Mathematics flagship

Mathematical Grammar Library

Contributors: *Jordi Saludes, Ares Ribó, Sebastian Xambó, Olga Caprotti, Aarne Ranta, Krasimir Angelov, Ramona Enach, Adam Slaski, Thomas Hallgren, Shafqat Mumtaz Virk*

- **Goal**:
  - To provide mathematics in natural language in many natural languages.
  - Including interfacing with math software.

Cloud services for mathematics

- Contributors: *Kaarel Kaljurand, Thomas Hallgren, Aarne Ranta, Jordi Saludes*
- The cloud services for mathematics provide linearization and parsing of mathematical text in many languages, as supported by the *Mathematical Grammar Library*.
- Math bar
- Multilingual semantic wiki on mathematics
- Exploitation scenarios: e.g. a browser plugin that, upon highlighting, translates the mathematical text, sends it to a computational engine, prompts the user for some decision, explains some symbol, searches in a mathematically meaningful way.

Multilingual semantic wiki

- Contributors: *Kaarel Kaljurand, Olga Caprotti, Jordi Saludes, Aarne Ranta, Ares Ribó*
- Running on ACE wiki here.
- Mixed formula/natural lenguage rendering of mathematics.
- Support for *qualified variables* in quantifiers:
  - for all *prime natural numbers n* and m, ...
- Aimed at expressing theorem statements and exercises.

Features

- A predictive parser allows a contributor to construct a well-formed statement or exercise.
- In English, Italian or Spanish.
- Automatically rendered in the reader language.
- A contributor is aware of ambiguous interpretations.

Example

- Translations

Sentence | Translations |

Let $ x $ be a prime number. Then for all integer numbers $ y $ and $ z $, if x divides the product of y and z then x divides y or x divides z. *- Translations*

English

Let $ x $ be a prime number. Then for all integer numbers $ y $ and $ z $, if x divides the product of y and z then x divides y or x divides z.

Español

Sea $ x $ un número primo. Entonces para todos números enteros $ y $ y $ z $, si x divide al producto de y y z entonces x divide a y o x divide a z.

Italiano

Sia $ x $ un numero primo. Allora per tutti numeri interi $ y $ e $ z $, se x divide al prodotto di y e z allora x divide ad y o x divide a z.

- Ambiguities
  - Let x be a prime number. Then for all integer numbers y and z, if x divides the product of y and z then x divides y or x divides z.
  - It has 3 different meanings. The wiki shows the tree structure of all of them.

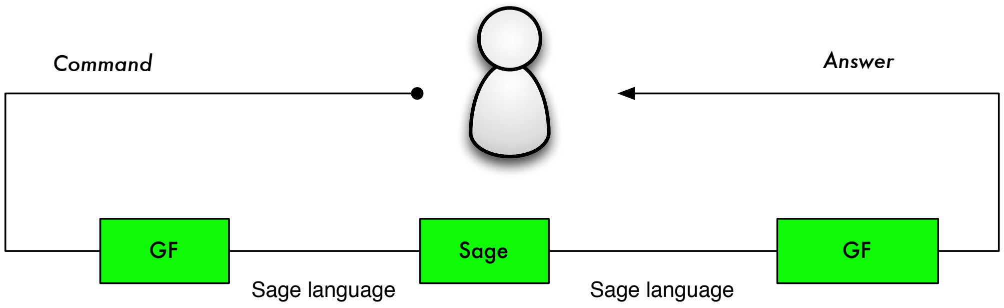
Multi-modal input/output of mathematics

- Contributors: *Ares Ribó, Jordi Saludes, Sebastian Xambó*
- **Goal**: Querying a Computer Algebra System by natural language.

Interfacing Sage

- A command line tool for computing using natural language and aural replies.
  - Screencast for basic usage
  - Screencast showing feedback
- An embedded interface in the Sage notebook. Using natural language in a Sage cell by prefixing it with expressions like `%english`.

