A Logiweb base page for IJCAR06

Klaus Grue

GRD-2006-02-24.UTC:10:23:46.350024

Contents

1 Introduction

This is a Logiweb *base page*[1]. The Logiweb system [6] is able to bootstrap using a page like the present one. Logiweb itself is a system for machine verification and distribution of formal mathematics.

The present page is a slightly modified version of the 'official' base page, constructed with the sole purpose of meeting a deadline for the IJCAR06 conference.

The base page contains quite a number of elementary definitions:

- On top of the Logiweb computing engine, the base page defines elementary operations on lists and cardinals (where *cardinal* means *natural number*).
- On top of the operations on lists and cardinals, the base page defines a macro expansion facility and a number of useful macros.
- On top of the operations on lists and cardinals, the base page defines a proof checker and a number of useful mathematical theories.
- Finally, the base page defines how to render the syntactic constructs defined on the page in the pyk and TEX languages.

1.1 The pyk language

From the point of view of Logiweb, the *pyk* language is a source language. A user may express a Logiweb page in pyk and then run the pyk source through the pyk compiler to obtain a Logiweb page.

A good way to learn the pyk language is to locate the pyk source of the present page and study it. The pyk source is likely to be in a file named "base.pyk". Plentiful comments in the pyk source explain what is going on.

The name "pyk" is constructed from the name "Volapyk" in the same way that Rene Thom construct the word "versal" from "universal": "pyk" is constructed by removing "Vola" from "Volapyk".

Volapyk was an artificial language constructed from several other languages by simplifying their words and their grammar. As an example, the name of the

[1] For the convenience of the reader, things that enter the index are in italics

language itself is constructed from "Vola" which is a simplification of "World" and "pyk" which is a simplification of "speak".

The pyk language may be used for "spoken mathematics" and may, among other, be entered through a microphone when editing mathematical text. The language is partly called "pyk" because of this speach aspect, partly because it looks like Volapyk in the other sense of that word. As an example, in a proper setup, the pyk phrase

> parenthesis var x plus var y end parenthesis square equals var x square plus two times var x times var y plus var y square

may correspond to the formula

$$(x + y)^2 = x^2 + 2xy + y^2$$

1.2 The TEX language

From the point of view of Logiweb, TEX is an output format intended for the creation of beautiful Logiweb pages—and especially for Logiweb pages that contain a lot of mathematics (c.f. the preface of the TEXbook [7]). The PDF version of the present page is produced by the TEX system (including LATEX) and a program named dvipdfm.

TEX is chosen for this purpose because of the high quality and great maturity of that system. Early version of Logiweb also used MathML, but support for that was removed for several reasons, one of which was a desire to reduce the number of output formats to exactly one.

One application of Logiweb could be straightforward translation of pyk to TEX. A major purpose of Logiweb, however, is to allow the Logiweb system to understand the mathematics present on Logiweb pages. In particular, Logiweb is able to check proofs and execute programs defined on Logiweb pages.

1.3 The attributes of a Logiweb page

In general, a page has several attributes:

The reference of a page is a cardinal that identifies the page. The reference of a page is world-wide unique. The Logiweb system includes Logiweb servers that can translate Logiweb references to Uniform Resource Locators (URLs) so that, having the reference of a page, one can locate the page without knowing which physical mirrors hold a copy of the page.

The vector of a page is the sequence of bytes that encode the page when it is stored on disk or transmitted over a network. All attributes of a page, including the reference, can be computed from the vector.

The bibliography of a page is a list of references to Logiweb pages. Entry number zero of the bibliography (i.e. the first element of the list of references) is the reference of the page itself. Appart from this self-reference, Logiweb pages and bibliographic references form a directed, acyclic graph.

The dictionary of a page is a list of all concepts defined on the page.

The body of a page comprises all the text and definitions of the page. When a user wants to read a Logiweb page, the system renders the body of the page using the TEX system and shows that to the user. What you read right now is such a rendering.

The expansion of a page is the macro expanded version of the body. Proof checking is done after macro expansion.

The codex of a page is an associative structure for fast lookup of all definitions made on a page.

The cache of a page is an associative structure for fast lookup of any attribute of any referenced page.

The diagnose (if any) of a page indicates what is wrong with the page (if anything).

1.4 Base pages

A Logiweb *base page* is a page whose Logiweb bibliography references no other pages.

As mentioned, Logiweb bibliographies form directed, acyclic graphs, so if one follows bibliographic references one can be sure to end up in a base page eventually.

What you read right now is the body of a base page. The body does not include the Logiweb bibliography, so you cannot verify that the present page is a base page by reading the present text. The bibliography at the end of the present body is an ordinary BIBTEX bibliography which is unrelated to the Logiweb bibliography.

To see the Logiweb bibliography of a page, you must view the page in a Logiweb browser and open the bibliography. If the page is generated from a pyk source, you may also see the Logiweb bibliography in that source.

2 Elementary definitions

2.1 Proclamations

2.1.1 Loading

When Logiweb reads a Logiweb page in order to "understand" it, we shall say that Logiweb *loads* the page. Logiweb loads a page as follows:

Resolving Given the reference of the page, Logiweb uses the mesh of Logiweb servers to *resolve* the reference, i.e. to locate a mirror that holds a copy of the page. That mirror typically is the server of the author of the page, but important pages may exist many places.

Retrieving Once the reference is resolved into a Uniform Resource Locator (URL), Logiweb *retrieves* the vector of the page, i.e. the sequence of bytes that encode the page when it is stored on disk or transmitted over a network.

Unpacking Once the vector is retrieved, Logiweb *unpacks* it into a bibliography, a dictionary, and a body. During this process, Logiweb recursively loads all pages referenced in the bibliography of the page.

Codifying Once the bibliography, dictionary, and body are available, Logiweb *codifies* the body. Logiweb does so by reading the body over and over again (c.f. Section 4.4.7). During these iterations, Logiweb is supposed to obtain a deeper and deeper "understanding" of the page. The iterations end when the understanding reaches a fixed point. This is similar to TEX that has to read a TEX source over and over again to get the references right. The outcome of the codification is a *codex* and an *expansion*. The expansion is a macro expanded version of the body. The codex is an associative structure for fast lookup of all definitions made on a page.

Verifying Once the page is codified, Logiweb *verifies* the page. Verification involves execution of the *claim* of the page as described later. The claim of a page typically runs a proof verifyer on all proofs on the page and typically also does several other chores to ensure that the formal mathematics presented on the page is correct.

Once a page is loaded, Logiweb can render it using TEX to produce a human readable version of it.

As a more exotic facility, Logiweb can also render a page in pyk to produce a pyk source file from which the page can be recompiled. This corresponds to "view source" when viewing html-pages but is more complex since Logiweb uses a compact, binary format from which the source has to be reverse engineered.

Finally, Logiweb is supposed to be able to *execute* the page, but that is not implemented at the time of writing. The execution facility will be a general programming facility including general I/O facilities. Logiweb itself is ultimately intended to be implemented using this facility. At the time of writing, Logiweb is implemented in Lisp.

2.1.2 Revelations

Logiweb has three kinds of *revelation* constructs that connect syntactic constructs with semantic concepts. The three kinds of revelations are called *proclamations*, *definititions*, and *introductions*.

Logiweb has a small number of predefined concepts; a proclamation connects a syntactic construct with one of these predefined concepts. Definitions and introductions allow to connect syntactic constructs with user defined concepts.

On the present page, proclamations are made using the *proclamation construct* [x ⋈ y][2]. As we shall see in Section 2.1.6, [x ⋈ y] proclaims [x] to denote the concept identified by [y].

2.1.3 Logiweb symbols

Every Logiweb construct is identified by a *reference* and an *identifier*, both of which are cardinals (i.e. natural numbers). The reference of a construct equals the reference of the page that introduces the construct. The identifier identifies the operation among all operations introduced by that page.

The reference of the [x ⋈ y] construct equals the reference of the present page (which is a big number) and the identifier equals one. As we shall see, it is important to Logiweb that the identifier equals one.

We shall refer to a pair consisting of the reference and identifier of a Logiweb construct as a Logiweb *symbol*. Logiweb symbols resemble Common Lisp symbols [11] which is the same as Lisp atoms in the original sense of the word [9].

There is a one-to-one correpondence between Logiweb symbols and Logiweb constructs. But the concepts are not identical; Logiweb constructs are things one may encounter on Logiweb pages, Logiweb symbols are pairs of integers that represent the constructs.

2.1.4 Preconceived opinions

To begin with, and with two exceptions, Logiweb does not assign any particular meaning to any particular construct. The general rule that Logiweb has no preconceived opinion about the meaning of constructs makes Logiweb flexible and ensures the notational freedom of each, individual author. The exceptions allow Logiweb to bootstrap.

The two exceptions relate to *page constructs* and proclamation constructs.

A page construct is a Logiweb construct whose identifier equals zero. There is a one-to-one correspondence between Logiweb pages and Logiweb page constructs: Given the reference of a page, one just adds a zero to get its page construct, and given a page construct one just removes the zero to get the reference of the page.

Page constructs are used whenever there is a need to reference a page in a context that requires a symbol. As an example, Logiweb allows to assign names (i.e. pyk names) to symbols. Logiweb does not allow to assign names to pages. But every page has a page symbol, and the name of the page symbol effectively becomes the name of the page.

Page constructs are unrelated to the bootstrapping of Logiweb. Rather, bootstrapping depends on proclamation constructs.

