The Default OLP Configuration File
`open-logic-config.sty`

OpenLogic Project

2017-06-03 7d6958d

Description

This file contains all commands and environments that are meant to be configured, changed, or adapted by a user generating their own text based on OLP text. **Do not edit this file to customize your OLP-derived text!** A file `myversion.tex` adapted from `open-logic-complete.tex` (or from any of the contributed example master files) will include `myversion-config.sty` if it exists. It will do so after it loads this file, so your `myversion-config.sty` will redefine the defaults. This means you won't have to include everythng, e.g., you can just change some tags and nothing else. You may copy and paste edfinitions you want to change into that file, or copy thi file, rename it `myversion-config.sty` and delete anything you'd like to leave as the default.

Symbols

Formula metavariabes

Use the exclamation point symbol `!` immediately in front of an uppercase letter in math mode for formula metavariables. By default, `!A`, `!B`, ... are typeset as φ, ψ, χ, ... if you use the command `\olgreekformulas`. If this is not desired, and you'd like A, B, C, ... instead, use `\ollatinformulas`. If you issue `\olalphagreekformulas`, you'll get α, β, γ, ...

Logical symbols

The following commands are used in the OLP texts for logical symbols. Their definitions can be customized to produce different output.

Truth Values

- `\True` defaults to 1 and `\False` to 0.

Propositional Constants and Connectives

- Falsity is `\lfalse` and defaults to $\bot$.
- Truth is `\ltrue` and defaults to $\top$.
- Negation is `\lnot` and defaults to $\neg$. To use a different symbol (e.g., tilde), use the following line.

```
\DeclareDocumentMacro \lnot {\mathord{\sim}}
```

- Conjunction is `\land` and deaults to $\wedge$. to use ampersand, uncomment the following line

```
\DeclareDocumentMacro \land {\mathbin{\&}}
```

- Disjunction is `\lor` and defaults to $\vee$.
- Conditional is `\lif` and defaults to $\rightarrow$. To use a different symbol, replace `\rightarrow` in the definition, e.g., by `\supset`
- The biconditional is `\liff` and defaults to $\leftrightarrow$. To use the triple bar $\equiv$ replace with `\equiv`.

Quantifiers

The quantifier symbols are provided as commands `\lexists` and `\lforall` which take two optional arguments. If no arguments are provided, it they just typeset the quantifier symbol. With one optional argument they produce the quantifier together with a variable, and this may include parenthesesaround the quantifier and variable. The second optional argument producesthe quantifier/variable combination plus the formula in the scope of the formula with appropriate spacing. For instance, `\lexists[x][!A(x)]` will, by default, produce $\exists x\, \varphi(x)$.

- The existential quantifier is `\lexists`. Replace `\exists` with `\boldsymbol{\exists}` for boldface, or redefine appropriately if you want parentheses around $\exists x$.
- The universal quantifier is `\lforall`.
- The identity relation is also provided as `\eq`. By itself, it produces the identity reation symbol (default: $=$) by itself. With two optional arguments, it typesets the corresponding atomic formula, e.g., `\eq[x][y]` produces $x = y$. `\eq/` produces the negated symbol (formula).

Proofs and Derivations

- The sequent symbol `\Sequent` produces $\Rightarrow$ by default. Change the definition for $\vdash$, or another symbol.

The sequent symbol in proofs displays as the above sequent symbol.

- Rule names: `\LeftR{Op}` typesets the name of a left rule for operator `Op`, e.g., `\LeftR{\land}` produces `$\land$L`. `\RightR{Op}` does the same for right rules.
- `\Weakening`: produces name or abbreviation for weakening rule, e.g., "W".
- Rule names: `\Intro{Op}` typesets the name of an intro rule for operator `Op`, e.g., `\Intro{\land}` produces `$\land$Intro`. `\Elim{Op}` does the same for elimination rules.
- `\Discharge{!A}{n}`: typesets a discharged assumption with label n, e.g., $[!A]^n$.
- `\DischargeRule{Rule}{n}`: used in a `prooftree` environment to provide the labels for an inference that discharges an assumption.

Metalogical Relations

Metalogical relationships, such as truth in a structure, validity, consequence, and provability, are also provided as commands. Uniform use of these commandsinstead of hard-coded typesetting according to specific conventions guarantees that by changing the definitions below you can uniformly change notation in the text.

Substitution

`\Subst{!A}{t}{x}`: The operation of substituting a term for a (free) variable in another term or in a formula. The default is $\varphi[x/t]$, other common notations are φ^t_x, $\varphi\{t \to x\}$, or $S^t_x\varphi$.