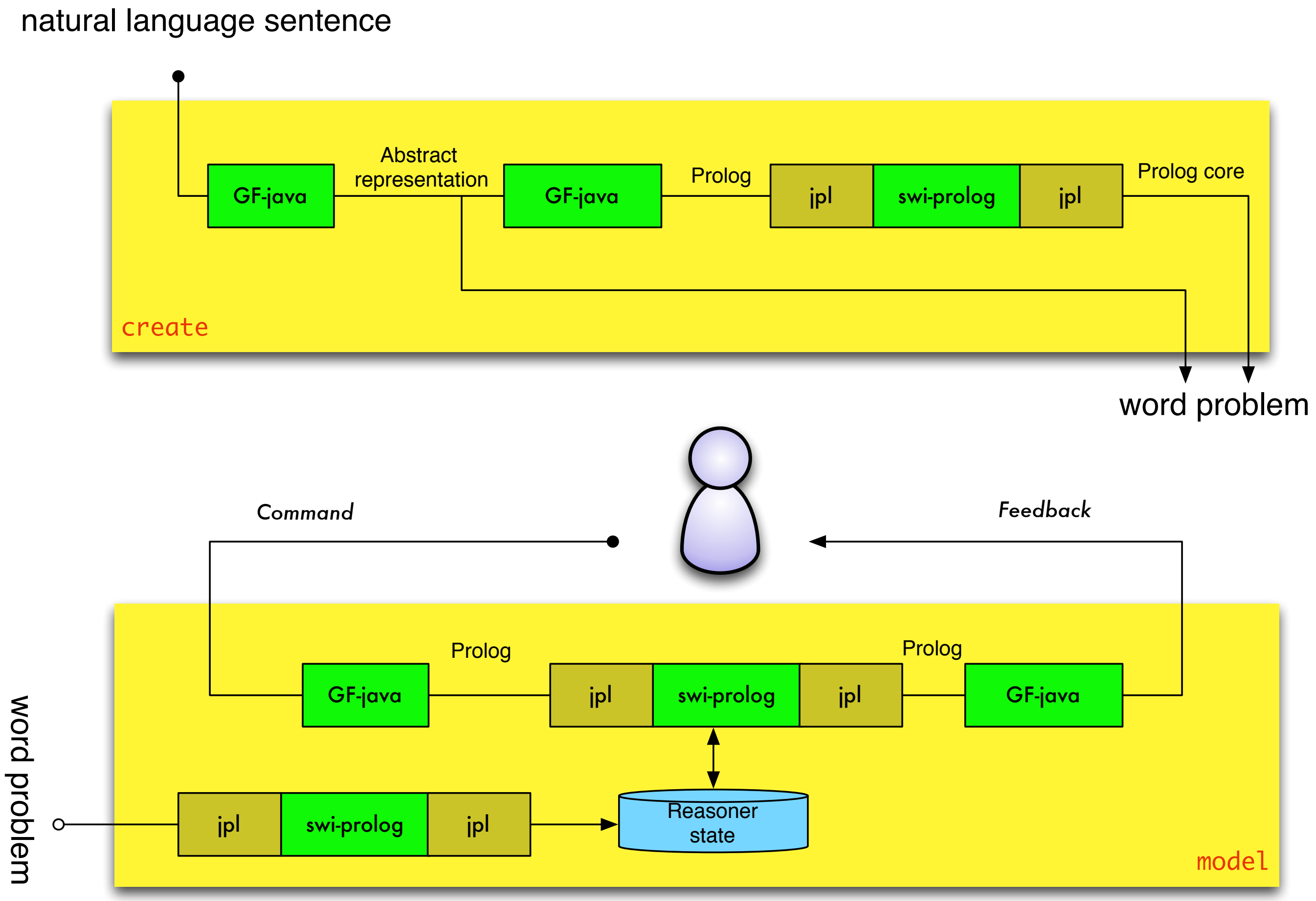
Sage interface workflow



Word problem solver dialog system

- Contributors: *Ares Ribó, Olga Caprotti, Thomas Hallgren, Aarne Ranta, Jordi Saludes*
- Tools for:
  - Writing a problem.
  - Screencast: Learning to model.

