



Verification of Programs with Inputs

Henrich Lauko, Vladimír Štill, Petr Ročkal, Jan Mrázek and Jiří Barnat

DIVINE

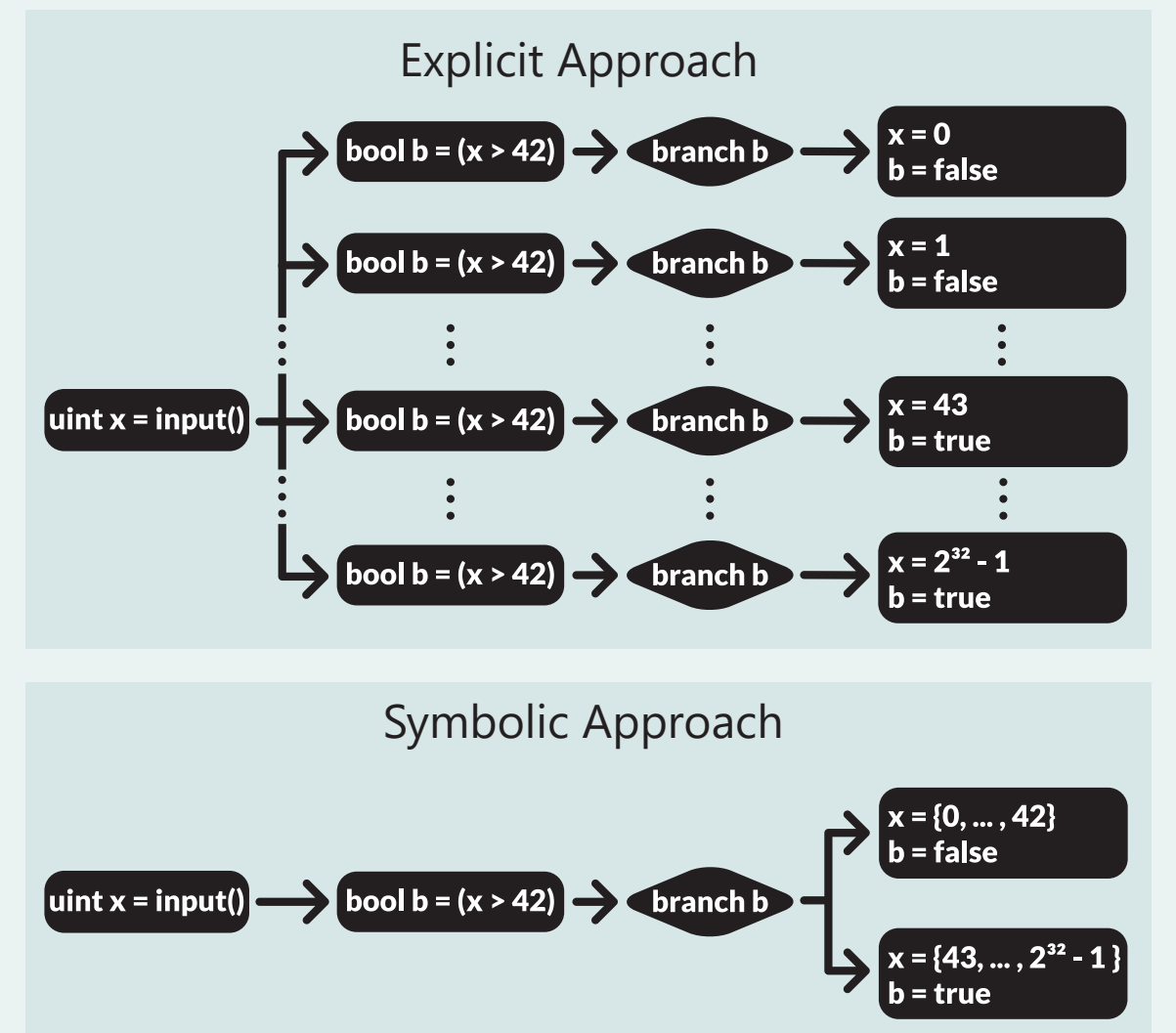


DIVINE is a tool for verification of parallel C++ programs. By using the LLVM compilation framework with the Clang compiler and the libc++ library it provides support for most of the standard C++ library and all the C++ language features. DIVINE is rather efficient when dealing with

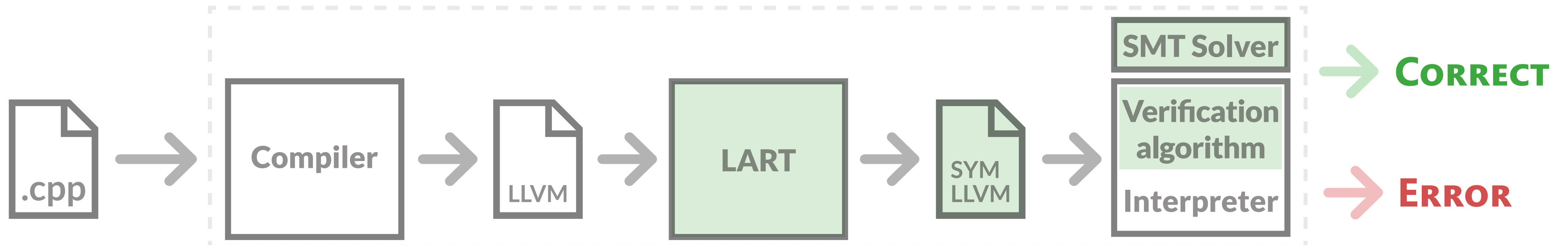
programs without inputs (for example test cases). A big downside of the current version of DIVINE is that for programs with inputs, this input has to be simulated by nondeterministic choice which is very inefficient. Therefore we present an approach for symbolic representation of inputs in DIVINE.

