Henkin Semantics for Reasoning with Natural Language
Instructions for Working with the Source Code

Michael Hahn   Frank Richter
mhahn@sfs.uni-tuebingen.de
f.richter@em.uni-frankfurt.de

2015

Contents

1 General

The present manual documents the source code for the paper

> Michael Hahn and Frank Richter. Henkin semantics for reasoning with natural language. *Journal of Language Modelling*, 2015

The software is in part adapted from the source code of

> Patrick Blackburn and Johan Bos. *Representation and Inference for Natural Language. A First Course in Computational Semantics.* CSLI, 2005

henceforth referred to as BB1, available at `http://www.let.rug.nl/bos/comsem/software1.html`. We generally followed the structure of their semantic fragments, and defined an analogous fragment named `HOI` ('higher-order intensional').

Part I of this manual presents exemplary inputs which illustrate the functionality of the package. They put you in a position to replicate our experiments. Part II shows where our code departs from BB1; it contains all necessary information to get started with manipulating the grammar fragment.

Part I
Typical Workflow: Examples

2 Preparation

- Install the reasoning engines on your machine from external sources. The program will call them with the commands

  `prover9 FILE`

  `eprover --tptp-in < FILE`

  `tptp2dfg FILE FILE2; SPASS FILE2` (where `FILE2` is a temporary file)

  `mace4 -c < FILE`

  Modify the predicates in `inferenceEngines.pl` if you do not want to use all supported engines.

- Adjust the definition of `translationStrategy/2` in `foTranslation.pl` to get the desired strength. In short:

  `translationStrategy(hybrid, weak).` = weak translation

  `translationStrategy(hybrid, medium).` = strong translation

  There are a few other options, described in the source code, for other translations which are not covered in the paper.

- Run a Prolog interpreter in the folder `src/` and consult `HOI.pl`.

3 Replicating the Experiments

For replicating our experiments or running your own experiments on modified or new grammar fragments, consult `fracas.pl` and remove everything from inside the directory `tpinput/`. Then run the following in Prolog:

```
initTheoremProversOnTestsuite.
```

Afterward, run the following in `src/`:

```
perl testTP.perl prover9 tpinput
perl testTP.perl eprover tpinput
perl testTP.perl spass tpinput
perl testMB.perl prover9 tpinput
perl printResults.perl tpinput
```

The last script will output the results in LaTeX and compute the statistics that is reported in the paper.

4 Processing Natural-Language Sentences

Now we turn to examples of how the analyses of natural language input can be inspected. For the formatting of Ty2 terms and first-order expressions, refer to Section 7.

- *Inspecting the analysis of a sentence in Ty2 and in the first-order translation:*

  ```
  lambdaHOI([a,man,thinks,mia,dances],[Ty2|_]),
     translate(Ty2,FO).
  ```

  Result:

  `Ty2`: Ty2 representation of the input sentence. If the sentence has multiple analyses in the grammar, the tail of the list will contain more Ty2 representations.

  `FO`: first-order translation of the Ty2 representation

- *Inspecting the first-order translation including the axioms and meaning postulates:*

  ```
  lambdaHOI([a,woman,dances],[Ty2|_]),
    translateWithAxioms(Ty2,FO).
  ```

  Result:

  `Ty2`: Ty2 representation of the first analysis of the input sentence

  `FO`: first-order translation including axioms and meaning postulates

- *Inspecting an analysis with LaTeX output:*

  ```
  lambdaHOI([every,man,dances],[Ty2|_]),
     ho2Latex(Ty2, short, fo).
  ```

Result:

`Ty2`: the Ty2 representation of the sentence

Furthermore, `Ty2` is printed in LaTeX. For more details on `ho2Latex`, see Section 9.3.

- *Inspecting the semantic analysis of a phrase:*

```
np(A,[john],[]),
  getFeatureValue(A,sem,Sem),
  completeAndUnifyTypes(Sem,Sem2),
  betaConvert(Sem2,Sem3),
  ho2Latex(Sem3,short,ho).
```

Result:

`A`: the feature structure assigned to the string 'John' by the grammar when it is analyzed as an NP

`Sem`: the semantic representation contained in `A`, encoding a partially-typed higher-order term (see Section 7 for this format)

`Sem2`: the semantic representation contained in `A`, encoded as a fully well-typed higher-order term

`Sem3`: the beta-converted version of `Sem2`

Finally, `Sem3` is printed in LaTeX format. See Section 9.3 on 'Output' for the options available for `ho2Latex/3`.