[2]For the convenience of the reader, mathematics is enclosed in brackets to distinguish it clearly from other text. Such use of brackets is a stylistic choice which is independent of Logiweb.

2.1.5 Bootstrapping and proclamation constructs

As mentioned in Section 2.1.1, Logiweb codifies a page by reading its body over and over again. At each reading, Logiweb "understands" the page in the light of what Logiweb already knows. This knowledge includes anything Logiweb managed to extract from pages referenced in the bibliography and anything Logiweb managed to extract from the present page at the previous reading.

In one situation, Logiweb has no prior knowledge when reading a page. That happens when Logiweb reads a base page first time (recall that a base page is a Logiweb page that references no other pages).

To get started, Logiweb has the preconceived opinion that, on first reading of a base page, the symbol whose identifier equals one is a proclamation symbol.

The proclamation construct in turn has the preconceived opinion that Logiweb constructs with identifiers from [97] to [122] denote the small letters from a to z and that certain sequences of such letters denote particular concepts. How the constructs with identifiers from [97] to [122] are used to form strings is treated in Section 2.4.

2.1.6 Self-proclamation

The first task of the proclamation symbol is to secure itself. Logiweb only has the preconceived opinion that the symbol whose identifier equals one is a proclamation symbol on first reading of a base page. On the second reading, Logiweb has no such preconceived opinion. For that reason, we make the following proclamation:

$$[[\mathsf{x} \bowtie \mathsf{y}] \bowtie \text{“proclaim”}]$$

The proclaimation construct happens to have the preconceived opinion that the string "proclaim" denotes the proclamation concept. Hence, during first reading of the present base page, Logiweb sees that $[\mathsf{x} \bowtie \mathsf{y}]$ is proclaimed to denote proclamation and, hence, $[\mathsf{x} \bowtie \mathsf{y}]$ denotes proclaimation during second reading of the base page. During second reading, $[\mathsf{x} \bowtie \mathsf{y}]$ is again proclaimed to denote proclamation and, hence, $[\mathsf{x} \bowtie \mathsf{y}]$ denotes proclaimation during third reading and so on.

Readers with plenty of spare time may find fun in constructing a base page for which the constructs whose identifiers equal two and three end up being proclamation constructs whereas the construct whose identifier equals one ends up not being a proclamation construct.

2.2 Definitions

2.2.1 Aspects

As mentioned in Section 2.1.2, Logiweb has three kinds of revelations: proclamations, definititions, and introductions. We now proclaim $[\mathsf{y} \xrightarrow{\mathsf{x}} \mathsf{z}]$ as a construct for definitions:

$$[[y \xrightarrow{x} z] \bowtie \text{“define”}]$$

The construct $[y \xrightarrow{x} z]$ states that the [x] *aspect* of [y] equals [z].

A mathematical definition like $[r(x,y) \doteq \sqrt{x^2+y^2}]$ corresponds to a Logiweb definition in which the value aspect of [r(x, y)] is defined to be $[\sqrt{x^2+y^2}]$. In general, ordinary, mathematical definitions correspond to Logiweb value definitions. Logiweb, however, allows the user to define a variety aspects of each construct.

2.2.2 The pyk aspect

A particular important aspect is the *pyk aspect*. We use [pyk] to denote the pyk aspect:

$$[\text{pyk} \bowtie \text{“pyk”}]$$

The pyk aspect of a construct indicates how to render the construct in the pyk language.

Until further, we have used the following constructs: $[x \bowtie y]$[3], $[y \xrightarrow{x} z]$[4], [pyk][5], [x][6], [y][7], and [z][8].

The footnote of $[x \bowtie y]$ reads

$$[[x \bowtie y] \overset{\text{pyk}}{=} \text{“proclaim} * \text{as} * \text{end proclaim”}]$$

As we shall see later, $[[x \bowtie y] \overset{\text{pyk}}{=} \text{“proclaim} * \text{as} * \text{end proclaim”}]$ macro expands into

$$[[x \bowtie y] \overset{\text{pyk}}{\rightarrow} \text{“proclaim} * \text{as} * \text{end proclaim”}]$$

which in turn defines the pyk aspect of $[x \bowtie y]$ as the string “proclaim * as * end proclaim”.

$[[x \bowtie y] \overset{\text{pyk}}{=} \text{“proclaim} * \text{as} * \text{end proclaim”}]$ differs from $[[x \bowtie y] \overset{\text{pyk}}{\rightarrow} \text{“proclaim} * \text{as} * \text{end proclaim”}]$ in a few places: The former protects the left hand side against macro expansion. That is important when defining the pyk aspect of macros. Furthermore, the former ensures that the right hand side is typeset as a string. Looking into the pyk source of the present page reveals that special measures are taken to get the string look right in $[[x \bowtie y] \overset{\text{pyk}}{\rightarrow} \text{“proclaim} * \text{as} * \text{end proclaim”}]$.

---

[3] $[[x \bowtie y] \overset{\text{pyk}}{=} \text{“proclaim} * \text{as} * \text{end proclaim”}]$

[4] $[[y \xrightarrow{x} z] \overset{\text{pyk}}{=} \text{“define} * \text{of} * \text{as} * \text{end define”}]$

[5] $[\text{pyk} \overset{\text{pyk}}{=} \text{“pyk”}]$

[6] $[x \overset{\text{pyk}}{=} \text{“var x”}]$

[7] $[y \overset{\text{pyk}}{=} \text{“var y”}]$

[8] $[z \overset{\text{pyk}}{=} \text{“var z”}]$

Note that the pyk aspect of $[\mathsf{x} \bowtie \mathsf{y}]$ has been defined seven times in total until now (once in a footnote and six times in the text). Logiweb just uses the definition that is leftmost after macro expansion. The pyk compiler is a little more touchy: it prints a warning if there are definitions that contradict one another. The pyk compiler accepts the seven pyk definitions of $[\mathsf{x} \bowtie \mathsf{y}]$ without notice since all the definitions are identical after macro expansion.

2.2.3 Uses of pyk

The pyk definitions stated so far allow to write e.g.

proclaim define var x of var y as var z end define as $\cdots$ end proclaim

in a pyk source file to obtain

$[[\mathsf{y} \xrightarrow{\mathsf{x}} \mathsf{z}] \bowtie \text{“define”}]$

In principle, the pyk language is not part of Logiweb. The Logiweb standard merely defines how to interpret Logiweb vectors, i.e. Logiweb pages on binary format. The only formal link between Logiweb and pyk is the fact that Logiweb has a predefined concept that is proclaimable under the name of “pyk”. The pyk language and compiler are just means for producing Logiweb pages. One could imagine other means for producing Logiweb pages that were completely independent of pyk.

Nevertheless, authors who publish on Logiweb are encouraged to give a pyk definition of each and every construct that is introduced on the page. In the future, the pyk definitions may be used for entering mathematics through a microphone or for reading mathematics e.g. for blind people.

At the time of writing, pyk definitions have a much more immediate use as explained in the following.

At the time of writing, the pyk compiler is the only realistic means for producing Logiweb pages. The pyk compiler takes a pyk source text as input and produces a Logiweb page as output. The pyk compiler also has facilities for rendering pages, for making various soundness checks, and for printing warnings and error messages.

A pyk source consists of a preamble and a body. The preamble defines the bibliography and dictionary of the page. The bibliography is a list of referenced pages and the dictionary is the list of all constructs introduced on the page. The preamble also defines other things like the priority and associativity of constructs and preliminary pyk definitions for all constructs.

When the pyk compiler reads the preamble, it loads all pages referenced in the bibliography. Then it extracts all pyk definitions from all referenced pages, merge them with the preliminary pyk definitions of the preamble, and use the resulting grammar for parsing the body of the page.

The pyk compiler does not care whether or not the resulting grammar is ambiguous as long as the body of the page has one and only one possible interpretation.

When the pyk compiler generates the Logiweb page, all the preliminary pyk definitions from the preamble are lost. Only pyk definitions explicitly included in the body by the author make their way to the resulting Logiweb page.

Hence, if you publish a Logiweb page and want to make the life easy for people who want to build on your work, you'd better include pyk definitions of each and every construct on your page. Do not omit constructs you think are unimportant or "internal to the page" in an information hiding sense. In the context of proof checking there is no such thing as an unimportant construct or an "internal construct" that one should not look at.

2.2.4 The tex aspect

The *tex aspect* is another important aspect. The tex aspect of a construct indicates how to render the construct in the TEX language. We use [tex] to denote the tex aspect:

$$[\mathsf{tex} \bowtie \text{“tex”}]^{9}$$

Tex definitions for the present page are collected in Appendix A.2. In that Appendix, one may find a definition like

$$[\mathsf{x} \overset{\mathsf{tex}}{=} \text{“}\backslash\mathsf{mathsf}\{\mathsf{x}\}\text{”}]$$

As we shall see later, $[\mathsf{x} \overset{\mathsf{tex}}{=} \text{“}\backslash\mathsf{mathsf}\{\mathsf{x}\}\text{”}]$ macro expands into

$$[\mathsf{x} \overset{\mathsf{tex}}{\rightarrow} \text{“}\backslash\mathsf{mathsf}\{\mathsf{x}\}\text{”}]$$

which in turn defines the tex aspect of [x] to be a string that starts with a *newline character* followed by \mathsf{x}.

Internally, Logiweb consistently uses Unicode 10 for the newline character. Whenever Logiweb runs on a host operating system that uses another character or character sequence between lines of text, newline characters have to be translated in the interface between Logiweb and the operating system.