The satisfaction/truth relation

`\Sat[/]{M}{!A}[s]`, the relation of being satisfied in a structure (relative to an assignment), is provided as the command `\Sat` with two mandatory arguents (the structure and the formula) and one optional argument (the assignment). Use `\Sat/` to create the negated relation. By default, `\Sat{M}{!A}[s]` is typeset as $\mathfrak{M}, s \models \varphi$.

The derivability relation

`\Proves[L]` is used to create the symbol for the derivability relation, `\Proves/` for the negation. By default this creates $\vdash$; e.g., `\Gamma \Proves !A` yields $\Gamma \vdash \varphi$. An optional argument may be used for the calculus or logic relative to which the provability relation is defined; by default it creates a subscript on the turnstile.

The semantic consequence relation relation

`\Entails` is the semantic counterpart of `\Proves` and defaults to $\vDash$. It also takes an optional `/` for $\nvDash$ and an optional argument for a subscript.

Model-theoretic notions and symbols

- `\Domain{M}` - domain of a structure, e.g., `\Domain{M}` gives $|\mathfrak{M}|$.
- `\Assign{R}{M}` - Assignment (value of) of a constant/predicate symbol in a structure; e.g., `\Assign{R}{M}` produces $R^{\mathfrak{M}}$.
- `\Value{t}{M}[s]` - Value of a term in a structure. Takes two mandatory arguments (term and structure) and one optional argument (variable assignment). By default, `\Value{t}{M}[s]` produces $\mathrm{Val}^{\mathfrak{M}}_{s}(t)$.
- `\substruct`: symbol for the substructure relation
- `\Theory{M}`: theory of a structure
- `\Mod[L](L'){T}`: class of models of a theory/sentence T in a language $\mathcal{L}$ and logic L'.
- `\elemequiv`: elementary equivalence (infix relation)
- `\iso[/][p]`: relation of being (partially) isomorphic
- `\ident`: syntactic identity between expressions (infix relation),
- `\QuantRank{!A}`: quantifier rank of a formula
- `\Expan{M}{R}`: expansion of a structure by a relation (etc.)

`\nssucc`, `\nsplus`, `\nstimes`, `\nsless`: non-standard arithmetical operations

Recursion-theoretic Notions and Symbols

- `\Proj{n}{i}`: projection functions
- `\Zero`: the constant zero function
- `\Succ`: the successor function

- `\Add`: the addition function
- `\Mult`: the multiplication function
- `\Exp`: the exponentiation function
- `\Pred`: the successor function
- `\tsub`: truncated subtraction function
- `\Char{R}`: characteristic function
- `\defis`: definitional identity
- `\defiff`: definitional equivalence
- `\concat`: concatenation of sequences
- `\umin{x}{!A}`: unbounded minimization
- `\bmin{x < y}{!A}`: bounded minimization
- `\bexists{x < y}{!A}`: bounded existential quantification
- `\bforall{x < y}{!A}`: bounded univeral quantification
- `\cfind{e}[n]`: partial computable function with index e
- `\redone`: one-step reduction
- `\red`: reduction
- `\lambd[x][!A]`: lambda abstract
- `\num{n}` : numeral corresponding to a number
- `\scode{s}`: code for a symbol
- `\Gn{!A}`: G"odel number of a string of symbols

Modal Logic

- `\mSat{M}{!A}[w]`: modal satisfaction relation

Special Sets and Mathematical Symbols

Set-theoretic operators

- Set abstracts: Use `\Setabs{x}{!A(x)}` to produce the set abstract $\{x : \varphi(x)\}$. If you prefer a | to :, change the definition accordingly.
- `\Pow{X}`: Power set, produces $\wp(X)$
- `\dom{f}`: domain of a function

- `\ran{f}`: range of a function
- `\len{s}`: length of a sequence
- `\emptyseq`: the empty sequence
- `\restrict`: restriction of a function to a set (infix operator)
- `\Complement{X}`: complement of a set
- `\card{X}`: cardinality of a set
- `\cardle{X}{Y}`: X is no larger than Y
- `\cardless{X}{Y}`: X is smaller than Y
- `\cardle{X}{Y}`: X is equinumerous with Y
- `\tuple{x,y}`: pairs, tuples, sequences
- `\comp{f}{g}`: composition of f with g, defaults to $g \circ f$
- `\pto`: partial function arrow
- `\defined`, `\undefined`: postfix for defined, undefined functions

