Karlsruher Institut für Technologie

Towards a Proof View at the Phase Trasnition between Autoactive and Interactive Verification

Sarah Grebing, Mattias Ulbrich

Institute for Theoretical Informatics, KIT

Motivation

Program verification is an iterative process.

- most verification attempts fail (at first)
- successful proof requires to find the reason for failure and correction of mistake/providing right information

For each iteration it is necessary to inspect different projections of the problem into different domains.

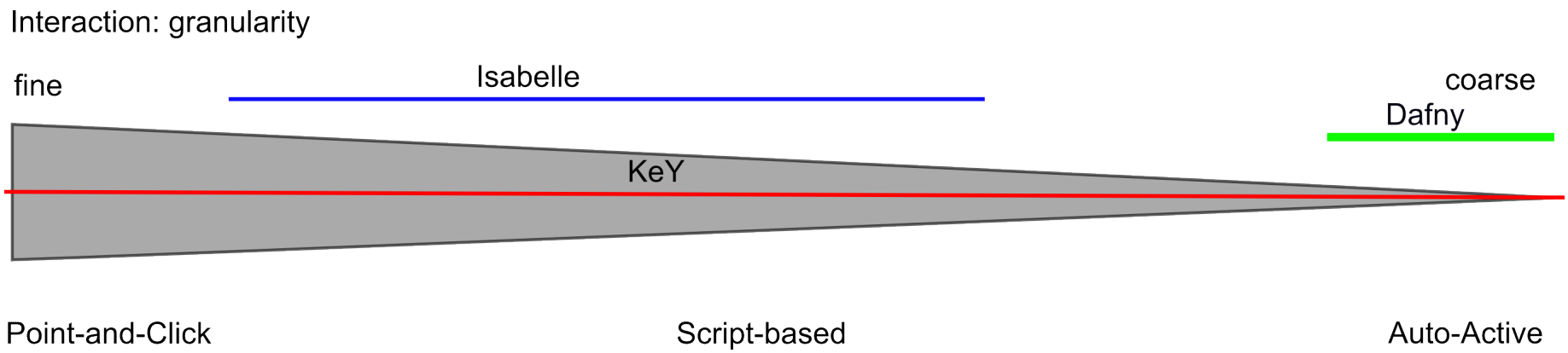
- Program and its specification
- logical encoding of the proof obligations
- counter examples
- ...

Switching between projections (if they exist) is costly and not well supported in the process.