The tex definition above states that whenever Logiweb renders an [x] using the TEX system, it dumps ↓\mathsf{x} to a file and runs it through TEX (where the down arrow represents the newline character).

As another example, the pairing construct [x :: y] introduced later has the following tex aspect:

$$[\mathsf{x} :: \mathsf{y} \overset{\mathsf{tex}}{=} \text{“}\#1.\ \backslash\mathrm{mathrel}\ \{\ :\ \backslash,\ :\ \}\#2.\text{”}]$$

[9] $[\mathsf{tex} \overset{\mathsf{pyk}}{=} \text{“tex”}]$

Arguments are represented by asterisks in pyk aspects. In tex aspects, arguments are represented by sequences of characters that start with a hash mark and end with a period. As an example, #117. denotes the 117'th argument of a construct. A hash mark immediately followed by a period as in "#." denotes a hash mark.

When sending text through TEX one has to be careful about newline characters. Firstly, one should avoid making lines that are too long for TEX or the host operating system. Secondly, one should avoid producing two newline characters in sequence since TEX assigns a special meaning to repeated newline characters.

To fulfill both, the present page makes the following convention for tex aspects: A newline character is added in front of the string whenever the string starts with a character rather than an argument. Furthermore, a newline character is inserted whenever a character follows an argument. Newlines are omitted, however, if they disturb TEX in achieving the intended rendering.

2.2.5 Interpretation of the tex aspect

Looking at the pyk source of the present page reveals that the tex aspect of the entire page looks like this:

```
File page.tex
...
End of file
File page.bib
...
End of file
latex page
makeindex page
bibtex page
latex page
makeindex page
latex page
```

The tex aspect above is not run directly through TEX. Rather, it instructs an interpreter to place certain text in certain files and then run latex, bibtex, and makeindex in a certain pattern.

For security reasons, the interpreter of the present system only allows execution of commands named tex, latex, bibtex, and makeindex. Furthermore, the interpreter only accepts file names made up from the characters a to z, A to Z, 0 to 9, and dots.

At present, security holes in tex, latex, bibtex, or makeindex may compromise security. For this and other reasons, these programs will run in a chroot gail in some future release of Logiweb.

2.2.6 The tex name aspect

Some constructs look different when using them and when talking about them.

As an example, consider ordinary typewriter text. When using a newline character, the character itself is invisible but has the effect that text following it starts on a new line. When talking about a newline character, one may call it "the newline character". This way, a newline character is invisible when using it and consists of 19 characters and two spaces when talking about it.

As another example, the TEX bold face command may be called `\bf` when talking about it and changes the text that follows it to bold face when using it.

To cope with this, we introduce a *tex name* or *name* aspect to supplement the tex aspect:

$$[\text{name} \bowtie \text{“texname”}]^{10}$$

The tex name aspect should be such that it can be typeset in TEX math mode. The tex name aspect defaults to the tex aspect, so if the tex and tex name aspects of a construct are identical then one should only define the tex aspect. (The tex aspect in turn defaults to something that is constructed from the pyk aspect, and the pyk aspect in turn defaults to something semi-readable constructed from the reference and identifier of the construct).

2.2.7 Uses of the tex name aspect

As an example of a construct for which the tex and tex name aspects differ, consider the following:

$$[\ \$x\$ \overset{\text{tex}}{=} \text{“}\$\#1.\$\text{”}]^{11}$$

$$[\ \$x\$ \overset{\text{name}}{=} \text{“}\ \backslash\ \backslash\$\#1.\backslash\$\backslash\text{linebreak}[0]\backslash\ \text{”}]$$

The "math * end math" construct allows to insert mathematics in TEX horizontal mode by changing to math mode temporarily. The tex name aspect allows to talk about the construct. The left hand sides of the three definitions above are typeset using the tex name aspect.

Elaborating the example from the previous section, the pyk source of a definition like

$$[[y \xrightarrow{x} z] \bowtie \text{“define”}]$$

could read something like

> math proclaim define var x of var y as var z end define as ⋯ end proclaim end math

[10] $[\text{name} \overset{\text{pyk}}{=} \text{“tex name”}]$

[11] $[\ \$x\$ \overset{\text{pyk}}{=} \text{“math} * \text{end math”}]$

The “math * end math” construct is invisible in the definition above but certainly affects the typography; without it, TEX would produce ugly error messages rather than beautiful typography.

By the way: note that “math * end math” changes to \rm immediately after changing to math mode. That is because Logiweb formulas contain loads of text and because the typography of e.g. variables is under tight control of Logiweb, so the defaults of TEX math mode are not the right ones for Logiweb.

2.2.8 Brackets

For completeness,

[bracket x end bracket $\stackrel{\text{tex}}{=}$ “$[#1.]$”][12]

and

[big bracket x end bracket $\stackrel{\text{tex}}{=}$ “$\left[#1.\right]$”][13]

are versions of “math * end math” that add brackets around the formula.

The latter adds large brackets using the large bracket facility of TEX, which is useful occasionally but which mainly leads to undesirable results. The large brackets in [5] adjust their height independently of their depth and allow line breaking of their argument. The brackets of [5] are not used here, however, because they would have an inconveniently large tex aspect.

By the way, two major Logiweb pages have been developed before Logiweb itself was developed. The first is [5] which is a three volume textbook on mathematics for first year computer science students used at the department of computer science at the University of Compenhagen as a replacement for discrete mathematics. The second is citegrue02b which contains a consistency proof for ZFC set theory expressed in Map theory. At the time of writing, these two are not yet published on Logiweb, but they have been drivers of the design of Logiweb.

2.2.9 The tex aspect of definitions

As stated in Appendix A.2, the TEX definition of [y $\xrightarrow{x}$ z] reads:

```
[[y -x-> z] =tex= “
[#2/tex name/tex.
\stackrel{#1.
}{\rightarrow}#3.
]”]
```

[12] [bracket x end bracket $\stackrel{\text{pyk}}{\longrightarrow}$ “bracket ∗ end bracket”]

[13] [big bracket x end bracket $\stackrel{\text{pyk}}{\longrightarrow}$ “big bracket ∗ end bracket”]

In the definition, "#1." says "insert the first argument (the aspect) here". Likewise, "#3." says "insert the third argument (the right hand side) here".

The second argument (the left hand side), is more complicated. A definition talks about the left hand side rather than just using it. For that reason, the left hand side should be rendered using the tex name aspect instead of the tex aspect. On the other hand, if the left hand side has parameters, then those parameters should be rendered normally, i.e. using the tex aspect.

"#2/tex name/tex." says "insert the second argument (the left hand side) here using the tex name aspect for the left hand side, but revert to the tex aspect for the parameters of the left hand side.

2.3 Associativity and priority

2.3.1 Introduction

A "preassociative" construct is left associative in text that runs left to right, right associative in text that runs right to left, top associative in text that runs from top to bottom, counterclockwise associative in text written in left turning spirals, and so on. Text on Logiweb pages may run in any conceivable direction. Pyk source text runs from left to right.

The pyk compiler uses associativities when parsing pyk source text. As an example, if the construct "* plus *" is preassociative then "var x plus var y plus var z" is interpretted as "(var x plus var y) plus var z. Likewise, if "* pair *" is postassociative then "var x pair var y pair var z" means "var x pair (var y pair var z)".

The pyk compiler also uses priorities. As an example, it "* plus *" has greater priority than "* pair *" then "var x plus var y pair var z" means "(var x plus var y) pair var z" and "var x pair var y plus var z" means "var x pair (var y plus var z)".

If two constructs have equal priority, then they are forced to have the same associativity as well. As an example, if "* minus *" has the same priority as "* plus *" then "* minus *" automatically becomes preassociative. Hence, "var x plus var y minus var z" means "(var x plus var y) minus var z" and "var x minus var y plus var z" means "(var x minus var y) plus var z".

2.3.2 Open and closed constructs

A construct is said to be "preopen" if it starts with an asterisk and to be "preclosed" otherwise. Likewise, a construct is "postopen" and "postclosed" if it does and doesn't end with an asterisk, respectively.

A construct is said to be "open" if it is pre- and postopen and "closed" if it is pre- and postclosed. A construct is said to be "prefix" if it is preclosed and post open and "suffix" if it is preopen and post closed. Here are some examples:

parenthesis * end	preclosed	postclosed	closed
* apply *	preopen	postopen	open
lambda * dot *	preclosed	postopen	prefix
factorial	preopen	postclosed	suffix

None of the four constructs above are declared on this page; they are just included for the example.

Associativity and priority is irrelevant for closed constructs.

Associativity does affect prefix and suffix constructs. As an example, if "* apply *", "lambda * dot *" and "* factorial" have the same priority, then "lambda var x dot var y apply var z factorial" means "((lambda var x dot var y) apply var z) factorial" if the constructs are preassociative and "lambda var x dot (var y apply (var z factorial))" if they are postassociative.

2.3.3 Priority interference

Prefixness and suffixness occasionally interfere with priorities. As an example, suppose "* apply *" has greater priority than "lambda * dot *" and consider the following pyk source:

var x apply lambda var y dot var y apply var y

First step in disambiguating the text above is to put parentheses around the operator that has the lowest priority and to put parentheses around its parameters:

var x apply (lambda (var y) dot (var y apply var y))

This divides the problem into two smaller problems: disambiguating "var x apply $\cdots$" and disambiguating "var y apply var y" both of which are trivial:

((var x) apply (lambda (var y) dot ((var y) apply (var y))))

Hence, the principal operator of "var x apply lambda var y dot var y apply var y" is the leftmost "apply" which may be surprising since the lambda is the operator with the lowest priority.