Particular sets

- Natural numbers: `\Nat`
- Integers: `\Int`
- Real numbers: `\Real`
- Rational numbers: `\Rat`
- The set $\{0, 1\}$: `\Bin`
- Identity relation: `\Id{X}`

Symbols for Turing Machines

- `\TMendtape` - symbol indicating left end of tape
- `\TMblank` - symbol for a blank
- `\TMstroke` - single stroke symbol on tape
- `\TMright` - symbol for move right instruction
- `\TMleft` - symbol for move left instruction
- `\TMstay` - symbol for the stay instruction
- `\TMtrans` - typeset a TM transition

Functions and Function/Relation symbols

- `\Part`: the parthood predicate
- `\Prf`: the proof relation
- `\Refut`: the refutation relation
- `\Prov`: the provability predicate
- `\RProv`: the Rosser provability relation
- `\OCon`: the consistency statement

Typesetting commands for logical concepts

In order to unformly typeset certain types of symbols uniformly, the OLP texts use special commands which carry out the typesetting. Thus, e.g., all structures, which appear in the texts, say, as `\Struct{M}`, can be typeset according to preference. By default, they are typeset in fraktur (e.g., $\mathfrak{M}$) but this can be configured by changing the definition of the `\Struct` command. Note that often the default behavior is to only apply the typeface command to the first token in the argument, e.g., `\Struct M_n` will generate $\mathfrak{M}_n$ and not $\mathfrak{M}_{\mathfrak{n}}$.

- `\Struct{M}` - First-order structures; by default, the first token in Fraktur
- `\mStruct{M}` - modal structures; default: set first token in Fraktur
- `\Lang{L}` - Languages; default: set first token in callgraphic font
- `\Log{L}` - Logics; default: set entirely in boldface
- `\Obj` - Object-language symbols; default: set entirely in sans-serif italics
- `\Atom{P}{t_1, t_2)` - Atomic formula or term; default produces predicate symbol followed by argumnets surrounded by parentheses. Some prefer no parentheses around the arguments.
- `\PIso` - Set of all partial isomorphisms
- `\fn{func}` – typeset a function name
- `\Th{T}` – typeset name of a theory

Sets of Expressions

The following commands provide uniform notation for sets of expressions, such as the set of formulas of a language. Typically the parameter (language) can be providedas an optional argument. By default, if the argument is given,

they aretypeset following the symbol within parentheses, i.e., the symbols are treated as operators (e.g., Frm($\mathcal{L}$). Redefine the commands to, e.g., typeset the parameters as subscripts (e.g., Frm$_{\mathcal{L}}$.

- `\Var`: the set of propositional variables
- `\Trm[L]`: the set of terms (of a language)
- `\Frm[L]`: the set of formulas (of a language)
- `\Trm2[L]`: the set of second-order terms (of a language)
- `\Frm2[L]`: the set of second-order formulas (of a language)
- `\SubFrm{!A}`: the set of subformulas of a formula (mandatory argument: formula)
- `\Sent[L]`: the set of sentences (of a language)

Commands for uniform formulations

Inductive definitions

Inductive definitions typically divide into cases depening on the form of a formula for which a concept if defined. To uniformly typeset these definitions flexibly, we provide a command `\indcase{formula}{complex formula}{case text}` which (a) typesets a uniform description of a case and (b) defines `\indfrm` and `\indcomplex` which can then be used in the definitions. For instance, in an inductive definition of $\models A$, you might say `\indcase{A}{B \land C}{$\models \indfrm$ iff $\models B$ and $\models C$}` to produce: "If $A \equiv B \land C$, then $\models A$ iff $\models B$ and $\models C$" or, alternatively and more succinctly: "$\models B \land C$ iff $\models B$ and $\models C$". Use the starred version for the atomic case, and a ! instead of the star for a case you want to leave as an exercise.

Tokens

The following terms are *tokenized* throughout OLP. This means that by changing the definition in the configuration file, the term as printed will also change. This is simpler than searching and replacing these terms in all OLP texts, and it will also treat plurals as well as occurrences of the term at the beginning of a sentence (where it should be capitalized) correctly.

Tokens are defined using the `\settexttoken` command.

