Proving Program Equivalence with Constrained Rewriting Induction and Ctrl

Carsten Fuhs
Birkbeck, University of London, UK

Cynthia Kop
Radboud University, The Netherlands

Naoki Nishida
Nagoya University, Japan

We prove equivalence of imperative programs by an automatic conversion of the functions in the input program to an equivalent *Logically Constrained Term Rewrite System (LCTRS)*, followed by constrained rewriting induction (either fully automatically or guided by the user) to analyze equivalence of the corresponding functions in the LCTRS. Our approach is implemented in the tool Ctrl.

Introduction. Total equivalence of two programs P and Q is the property that (a) both P and Q terminate on all inputs (termination) and (b) for the same inputs, the result of P is always the same as the result of Q (partial equivalence). For termination, one can use existing push-button provers (e.g., [1, 2, 4, 6]). How to prove property (b) is less obvious. In this abstract, we sketch an approach for proving partial equivalence of terminating imperative programs P and Q. Concretely, we show that calls to two different functions f_P and f_Q will lead to the same result for all possible inputs (possibly for some precondition).

We transform P and Q in an equivalence-preserving way to a *Logically Constrained Term Rewrite Systems (LCTRS)* [5]. On this LCTRS, we prove equivalence by rewriting induction [8] with support for logical constraints. We sketch the approach by example. An extended version of the present abstract with technical details, correctness proofs, generalization heuristics, further examples, and an experimental evaluation is available in the journal paper [3].

Motivating Example. Consider the following imperative program (in C syntax, where we consider `int` as *unbounded* integers).

```
int fibrec(int x) {
  if (x <= 0)
    return 0;
  else if (x == 1)
    return 1;
  else
    return fibrec(x-1) + fibrec(x-2);
}
```

```
int fibiter(int x) {
  int y = 0, z = 1, t;
  for (int i = 1; i <= x; i++) {
    t = y + z;
    y = z;
    z = t;
  }
  return y;
}
```

The program provides a recursive and an iterative implementation of the Fibonacci function on natural numbers (negative inputs are mapped to 0). Both functions are easily proved terminating. Now we want to prove that for all inputs, the functions `fibrec` and `fibiter` return the same results. The functions can be expressed equivalently by an LCTRS with the following constrained rewrite rules:

$$
\begin{array}{lrcll}
(1) & \mathsf{fibrec}(x) & \to & 0 & [x \leq 0] \\
(2) & \mathsf{fibrec}(1) & \to & 1 & \\
(3) & \mathsf{fibrec}(x) & \to & \mathsf{plus}(\mathsf{fibrec}(x-1), \mathsf{fibrec}(x-2)) & [x \geq 2] \\
(4) & \mathsf{plus}(\mathsf{return}(x), \mathsf{return}(y)) & \to & \mathsf{return}(x+y) &
\end{array}
$$

Submitted to:
PERR 2019

$$\begin{array}{lrcll}
(5) & \mathsf{fibiter}(x) & \to & \mathsf{iter}(x,1,0,1) & \\
(6) & \mathsf{iter}(x,i,y,z) & \to & \mathsf{iter}(x,i+1,z,y+z) & [x \geq i] \\
(7) & \mathsf{iter}(x,i,y,z) & \to & \mathsf{return}(y) & [x < i]
\end{array}$$

Our equivalence claim can be expressed by the equation $\mathsf{fibrec}(x) \approx \mathsf{fibiter}(x)$ $[\mathsf{true}]$. An attempt to prove that this equation holds for all $x \in \mathbb{Z}$ results in a divergence with more and more proof obligations:

$$\begin{array}{lcll}
\mathsf{iter}(n,3,1,2) & \approx & \mathsf{plus}(\mathsf{iter}(m,\mathsf{iter}(m,2,1,1)),\mathsf{iter}(k,\mathsf{iter}(k,1,0,1))) & [m = n-1 \wedge k = n-2] \\
\mathsf{iter}(n,4,2,3) & \approx & \mathsf{plus}(\mathsf{iter}(m,\mathsf{iter}(m,3,1,2)),\mathsf{iter}(k,\mathsf{iter}(k,2,1,1))) & [m = n-1 \wedge k = n-2] \\
\mathsf{iter}(n,5,3,5) & \approx & \mathsf{plus}(\mathsf{iter}(m,\mathsf{iter}(m,4,2,3)),\mathsf{iter}(k,\mathsf{iter}(k,3,1,2))) & [m = n-1 \wedge k = n-2]
\end{array}$$

The key to a successful proof is our generalization technique [3] which abstracts variable initializations:

$$\begin{array}{c}
\mathsf{iter}(n_3,i_3,z_3,z_4) \approx \mathsf{plus}(\mathsf{iter}(n_2,i_2,z_2,z_3),\mathsf{iter}(n_1,i_1,z_1,z_2)) \\
[n_2 = n_3 - 1 \wedge n_1 = n_2 - 2 \wedge i_3 = i_2 + 1 \wedge i_2 = i_1 + 1 \wedge z_3 = z_1 + z_2 \wedge z_4 = z_2 + z_3]
\end{array}$$

This equation can be shown using constrained rewriting induction. This implies equivalence of `fibrec` and `fibiter`. Thus, we can prove equivalence of functions with wildly different time complexities: `fibrec`'s running time is exponential in the input value, whereas that of `fibiter` is linear.

Another interesting example where our approach automatically proves equivalence has a function to sum up all numbers from 0 to n via a `for`-loop (linear running time) and a function that immediately returns $n(n+1)/2$ (constant running time). Here, equivalence is proved for all non-negative inputs n.

Implementation. In addition to reasoning about integer arithmetic, our implementation Ctrl [6] provides automation for reasoning about arrays. Ctrl also performs preprocessing to simplify the input LCTRSs. Reasoning about the underlying constraint theory is performed by the SMT solver Z3 [7]. Ctrl can use a theory that a user specifies by means of SMT-LIB logics such as bitvectors, and thus, we will apply our approach to more practical programs, e.g., automotive embedded systems written as C programs with structures and unions.

References

[1] Marc Brockschmidt, Byron Cook, Samin Ishtiaq, Heidy Khlaaf & Nir Piterman (2016): *T2: Temporal Property Verification*. In: *TACAS '16*, pp. 387–393.

[2] Stephan Falke, Deepak Kapur & Carsten Sinz (2011): *Termination Analysis of C Programs Using Compiler Intermediate Languages*. In: *RTA '11*, pp. 41–50.

[3] Carsten Fuhs, Cynthia Kop & Naoki Nishida (2017): *Verifying Procedural Programs via Constrained Rewriting Induction*. *ACM Trans. Comput. Log.* 18(2), pp. 14:1–14:50.

[4] Jürgen Giesl, Cornelius Aschermann, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Jera Hensel, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Stephanie Swiderski & René Thiemann (2017): *Analyzing Program Termination and Complexity Automatically with AProVE*. *J. Aut. Reasoning* 58(1), pp. 3–31.

[5] Cynthia Kop & Naoki Nishida (2013): *Term Rewriting with Logical Constraints*. In: *FroCoS '13*, pp. 343–358.

[6] Cynthia Kop & Naoki Nishida (2015): *Constrained Term Rewriting tooL*. In: *LPAR '15*, pp. 549–557. Tool available at `http://cl-informatik.uibk.ac.at/software/ctrl/`.

[7] Leonardo de Moura & Nikolaj Bjørner (2008): *Z3: An Efficient SMT Solver*. In: *TACAS '08*, pp. 337–340.

[8] Uday S. Reddy (1990): *Term Rewriting Induction*. In: *CADE '90*, pp. 162–177.