2.3.4 Openness coherence

When defining the pyk and tex aspects of a construct it is important to let them have the same "openness", i.e. to make them both open, both closed, both prefix, or both suffix. As an example, if one renders a pair as e.g. $x :: y$ which is open then one should ensure that the pyk aspect is also open as in "* plus *". If one renders the pair as e.g. (x, y) which is closed then one should ensure that the pyk aspect is closed as in "pair * comma * end pair".

Failure to make openness coherent may seriously baffle the reader since it may make the pyk compiler and a human reader interpret terms differently.

Some constructs are non-trivial to classify. As an example, exponentiation x^y is best classified as a suffix construct because $a + x^y$ could mean $(a + x)^y$ or $a + (x^y)$ whereas $x^y + a$ can only mean $(x^y) + a$ and, thus, x^y is ambiguous the way suffix constructs are. Hence, the pyk name of exponentiation should be something like "* power * end power".

2.3.5 Priority tables

A Logiweb page may contain constructs from the page itself as well as constructs from all pages it references in its Logiweb bibliography.

The pyk compiler insists on knowing the associativity and priority af each non-closed construct that might possibly appear on a page. The pyk compiler is somewhat relaxed about whether or not priorities and associativities are specified for closed constructs because the associativities and priorities of such constructs have no effect.

To avoid double work, a Logiweb page should contain a priority table which the pyk compiler can import when the page is included in the Logiweb bibliography of another Logiweb page.

A page should contain one priority table, not one for each construct. Hence, the priority table is an aspect of the page rather than an aspect of each, individual construct. This is a problem because Logiweb allows to define aspects of constructs but does not allow to define aspects of pages. So what do we do?

2.3.6 The page symbol

As mentioned, every construct in Logiweb is identified by a reference [r] and an id [i], both of which are cardinals. The reference uniquely identifies the Logiweb page that introduces the construct and the id identifies the construct within the page. We shall refer to a construct with reference [r] and id [i] as the [i]'th construct of page [r].

The zero'th construct of a Logiweb page will be referred to as the "page construct" of the page. Aspects of the page construct should be thought of as aspects that concern the entire page.

As an example, consider the following pyk definition of the page construct of the present page:

[ijcar base $\stackrel{\text{pyk}}{\longrightarrow}$ "ijcar base"]

Because of the definition above we shall say that the present page is named "ijcar base".

To see that [ijcar base] is the page construct of the present page one has to open the present page in a Logiweb browser, find the id of the construct, and see that the id is zero. In the Logiweb crossbrowser, one may do that by opening the "dictionary" window. More advanced browsers would probably allow the user to click on [ijcar base] and view its properties somehow.

We do not define a tex aspect of the page construct, and for that reason the tex aspect defaults to `\mathrm{ijcar base}`. We do not define a tex name aspect either, and for that reason the tex name aspect defaults to be identical to the tex aspect.

The page construct is forced to have arity zero, i.e. it has no parameters.

2.3.7 The priority aspect

The priority table is defined as a "priority aspect" of the page construct. We introduce the priority aspect thus:

[prio $\bowtie$ "priority"]

[prio $\stackrel{\text{pyk}}{\longrightarrow}$ "priority"]

[prio $\overset{\text{tex}}{\to}$ “
\mathrm{prio}”]

2.3.8 Flush left environment

The priority table is a long, heterogeneous formula which is difficult to linebreak. For that reason we shall typeset it *ragged right* or *flush left*. To do so, we introduce a [**flush left** [x]][14]construct which typesets its argument with a ragged right margin.

2.3.9 The priority table

For convenience, we introduce a variable named [∗] which we use in pyk, TEX, and priority definitions where parameter names are ignored.

The priority table itself is given in Appendix C. The number of constructs in it is colossal, but that is normal for large base pages since the table contains lots of trivia like characters and variable names.

2.3.10 Priority table constructors

The priority table in Appendix C is constructed from the following four constructs:

[**Preassociative** x; y $\overset{\text{pyk}}{\to}$ “preassociative ∗ greater than ∗”]

[**Postassociative** x; y $\overset{\text{pyk}}{\to}$ “postassociative ∗ greater than ∗”]

[[x], y $\overset{\text{pyk}}{\to}$ “priority ∗ equal ∗”]

[priority x end $\overset{\text{pyk}}{\to}$ “priority ∗ end priority”]

The pyk source of the table looks something like

```
math define priority of ijcar base as

preassociative
priority bracket var x end bracket equal
priority math var x end math equal
...
priority priority end priority

greater than preassociative
priority unicode start of text var x end unicode text equal
...
priority priority var x end priority end priority

greater than postassociative
priority var x then var y equal
priority var x begin var y end var z end priority
```

[14] [**flush left** [x] $\overset{\text{pyk}}{=}$ “flush left ∗ end left”]

```
greater than ijcar base end define end math
```

From the structure of the table, the pyk compiler can guess that "priority * equal *" denotes "equal priority" and that "priority * end priority" marks the end of a list of equal priorities.

The pyk compiler can also guess that "preassociative * greater than *" and "postassociative * greater than *" denote pre- and postassociativity but cannot guess which is which. For that reason, we have to reveal that to Logiweb explicitly using proclamations:

[**Preassociative** x; y ⋈ "pre"]

[**Postassociative** x; y ⋈ "post"]

2.3.11 Tex aspects

Tex aspects of priority table constructors read:

[**Preassociative** x; y $\overset{\text{tex}}{\to}$ "
\newline \mathbf {Preassociative} \newline #1.
; #2."]

[**Preassociative** x; y $\overset{\text{name}}{\to}$ "
\mathbf{Preassociative}\,#1.
; #2."]

[**Postassociative** x; y $\overset{\text{tex}}{\to}$ "
\newline\mathbf{Postassociative} \newline #1.
; #2."]

[**Postassociative** x; y $\overset{\text{name}}{\to}$ "
\mathbf{Postassociative}\,#1.
; #2."]

[[x], y $\overset{\text{tex}}{\to}$ "
[#1/tex name/tex.
] , \linebreak [0] #2."]

[[x], y $\overset{\text{name}}{\to}$ "
[#1.
] , \linebreak [0] #2."]

[priority x end $\overset{\text{tex}}{\to}$ "
[#1/tex name/tex.
]"]

[priority x end $\overset{\text{name}}{\to}$ "
\mathrm{priority} \,#1.
\, \mathrm{end}"]

2.4 Strings

2.4.1 Introduction

The pyk language itself is a rather purist language that merely uses the small letters from a to z and space and newline characters for expressing Logiweb pages of arbitrary complexity. Capital letters and punktuation marks have been omitted to prepare the language for input via a microphone (it is easy, though, to adapt pyk to allow letters from other alphabets).

Furthermore, the rather purist Logiweb computing engine described elsewhere has sufficient power to render pages all the way from layout to pixels without resorting to external systems like TEX or BIBTEX or whatever. Or, at least, programs like TEX may be translated to the Logiweb language, placed on a Logiweb page, and executed from there by the Logiweb engine.

During the early phases of the development of Logiweb, however, too much purism is unaffordable. TEX is a very mature system that cannot be replaced overnight. To allow Logiweb pages to include TEX source, the pyk language has support for strings.

The treatment of strings in the following should make sense in itself, but the motivation for going through the pain of introducing strings requires quite some overview of how all the parts of Logiweb play together.

2.4.2 Construction of strings

As mentioned earlier, every construct in Logiweb is identified by a reference [r] and an id [i], both of which are cardinals.

We shall say that a construct is "blind" if its id is between zero and nine, inclusive, or between 11 and 31, inclusive. We shall say that constructs whose ids equal 10 or are greater than 31 are non-blind.

When used in strings, non-blind constructs represents the character whose unicode equals the id of the construct. Blind constructs represent no characters.

As an example, The id's of the pyk construct "unicode small a *", "unicode small b *", "unicode small c *", and "unicode end of text" equal 97, 98, 99, and 3, respectively (to see that one needs access to the pyk source of the present page or needs to view the page in a Logiweb browser). The expression

> unicode small a unicode small b unicode small c unicode end of text

is interpretted as

> unicode small a (unicode small b (unicode small c (unicode end of text)))

When read from left to right, the id's of the constructs are 97, 98, 99, and 3. Ignoring 3 which is blind, we are left with 97, 98, and 99 which are the unicodes for "a", "b", and "c", respectively. For that reason,

> unicode small a unicode small b unicode small c unicode end of text

represents the string "abc".

2.4.3 Pyk strings

Continuing the previous example, the id of the pyk construct “unicode start of text * end unicode text” equals two. Whenever the pyk compiler sees a string like `"abc"` inside a mathematical expression, it translates it to

```
unicode start of text
  unicode small a
  unicode small b
  unicode small c
  unicode end of text
end unicode text
```

Ignoring blind constructs, the expression above represents the string “abc”.

Logiweb itself assigns no particular semantics to constructs like “unicode start of text * end unicode text” and “unicode end of text” whose ids equal two or three. But the pyk compiler consistently add constructs with these particular ids whenever it translates a string.

The representation of Logiweb pages is such that characters with codes below 128 take up one byte. A string like “abc” takes up five bytes because the start and end of text characters also take up one byte.

