic actions

Challenges:

In general transitive closure cannot be expressed in FOL

Only some (important) classes of τ's allow the definability of τ^+

- 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 Σ_2^0-assignments

Acceleration for arrays
Example

$$\tau_1 := pc = l_L \quad \wedge \quad \underbrace{\mathtt{i} < \mathtt{L} \wedge \mathtt{a}[\mathtt{i}] \neq \mathtt{e}}_{\text{guard}} \quad \wedge \quad \underbrace{\mathtt{i}' = \mathtt{i} + 1}_{\text{update}}$$

Acceleration for arrays

Example

$$\tau_1 := pc = l_L \quad \wedge \quad \underbrace{\mathtt{i} < \mathtt{L} \wedge \mathtt{a}[\mathtt{i}] \neq \mathtt{e}}_{\text{guard}} \quad \wedge \quad \underbrace{\mathtt{i}' = \mathtt{i} + 1}_{\text{update}}$$

$$\Downarrow$$

Acceleration for arrays

Example

$$\tau_1 := pc = l_L \quad \wedge \quad \underbrace{\mathtt{i} < \mathtt{L} \wedge \mathtt{a}[\mathtt{i}] \neq \mathtt{e}}_{\text{guard}} \quad \wedge \quad \underbrace{\mathtt{i}' = \mathtt{i} + 1}_{\text{update}}$$

$$\Downarrow$$

$$\tau_1^+ := \exists y. \begin{pmatrix} y > 0 \wedge pc = l_L \ \wedge \\ \forall j.(\ \mathtt{i} \leq j < \mathtt{i} + y \quad \rightarrow \quad j < \mathtt{L} \wedge \mathtt{a}[j] \neq \mathtt{e}\ ) \wedge \\ \mathtt{i}' = \mathtt{i} + y \end{pmatrix}$$

Acceleration for arrays
Example

$$\tau_1 := pc = l_L \quad \wedge \quad \underbrace{\mathtt{i} < \mathtt{L} \wedge \mathtt{a}[\mathtt{i}] \neq \mathtt{e}}_{\text{guard}} \quad \wedge \quad \underbrace{\mathtt{i}' = \mathtt{i} + 1}_{\text{update}}$$

$$\Downarrow$$

$$\tau_1^+ := \exists y. \begin{pmatrix} y > 0 \wedge pc = l_L \ \wedge \\ \forall j.(\ \mathtt{i} \leq j < \mathtt{i} + y \quad \rightarrow \quad j < \mathtt{L} \wedge \mathtt{a}[j] \neq \mathtt{e}\ ) \wedge \\ \mathtt{i}' = \mathtt{i} + y \end{pmatrix}$$

Acceleration for arrays

Example

$$\tau_1 := pc = l_L \quad \wedge \quad \underbrace{\mathtt{i} < \mathtt{L} \wedge \mathtt{a}[\mathtt{i}] \neq \mathtt{e}}_{\text{guard}} \quad \wedge \quad \underbrace{\mathtt{i}' = \mathtt{i} + 1}_{\text{update}}$$

$$\Downarrow$$

Number of iterations

$$\tau_1^+ := \exists y. \begin{pmatrix} y > 0 \wedge pc = l_L \ \wedge \\ \forall j.(\ \mathtt{i} \leq j < \mathtt{i} + y \ \rightarrow \ j < \mathtt{L} \wedge \mathtt{a}[j] \neq \mathtt{e}\ ) \wedge \\ \mathtt{i}' = \mathtt{i} + y \end{pmatrix}$$

Acceleration for arrays

Example

$$\tau_1 := pc = l_L \quad \wedge \quad \underbrace{\mathtt{i} < \mathtt{L} \wedge \mathtt{a}[\mathtt{i}] \neq \mathtt{e}}_{\text{guard}} \quad \wedge \quad \underbrace{\mathtt{i}' = \mathtt{i} + 1}_{\text{update}}$$

$$\Downarrow$$

Number of iterations

The guard is satisfied for all iterations

$$\tau_1^+ := \exists y. \begin{pmatrix} y > 0 \wedge pc = l_L \wedge \\ \forall j.(\ \mathtt{i} \leq j < \mathtt{i} + y \ \rightarrow \ j < \mathtt{L} \wedge \mathtt{a}[j] \neq \mathtt{e}\ ) \wedge \\ \mathtt{i}' = \mathtt{i} + y \end{pmatrix}$$

Acceleration for arrays

Example

$$\tau_1 := pc = l_L \quad \wedge \quad \underbrace{\mathtt{i} < \mathtt{L} \wedge \mathtt{a}[\mathtt{i}] \neq \mathtt{e}}_{\text{guard}} \quad \wedge \quad \underbrace{\mathtt{i}' = \mathtt{i} + 1}_{\text{update}}$$

$$\Downarrow$$

Number of iterations

The guard is satisfied for all iterations

$$\tau_1^+ := \exists y. \begin{pmatrix} y > 0 \wedge pc = l_L \ \wedge \\ \forall j.(\ \mathtt{i} \leq j < \mathtt{i} + y \ \ \rightarrow \ \ j < \mathtt{L} \wedge \mathtt{a}[j] \neq \mathtt{e}\ ) \wedge \\ \mathtt{i}' = \mathtt{i} + y \end{pmatrix}$$

Do the "jump"

Contribution

- ✘ Σ_2^0-formulæ over T may not admit decision procedures

Contribution

✘ Σ^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

✘ Σ_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 Σ_1^0-assignment
- every loop edge is labeled with a *basic-assignment*.

Contribution

✘ Σ_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 Σ_1^0-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_{init} to l_{error}

   $\Rightarrow$ Feasible iff the corresponding *Array Property formula* is satisfiable

Basic-Flat-Programs

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

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 Σ_2^0-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 Σ_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.

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 Σ_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

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 Σ_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?

References I

Francesco Alberti, Silvio Ghilardi, and Natasha Sharygina.
Definability of accelerated relations in a theory of arrays and its applications.
In *FroCos*, pages 23–39, 2013.

Gerd Behrmann, Johan Bengtsson, Alexandre David, Kim G. Larsen, Paul Pettersson, and Wang Yi.
UPPAAL implementation secrets.
In Werner Damm and Ernst-Rüdiger Olderog, editors, *FTRTFT*, volume 2469 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2002.

References II

Marius Bozga, Radu Iosif, and Filip Konecný.
Fast acceleration of ultimately periodic relations.
In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *CAV*, volume 6174 of *Lecture Notes in Computer Science*, pages 227–242. Springer, 2010.

Aaron R. Bradley, Zohar Manna, and Henny B. Sipma.
What's decidable about arrays?
In E. Allen Emerson and Kedar S. Namjoshi, editors, *VMCAI*, volume 3855 of *Lecture Notes in Computer Science*, pages 427–442. Springer, 2006.