When applied to other phrasal categories, the first line must be changed to match the corresponding syntactic label in the grammar in `englishGrammar.pl` (which is adapted from the BB1 grammar).

5 Accessing the Test Suite

- *Accessing test items from our test suite* (`hoOnferenceTestSuite.pl`)

```
testItemFO(ID,C,D-E,HO,FO-FONeg).
```

Result: retrieves item `ID` (replace with any integer for which there is a test item) from the test suite.

`C`: label (valid, contingent, contradictory)

`D`: list of premises

`E`: conjecture

`HO`: Ty2 representation of the inference

`FO`: first order translation of the positive formula ($p \to \gamma$ in Section 4.4 in the paper)

`FONeg`: first order translation of the negative formula ($p \to \neg\gamma$ in Section 4.4 in the paper)

- *Accessing FraCaS items:* after consulting `fracas.pl`, use

```
fracasTestItemFO(Section,ID,C,D-E,HO,FO-FONeg).
```

Result: similar to before. `Section` matches the section (an integer ranging from 1 to 9), `ID` the id of the item (an integer between 1 and 346), and the other variables have the same meaning as before.

- *Print the entire test suite*

```
lambdaHOITestSuite.
```

Result: prints the Ty2 analyses for the test items

6 Running Inference Engines

For the commands below, at least one of the supported inference engines must be installed.

- *Testing validity of natural language inferences (with meaning postulates):*

```
testInferenceSentence([mia,dances,or,does,not,dance]).
```

Result: terminates quickly with success messages from the provers.

```
testInferenceSentence([mia,dances]).
```

Result: does not terminate (or terminates without success message)

- *Testing validity of Ty2 terms:*

```
op(500,xfx,@).
testInferenceFormula((or @ (not @ a)) @ a).
```

Result: terminates quickly with success messages from the provers.

Any partially-well-typed higher-order term that can be resolved unambiguously to a term of type t can be the argument of `testInferenceFormula`.

- *Creating input files for the inference engines:*

First consult `fracas.pl` and remove everything inside `tpinput/`. Then run

```
initTheoremProversOnTestsuite.
```

Result: creates files in `tpinput/` for all test items (both ours and FraCaS)

- *Run theorem provers on the full test suite:*

After running the previous command, execute the following on the command line:

```
perl testTP.perl prover9 tpinput
```

Result: Runs Prover9 on the entire test suite, with timeout set to 30 seconds, and creates a logfile at `tpinput/tp-prover9.log`. The timeout is set in `testTP.perl`. The parameter `prover9` can be replaced by `eprover, spass`.

- *Run model builders on the full test suite:*

  ```
  perl testMB.perl prover9 tpinput
  ```

  Result: Runs Mace4 on the entire test suite, with timeout set to 30 seconds, and creates a logfile at `tpinput/mb-prover9.log`.

- *Printing results and computing statistics:*

  After running `testTP.perl` on `prover9`, `eprover`, and `spass`, and after running `testMB.perl` on `prover9`, execute

  ```
  perl printResults.perl tpinput
  ```

  Result: Will print the results of the inference engines as LaTeX source code, and will compute some statistics. With this script the results reported in the paper can be verified directly.

Part II
Documentation

7 Data formats encoding formulae

This section describes the syntactic formats for encoding first-order formulae and Ty2 terms.

- *Blackburn & Bos first-order formulae.* This is the input format used by the predicates creating output files for inference engines. The predicate `fo2bb(In,Out)` converts from our format to BB1 format

- *Our syntax for first-order formulae.* Extends the BB1 format with more logical operators, otherwise the same. Inspect `fo2bb(In,Out)` for details.

- *(Well-typed) Higher-order terms.* This is the output format of `lambdaHOI/2`. It is defined as follows:

  `TermWithType` → `(Term,Type)`

  `Term` → `Atom`, where `Atom = Functor(Type, ..., Type)`

  `Term` → `lam(VariableWithType,Term)`

  `Term` → `(TermWithType @ TermWithType)`

  `Term` → `PrologVariable`

  `VariableWithType` → `(PrologVariable,Type)`

  `Type` → `e|s|t|PrologVariable`

Type → (Type,Type)

Some predicates will assume that the term is also well-typed, i.e., that lambda abstraction and functional application result in terms of the correct type, and that the logical constants have the types stated in the paper. There might be additional restrictions, use `syntaxCheck/1` to make sure your term will be processed properly.