2.4.4 Rendering of strings

Rendering of “a”, “b”, and “c” is completely straighforward:

[ax $\overset{\text{tex}}{\to}$ “a#1.”]

[bx $\overset{\text{tex}}{\to}$ “b#1.”]

[cx $\overset{\text{tex}}{\to}$ “c#1.”]

Pyk and tex definitions for “a”, “b”, “c”, and many other non-blind characters are collected in the appendix.

The tex aspects of the start of text character clearly show that it is blind:

[“x” $\overset{\text{tex}}{\to}$ “#1.”]

However, when talking about the start of text character it is inconvenient if it is blind. Instead, we represent it by double quotes:

[“x” $\overset{\text{name}}{\to}$ “
\mbox{‘‘}#1.
\mbox{’’}”]

We shall take the liberty to make the end of text character completely blind:

[$\overset{\text{tex}}{\to}$ “”]

As you can see (or, rather, cannot see), there are no glyphs in the left hand side of the definition above. That indicates that the left hand side comprises a unicode end of text. We shall define no other constructs that are completely blind, i.e. blind when talking about them. Lots of constructs are blind when using them, but the unicode end of text is the only one that is blind when talking about it. The unicode end of text should be used with caution is it may baffle the reader.

For completeness, the pyk aspects of the start and end of text characters read:

[$\overset{\text{pyk}}{\rightarrow}$ "unicode end of text"]

[“x” $\overset{\text{pyk}}{\rightarrow}$ "unicode start of text $*$ end unicode text"]

2.4.5 Rendering of special characters

The rendering of a backslash reads:

[\x $\overset{\text{tex}}{\rightarrow}$ "\#1."]

[\x $\overset{\text{name}}{\rightarrow}$ "
\mbox{$\backslash$}#1."]

The tex aspect of a backslash generates a backslash character in the input to TEX. In contrast, the tex name aspect generates a backslash in the output from TEX.

2.4.6 Talking about strings

The id of the pyk construct "text * end text" equals six so it is yet another blind construct. The construct is rendered thus:

[$(\mathsf{x})^{\mathbf{t}}$ $\overset{\text{pyk}}{\rightarrow}$ "text $*$ end text"]

[$(\mathsf{x})^{\mathbf{t}}$ $\overset{\text{tex}}{\rightarrow}$ "\#1/tex name."]

[$(\mathsf{x})^{\mathbf{t}}$ $\overset{\text{name}}{\rightarrow}$ "
(\#1.
)\^{}{\bf t}"]

The tex aspect of "text * end text" is invisible in itself but has the effect that all of its argument is rendered using the tex name aspect. As an example, the expression

```
text
  unicode start of text
    unicode backslash
    unicode small b
    unicode small f
    unicode end of text
  end unicode text
end text
```

generates a start double quote, a backslash, a "b", an "f", and an end double quote in the output from TEX. Without the "text * end text", the expression generates `\bf` in the input to TEX which makes TEX change to bold face.

2.4.7 Breaking strings

The id of the pyk construct "text * plus *" equals seven and the construct is rendered thus:

[string(x) + y $\overset{\text{pyk}}{\rightarrow}$ "text $*$ plus $*$"]

[string(x) + y $\overset{\text{tex}}{\rightarrow}$ “
\mathrm{#1/tex name.
}+\newline#2.”]
[string(x) + y $\overset{\text{name}}{\rightarrow}$ “
\mbox{string}(#1.
)+#2.”]
A term like

```
text
  unicode start of text
    unicode small a
    unicode small b
    unicode end of text
  end unicode text
plus
  text
    unicode start of text
      unicode small c
      unicode end of text
    end unicode text
  end text
```

denotes the string “abc”. This is so because if one scans the text left to right (or scans the associated parse tree root to leaf, left to right) then one encounters the id’s 7, 2, 97, 98, 3, 6, 99, 3 in that order. Ignoring blind codes leaves one with the sequence 97, 98, 99 which represents “abc”.

The rendering of the string consists of “ab” on one line and “c” on the next. The “text * plus *” construct allows the author of a Logiweb page to split a long string over several lines.

The id of the “text * plus indent *” construct equals eight; the construct does the same as “text * plus *” but also indents the second line:

[string(x) ++ y $\overset{\text{pyk}}{\rightarrow}$ “text $*$ plus indent $*$”]
[string(x) ++ y $\overset{\text{tex}}{\rightarrow}$ “
\mathrm{#1/tex name.
}\;{++}\newline{}\qquad#2.”]
[string(x) ++ y $\overset{\text{name}}{\rightarrow}$ “\mbox{string}(#1.
)\mathrel{++}#2.”]

2.4.8 Formulas in text

The id of the pyk construct “* begin * end *” equals five and the construct is rendered thus:

[x[y]z $\overset{\text{pyk}}{\rightarrow}$ “$*$

begin $*$

end ∗”]
[x[y]z $\overset{\text{tex}}{\rightarrow}$ “#1.#2.#3.”]
[x[y]z $\overset{\text{name}}{\rightarrow}$ “#1.
{[}#2.
{]}#3.”]

As an example of use, the pyk source

```
"abc" begin math var x end math end "def"
```

is equivalent to

```
  unicode start of text
    unicode small a
    unicode small b
    unicode small c
    unicode end of text
  end unicode text
begin
  math var x end math
end
  unicode start of text
    unicode small d
    unicode small e
    unicode small f
    unicode end of text
  end unicode text
```

The tex face of the expression above reads

```
abc$\mathsf{x}$def
```

which, when run through TEX, places the variable “x” in the middle of the string “abcdef”.

In general, the “* begin * end *” construct allows to place mathematical formulas in text.

As a violation of purism, the pyk compiler allows the construct “*[*]*” to occur in pyk source files. Furthermore, the pyk compiler translates such constructs into constructs whose id equals five. For that reason one may write

```
"abc"[ math var x end math ]"def"
```

in place of the pyk source above.

Due to a peculiarity in the current implementation of the pyk compiler, the space between the left bracket and the “m” in “math” above is obligatory.

2.4.9 Juxtaposition

The id of the pyk construct “* then *” equals four and the construct is rendered thus:

[x then y $\overset{\text{pyk}}{\rightarrow}$ “∗

then ∗”]

[x then y $\overset{\text{tex}}{\rightarrow}$ “#1.#2.”]

[x then y $\overset{\text{name}}{\rightarrow}$ “#1.
\mathrel{\mathrm{then}}#2.”]

As a violation of purism, the pyk compiler allows the construct “*,*” to occur in pyk source files. Furthermore, the pyk compiler translates such constructs into constructs whose id equals four. For that reason one may write

```
var x, "\cdots", var y
```

to obtain [x · · · y].

2.4.10 Purism versus pragmatism

Purism is a good thing when defining, designing, implementing, debugging, testing, explaining, teaching, learning, understanding, and using a computational system, and one should only resort to pragmatism when neccessary. The pyk compiler is rather puristic, but resorts to pragmatism in the treatment of strings: Double quotes, brackets, and commas are treated specially and constructs whose id equal two, three, four, and five are used when translating those special characters.

3 The Engine

This section describes the Logiweb computing engine.

The Logiweb engine is the computing machinery that performs proof checking, macro expansion, and general evaluation in the Logiweb system.

At a later stage of the development of Logiweb, the engine is supposed to take over the layout and rendering of Logiweb pages. As mentioned previously, that task is currently done by TEX[7] and related programs.

The engine is just a computing engine; it has no facilities to communicate with the outside world. The Logiweb “machine” described elsewhere contains the Logiweb engine and also contains facilities for input/output.

Apart from introducing the engine, the present page also introduces a number of operations suited for manipulation of “tagged trees”. Tagged trees are to Logiweb what Lisp S-expressions are to Lisp[9].

3.1 Elementary concepts

3.1.1 Fundamental, computable constructs

The Logiweb engine has four fundamental, computable constructs: lambda abstraction [λx.y][15], functional application [x ' y], truth [T], and conditional [if(x, y, z)].

[λx.y] denotes lambda abstraction[2]. Lambda abstraction is a *predefined concept* of Logiweb in the sense that knowledge of lambda abstraction is hardwired into Logiweb. Logiweb does not connect lambda abstraction with any, particular syntax, so we must connect the syntactic construct [λx.y] with the concept of lamdba abstraction. We do so using a proclamation: [λx.y $\bowtie$ "lambda"].

The proclamation [λx.y $\bowtie$ "lambda"] makes [λx.y] denote lambda abstraction on the present Logiweb page and on all Logiweb pages that include the present page in their Logiweb bibliography. This does not exclude other constructs from denoting lambda abstraction as well. One may proclaim any number of binary constructs to denote lambda abstraction. Neither does this exclude the possibility of using the notation [λx.y] for other concepts than lambda abstraction on other Logiweb pages.

[x ' y] denotes functional application. As an example, if [x] equals [λz.z + 3] then [x ' 2] equals [2 + 3] which in turn equals [5]. We connect syntax and semantics of functional application thus: [x ' y $\bowtie$ "apply"].

All computable functions can be expressed in *pure lambda calculus*, i.e. using only [λx.y] and [x ' y]. Doing so is inconvenient, however. As an example, the range of any function of pure lambda calculus contains one or infinitely many values, and one often wants a computer program to have a finite range of possible responses. Just think of a computer program which is required to loop indefinitely or answer "yes" or "no" for arbitrary input, and which is required to answer "yes" in certain situations and "no" in certain, other, situations. Such a computer program cannot be implemented in pure lambda calculus since its range is finite and contains more than one possible response.

