MUltseq v2.0-2-gf2ad99b

Contents

1 Installation & Configuration

MUltseq consists of the following files:

- `multseq/` Directory: The Prolog code for MUltseq
  - `mscalcul.pl` kernel of MUltseq: proof construction and transformations
  - `msconf.pl` OS- and Prolog-specific settings
  - `msconseq.pl` converting consequences between formulas and sequents, equalities, and quasi-equations to (lists of) sequents.

  - `mslgcin.pl` routines for reading logic specification
  - `msoption.pl` processing of options
  - `mstex.pl` output routines (TeX)
  - `msutil.pl` auxiliary predicates
  - `multseq.pl` main file; loads all other parts of MUltseq listed above

- `examples/` Directory:
  - Logic definition files
    * `classical.msq` Classical logic
    * `fde.msq` First-Order Entailment Logic (4-valued)
    * `goedel.msq` 3-valued Gödel logic
    * `lukasiewicz.msq` 3-valued Łukasiewicz logic
    * `shramko-wansing.msq` Shramko & Wansings SIXTEEN (16-element bilattice logic)
  - Property files (to use with `chkProp`)
    * `properties.pl` List of popular properties
  - Example experiment batch files
    * `ex_classical.pl` Checks that MUltseq can prove many tautologies, consequences, equivalences, etc. that all hold in classical logic.
    * `ex_lukasiewicz1.pl` Checks which classical tautologies, consequences, etc. hold in 3-valued Łukasiewicz logic (with weak $\wedge$ and $\vee$.
    * `ex_lukasiewicz2.pl` Checks all properties involving $\wedge$ and/or $\vee$ and compares the strong and weak Łukasiewicz operators.
    * `ex_sixteen.pl` Tests some properties in SIXTEEN, finds which versions of $\neg A \vee B$ (interpreted as implication) satisfy modus ponens and deduction theorem) and compares combinations of $\neg$, $\wedge$, $\vee$ in their truth- and falsity-order versions
  - Auxiliary files for producing LaTeX output
    * `mspost.tex` postamble of TeX documents created by MUltseq
    * `mspre.tex` preamble of TeX documents created by MUltseq
    * `proof.sty` style file for typesetting derivations (by Makoto Tatsuta)

- `doc/` Directory:
  - `multseq.md` this file; the MUltseq manual

Check file `msconf.pl` and edit it to fit your operating system (DOS/Windows, Unix, or Mac).

2 How to use MUltseq

2.1 For the impatient

Download and unzip the code or clone MUltseq using Git:

```
git clone git@github.com:rzach/multseq.git
```

Put yourself in the directory containing the examples

```
cd multseq/examples
```

Load Prolog

```
prolog
```

Pick an example experiment file and run it by entering, e.g.,

```
[ex_lukasiewicz1].
```

at the `?-` prompt.

This should generate a file `ex_lukasiewicz1.tex` which you can run through `pdflatex`.

3 Interface

3.1 Main predicates

- `provable(Sequent, Proof).`

  `Proof` is a derivation of `Sequent` from axioms only; fails if `Sequent` has no proof.

- `derivable(Sequent, Proof).`

  Same as `provable/2` but generates a derivation from hypotheses if no correct proof exists.

- `collect_hyps(Proof, ListOfHypotheses).`

  `ListOfHypotheses` is the minimized list of hypotheses used in `Proof`.

- `proof_skeleton(Proof, Skeleton, SequentTable).`

  `Skeleton` is the skeleton of `Proof`: all sequents are replaced by unique numbers. The correspondence between numbers and sequents is given by `SequentTable` which is a list of `Number : Sequent` pairs.

4 Logic definition files

Logic definition files tyically use the `.msq` extension. They provide the information MUltseq needs to convert test queries to sequents (i.e., what the truth values and designated truth values are), the sequent rules, and definition of how to output operators and formulas using LaTeX.

MUltlog can produce `.msq` files from MUltlog's `.lgc` and `cfg` files, computing optimized sequent rules. (Use the `lgc2msq/3` predicate in `ml_msq.pl`).

To load a logic definition file, say `load_logic(File)`, e.g.,

```
:- load_logic('lukasiewicz.msq').
```

- List of truth values

  ```
  truth_values(TVs).
  designated_truth_values(DTVs).
  ```

  where `TVs` is a list of atoms, and `DTVs` is a sublist of `TVs` (the truth values that are designated). This is used to compute sequents for consequence relations.

- Operator declarations:

  ```
  operator(Name, Arity, TeXName).
  ```

  where `Name` is the name of the oerator, `Arity` is the number of arguments, and `TeXName` is a string that provides the LaTeX code for printing the operator symbol. `op(Precedence, Type, Name).` This is optional and used to define the operator in Prolog: the parameters of are the same as those of the built-in `op/3` predicate. If it is ommitted, formulas have to be written in the standard Prolog prefix notation.

