Synthesis of Proven-to-be Correct Programs

Eva Maria Wagner, TU WIEN

acm's women in computing
acm-w Europe

Deductive Program Synthesis

Correctness of a program according to a given specification can be verified using an *automated theorem prover* (ATP), such as Vampire [1]. Although this process produces highly reliable outputs, it requires the program to be written manually. Deductive program synthesis involves constructing the program as part of the automation process [2, 3]. In other words, provide an ATP with only the specification as input and let it simultaneously construct the correct program and prove the specification. This not only reduces the need for the extra manual step of writing the code but also avoids inaccurate implementations (see Figure 1).

Saturation

ATPs use saturation as the main principle of proof searching. To prove that a statement is valid, the ATP searches for a contradiction in the negated input ('proof by contradiction'). The negated input is first transformed into a set of clauses. The ATP applies a set of inference rules exhaustively to generate further clauses. If the empty clause □ is derived, the proof search terminates. From a logical viewpoint, the empty clause is the sought-for contradiction (see Figure 2).

Specification Format

Specifications express the requirements of programs that are to be synthesised. Different synthesis approaches use different specification formats. ATP inputs are expressed in *first-order logic*. For synthesis, in particular, the specification has the form ∀x∈S∃y∈T.F[x,y] (1). Variable x represents the input of a program p ("for any input x of type S"); the variable y represents the output of program p ("there is an output y of type T") and the formula F expresses program p's requirement or task ("it holds that F[x,y]"). On top of proving (1) the goal of an ATP is to construct a *computable term* p(.) such that ∀x∈S.F[x,p(x)] is valid. This proof serves as certification of correctness and p(x) can be translated into code of any programming language.

