Projects and technical details

Johan Jeuring

Joint work with Alex Gerdes and Bastiaan Heeren

Utrecht University and Open Universiteit Nederland
Computer Science

CEFP, Budapest, Hungary

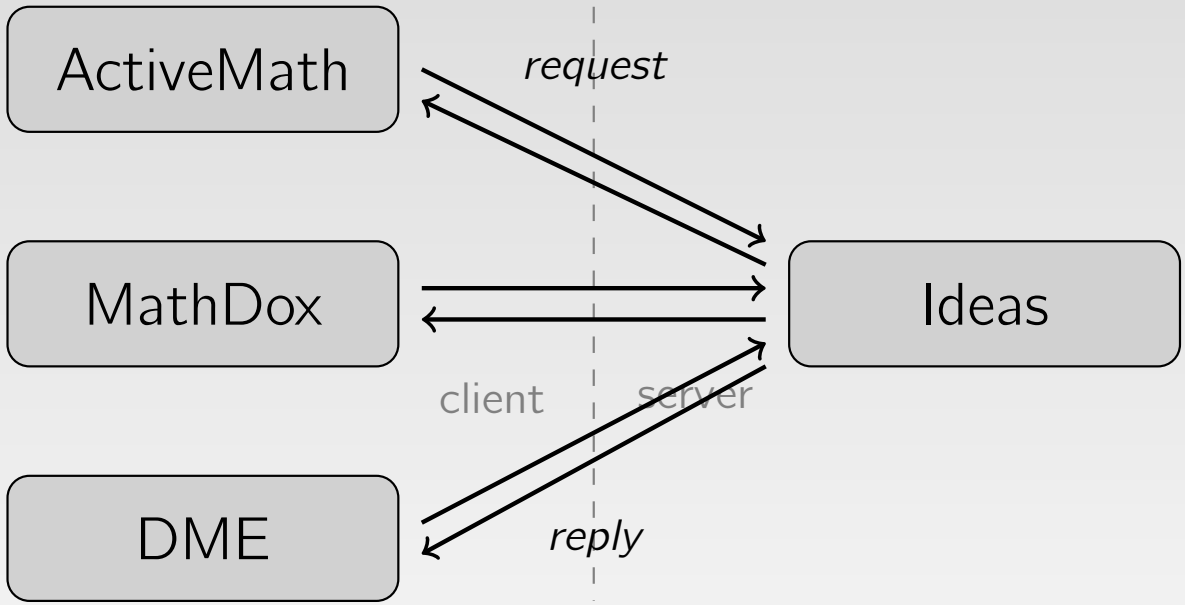
June 2011

Open Universiteit
www.ou.nl

Introduction

- We have developed several exercises and projects related to our programming tutor
- For some of these, you need to adapt the software
- This lecture gives an overview of the software, and a brief introduction to the projects.

Feedback services



A hint request

```
<request service    = "onefirst"
         exerciseid = "math.lineq"
         encoding   = "string">
  <state>
  <expr>5*(x-1) == x/2</expr>
  </state>
</request>
```

A response

```
<reply result="ok" version="0.7.4 (0)">
  <elem ruleid="algebra.equations.linear.remove-div"
        location="[]">
    <argument description="factor">
      2
    </argument>
    <state>
      <prefix>
        [4,1,0]
      </prefix>
      <expr>
        10*(x-1) == x
      </expr>
    </state>
  </elem>
</reply>
```

Services

allfirsts.	all next steps that are allowed by a strategy
onefirst.	a single possible next step that follows a strategy. You can specify an order on steps, to select a single step among multiple possible steps
derivation.	a worked-out solution starting with the current program
ready.	is the program accepted as a final answer?
diagnose.	diagnoses a program submitted by a student

An examples exercise package

```
dnfExercise :: Exercise SLogic
dnfExercise = makeExercise
    { parser          = parseLogicPars
    , prettyPrinter   = ppLogicPars
    , equivalence     = withoutContext eqLogic
    , similarity      = withoutContext equalLogicA
    , isReady         = isDNF
    , strategy        = dnfStrategyDWA
    , navigation      = navigator
    , extraRules      = map liftToContext
                            (extraLogicRules ++ buggyRules)
    , randomExercise = useGenerator (const True)
                                    logicExercise
    ...
    }
```