Symbolic States

Consider a simple program with 32 bit input variable x and a branch on the value of this variable. In the current DIVINE, this program gives rise to 2^{32} possible memory configurations. In symbolic version, possible values of variables x and b are represented symbolically using bitvector formulae, therefore, there are only two possible configurations at the end of the program.



Proposed Approach



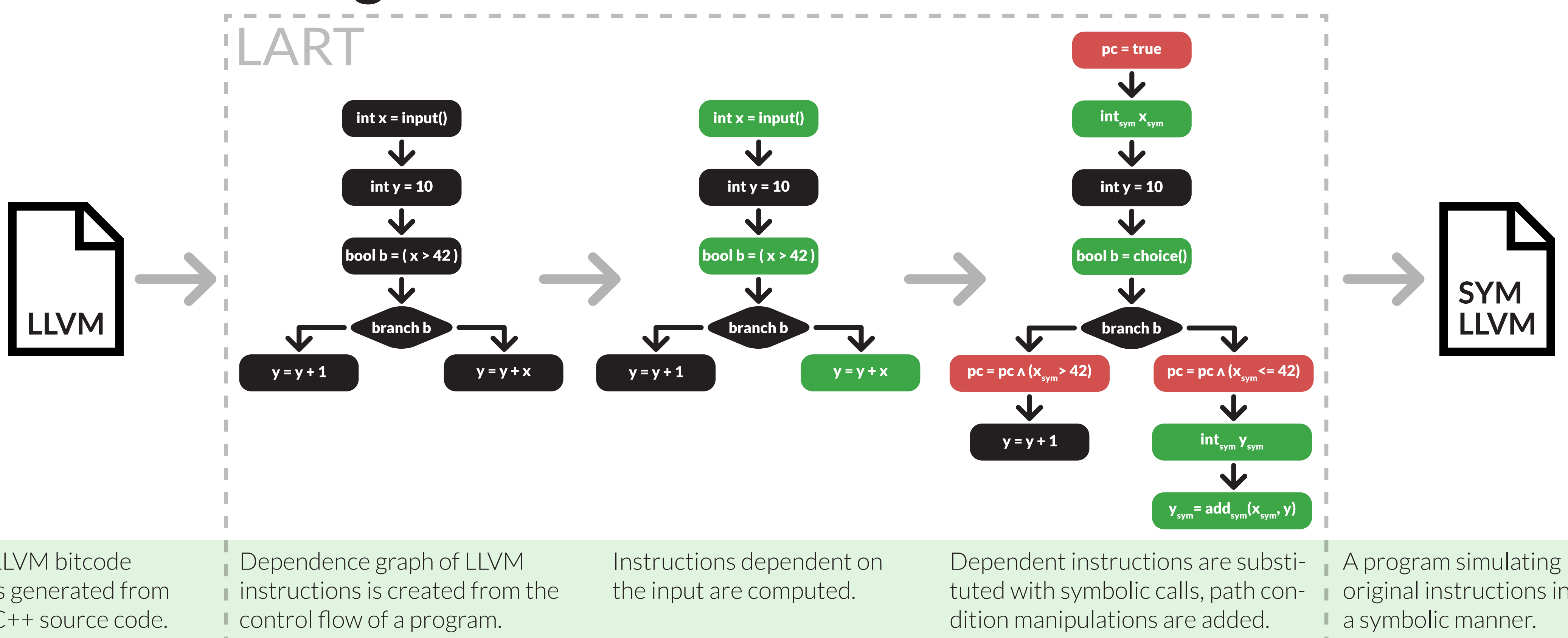
To take advantage of symbolic representation of states, we transform the LLVM bitcode in such a way that it represents variables which can contain values dependent on inputs symbolically. This transformation is performed by LART and is resented

in detail later. Apart from that, the verification algorithm is modified to handle symbolic states with the help of an SMT solver.

Our approach aims for minimizing changes to the LLVM interpreter that is used to execute instructions in DIVINE. The reason is that the interpreter is complex and performance tuned and therefore it is not desirable to make it even more complex by adding symbolic data manipulation into it. Instead, symbolic data are to be handled by the program itself. To encode symbolic manipulations into the program we transform the LLVM bitcode produced by the compiler and create symbolic LLVM from it. This not only minimizes changes to the interpreter, but

the transformation can also be used for different representation of symbolic data quite easily. The transformation is handled by LART – LLVM Abstraction & Refinement Tool. Furthermore, DIVINE's verification algorithm has to be modified. It has to check if symbolic states are valid (nonempty), that is if they can represent at least one concrete state. It also has to handle comparison of symbolic states. For both of these tasks, DIVINE has to extract SMT formulae from the program state and use SMT solver.

Details of Program Transformation



LART takes the LLVM bitcode of the program and libraries produced by the compiler and transforms it into a bitcode which manipulates data symbolically. In this modified program, any variable which can depend on an input value is represented symbolically using bitvector formulae. Bitvector formulae describe integers of fixed bit width with overflow and bitwise operations, and therefore

are well suited for exact representation of computer integers. All the manipulations with such variables have to be transformed to their symbolic versions which modify the formulae accordingly. Furthermore, any branch which depends on an input value has to put constraints on the possible values of symbolic variables (this constraint is given in the form of a path condition formula).