`\settexttoken{token}{singular}{plural}[Singular][Plural]`

Here `token` is the term as it is used in the source text, where it typically is used as `!!{token}`. `singular` and `plural` are the text you want printed wherever an OLP text contains `!!{token}` or `!!{token}s`. In sentence-initial position, OLP texts would use `!!^{token}` and `!!^{token}s` to create capitalized versions of these token replacements. By default, they are generated from `singluar` and `plural` by capitalizing the first character, but can be provided explicitly to `\settexttoken` as optional arguments. `!!a{token}` produces an indefinite article plus the token replacement. This is usually "a" unless the settexttoken command is used with a star, as in `\settexttoken{element}*{element}{elements}`, in which case it produces "an". This can be combined with `^` to produce the uppercase version, e.g., `!!^a{element}` for "An element". `\article{token}` and `\Article{token}` produce just the article by itself (lower or uppercase, respectively).

- `language`: defaults to "language", redefine for, e.g., "signature"
- `formula`: defaults to "formula/formulas", redefine for plural "formulae," or for "wff".
- `subformula`: defaults to "subformula", redefine for plural "subformulae," hyphenated spelling "sub-formula", or "sub-wff".
- `sentence`: defaults to "sentence", redefine for "closed formula" etc.
- `variable`: defaults to "variable/variables", redefine to be more specific, e.g., "individual variable", "object veriable".
- `constant`: defaults to "constant", redefine for "individual constant", "constant symbol."
- `predicate`: defaults to "predicate symbol".
- `function`: defaults to "function symbol".
- `operator`: defaults to "logical operator", redefine for "connective".
- `main operator`: defaults to "main operator", redefine for "outermost operator".
- `free for`: defaults to "free for", redefine for "substitutable for" as in Enderton.
- `identity`: defaults to "identity predicate", redefine for "equality predicate."
- `conditional`: defaults to "conditional", redefine for "implication."
- `biconditional`: defaults to "biconditional", redefine for "equivalence."
- `falsity`: name of the falsity symbol, defaults to "falsity", redefine for "absurdity."
- `truth`: name of the truth symbol, defaults to "truth", redefine for "verum" or "top".

- `structure`: term for first-order structures, defaults to "structure", redefine for "interpretation", "model".
- `domain`: domain of a structure
- `\value`: value (denotation) of a term
- `derivation`: derivation in a calculus, proof
- `derive`: derive in a calculus, prove
- `derivable`: derivable in a calculus, provable
- `derivability`: derivability in a calculus, provability
- `nonderivability`: derivability in a calculus, unprovability

`-complete`: negation complete, syntactically complete (of theories)

`-axiomatizable`: effectively/recursively axiomatizable

`-represents`, `representable`

- `discharge`: discharge an assumption in a natural deduction proof, cancel, close. Also: undischarged, non-cancelled, open.
- `enumerable`: term for finite or countably infinite; defaults to "enumerable", redefine for "countable".
- `nonenumerable`: term for uncountable; defaults to "non-enumerable", redefine for "uncountable".
- `denumerable`: term for countably infinite; defaults to "denumerable".
- `element`: element of a set; redefine for "member"
- `injective`, `injection`: redefine for "one-one" and "one-one function"
- `surjective`, `surjection`: redefine for "onto" and "onto function"
- `bijective`, `bijection`: redefine for "one-one onto" and "one-one onto function" or "correspondence"
- `decidable`:

Tags

Tags are used to guide selective compilation, using `sty/open-logic-selective.sty`. Tags are initialized using `\tagtrue` and `\tagfalse`, which also initialize, for each `tag` given as argument, a corresponding tag `nottag` with opposite truth value.

- `prvNot`, `prvOr`, `prvAnd`, `prvIf`, `prvIff`, `prvTrue`, `prvFalse`, `prvEx`, `prvAll` - Primitives: tags for (sets of) operators which are treated as primitives. Default: all as primitives.
- `defNot`, `defOr`, `defAnd`, `defIf`, `defIff`, `defTrue`, `defFalse`, `defEx`, `defAll` - Defined operators: tags for operators which are defined, not primitive. Default: none. Note: Not all combinations of primitive/defined operators will result in complete definitions!
- `probNot`, `probOr`, `probAnd`, `probIf`, `probIff`, `probEx`, `probAll` - Cases in proofs: tags for (sets of) operators for which cases are proved; if the corresponding tag is off then the case is added as a problem. Default: prove all cases. Note: propositional constants are part of induction base case and are always treated, if included.
- `limitclause` - Limit clause in inductive definitions: do you want inductive definitions to have a "Nothing else is a . . . " clause?
- `tagTrue` - a true tag respectively.
- `TMs` - Turing machines have been discussed
- `lambda` - Lambda calculus has been discussed
- prfND, prfSC - alternative coverage of proof systems. Set `prfND` to true if you want to include material on natural deduction, and `prfSC` for the sequent calculus. By default, we include both.