Workflow


natural language sentence
GF-java
Abstract representation
GF-java
Prolog
jpl
swi-prolog
jpl
Prolog core
create
word problem
Command
Feedback
GF-java
Prolog
jpl
swi-prolog
jpl
Prolog
GF-java
word problem
jpl
swi-prolog
jpl
Reasoner state
model

Word problem schemata

The current prototype allows to state word problems of the following form:

- **Human** (has *or* owns) (**numeral** *or* some) **class**
- how many **class** does **Human** (have *or* own)?

  where **class** is: *animals*, *rabbits*, *cows*, *fruits*, *apples*, etc. and **Human**: *John* or *Mary*.
- in the languages: English, Swedish, Spanish and Catalan.
- Classes and objects can be extended by adding entries to the `WPEntities` module.

Setting the problem model

We considered two levels of discourse:

- The *plain* language is for direct communication with the user, natural language;
- The *core* language is for the reasoner to work with.

The fact that **John has seven fruit** is represented by the GF tree:

```
fromProp (E1owns john (gen Fruit n7))
```

and in Prolog by:

```
own(john, 7 * fruit)
```

while in *core*, it converts to the Prolog assertion:

```
p(X, fruit, own(john, X)) - 7 * unit(fruit)).
```

The latter is more suited to reasoning with it.

Normalization

Conjunctions are *disaggregated*: namely

- **John has three apples and six bananas**

is converted into:

- **John has three apples**
- **John has six bananas**

Questions make the *unknown explicit* in the *core* expressions:

- **how many apples does Mary have?**

is represented in *plain* as:

```
find(own(mary,apple))
```

and in *core* as:

```
find(X, apple, own(mary,X))
```

Example

```
% abs:fromProp (E1owns john (gen Fruit n7))
% Eng:John has seven fruit .
p(X, fruit, own(john, X)) - 7 * unit(fruit).

% abs:fromProp (E1owns john (aplus (ConsAmount (gen Apple n2)
%              (BaseAmount (some Orange) (gen Banana n3)))))
% Eng:John has two apples , some oranges and three bananas .
p(Y, apple, own(john, Y)) - 2 * unit(apple).
p(Z, banana, own(john, Z)) - 3 * unit(banana).
p(V, orange, own(john, V)) - some(orange).

% abs:fromQuestion (Q1owns john Orange)
% Eng:how many oranges does John have ?
find(W, orange, own(john, W)).
```

Solving a problem

To process a given word problem, the student must construct a set of of statements in *core language* that model that given word problem.

The authoring interpreter saves a word problem in a Prolog file consisting of:

- A GF abstract tree for the plain sentence of a problem. This is written as a Prolog comment.
- Core statements in Prolog format that correspond to the plain expression.

Modeling a problem

When the dialog interpreter is started on a word problem file, the system uses the GF abstract lines to display the statement of the problem in the selected language.

Next, the student must go through a sequence of steps to have the problem correctly modeled by:

1. assigning variables to given data and to unknowns
2. discovering relations among the given data
3. stating equations involving these unknowns

Steps 1 and 2

- Assigning variables

  At the beginnig the student must choose variables to designate unknowns that are relevant to the problem.

  This includes the target unknowns (they appear as arguments of `find` clauses) and expressions like **some apples**.

- Discovering relations

  In this step the student has to combine information from different statements into new relations.

  For example, *decomposing* **the fruits that John has** into **the apples and bananas that John has**.

Steps 3 and 4

- Stating equations

  In the next step, the student converts the relations uncovered in the previous step into numerical equations.

  This step finishes when there are enough equations to determine the unknowns of the problem.

  The system checks that the student's equations are consistent equations and are entailed by the problem information.

- Final

  At the last step, the system displays the solution for the unknowns of the problem and exits.