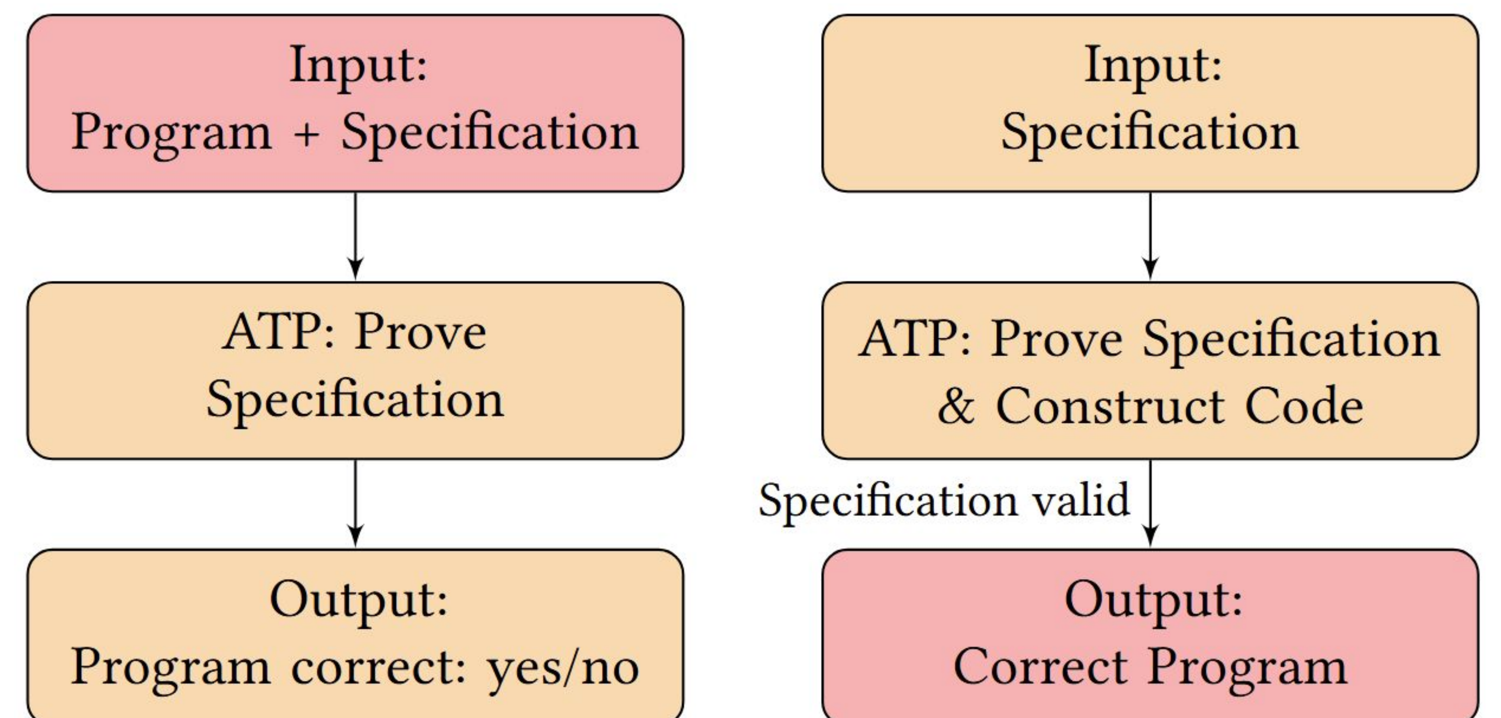

Fig. 1: Program Verification (left) & Synthesis (right)

