# Auto-Grading for Elements of Haskell Programming

Johannes Waldmann, HTWK Leipzig

29. Sept 2026

# Algebraic Data Types: Completeness of Patterms

```
For these data declarations
  data Q = F P | D
  data P = C Q Q | E P P

prove that this set  A  of patterns
  [ E u v, E u (C v w), C D D ]
is not complete for the type P

by giving a term  t  of type P
that is not an instance of any pattern in  A.
```

# Completeness and Disjointness of Patterms

```
For these data declarations
  data Q = F P | D
  data P = C Q Q | E P P
and set  A  of patterns
  [ E u v, E u (C v w), C D D ]

... give a set  B  of patterns such that
each pattern in B is disjoint to each
other pattern in B, and to each pattern in A,
and  (A union B)  is complete for the type P
```

# autotool Overview

- Waldmann et al. 2000– `https://git.imn.htwk-leipzig.de/waldmann/autotool`
  automatic grading of student homework in (theoretical) computer science
- semantics back-end (stateless) contains, for each exercise type, grader and generator. JSON-RPC (servant)
- web front-end (learn management system, for courses, problems, results, leaderboards) yesod, persistent
- recent additions (semantics)
  - fill holes in formal proofs: cyp (simplified Isabelle), Agda
  - exercises on pattern matching and strictness (this talk)
- ongoing work (infrastructure) (following talks)
  - LTI interface (use autotool from moodle etc.)
  - agentic usage for LLM auto-grader

# Instance Generator

- for previous example exercise

```
Config
  { num_data_decls = 2
  , num_constructors = 4
  , max_arity = 2
  , num_patterns = 3
  , height_patterns = 2
  , candidates =  101
  }
```

# Size of Finite Algebraic Data Type – Monomorphic

```
Fill holes (_) (replace with one name or expression)
in the following,
so that the claim at the top becomes true.

For this exercise, each data declaration
can only refer to types defined earlier -
not to itself or later (it cannot be recursive).

size_of (E) == 8 where
{ data Bool =  False | True
; data H =  D _
; data E =  G H | B H _ | A _
}
```

# Size of Finite Algebraic Data Type – Polymorphic

```
Fill holes (_) (replace with one name or expression)
in the following, so that the claim at the top becomes

For this exercise, each data declaration can only refe
defined earlier - not to itself or later (it cannot be

size_of (_ (Maybe (Maybe C))) == 9 where
{ data Unit =  Unit
; data Maybe a =  Nothing | Just _
; data E u = D u _
; data C = G Unit
}
```

# Prove Non-Strictness of Function

```
For this set of declarations
{ g p y =
    case (f y True True) of
      { False -> False; True -> f True p p }
; f q p z =
    case (case False of
      { False -> z; True -> p }) of
      { False -> z; True -> q }
}
Prove that function g is not strict:
Write a call where at least one argument is  undefined
such that the result is defined
(it has a head normal form with a constructor on top)
and will be obtained after at most 40 steps.
```

# Implement a strict and non-strict specification

```
Find a replacement expression
for the hole (_) in the program
  { f z y = g (g False z) True; g y x = _ }
such that evaluations of the following expressions
to head normal form gives these results:
  [ ( f True undefined, Defined )
  , ( f undefined False, Bottom )
  ]
```

# Discussion

- exercises illustrate concepts of programming
  but not by plain "write a program that ..."
- use Haskell semantics, but sometimes re-implement (for better tracing, error messages)
- can an LLM guess the answers? Maybe.
  - homework: any help (human, tool) is allowed anyway.
    Selected homework needs to be presented (explained) in class as Prüfungsvorleistung
  - exam: autotool in classroom (supervised, locked down network)