One symptom of the inconvenience of pure lambda calculus is the difficulty of constructing models for it. Another symptom is the inability so far to construct a foundation of mathematics upon pure lambda calculus.

If one adds two more constructs, truth [T] and conditional [if(x, y, z)], then the problems disappear: One may express functions like [λx.if(x, y, z)] whose range is finite, one may construct rather simple models[1], and one may extend the system into a foundation of mathematics by the drop of a quantifier[4].

[$\mathsf{T} \bowtie$ "true"] denotes truth. The main property of [T] is that it differs from [λx.y] for all variables [x] and all terms [y] (of course, we make the convention that [λx.y] represents falsehood for all variables [x] and all terms [y]).

Since [T] is no function, it does not make sense to apply it to an argument. For completeness, however, we make the convention that [T] applied to anything

[15] For the convenience of the reader, mathematics is enclosed in brackets to distinguish it clearly from other text. Such use of brackets is a stylistic choice which is independent of Logiweb.

equals [T] itself: $[\mathsf{T}\,\text{'}\,\mathsf{x} = \mathsf{T}]$.

$[\mathrm{if}(\mathsf{x},\mathsf{y},\mathsf{z}) \bowtie \text{“if”}]$ is a conditional which allows to test whether [x] equals [T] or is a function:

$$\left[\begin{array}{lcl} \mathrm{if}(\mathsf{T},\mathsf{u},\mathsf{v}) & = & \mathsf{u} \\ \mathrm{if}(\lambda\mathsf{x}.\mathsf{y},\mathsf{u},\mathsf{v}) & = & \mathsf{v} \end{array}\right]$$

3.1.2 Reduction system

The Logiweb *reduction system* is thus:

$$\left[\begin{array}{lcl} (\lambda\mathsf{x}.\mathsf{y})^{\mathrm{I}}\,\text{'}\,\mathsf{z} & \xrightarrow{+} & \langle \mathsf{y}\,|\mathsf{x}{:=}\mathsf{z}\rangle \\ \mathsf{T}\,\text{'}\,\mathsf{z} & \xrightarrow{+} & \mathsf{T} \\ \mathrm{if}(\mathsf{T},\mathsf{u},\mathsf{v}) & \xrightarrow{+} & \mathsf{u} \\ \mathrm{if}(\lambda\mathsf{x}.\mathsf{y},\mathsf{u},\mathsf{v}) & \xrightarrow{+} & \mathsf{v} \end{array}\right]$$

$\langle \mathsf{y}\,|\mathsf{x}{:=}\mathsf{z}\rangle$ denotes the result of replacing all free occurrences of the variable [x] in the term [y] by the term [z], possibly renaming bound variables as needed.

We shall say that a term [z] is on *truth normal form* if the term [z] is identical to [T].

We shall say that a term [z] is on *function normal form* if the term [z] has form $[\lambda\mathsf{x}.\mathsf{y}]$ where [x] is a variable and [y] is a term.

We shall say that a term is on *root normal form* if the term is on truth or function normal form.

When given a term, the Logiweb engine starts reducing the term using left-most reduction. The Logiweb engine stops again if the term reaches root normal form.

3.1.3 Root equivalence

We shall say that a term is a *true term* if the engine can reduce it to truth normal form and that it is a *function term* if it can reduce it to function normal form. We shall say that a term is *perpetual* if the engine reduces it forever without reaching a root normal form.

We shall say that two terms are *root equivalent* if they are both true terms, both function terms, or both perpetual. Root equivalence is undecidable in general but is decidable on terms that have a root normal form.

3.1.4 Mathematical equality

The Logiweb computing engine does not support the parallel “or” operation $[\mathsf{x} \parallel \mathsf{y}]$, but the operation is interesting for theoretical reasons. The parallel “or” operation has the following properties:

$$\left[\begin{array}{lcl} \mathsf{T} \parallel \mathsf{y} & = & \mathsf{T} \\ \mathsf{x} \parallel \mathsf{T} & = & \mathsf{T} \\ (\lambda\mathsf{x}.\mathsf{y})^{\mathrm{I}} \parallel (\lambda\mathsf{u}.\mathsf{v})^{\mathrm{I}} & = & \lambda\mathsf{x}.\mathsf{T} \end{array}\right]$$

If the parallel "or" operation were added to the Logiweb computing engine, then the engine would be required to reduce both [x] and [y] in parallel whenever the engine had to reduce [x ‖ y]. If the engine managed to reduce [x] to [T] then it should stop reducing [y] and vice versa.

We shall refer to the Logiweb engine extended with parallel "or" as the *parallel engine*.

The notions of "root normal form" and "root equivalence" are defined for the parallel engine as they are for the Logiweb engine.

Two terms [x] and [y] are considered *mathematically equal*, written [x = y], if [z ' x] is root equivalent to [z ' y] for all terms [z] (where the term [z] is allowed to include parallel "or"). This defines equality of term [x] and [y] that may include parallel "or" and, in particular, defines equality of terms that do not include paralle "or". Mathematical equality is undecidable in general.

At this point, parallel "or" has played its role and we shall not refer to it anymore. But we shall use mathematical equality to state things like $[(\lambda x.x)^{I}\text{'}y = y]$. For an axiomatization of mathematical equality see [4].

3.2 Definitions

3.2.1 The value aspect

One may define many "aspects" of each construct. As an example, the "tex" aspect of a construct defines how the construct should be rendered using TEX. Likewise, the "pyk" aspect of a construct defines what the construct looks like in a pyk source file from which a Logiweb page may be compiled.

An ordinary mathematical definition of a construct corresponds to a definition of the *value aspect* of the construct in Logiweb. The value aspect is proclaimed thus: [val ⋈ "value"].

As an example of a value definition, consider the following:

$$[f(x) \overset{val}{\to} if(x, T, f(x\text{ ' }T))]$$

With the definition above we have $[f(\lambda x.\lambda x.\lambda x.T) = f(\lambda x.\lambda x.T) = f(\lambda x.T) = f(T) = T]$

3.2.2 Parentheses

It is best to macro define parentheses, i.e. to define them by defining the macro aspect of the parenthesis construct. To keep the present page simple and to the point, however, we shall not deal with macro definitions here. Instead, we value define parentheses:

$$[(x)^{I} \overset{val}{\to} x]$$

3.2.3 Bottom

As mentioned, a term is perpetual if reduction of the term proceeds indefinitely without yielding a root normal form. As an example, $[(\lambda x.x\text{ ' }x)^{I}\text{ ' }(\lambda x.x\text{ ' }x)^{I}]$ is a perpetual term. Some authors prefer to use a capital omega for that term.

There are different opinions on whether or not all perpetual terms are equal. Intuitionists will typically say that there are perpetual terms that are not provably equal. Other blends of mathematicians go further and claim there are perpetual terms that are provably different. With the definition of mathematical equality stated earlier, however, all perpetual terms are equal.

Having decided that all perputual terms are equal or, equivalently, that they all have the same value, it is reasonable to introduce a name for that unique value. We shall follow tradition and call that value *bottom* since it lurks at the bottom of kappa-Scott domains.

Furthermore, we shall follow tradition and use $[\bot]$ to denote bottom. We do so by making the following definition:

$$[\bot \overset{\text{val}}{\Rightarrow} (\lambda \mathsf{x}.\mathsf{x}\,\text{'}\,\mathsf{x})^{\mathrm{I}}\,\text{'}\,(\lambda \mathsf{x}.\mathsf{x}\,\text{'}\,\mathsf{x})^{\mathrm{I}}]$$

The equal sign in the definition deviates slightly from the equal sign used previously. Formally, that has no effect on the definition. See Section 3.6 for further details.

With the definition above we have that $[\bot]$ is perpetual, since evaluating it causes a computer to look indefinitely. For that reason, the value of $[\bot]$ is bottom.

Mathematics is referentially transparent, meaning that any term silently denotes the value of the term. For that reason, $[\bot]$ denotes bottom.

By abuse of the language, we shall say that an operator *returns* $[\bot]$ when the operator does not return anything ever.

3.3 Peano trees

3.3.1 Raw pairs

Define

$$[\mathsf{y} \mathrel{\therefore} \mathsf{z} \overset{\text{val}}{\Rightarrow} \lambda \mathsf{x}.\mathrm{if}(\mathsf{x},\mathsf{y},\mathsf{z})]$$

$$[\mathsf{F} \overset{\text{val}}{\Rightarrow} \mathsf{T} \mathrel{\therefore} \mathsf{T}]$$

$$[\mathsf{x}^{\mathrm{H}} \overset{\text{val}}{\rightarrow} \mathsf{x}\,\text{'}\,\mathsf{T}]$$

$$[\mathsf{x}^{\mathrm{T}} \overset{\text{val}}{\rightarrow} \mathsf{x}\,\text{'}\,\mathsf{F}]$$

We shall use $[\mathsf{T}]$ and $[\mathsf{F}]$ to denote truth and falsehood, respectively.