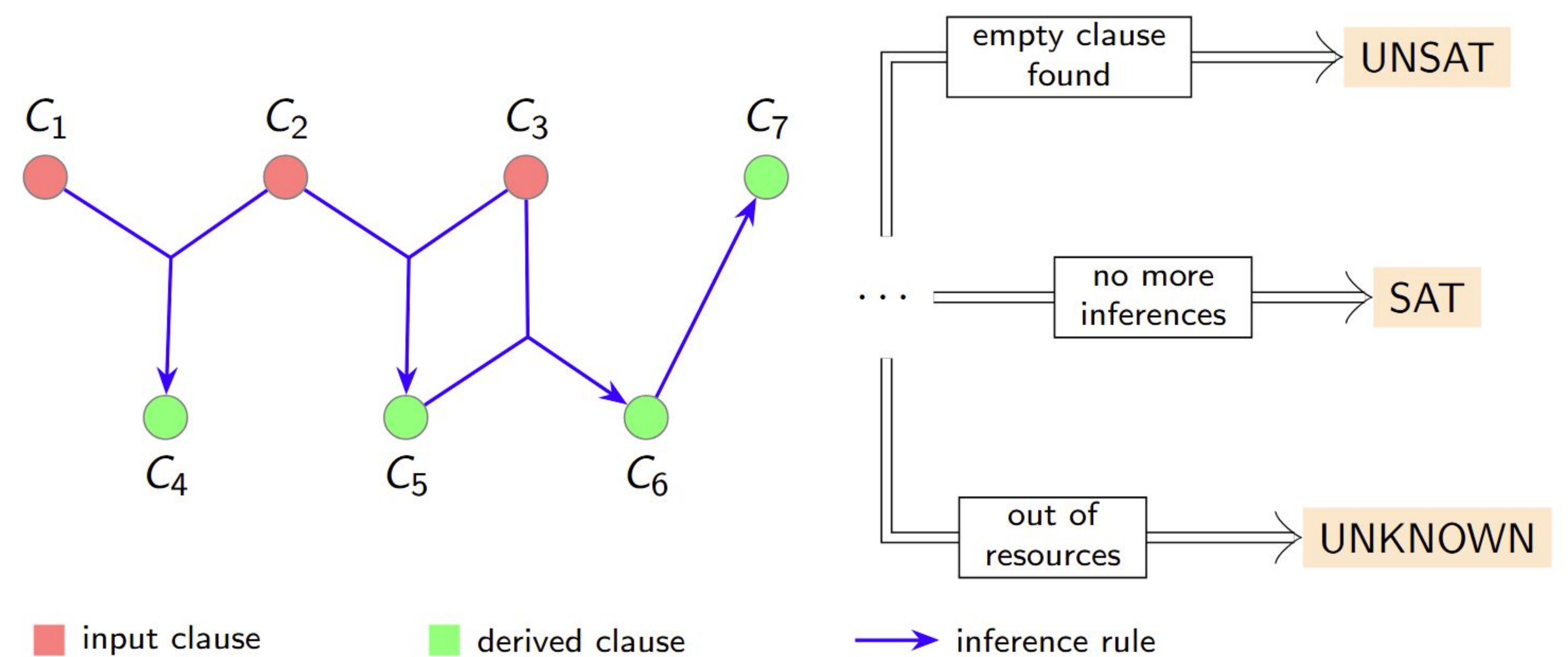

Fig. 2: Saturation Process

Specification (Maximum of two numbers): $\forall x_1, x_2 \in \mathbb{Z}\, \exists y \in \mathbb{Z}.\ x_1 \leq y \wedge x_2 \leq y.$
Axioms: $\forall z_1, z_2 \in \mathbb{Z}.(z_1 \leq z_2 \vee z_1 \neq z_2)$ $(A1)$, $\forall z_1, z_2 \in \mathbb{Z}.(z_1 \leq z_2 \vee z_2 \leq z_1)$ $(A2)$.

1. $\neg x_1^{sk} \leq y \vee \neg x_2^{sk} \leq y \vee ans(y)$ [input]
2. $x_1^{sk} \neq y \vee \neg x_2^{sk} \leq y \vee ans(y)$ [BR 1, A1]
3. $\neg x_2^{sk} \leq x_1^{sk} \vee ans(x_1^{sk})$ [ER 2]
4. $\neg x_1^{sk} \leq y \vee x_2^{sk} \neq y \vee ans(y)$ [BR 1, A1]
5. $\neg x_1^{sk} \leq x_2^{sk} \vee ans(x_2^{sk})$ [ER 4]
6. $x_2^{sk} \leq x_1^{sk} \vee ans(x_2^{sk})$ [BR 5, A2]
7. $ans(\texttt{if } x_2^{sk} \leq x_1^{sk} \texttt{ then } x_1^{sk} \texttt{ else } x_2^{sk})$ [BR' 3, 6]
8. $\square$ [answer literal removal]

Fig. 3: Example of Proof Derivation with Answer Literals

Answer Literals

Figure 3 shows an example derivation of computing the maximum of two integers. There are two integer-type input variables, x1 and x2. The clauses are numbered from 1 to 8. Inference rules applied to clauses are written in the square brackets on the left. During preprocessing, an answer literal containing the output variable y is added to the input clause. Answer literals track the behaviour of the to-be-synthesised program at certain proof steps. Once the proof has terminated the answer literal in clause 7 contains the desired program: an if-then-else branch that checks whether x1 is greater than x2, determining whether to return x1 or x2.

References

1. Laura Kovács and Andrei Voronkov. 2013. First-Order Theorem Proving and Vampire. In CAV. 1–35.
2. Hozzová, P., Kovács, L., Norman, C., Voronkov, A.: Program Synthesis in Saturation. In: CADE. pp. 307–324 (2023).
3. Hozzová, P., Amrollahi, D., Hajdu, M., Kovács, L., Voronkov, A., Wagner, E.M.: Synthesis of recursive programs in saturation. In: Benzmüller, C., Heule, M.J., Schmidt, R.A. (eds.) Automated Reasoning. pp. 154–171. Springer Nature Switzerland, Cham (2024).

12th ACM Celebration of Women in Computing: womENcourage™
Brașov, Romania
17-19 September, 2025
Theme: Computer Science: a Catalyst for Educational Change