- Sequent rules:

  ```
  rule(Conclusion, Premises, Name).
  ```

  One `rule` declaration is needed for each sequent rule. `Conclusion` is a single signed formula (e.g., `and(X,Y)^t`) and `Premises` is a list of lists of signed formulas assumed to be the variables occuring in `Conclusion` (e.g., `[[X^t], [Y^t]]`).

- LaTeX formatting

  ```
  tex_tv(TV, String).
  ```

  The truth value `TV` is typeset using `String` (in math mode). If no such directive exists for a truth value, `TV` is converted to a string using `name/2`. This may lead to an error if TV is no valid argument for `name/2`.

  ```
  tex_rn(RN, String).
  ```

  The rule name `RN` is typeset using `String` (in math mode).

  ```
  tex_op(Expr, List).
  ```

  `Expr` is a term built from a connective and Prolog variables as arguments. These variables can be reused in `List` to typeset a connective in prefix/infix/postfix notation. To obtain a backslash, use "bslash". Example:

```
tex_op(and(A, B), ["(", A, "\\land", " ", B, ")"])
    .
```

If there is no "tex_op" directive for a connective, the expression will be typeset in functor notation, using a tex_op directive for the connective if available.

5 Properties of a logic

A property of a logic is a fact of the form

```
property(Name, OPList, PropSpec).
```

where `Name` is an atom (a label for the property), `OpList` is a list of `Operator/Arity` pairs, and `PropSpec` is a specification of the property. The `OpList` must contain all operators used in `PropSpec` with the corresponding number of places.

`PropSpec` can be one of the following:

- Tautology:

  ```
  tautology(Fmla)
  ```

- Consequence (implication):

  ```
  consequence([Prems] => Concl)
  ```

- Equivalence (bi-implication)

  ```
  equivalence(Fmla1, Fmla2)
  ```

- Algebraic equality (identity of truth values)

  ```
  equality(Fmla1, Fmla2)
  ```

- Metaconsequence

  ```
  metaconseq([Hyps], Concl)
  ```

In all but the last, `Fmla`, `Fmla1`, `Fmla2`, and `Concl` are single formulas and `Prems` is a list of formulas. In `metaconseq`, `Concl` is a consequence judgement `[Prems] => Fmla` and `Hyps` a list of such.

Examples:

```
property(contrapos, [(>)/2, (-)/1], tautology((a > b)
   > (-b > -a))).
property(constrdilemma, [(\/)/2, (>)/2, (-)/1],
   consequence([(-a) \/ (-b), a > c, b > d] => ((-c)
   \/ (-d)))).
```

```
property(ldistrleft, [(/\)/2,(\/)/2], equivalence( a
   /\ (b \/ c), (a /\ b) \/ (a /\ c) )).
property(residuation, [(>)/2, (/\)/2], metaconseq([[p
   /\ q] => r], [p] => (q > r) )).
```

It is possible to combine property specifications in a `property` declaration using Prolog's and (`,`), or (`;`), and not (`\+`) operators, e.g.,

```
property(demorgan, [(/\)/2, (\/)/2, (-)/1],
  ( equivalence(-(a /\ b), (-a) \/ (-b)),
    equivalence(-(a \/ b), (-a) /\ (-b)))).
```

6 Checking properties

If you have loaded a logic and a list of properties, you can check whether the properties hold in the logic using the predicate

```
chkProp(Name).
```

This only works directly if the operators used to define properties are the same as the operators in the logic definition file. However, very often you'd like to use properties for different logics, or even for the same logic but for different operators. E.g., a logic might have two negations, and you want to check which (if any) satisfy De Morgan's laws. For that reason it is possible to provide a list `Omap` as a second argument to `chkProp` that specifies which operators of the logic should map to which operators in the property specification. `Omap` is a list of pairs `OpL/OpP` where `OpL` is an operator of the logic and `OpL` is an operator in the property specification. E.g.,

```
:- chkProp([and/(/\),or/(\/)], ldistrleft).
```

You can define a default operator mapping using `setOmap(Omap)`. `chkProp/1` uses this; by default it is empty, i.e., no replacement of operators happens.

You can also use the property specifications above directly, e.g.,

```
:- tautology(a imp (a or b)).
```

will test if `a imp (a or b)` is a tautology. This assumes the logic definition contains operators `imp` and `or` defined as infix. Functor notation must be used if the operators are not defined as infix (using `op/3`), e.g.,

```
:- tautology(imp(a, or(a, b))).
```

7 Logging

The commands

```
start_logging.
stop_logging.
```

turn on logging of user input and Prolog query results. If the option `tex_output` is either `terse` or `verbose`, then any LaTeX output is written to the log file as well.

Without arguments, `start_logging` writes to a file `session_TIMESTAMP.tex`. You can change the default extension `.tex`, or the filename and extension by providing additional arguments:

```
start_logging(Extension).
start_logging(Filename, Extension).
```

where `Extension` is the extension to use *with period*, e.g., `'.tex'` or `'.log'`. E.g., to write to file `lukasiewicz.tex`, say

```
start_logging(lukasiewicz,'.tex').
```

8 LaTeX Output

The option `tex_output` controls whether MUltseq queries merely run checks on properties (and fail or succeed accordingly), or if MUltseq also provides output suitable for printing using LaTeX. By default, `tex_output` is `off`, so no output is generated. If it is set to `terse`, MUltseq will generate output that records the queries and results in human readable format, e.g.,

Proposition. The formula $A \to (A \vee B)$ is a tautology.