Furthermore, we shall use $[\mathsf{y} \mathrel{\therefore} \mathsf{z}]$ as the most fundamental (most *raw*) among several pair constructs. The components of a pair may be found using the head operation $[\mathsf{x}^{\mathrm{H}}]$ and tail operation $[\mathsf{x}^{\mathrm{T}}]$:
$[(\mathsf{y} \mathrel{\therefore} \mathsf{z})^{\mathrm{IH}} = \mathsf{y}]$
$[(\mathsf{y} \mathrel{\therefore} \mathsf{z})^{\mathrm{IT}} = \mathsf{z}]$

In principle, we may use any definition for $[\mathsf{y} \mathrel{\therefore} \mathsf{z}]$ as long as it is possible to compute $[\mathsf{y}]$ and $[\mathsf{z}]$ from $[\mathsf{y} \mathrel{\therefore} \mathsf{z}]$. The definition above differs from anything

tenable in pure lambda calculus: The range of any function in pure lambda calculus has one or infinitely many elements whereas the range of $[(\mathsf{y} \therefore \mathsf{z})^{\mathsf{I}}\ \mathsf{x}]$ comprises $[\mathsf{y}]$, $[\mathsf{z}]$, and $[\bot]$.

3.3.2 Peano trees

As is well known, Dedekind and Peano both managed to axiomatize the natural numbers. Dedekind was first, but Peano had best public relations, so we know the axioms as the Peano axioms today.

In short, the Peano axioms are axioms for the smallest set [N] which contains [0] and which contains the successor of [x] whenever it contains [x].

Now consider the smallest set [P] which contains $[\mathsf{T}]$ and which contains $[\mathsf{x} \therefore \mathsf{y}]$ whenever it contains [x] and [y]. We shall refer to the elements of [P] as *Peano trees* due to the similarity to the structure of natural numbers.

Peano trees are good substitutes for natural numbers in elementary recursion theory because it is much easier to work with "Gödel trees" than to work with "Gödel numbers". Peano trees are early precursors of the maps used in Map Theory and Logiweb, and they constitute convenient stepping stones for building up more advanced data structures.

In early versions of Map Theory and Logiweb, Peano trees have been called "Binary trees", "Finite trees", and "Bintrees", but those names are inconvenient because it is convenient to reserve "B" for "Boolean" and "F" for "False". The "P" in "Peano tree" may be mistaken to denote "Pair", but that is acceptable since Peano trees are little more than a bunch of pairs.

As far as possible, we shall represent data structures by Peano trees. As an example, we represented truth values by Peano trees in the previous section.

3.3.3 Canonical and liberal representations

In Logiweb (as in most computational systems) one has to distinguish between *canonical* and *liberal* representations. By a canonical representation we shall mean a representation scheme that assigns one and only one representation to each concept to be represented. By a liberal one we shall mean a representation scheme that assigns at least one.

For each set of concepts to be represented, we shall assign a canonical as well as a liberal representation. And we shall make sure the representations are *compatible* in the sense that the canonical representation of any concept is among the liberal representations of the concept.

We make the convention that $[\mathsf{T}]$ liberally represents truth and that any lambda abstraction liberally represents falsehood. This convention is compatible with the convention that $[\mathsf{T}]$ represents truth and $[\mathsf{T} \therefore \mathsf{T}]$ represents falsehood.

$[\mathsf{T}]$[16] of Map Theory and Logiweb corresponds to Nil in Lisp and Null in C. Hence, the convention to let $[\mathsf{T}]$ represent truth and anything else falsehood is

[16] It is generally considered to be bad style to start a sentence with a formula. But, in my opinion, the reasons for deprecating such opening formulas disappear when formulas are consistently surrounded by brackets.

opposite to the choice made in Lisp and C. But the choice is in line with Gödels choice to let 0 denote truth. Gödels choice to make the simplest possible data structure represent truth and everything else denote falsehood is a smart one; it has the consequence that when programming primitive recursive functions, one tends to treat the simple case first and the recursive case afterwards which enhances the readability of the code. Gödels 1931 paper [3] is the best example I have ever seen of "literate programming", and the syntax of that particular paper has been a driving force in the development of Logiweb since 1984 where the first version of the "Pyk" compiler was implemented. The present implementation of Logiweb depends heavily on the existence of TEX and the World Wide Web.

3.3.4 Eager pairs

We now introduce an eager version of $[\mathsf{x} \therefore \mathsf{y}]$. In many situations, the eager version is more efficient to compute than the lazy one.

Define

$$[\mathsf{x} : \mathsf{y} \overset{\text{val}}{\rightarrow} \text{if}(\mathsf{x}, \mathsf{y}, \mathsf{y})]$$

$$[\mathsf{x} \underline{\therefore} \mathsf{y} \overset{\text{val}}{\rightarrow} \mathsf{x} : \mathsf{y} : \mathsf{x} \therefore \mathsf{y}]$$

The *guard* construct $[\mathsf{x} : \mathsf{y}]$ equals $[\bot]$ whenever $[\mathsf{x}]$ equals $[\bot]$ and equals $[\mathsf{y}]$ otherwise. In Logiweb, one may use guards when one needs the value of one construct $[\mathsf{y}]$ but also wants to force another construct $[\mathsf{x}]$ to be computed.

The guard construct is used in the definition of the *eager* pair $[\mathsf{x} \underline{\therefore} \mathsf{y}]$ above. Computation of $[\mathsf{x} \underline{\therefore} \mathsf{y}]$ forces $[\mathsf{x}]$ and $[\mathsf{y}]$ to be computed. Whenever $[\mathsf{x}]$ and $[\mathsf{y}]$ differ from $[\bot]$, the eager pair equals the raw pair.

We shall say that $[\mathsf{x} \underline{\therefore} \mathsf{y}]$ is *strict* in $[\mathsf{x}]$ because $[\bot \underline{\therefore} \mathsf{y} = \bot]$ for all maps $[\mathsf{y}]$.

Similarly, we shall say that $[\mathsf{x} \underline{\therefore} \mathsf{y}]$ is *strict* in $[\mathsf{y}]$ because $[\mathsf{x} \underline{\therefore} \bot = \bot]$ for all maps $[\mathsf{x}]$.

Furthermore, we shall say that $[\mathsf{x} \underline{\therefore} \mathsf{y}]$ is "strict" (without mentioning a particular argument) because it is strict in all arguments.

We shall say that $[\mathsf{x} \therefore \mathsf{y}]$ is *lazy* in $[\mathsf{x}]$ because it is not strict in $[\mathsf{x}]$ and lazy in $[\mathsf{y}]$ for similar reasons. We shall say that $[\mathsf{x} \therefore \mathsf{y}]$ is "lazy" (without mentioning a particular argument) because it is lazy in at least one argument.

3.4 Cardinals

3.4.1 Untagged cardinals

Cardinal numbers are the numbers "one", "two", "three", and so on as opposed to the *ordinal numbers* "first", "second", "third", and so on.

In set theory, the words "cardinal" and "ordinal" are used in a somewhat technical sense in which cardinals and ordinals may be infinite. We shall not use the words in that sense here.

In programming context, the word "cardinal" is sometimes simply used as a synonym for "natural number". We shall use the word in that sense here.

We shall use $[\mathsf{b}_0 \mathbin{\underline{\therefore}} \mathsf{b}_1 \mathbin{\underline{\therefore}} \cdots \mathbin{\underline{\therefore}} \mathsf{b}_{n-1} \mathbin{\underline{\therefore}} \mathsf{T}]$ to represent

$$\left[\textstyle\sum_{i=0}^{n-1} 2^i \mathrm{if}(\mathsf{b}_i, 0, 1)\right]$$

As an example, $[\mathsf{T} \mathbin{\underline{\therefore}} \mathsf{F} \mathbin{\underline{\therefore}} \mathsf{F} \mathbin{\underline{\therefore}} \mathsf{T}]$ represents six. The rightmost $[\mathsf{T}]$ marks the end of the structure. The other elements, $[\mathsf{T}]$, $[\mathsf{F}]$, and $[\mathsf{F}]$, are the bits of $[6 = 110_2]$, stated with the least significant bit first.

3.4.2 Operations on untagged cardinals

Define

$$[\mathsf{x} \mathbin{\underline{+2*}} \mathsf{y} \overset{\mathrm{val}}{\to} \mathrm{if}(\mathsf{x}, \mathrm{if}(\mathsf{y}, \mathsf{T}, \mathsf{x} \mathbin{\underline{\therefore}} \mathsf{y}), \mathsf{x} \mathbin{\underline{\therefore}} \mathsf{y})]$$

If $[\mathsf{y}]$ is an untagged cardinal, then $[\mathsf{T} \mathbin{\underline{+2*}} \mathsf{y}]$ represents two times $[\mathsf{y}]$ and $[\mathsf{F} \mathbin{\underline{+2*}} \mathsf{y}]$ represents one plus two times $[\mathsf{y}]$.

$[\mathsf{x} \mathbin{\underline{+2*}} \mathsf{y}]$ is right associative so that e.g. $[\mathsf{T} \mathbin{\underline{+2*}} \mathsf{F} \mathbin{\underline{+2*}} \mathsf{T}]$ equals $[\mathsf{T} \mathbin{\underline{+2*}} (\mathsf{F} \mathbin{\underline{+2*}} \mathsf{T})^{\mathrm{I}}]$ which in turn equals two.

The set of untagged cardinals is the smallest set that contains $[\mathsf{T}]$ (which represents zero) and which contains $[\mathsf{T} \mathbin{\underline{+2*}} \mathsf{y}]$ and $[\mathsf{F} \mathbin{\underline{+2*}} \mathsf{y}]$ whenever it contains $[\mathsf{y}]$.

