A major challenge in software verification is choosing a suitable abstraction which yields a tractable yet still precise enough program model. However, automatically picking a suitable abstraction requires non-trivial insights and is usually implemented using heuristics. Though heuristics determine the efficiency of most verification tools, they are typically not formally described and hard-coded in the source code of tools. Furthermore, heuristics tight a verification tool to a restricted application domain. This causes another verification challenge – an optimal choice of a verification tool for a given task.

We conjecture that the two challenges, namely automatically choosing program-specific abstraction and automatically picking an optimal tool for a verification task, can be solved by identifying in a program typical patterns of variable use. Examples of these patterns are bitvectors, counters, loop iterators, and so on. We formalise and study the concept of a variable role, which captures the implicit knowledge about these patterns.

We identify most frequent variable roles in practical open source programs and define a framework for a formal specification of variable roles using Datalog. We explore the application of variable roles in software verification in two settings.

First, we create a portfolio solver which chooses a tool for a given verification task, using variable-role-based program metrics. We evaluate our
portfolio solver in the setting of the software verification competition SV-COMP.

Second, we define a framework to systematically specify variable-role-based heuristics to automatically generate program abstraction. We integrate our method in the model checker Eldarica to generate predicates for initial abstraction and to guide abstraction refinement through templates provided for Craig interpolation. We evaluate our approach on a subset of benchmarks from SV-COMP and verification benchmarks from the literature.