If it is set to `verbose`, MUltseq will also output the sequent corresponding to the query as well as its proof (or, if the query fails, the derivation from non-axiom hypotheses and the corresponding list of counterexamples).

LaTeX output is written to the log file if logging is on (i.e., `start_logging` has been called) and to the standard error stream if not.

9 Options

- `reset_options.`

  Resets all options to their defaults.

- `set_option(Option).`

  Sets option `Option`. See below for the list of options.

- `list_options.`

  Lists all active as well as all available options.

Currently the following options are implemented:

- `strategy(S)`.

  Defines the strategy for selecting an applicable rule. The following strategies `S` exist:

  - `leftright` (default).

    The sequent is scanned from left to right, and the first non-variable expression found is decomposed using the first rule (searching top-down) which is applicable to the expression.

  - `topdown`.

    The rules are scanned top-down, and the first rule applicable to any expression in the sequent is chosen; if the rule applies to several expressions, the first one from the left is taken.

  - `ordering(ListOfRuleNames)`.

    Refinement of the strategy "topdown". The rules are tried in the order given by `ListOfRuleNames`. The same effect could be reached (less elegantly) by reordering the rules in the input file and using `top_down`. A good heuristic might be to order the rules according to their branching degree, small branching degrees first. This will result in narrow proof trees and less duplication. Within the same branching degree one might prefer rules with more formulas per sequent (with the intuition behind that this will lead faster to an axiom); or one might prefer rules with a small number of formulas per sequent (to avoid duplication).

  - `interactive`

    MUltseq collects all rules applicable to any part of the sequent. If there is only one possibilty to apply a rule, it is applied. Otherwise the rules are listed on standard output and the user is given the choice.

- `tex_rulenames(OnOff)` (default: `OnOff=off`).

  If `OnOff=on` then each step in a TeX-derivation is labeled by the applied rule. Otherwise, if `OnOff=off`, no labels are added.

- `tex_sequents(Style)` (default: `Style=signed`).

  `Style=signed` prints sequents as signed formulas, whereas with `Style =multidimensional`, sequents are represented as (n)-tuples, where the truth value assigned to a formula is given implicit by the position of the formula within the sequent.

- `tex_proofstyle(Style)` (default: `Style=verbose`)

  `Style=verbose` means that axioms and hypotheses are marked by the phrases "axiom for (A)" and "hypothesis". `Style=compact` results in the phrases "ax(A)" and hyp". `Style=bare` suppresses the phrases altogether.

- `tex_output(Level)` (default `Level=off`),

  If `Level=off`, no LaTeX output is produced by `load_logic` or the various property checking predicates. If `Level=terse`, the results of property checks are recorded as propositions in human-readable format. If `Level =verbose`, MUltseq additionally prints the evidence, i.e., which sequents correspond to the property queries, as well as the derivations and, if applicable, counterexamples.

- `tex_success(OnOff)` (default `OnOff=on`)

- `tex_failure(OnOff)` (default `OnOff=on`),

  If `OnOff` is `on`, then LaTeX output is produced whenever a property check succeeds/fails. To print only for successful checks, set `tex_failure(off)`; top print only failed tests, `tex_success(off)`.

10 The prover

10.1 Data structures

- Sequent: list of signed formulas
- Signed formula: `F^S` where `F` is a many-valued formula (any term) and `S` is a truth value.
- Proof tree: `ra(Name, Conclusion, Proofs)` `ra` stands for "rule application". Rule `Name` is applied to the conclusions of `Proofs`, giving `Conclusion`. `Conclusion` is a sequent and `Proofs` a list of proof trees. prove(Sequent, Prooftree). Constructs a for .

10.2 Predicates

```
select_rule(S, Sl, Sr, Ps, R)
```

Select a formula `F` from sequent `S` and choose an applicable rule. `Sl` are the signed formulas to the left of `F`, `Sr` are those to the right of `F`, `Ps` are the premises of the chosen rule, and `R` is its name.

```
appseqs(Sl, Sr, S0, S)
```

`Sl` is added to the left of `S0` and `Sr` to the right, the result being `S`: `S = [Sl, S0, Sr]`.

```
split(S, Sl, F, Sr)
```

Selects an element `F` from `S`, and instantiates `Sl` (`Sr`) with the elements to the left (right) of `F`: `S = [Sl, [F], Sr]`

```
contains_axiom(Sequent, A)
```

Checks whether `Sequent` contains axiom `A|...|A`. Depends on the fact that truth values and sequents are kept sorted.

```
collect_hyps(Proof, MinHypotheses)
```

Gathers all hypotheses occurring in `Proof` and returns a minimized list of hypotheses in `MinHypotheses`.