BRAIN: Backward Reachability Analysis with INtegers

Tatiana Rybina and Andrei Voronkov

University of Manchester
{rybina,voronkov}@cs.man.ac.uk

1 General description

BRAIN is a tool for analysis of infinite-state systems in which a state can be represented by a vector of integers. Currently it is used to verify safety and deadlock properties expressed as reachability statements in a quantifier-free language with variables ranging over natural numbers.

2 Input language

The input language of BRAIN consists of problem descriptions specifying the following.

1. The sets of initial and final states by a quantifier-free formula of Presburger arithmetic. Instead of final states, one can specify the set of deadlock states, which will be computed by BRAIN automatically from the description of transitions.
2. Transitions expressed as *guarded assignments* `guard -> assignment` (see [2]), where `guard` is again a quantifier-free formula of Presburger arithmetic and `assignment` is a sequence of assignments to variables. Examples will be given below.
3. Queries of two types: reachability of a set of states from another set of states, or reachability of a deadlock state from a set of states.

3 Transition systems

The semantics of a problem expressed in the language can be defined as follows. Let $\mathcal{V}$ be the set of all variables declared in the problem. We call a *state* a mapping from the set of variables into the corresponding domain (currently, only the domain of natural numbers is supported). Such a mapping can be homomorphically extended to arbitrary terms of Presburger arithmetic over $\mathcal{V}$ by defining $s(n) = n$, for every non-negative integer n and $s(u_1 + u_2) = s(u_1) + s(u_2)$. A *guard* G is a quantifier-free formula of Presburger arithmetic with variables in $\mathcal{V}$. In addition to the function symbol $+$ and equality BRAIN allows one to use $-$ and all the standard comparison operators, such as $\leq$. If a formula C of variables $\mathcal{V}$ is true in a state s, we write $s \models C$. Every such formula C *symbolically represents* a set of states, namely the set of states in which it is true, i.e., $\{s \mid s \models C\}$. A *transition* t is a set of pairs of states. Transitions are declared in BRAIN using *guarded assignments* of the form $G \to v_1 := u_1, \ldots, v_n := u_n$, where all the v_i's are variables and the u_i's are terms. This guarded assignment defines a (deterministic) transition t with the following properties: $(s, s') \in t$ if $s \models G$, $s'(v_1) = s(u_1), \ldots, s'(v_n) = s(u_n)$, and for every variable $v \in \mathcal{V} - \{v_1, \ldots, v_n\}$ we have $s'(v) = s(v)$. Though every guarded assignment specifies a deterministic transition, the transition relation specified by a set of guarded assignments may be non-deterministic.

4 The reachability algorithm

The backward reachability algorithm LocalBackward of BRAIN is implemented via a saturation algorithm given in Figure 1. This algorithm is taken from [3], where implementation of BRAIN is discussed in more detail. Backward reachability was chosen for a simple reason: symbolic application of a guarded assignment to a set of states represented by a formula $C(\mathcal{V})$ gives a quantified formula, while quantifiers arising from the backward application of a guarded assignment can be immediately eliminated.

```
procedure LocalBackward
input: quantifier-free formulas In, Fin,
       finite set of simple guarded assignments U
output: "reachable" or "unreachable"
begin
  IS := pdnf(In); FS := pdnf(Fin)
  if there exist I ∈ IS, F ∈ FS such that ⊨ ∃𝒱(I ∧ F) then return "reachable"
  unused := FS; used := ∅
  while unused ≠ ∅
    S := select(unused)
    used := used ∪ {S}; unused := unused − {S}
    forall u ∈ U
      (* backward application of u *)
      N := S^{-u}
      (* satisfiability-check for the new constraint N *)
      if ⊨ ∃𝒱(N) then
        (* intersection-checks *)
        if there exists I ∈ IS such that ⊨ ∃𝒱(N ∧ I) then return "reachable"
        (* entailment-checks *)
        if for all C ∈ used ∪ unused we have ⊭ ∀𝒱(N ⊃ C) then
          unused = unused ∪ {N}
          forall C′ ∈ used ∪ unused
            (* more entailment-checks *)
            if ⊨ ∀𝒱(C′ ⊃ N) then remove C′ from used or unused
  return "unreachable"
end
```

Fig. 1. Local backward reachability algorithm used in BRAIN

5 Algorithms over simple constraints

The backward reachability algorithm of BRAIN requires efficient algorithms for checking satisfiability and entailment of constraints. These algorithms are described in [3]. They are based on an algorithm [4] for building the basis of simple constraints (systems of linear equalities and inequalities over nonnegative integers) similar to the algorithm of [1]. The algorithm computes the basis of a constraint consisting of non-decomposable solutions. Incremental computation of the basis is very useful for checking intersection with the set of the initial states and also for deleting redundant inequalities or equalities.

6 Performance

Paper [3] contains experimental results comparing BRAIN with Action Language Verifier, HyTech and DMC. For integer-valued problems HyTech and DMC use relaxation, i.e., they solve real reachability problems instead of integer reachability problems. Therefore, they are correct only when they report non-reachability. Our results reported in [3] show that BRAIN is usually faster that HyTech and considerably faster than the other two systems, often by several orders of magnitude. BRAIN also consumes much less memory that HyTech and Action Language Verifier. There are several problems which could only be solved by BRAIN, but not by any of the other systems. On some of these benchmarks, BRAIN made about 10^9 entailment checks within the overall running time of about 15 minutes. However, we would like to note that all of these systems are on some benchmarks more powerful than BRAIN since they can use techniques such as widening or transitive closure which the current version of BRAIN does not have. The system is available at `http://www.cs.man.ac.uk/~voronkov/BRAIN/`.

References

1. F. Ajili and E. Contejean. Avoiding slack variables in the solving of linear diophantine equations and inequations. *Theoretical Computer Science*, 173(1):183–208, 1997.
2. T. Rybina and A. Voronkov. A logical reconstruction of reachability. In *Submitted to FME 2002*, 2002.
3. T. Rybina and A. Voronkov. Using canonical representations of solutions to speed up infinite-state model checking. In *Submitted to CAV 2002*, 2002.
4. A. Voronkov. An incremental algorithm for finding the basis of solutions to systems of linear Diophantine equations and inequations. unpublished, January 2002.