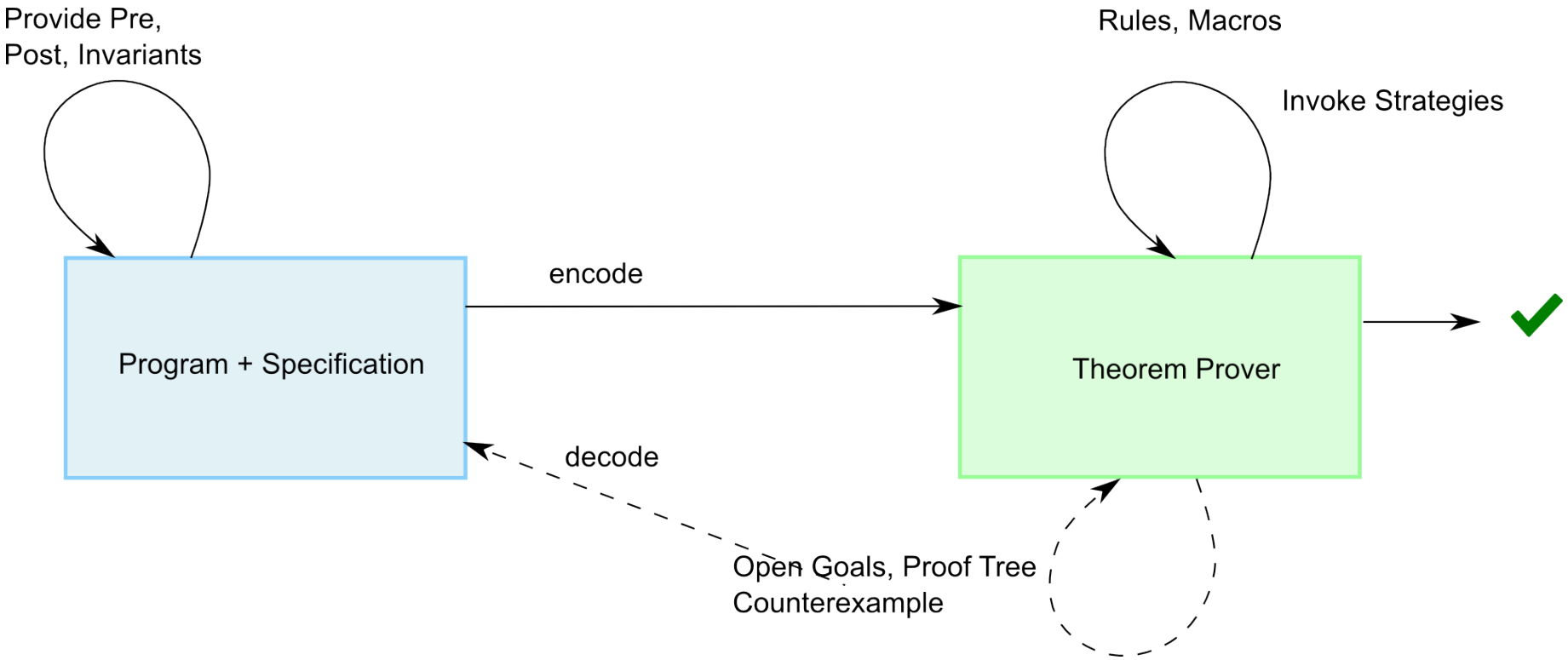
State-of-the-Art Verification Systems

Three interaction principles for guiding the proof search have emerged:

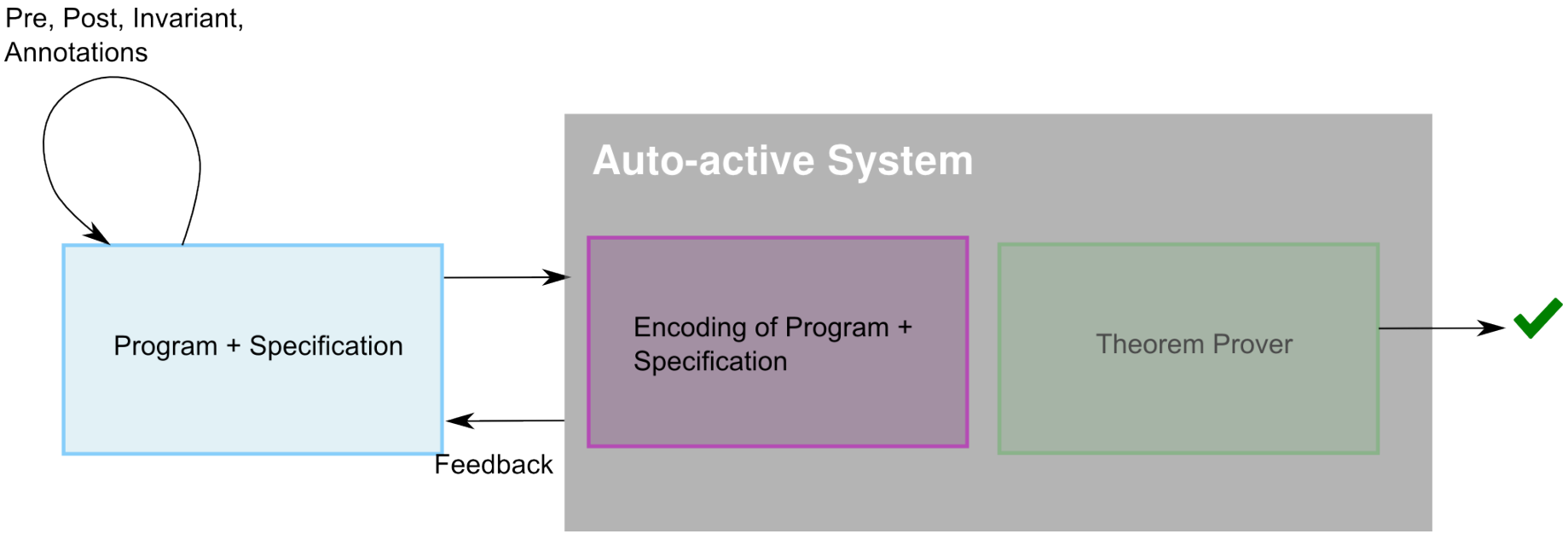
- autoactive
- point-and-click style
- script-based