Exercise

```
data Exercise a = Exercise
    { parser          :: String -> Either String a
    , prettyPrinter   :: a -> String
        -- syntactic and semantic checks
    , equivalence     :: Context a -> Context a -> Bool
    , similarity      :: Context a -> Context a -> Bool
    , ordering        :: a -> a -> Ordering
    , isReady         :: a -> Bool
        -- strategies and rules
    , strategy        :: LabeledStrategy (Context a)
    , navigation      :: a -> Navigator a
    , extraRules      :: [Rule (Context a)]
        -- testing and exercise generation
    , examples        :: [(Difficulty, a)]
    ...
    }
```

Structure of the software

The code for the programming tutor is built on top of our Ideas framework.

- `FPTutor/trunk/src/` describes the functional programming domain
- `Feedback/trunk/src` describes the Ideas framework

The Ideas framework

The Ideas framework consists of a number of directories, of which the following three are most interesting:

- `Common`: General machinery dealing with rewriting, strategies, contexts, exercises, etc.
- `Domain`: Domain reasoner instances dealing with several domaims such as linear algebra, logic, math, etc.
- `Service`: Implementing the various services we offer

The FPTutor framework

The FPTutor framework consists of a number of directories, of which these are the most interesting:

- `src/Domain/FP`: Domain reasoner instance for the programming domain
- `src/Domain/FP/Transformations`: Program transformations for normalisation
- `src/WebApp`: Web application implementation (HTML and JavaScript)
- `models`: directory containing model solutions
- `scripts`: directory with feedback scripts

Exercises, projects, slides, and notes

We have made all our (textual) material available on

`http://people.cs.uu.nl/johanj/homepage/Publications/CEFP/`

- Exercises: `exercises.pdf`
- Slides:
  - `slides1.pdf`: Introduction, overview, tutors, strategies
  - `slides2.pdf`: The strategy language
  - `slides3.pdf`: A strategy recogniser
  - `slides4.pdf`: Brief overview of the ideas framework. Introduction to the exercises/project work
- Lecture notes: `notes.pdf`

Software

Experiment on-line:

`http://ideas.cs.uu.nl/ProgTutor/`

Build the tutor on your own machine:

`http://ideas.cs.uu.nl/trac/wiki/Download`

Project 1: Adapting feedback I

A teacher should be able to add her own feedback to a model solution.

> $reverse = foldl$ {-# FEEDBACK Note ... #-} $(flip\ (:))\ []$

and it should be possible to disallow or enforce particular solutions described by a strategy:

> $reverse =$ {-# USEDEF #-} $foldl\ (flip\ (:))\ []$

Implement these ideas for adapting strategies.

Project 1: Adapting feedback II

We might want to add a property to a function, and use that in a strategy:

```
reverse =
    {-# PROP foldl op e == foldr (flip op) e . reverse #-}
  foldl (flip (:)) []
```

Inspiration for desirable properties can be obtained from the file `data/Default.hs` in the HLint distribution:

```
error  = zipWith (,) ==> zip
error  = foldr (&&) True ==> and
error  = (\x -> x) ==> id
```

Implement these ideas for adapting strategies.

Project 2: Automatic contract checking

We want the student's definition $reverse = <?>$ to satisfy the function contract:

> $(x : true) \to \{y \mid y \equiv reverse\ x\}$

for some model solution of *reverse*. If a student refines with $<?> \Rightarrow foldl\ <?_1>\ <?_2>$, this holds if both

> $assert\ ((x : true) \to (y : true) \to \{z \mid z \equiv flip\ (:)\ x\ y\}\ <?_1>$
> $assert\ (\equiv [])\ <?_2>$

Strategies (and normalisation) help in constructing such refinement (proof) steps.

Investigate if we can use contracts for blaming incorrect steps.

Conclusions

The Ideas framework is a sizable application, mainly written in Haskell

We welcome contributors to the framework, test users, etc.

All feedback is welcome

- `http://ideas.cs.uu.nl/`
- johanj@cs.uu.nl
- alex.gerdes@ou.nl