- *Partially typed higher-order terms.* This is the output format of the grammatical analysis, and the format used in writing the fragment itself. It is defined by adding the rule: `TermWithType → Term` .

  There are problems if the term does not really make sense, e.g., if the same Prolog variable is used both for a type and for a term. Moreover, there can be problems if the type of some variable or constant is not completely determined by the information given in the term.

  `completeAndUnifyTypes(IN,OUT)` from `betaConversion.pl` converts partially typed terms to full higher-order terms, performing type inference to the extent that it is possible.

8 Grammar Fragment and Meaning Postulates

This section describes the encoding of the grammar fragment, the meaning postulates, and the test suite. They all can of course be freely modified.

CFG rules and lexicon are essentially the same as in the BB1 grammar, extended by some words and constructions.

Semantic Composition is defined by the predicate `combine/2` of BB1 in `semRulesHOI.pl`. The Ty2 representations of syntactic categories may be partially typed terms up to the sentential category 'S', determined by composition rules such as

```
combine(s:(A @ B),[np:A,vp:B]).
```

At the level of the text category 'T', the relevant combination rule transforms the entire term into a well-typed Ty2 term (by two rules at the beginning of the file), and, for every subterm of the form `lam((W,s),X)`, it unifies `W` with every free world variable that occurs in `X`.

Semantic Lexicon is in `semlexHOI.pl`, as in the BB1 fragments. Format:

```
semLexBB(SyntacticCategory,FeatureStructure)
```

where the features of the feature structure depend on the category as in BB1's grammar, and the `sem` value (which must be the last value for the generation of `semLex/2` entries to work) should be a partially typed Ty2 term.

Upon compilation, these entries are converted to the `semLex/2` assertions used by the BB1 grammar (in which the `sem` values are well-typed Ty2 terms) by code at the end of `semlexHOI.pl`.

Meaning Postulates are in `linguisticAxioms.pl`. Format:

```
lingaxHO(ListOfTriggeringTy2ConstantSymbols, ListOfFreeTypeVariables,
   Ty2Term, Description)
```

where `Ty2Term` can be partially typed. `ListOfFreeTypeVariables` is currently not supported and should be `[]` to be safe. The types of the triggering constant symbols do not have to be given, but underspecifying argument types of polymorphic constants might lead to problems. Upon compilation, these entries are converted to assertions of

```
foTranslation:lingax(Triggers,FirstOrderTranslationOfPostulate,...)
```

The postulates are printed in LaTeX format by `printMeaningPostulates/0`.

Test suite see Section 5.

Axioms are contained in `translationTypedTotalAxioms.pl`.

9 Important Predicates

9.1 Manipulating higher-order terms

`betaConversion.pl`:

- `syntaxCheck(In)`: checks the format of a higher-order term
- `betaConvert(In,Out)`

9.2 Translation

The predicates for FO translation are:

- `translate(In,Out)`
- `translateWithAxioms(In,Out)`

The translation is implemented in `translationTypedTotalHybrid.pl`. Some other translations that are not described in the paper are also implemented (inspect `foTranslation.pl`).

Choosing translation see Section 2, 'Preparation'.

9.3 Output

Ty2 formulae can be printed in LaTeXwith the predicate
`ho2Latex(Term,Mode,SMode)`
where
`Term` is a Ty2 term
`Mode` is one of `strict(total)` [with all types attached], `strict(partial)` [no types attached to complex terms], `short` [no types].
`SMode` is one of `ho` (curried), `fo` (uncurried + primitive quantifiers)

9.4 Passing formulae to inference engines

The predicates are directly inherited from BB1 and work analogously. They take FO formulae in BB1's format. See Section 6 for examples.

`callInference.pl`:

- `callTP(Formula,Solved,Engine)`
- `callMB(Problem,DomainSize,Model,Engine)`

The interface to the engines itself is provided by Perl scripts as in BB1 (see Section 6 for instructions on how to run them). To add a new reasoning engine, modify the Perl scripts, and supply a suitable declaration of `proverCall/3` or `initModelBuilders/2` in `callInference.pl`.

References

[1] Patrick Blackburn and Johan Bos. *Representation and Inference for Natural Language. A First Course in Computational Semantics.* CSLI, 2005.

[2] Michael Hahn and Frank Richter. Henkin semantics for reasoning with natural language. *Journal of Language Modelling*, 2015.