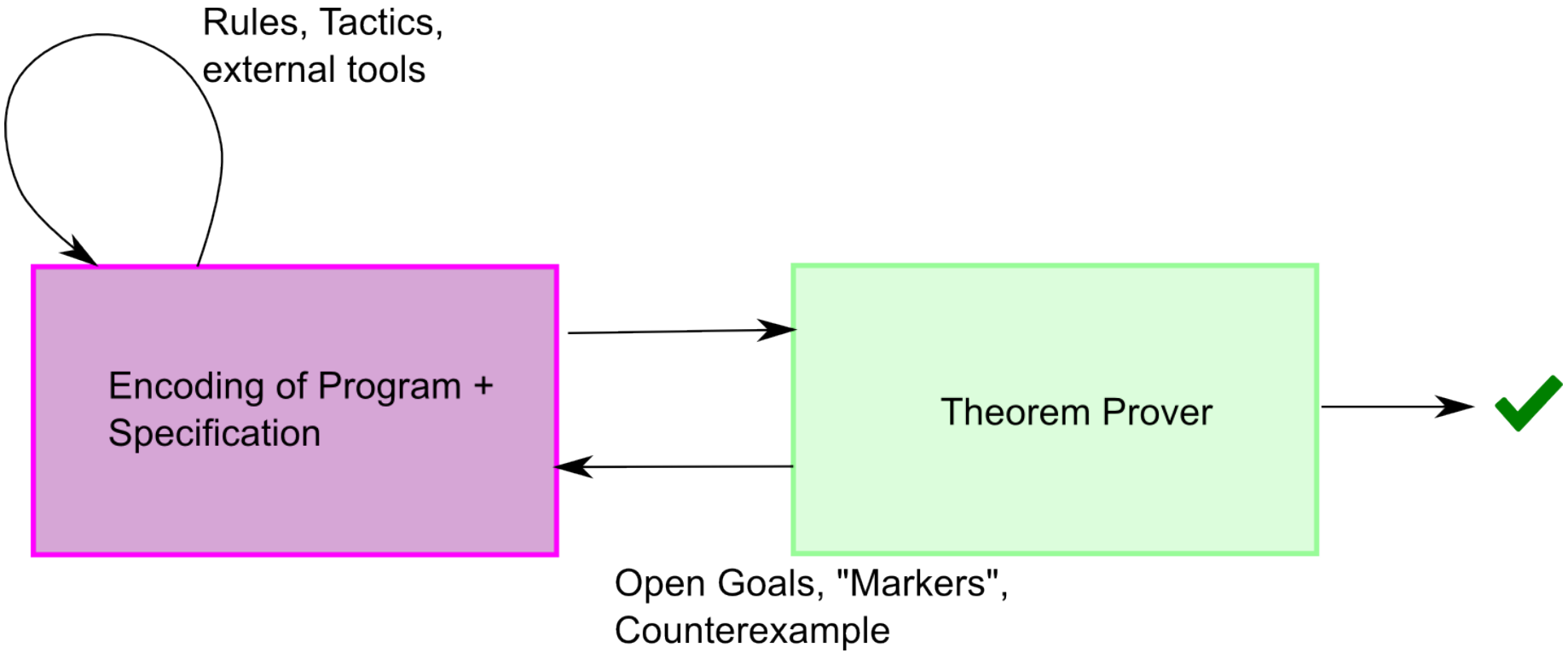
Interactive: Point-and-Click



Autoactive


Pre, Post, Invariant,
Annotations
Program + Specification
Feedback
Auto-active System
Encoding of Program +
Specification
Theorem Prover

Text-Based Interaction



Interactive: Point-and-Click

Level of Interaction (proof guidance)

Problem input on program level, proof guidance on logical level (two mental models)

Level of Feedback

On logical level by showing the proof tree with open goals and sequent, macro (auto-pilot)

User support during proof construction

automatic heuristics, macros, counter examples, symbolic execution tree

Pros and Cons

- +/- all necessary information available
- + full proof control
- - interaction can be tedious
- - error recovery
- - two mental models have to be kept in sync

Auto-Active

Level of Interaction (proof guidance)

Program level using annotations and programing constructs

Level of Feedback

Program level using squiggly red lines, state inspection and counter example, path in program, hover text with useful information

