Seamless Interaction Concept for Interactive Program Verification

Sarah Grebing, Philipp Krüger, Mattias Ulbrich

Sarah Grebing, Philipp Krüger, Mattias Ulbrich

Institute for Theoretical Informatics, KIT

21. Juni 2019

Interaction in Interactive Program Verification

Interaction on:

- different levels of abstraction for interaction
- different representations of the same problem

Switch between levels and/or representations is necessary.

Involved Entities in Interactive Program Verification

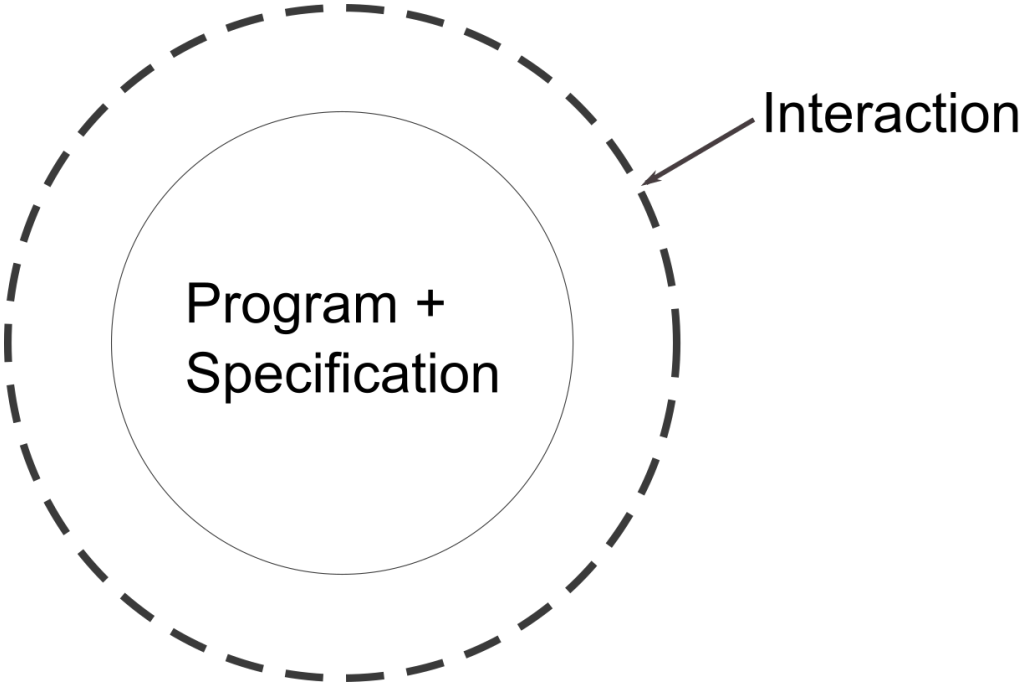
- program code
- specification
- proof representation/proof obligation
- *proof guidance/interaction*

Examples for State-of-the-Art Systems

Three different kinds of Interaction Concepts:

- auto-active
- point-and-click
- text-based

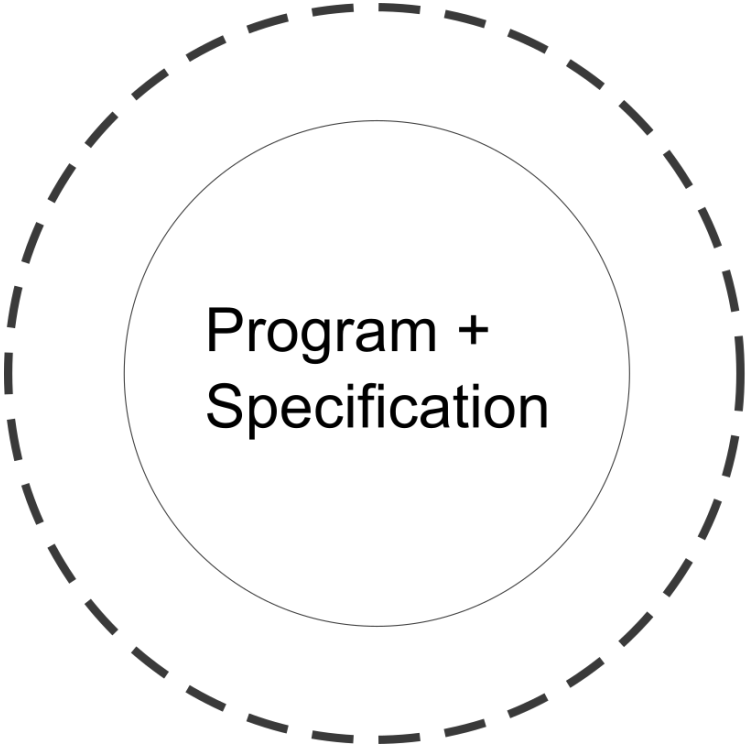
Auto-active



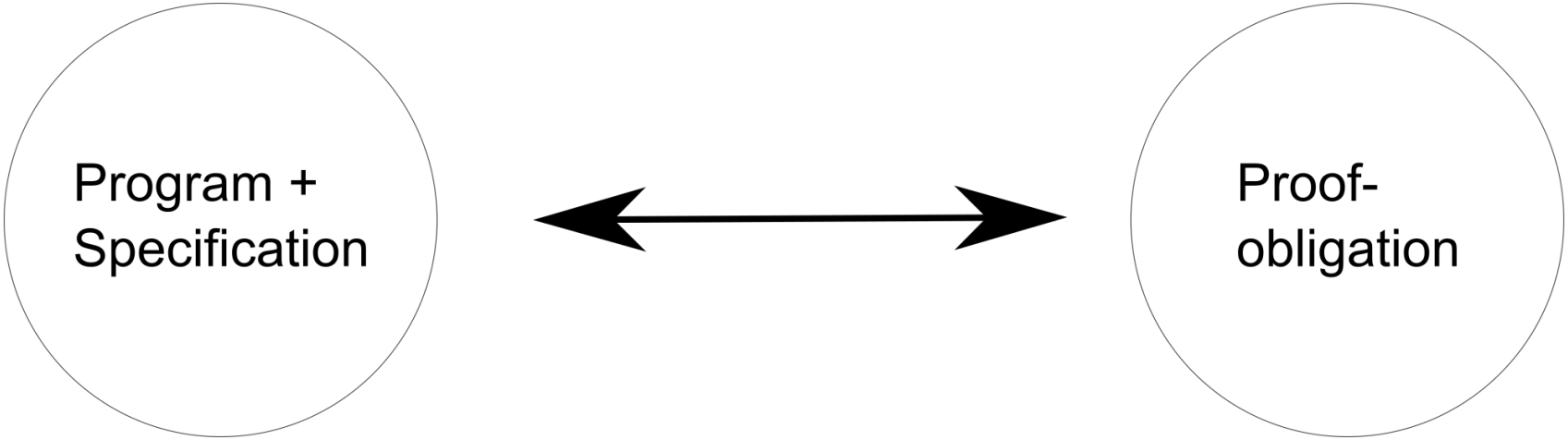
Point-and-Click

Point-and-Click

Point-and-Click

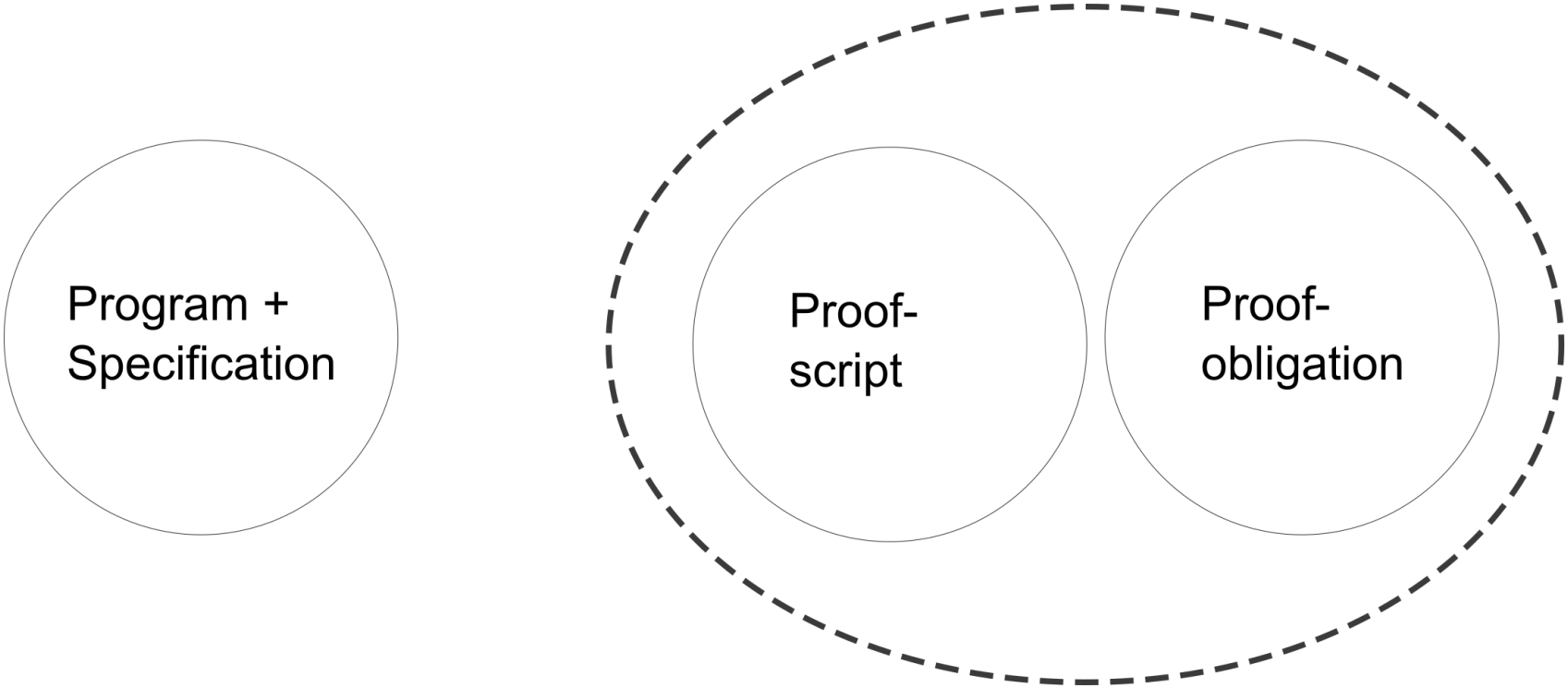


Point-and-Click



Text-based

Text-based



Problems with Interaction in State-of-the-Art Systems

- interaction on different representations
- hidden dependencies between representations
- context change cognitively challenging for the user
- missing interaction possibilities on representations

Goal of our concept

An interactive program verification system that allows implementing and researching different interaction concepts:

- integration of different representations as views
- integration of different interaction concepts
- seamless transition between views

