State of the project

Pierre Misse (Hogo), Théo Rogliano (Alisu)

December 2018

1 Sources

Every classes used in this project are available in the _resolution_1erOrdre package, in the ./Sources/ directory. Just take the higher version.

2 Known Limitations

- The quantifier don't take terms as arguments, but strings. This forbids the construct $\exists f(x), P(f(x))$. The treatment for functal terms is defined recursively though, and should work on when this feature will be implemented.
- The quantifier aren't variadic, although they are in the propGeneration tool.

3 General testing

There's two types of tests available. Automated testing for some part of the program through Unit Testing. Those are available in each test class documentation and in the state of the program part of this file. Textual testing available in the Prop metaclass. They can be run in the playground using 'Prop TestName.' and using the Ctrl+D command on that line, and the result will show on a Transcript (which needs to be open). Several smaller tests for less important sections of the program are available only textually in the Prop metaclass.

Traces can be activated for the 4 algorithmic process described in the next section. They are all available in the Prop metaclass method "initialize", and will be run before every tests, so the traces are up to date.

4 State of the program

The resolution method was build using 4 distinct steps

- Skolemisation
- Clausification
- Formal Clause
- Resolution method
  - Unification

We'll describe each of those process using the folowing pattern:

- State of implementation
- State of automated testing
- How to add tests
- eventual comment on the implementation
- eventual questions concerning the specification

5 Skolemisation

The process is completely implemented. Some automated tests are available in the TestSkolemnisation class, but aren't covering (yet).

Since it's the first step of the process (modulo the renaming of the variables), this is where the propGeneration tool shines. See the referring manual to use it.

To apply only this process to this proposition, you just have to send the "StartSkolemnisation" message to the concerned proposition.

$$aProposition\ StartSkolemnisation.\ (ctrl + P\ to\ run\ and\ show\ the\ result)$$

Although there was a question (left unanswered so far) of the validity of the rules in the specifications. we can find this in the slides:
$s(\exists x.\phi) = s(\phi)$
Given that every other rules are symmetric, and that the symmetric change function:
$h(\forall x.\phi) = s(\phi)$
We weren't able to find another source to confirm or deny this.

In the implementation, we used this interpretation:
$s(\exists x.\phi) = s(\phi)$
$h(\forall x.\phi) = h(\phi)$

6 Clausification

The process is completely implemented, and the automatic tests are divided in two sections.
The isClause test, which check that a proposition is a clause, and the one testing the clausification process. They should be covering everything for this part.

The terms in the propositions are completely ignored by this process which allows us to use the tool propGenerator as-well.

To apply this process to a proposition, send the message "fullClausification" to a skolemised proposition.

$$aSkolemisedProposition\ fullClausification\ (crtl + P\ to\ run)$$

7 Switch to Clausal Form

After the clausification, we change the format of the data from a tree to a set of clauses (which are sets too). There's no automatic tests so far. To do that switch, we just have to give a clausified prop to the default constructor of ClausalForm through the message "new".

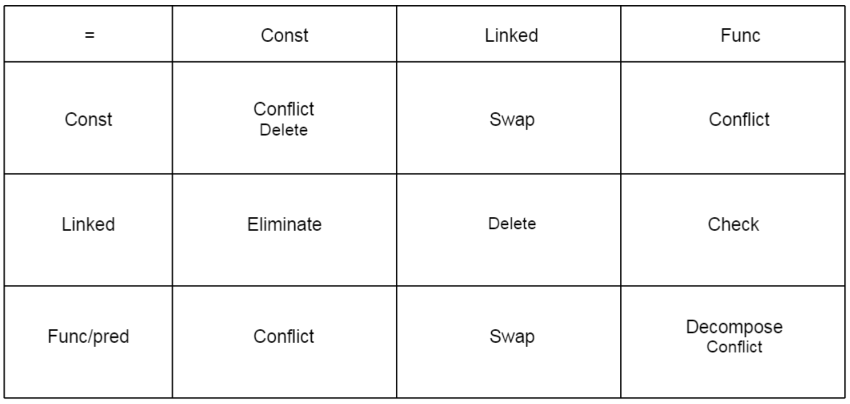
$$ClausalForm\ new : aClausifiedProposition$$

From that point on, we concentrated on the understanding of the algorithms, rather than the design. It's poorly done, but it's functional nonetheless.

An important thing to note is that the resolution method affect the data structure, we don't work on a copy. If you want to apply the method resolution several times on a clausal form, you'll have to refill it every time. The terms are also specialized after the skolemisation process, and are important here. So it'll be harder to create new tests.

8 Unification

The unification process being widely use during the clausification, we'll start by talking about it. Partial automated testing are available, not covering yet.



=	Const	Linked	Func
Const	Conflict Delete	Swap	Conflict
Linked	Eliminate	Delete	Check
Func/pred	Conflict	Swap	Decompose Conflict

Figure 1: Table used to implement the unification of terms.

For reference, the table of possible operations we used is represented in figure 1.
And is fully implemented.

Very unsure on some of those, but we're missing some examples.
VERY unsure about what to do on a const=Func or the other way around.

9 Resolution Method

The process is fully implemented. A few automated tests are available, which correspond to the proofs that had to be done during the course.

The resolution method can be applied :

- On a clausal form, using the message "resolutionMethod". This is only the last part of the process, so this method returns the unsatisfiability of the proposition, since we didn't negate it (pragmatically). This method affect the clausal form, as previously stated. aClausalForm resolutionMethod.
- On a Proposition using the amIValid message. This time, the proposition goes through all the processes before returning the validity of the We negate the proposition at the start of this method, so the validity of the proposition is returned this time. aProposition amIValid.

10 Optional renaming

This is an optional pre skolemisation step, in which we rename the terms with similar names, that aren't in the same scope, or that are masked.
$\forall x.\forall x.\ P(x)$ becomes: $\forall x.\forall x1.\ P(x1)$

This process if fully implemented, with an automated test available in General Tests.

This is applied using the launchRenameVariables on a proposition:

$$aProposition\ launchRenameVariables.(ctrl + p\ to\ print\ the\ result)$$