User support during proof construction

inference of simple annotations, feedback on program level, useful hints in hover text of feedback, fast error recovery

Pros and Cons

- + interaction on input representation $\rightarrow$ only one mental model
- + comprehensible annotations
- - missing detailed insight into logical level when proof attempt fails

Text/Script-Based interaction

Level of Interaction (proof guidance)

Problem input on program level, proof guidance on logical level (two mental models)

Level of Feedback

On logical level by showing open goals, Nitpick, Sledgehammer

User support during proof construction

tactics, counter examples, *sorry* to postpone proof tasks and concentrate on one problem at a time

Pros and Cons

- + problem/proof decomposition in smaller parts
- + interaction steps more coarse grained than point-and-click $\rightarrow$ more readable/comprehensible and proof plan expressible
- +/- limited insight into logical level
- - two mental models may have to be kept in sync

Seamless Integration

Overall Goal

Interaction concept supporting interactive, semi-automated proof guidance for an effective and efficient proof process.

- support the user in the decision why a proof attempt failed
- support the user in developing and enhancing proof

Decrease Costs

Allow for interaction on suitable abstraction level (problem and user dependent)

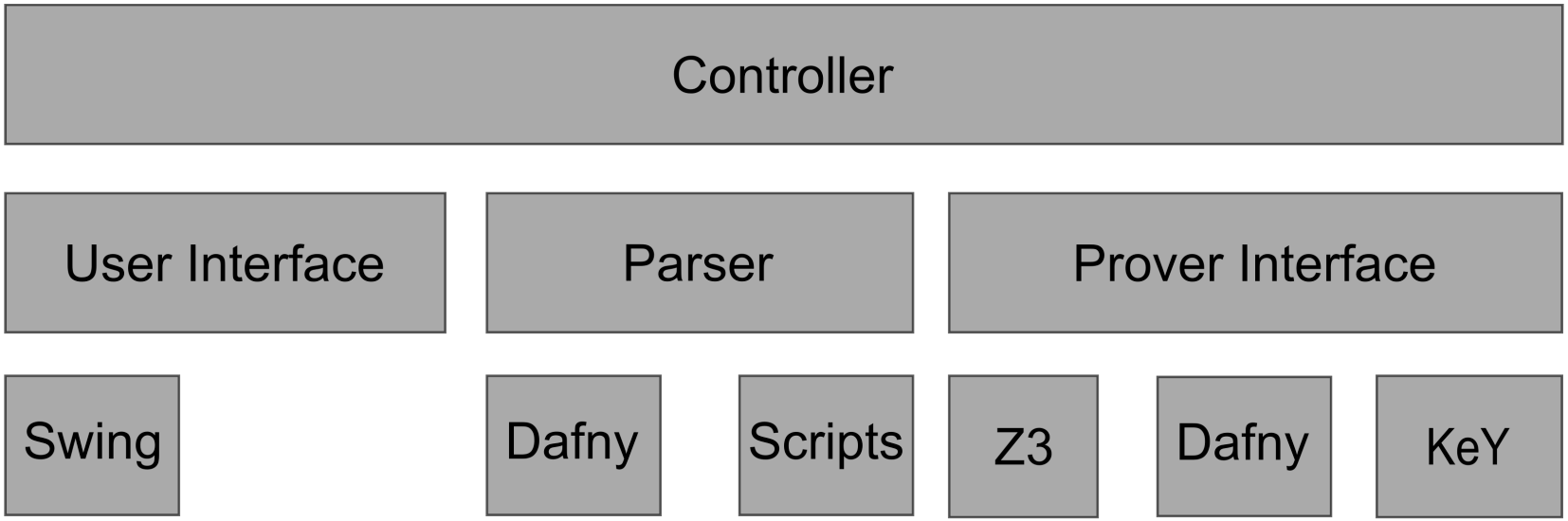
Decrease amount of changes

- Stay on input language as long as possible (input and feedback)
- Allow for coarse proof steps (proof plan)
- Give clear overview over overall proof state

Decrease cost per iteration

- If necessary allow for state inspection on logical level
- Making dependencies visible between levels
- Allow for proof exploration on logical level

Architecture



Example

Discussion

From your experience (especially using KeY) what takes more time and why?

- finding cause of failed proof?
- convince KeY, that proof can be found?