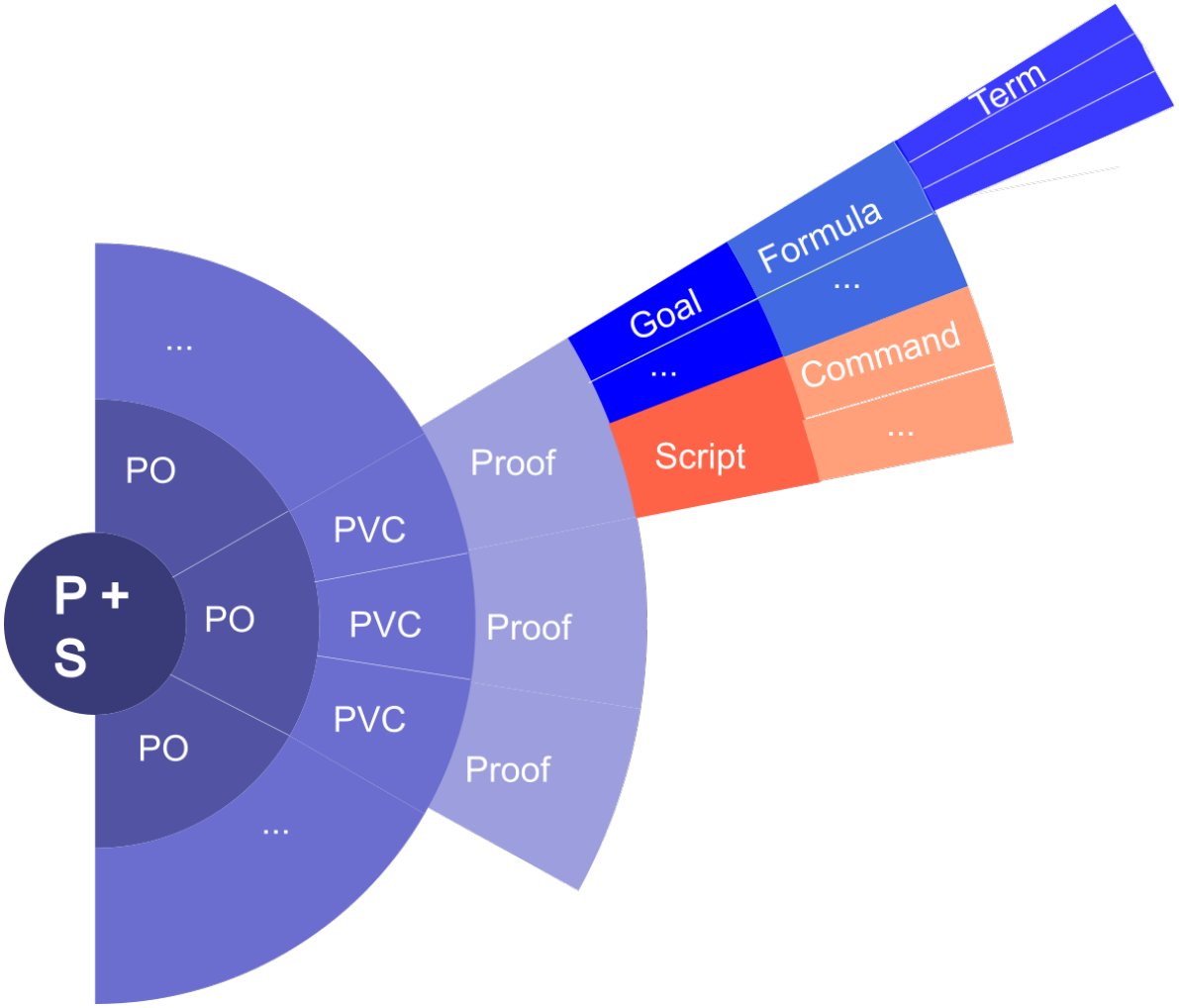
Hypothesis

User interaction needs, depending on the context,

(a) a focussed view on specific elements or

(b) an overview of the bigger picture.

Structure of the Program Verification Problem



Objectives

The user is ...

1. ... able to use appropriate view at all times
2. ... can easily switch views without loosing focus
3. ... is able to determine the results of costly actions before executing them

Objective 1: Appropriate View at All Times

The user needs different views (problem and user dependent)

(a) overview over the whole system state (global system state, proof state, remaining proof tasks, ...)

(b) tailor-made views that focus on a specific part of the proof problem (e.g., single proof verification condition)

Our concept integrates all views into one system and allows suitable interaction on the views.

Objective 2: Easy switch of Views

For the user: Switching view is switching context.
$\Rightarrow$ requires cognitive resources

Goal: Reduce cognitive resources for context switch by supporting fluent switches between views:

- show similar things in proximity to each other (e.g., adjacent elements of problem structure)
- show effect of user interaction in all visible views to keep track of changes
- show dependencies between entities to make hidden dependencies clear

Objective 3: Determine Action Results

Show the user different variant of the *future* (principle of least surprise).

- allow light-weight tools to discharge simple proof obligations ($\Rightarrow$ let the user concentrate on hard tasks)
- reduce unrecoverable errors by showing action results in context
- integration of proof exploration techniques

Demo

name	status
Classes	✔ 0/0
▼ Methods	12/22
▼ method max	12/22
max/InitInv[inbounds]	✔ Edit
max/InitInv[greater]	! Edit
max/InitInv[witness]	✔ Edit
max/InitInv[witness_in_bo...	✔ Edit
max/loop/else/Inv[inbounds]	✔ Edit
max/loop/else/Inv[greater]	! Edit
max/loop/else/Inv[witness]	! Edit
max/loop/else/Inv[witness...	! Edit
max/loop/else/Var	! Edit
max/loop/then/Inv[inbounds]	✔ Edit
max/loop/then/Inv[greater]	! Edit
max/loop/then/Inv[witness]	! Edit
max/loop/then/Inv[witness...	✔ Edit

arrayMax.dfy

```
01 method max(a : array<int>) returns (m : int)
02   requires a.Length > 0
03   ensures label greater: (forall i:int :: 0 <= i && i < a.Leng
04   ensures label witness: (exists i:int :: 0 <= i && i < a.Leng
05 {
06   var i:int := 0;
07   m := 0;
08   label mainLoop: while i < a.Length
09     invariant label inbounds: 0 <= i && i <= a.Length
10     invariant label greater: (forall j:int :: 0 <= j && j < i
11     invariant label witness: m == 0 || (exists j:int :: 0 <= j
12     invariant label witness_in_bounds: 0 <= m && m < a.Length
13     decreases a.Length - i
14   {
15     if (a[i] > a[m])
16     {
17       m := i;
18     }
19     i := i+1;
20   }
21 }
22
23
24
```