Synthesizing Invariants for Arrays

Manuel Montenegro, Susana Nieva, Ricardo Peña, Clara Segura

Departamento de Sistemas Informáticos y Computación
Facultad de Informática, Universidad Complutense de Madrid

PROLE 2016
14-16 september 2016

Outline

Motivation

- Liquid types are a variant of dependent types which have been successfully used for automatically verifying a number of nontrivial properties of programs.
- A liquid type is a refinement of an ordinary type, defined by restricting the set of possible values to those satisfying a predicate, which may mention variables in scope and consequently the type may depend on the values computed by the program.
- They allow to express invariants of recursive data structures: sorted lists, binary search trees, binary heaps. The system assists the programmer in veryfing that the functions manipulating the data structure actually preserve the invariant.
- **Our contribution:** extending the liquid types to properties on arrays, which very frequently need predicates that are universally quantified over the array indices.

Liquid types

- Liquid types are Hindley-Milner types refined by predicates: $\{\nu : \tau \mid e\}$

$$\{\nu : int \mid \nu \geq 0\}$$

- The refinement predicate may mention program variables, so they are dependent types:

  `substract` $:: (x : \{\nu : int \mid \nu \geq 0\}) \rightarrow (y : \{\nu : int \mid \nu \leq x \wedge \nu \geq 0\}) \rightarrow \{\nu : int \mid \nu = x - y \wedge \nu \geq 0\}$

  `sort` $:: \forall \alpha.(a : array\ \alpha) \rightarrow \{\nu : array\ \alpha \mid (\forall i.0 \leq i < (len\ \nu) - 1 \rightarrow \nu[i] \leq \nu[i+1])\}$

- The rules of the liquid type system are an extension of standard typing rules with a subtyping relation between refined types:

$$\Gamma \vdash \tau_1 <: \tau_2$$

  which generates logical formulas:

$$\frac{[\![\Gamma]\!] \wedge e_1 \Rightarrow e_2 \text{ is valid}}{\Gamma \vdash \{\nu : B \mid e_1\} <: \{\nu : B \mid e_2\}}$$

  The formula is sent to an SMT solver in order to check its validity.

Inferring liquid types

- The inference algorithm assumes that a standard Hindley-Milner inference has been applied previously, then each type is refined with a fresh template variable κ:
$$\{\nu : \tau \mid \kappa\}$$
Inferring the types involves finding a mapping A from variables κ to predicates.
- Liquid type inference becomes decidable by:
  - restricting the boolean expressions to formulas in the logic of linear arithmetic, equality and uninterpreted functions (EUFA), and
  - bounding the search space of refinements with the help of logical qualifiers.

Logical qualifiers

- A logical qualifier is an atomic predicate which depends on the ν variable and a placeholder variable denoted by the $\star$ symbol e.g.
  $$\star \geq 0$$
  which may be instantiated by replacing the $\star$ by any (type appropriate) program variable e.g.
  $$x \geq 0$$
- The search space becomes finite by restricting the refinement predicates to conjunctions of instances of a set of qualifiers $\mathbb{Q}$ given by the programmer, e.g.
  $$\mathbb{Q} = \{\nu \geq 0, \nu \leq \star, \nu = \star - \star\}$$
  The instances are denoted by $\mathbb{Q}^*$.
- The inference algorithm looks for a mapping A from variables κ to conjunctions of $\mathbb{Q}^*$ such that, when applied to the typing derivation, the expression type checks.

Inference algorithm

- Initially all the template variables are mapped to $\bigwedge_{q \in \mathbb{Q}^*} q$, which is the strongest refinement. If it is a valid solution, the algorithm terminates.
- Otherwise, there is an assertion $[\![\Gamma]\!] \wedge A(\kappa_1) \Rightarrow A(\kappa_2)$ which is not valid. Remove from $A(\kappa_2)$ the qualifiers not being satisfied in the formula.
- With the new mapping repeat the process until a solution is found if there is any. Termination is guaranteed as $\mathbb{Q}^*$ is finite.