Now define

$$[\mathsf{x}^{\mathrm{C}} \overset{\mathrm{val}}{\to} \mathrm{if}(\mathsf{x}, \mathsf{T}, \mathsf{x}^{\mathrm{HB}} \mathbin{\underline{+2*}} \mathsf{x}^{\mathrm{TC}})]$$

For all untagged cardinals $[\mathsf{x}]$ we have $[\mathsf{x}^{\mathrm{C}} = \mathsf{x}]$. Furthermore we have $[\bot^{\mathrm{C}} = \bot]$.

We shall say that $[\mathsf{x}]$ is a *lifted, untagged cardinal* if $[\mathsf{x}]$ is an untagged cardinal or equals $[\bot]$. $[\mathsf{x}^{\mathrm{C}} = \mathsf{x}]$ if and only if $[\mathsf{x}]$ is a lifted, untagged cardinal.

Moreover, $[\mathsf{x}^{\mathrm{C}}]$ is a lifted, untagged cardinal for all maps $[\mathsf{x}]$. Hence, $[\mathsf{x}^{\mathrm{C}}]$ is a construct that leaves lifted, untagged cardinals alone and converts everything else to lifted, untagged cardinals.

From the remarks above it follows that $[\mathsf{x}^{\mathrm{C}}]$ is *idempotent* in the sense that $[\mathsf{x}^{\mathrm{CC}} = \mathsf{x}^{\mathrm{C}}]$ for all maps $[\mathsf{x}]$.

We shall say that an operation is a *normalization construct* or *retract* for a set $[\mathrm{S}]$ if the operation is unary, strict, idempotent, and its range equals $[\mathrm{S} \cup \{\bot\}]$. Hence, $[\mathsf{x}^{\mathrm{C}}]$ is a retract for untagged cardinals.

3.4.3 Retracts and representations

Given a map $[\mathsf{x}]$ one may use the liberal representation of Booleans to translate the map to a Boolean and then use the canonical representation to translate the Boolean back to a map. The operation thus defined is a retract of Booleans:

$$[\mathsf{x}^{\mathrm{B}} \overset{\mathrm{val}}{\to} \mathrm{if}(\mathsf{x}, \mathsf{T}, \mathsf{F})]$$

In general, any pair consisting of a liberal and a canonical representation defines a retract.

3.4.4 The untagged cardinals from zero to nine

Define $[\underline{0} \overset{\text{val}}{\to} \mathsf{T}]$, $[\underline{1} \overset{\text{val}}{\to} \mathsf{F}\ \underline{+2*}\ \underline{0}]$, $[\underline{2} \overset{\text{val}}{\to} \mathsf{T}\ \underline{+2*}\ \underline{1}]$, $[\underline{3} \overset{\text{val}}{\to} \mathsf{F}\ \underline{+2*}\ \underline{1}]$, $[\underline{4} \overset{\text{val}}{\to} \mathsf{T}\ \underline{+2*}\ \underline{2}]$, $[\underline{5} \overset{\text{val}}{\to} \mathsf{F}\ \underline{+2*}\ \underline{2}]$, $[\underline{6} \overset{\text{val}}{\to} \mathsf{T}\ \underline{+2*}\ \underline{3}]$, $[\underline{7} \overset{\text{val}}{\to} \mathsf{F}\ \underline{+2*}\ \underline{3}]$, $[\underline{8} \overset{\text{val}}{\to} \mathsf{T}\ \underline{+2*}\ \underline{4}]$, and $[\underline{9} \overset{\text{val}}{\to} \mathsf{F}\ \underline{+2*}\ \underline{4}]$.

3.4.5 Tagged cardinals

For all untagged cardinals $[\mathsf{y}]$ we shall refer to $[\mathsf{T}\ \underline{\therefore}\ \mathsf{y}]$ as the corresponding *tagged cardinal*. $[\mathsf{T}]$ in $[\mathsf{T}\ \underline{\therefore}\ \mathsf{y}]$ is a *tag* which, in a moment, allows to distringuish tagged cardinals from other data structures.

3.4.6 Operations on tagged cardinals

For all tagged cardinals $[\mathsf{x}]$,

$$[\mathsf{x}^{\mathrm{U}} \overset{\text{val}}{\to} \mathrm{if}(\mathsf{x}^{\mathrm{H}}, \mathsf{x}^{\mathrm{T}}, \mathsf{T})]$$

is the corresponding untagged cardinal. The untag operation maps data other than tagged cardinals to untagged zero or bottom.

Now define

$$[\mathsf{x}\ +2*\ \mathsf{y} \overset{\text{val}}{\Rightarrow} \mathsf{T}\ \underline{\therefore}\ \mathsf{x}^{\mathrm{B}}\ \underline{\therefore}\ \mathsf{y}^{\mathrm{UC}}]$$

If $[\mathsf{y}]$ is a tagged cardinal, then $[\mathsf{T}\ +2*\ \mathsf{y}]$ represents two times $[\mathsf{y}]$ and $[\mathsf{F}\ +2*\ \mathsf{y}]$ represents one plus two times $[\mathsf{y}]$.

$[0 \overset{\text{val}}{\Rightarrow} \mathsf{T} \therefore \mathsf{T}]$ is a tagged zero.

The set of tagged cardinals is the smallest set that contains $[0]$ and which contains $[\mathsf{T}\ +2*\ \mathsf{y}]$ and $[\mathsf{F}\ +2*\ \mathsf{y}]$ whenever it contains $[\mathsf{y}]$.

3.4.7 The tagged cardinals from one to nine

Define $[1 \overset{\text{val}}{\to} \mathsf{F}\ +2*\ 0]$, $[2 \overset{\text{val}}{\to} \mathsf{T}\ +2*\ 1]$, $[3 \overset{\text{val}}{\to} \mathsf{F}\ +2*\ 1]$, $[4 \overset{\text{val}}{\to} \mathsf{T}\ +2*\ 2]$, $[5 \overset{\text{val}}{\to} \mathsf{F}\ +2*\ 2]$, $[6 \overset{\text{val}}{\to} \mathsf{T}\ +2*\ 3]$, $[7 \overset{\text{val}}{\to} \mathsf{F}\ +2*\ 3]$, $[8 \overset{\text{val}}{\to} \mathsf{T}\ +2*\ 4]$, and $[9 \overset{\text{val}}{\to} \mathsf{F}\ +2*\ 4]$

3.5 Tagged trees

3.5.1 Tagged pairs

For all $[\mathsf{x}]$ and $[\mathsf{y}]$, we shall refer to

$$[\mathsf{x}\ \underline{::}\ \mathsf{y} \overset{\text{val}}{\to} (\underline{0}\ \underline{\therefore}\ \underline{0}\ \underline{\therefore}\ \mathsf{T})^{\mathrm{I}}\ \underline{\therefore}\ \mathsf{x}\ \underline{\therefore}\ \mathsf{y}]$$

as the *tagged pair* of $[\mathsf{x}]$ and $[\mathsf{y}]$. In $[(\underline{0}\ \underline{\therefore}\ \underline{0}\ \underline{\therefore}\ \mathsf{T})^{\mathrm{I}}\ \underline{\therefore}\ \mathsf{x}\ \underline{\therefore}\ \mathsf{y}]$ we shall refer to $[\mathsf{x}]$ and $[\mathsf{y}]$ as the head and tail, respectively, of the tagged pair.

The set of *tagged Peano trees* is the smallest set which contains $[\mathsf{T}]$ and which contains $[\mathsf{x}\ \underline{::}\ \mathsf{y}]$ whenever $[\mathsf{x}]$ and $[\mathsf{y}]$ are tagged Peano trees.

3.5.2 Cardinal trees

The set of *cardinal trees* is the smallest set which contains $[\mathsf{T}]$ and all tagged cardinals and which contains $[\mathsf{x} \mathbin{\underline{::}} \mathsf{y}]$ whenever $[\mathsf{x}]$ and $[\mathsf{y}]$ are cardinal trees.

Cardinal trees are even more suited to replace Gödel numbers than Peano trees are. For that reason, Logiweb represents terms by Cardinal trees. In Logiweb, terms are represented by Cardinal trees as follows:

Every operation in Logiweb is identified by a *reference cardinal* $[\mathsf{r}]$ and an *id* $[\mathsf{i}]$, which is also a cardinal. The reference cardinal uniquely identifies the home page of the operation, which is the page that introduces the operation. The id is a cardinal which identifies the operation among all operations introduced by that page.

In Logiweb, an operation with reference cardinal $[\mathsf{r}]$ and id $[\mathsf{i}]$ applied to arguments is represented by the cardinal tree

$[((\mathsf{r} \mathbin{\underline{\therefore}} \mathsf{i} \mathbin{\underline{\therefore}} \mathsf{x})^{\mathrm{I}} \mathbin{\underline{\therefore}} \mathsf{y} \mathbin{\underline{\therefore}} \cdots \mathbin{\underline{\therefore}} \mathsf{z} \mathbin{\underline{\therefore}} \mathsf{T})^{\mathrm{I}}]$

where $[\mathsf{y}, \ldots, \mathsf{z}]$ are representations of the arguments and $[\mathsf{x}]$ is a structure which is ignored by Logiweb except when generating human readable error messages ($[\mathsf{x}]$ indicates the location the operation in the body before macro expansion so that a browser can locate and highlight it).

3.5.3 Tagged maps

We shall refer to

$$[\mathcal{M}(\mathsf{x}) \overset{\mathrm{val}}{\to} (\underline{0} \mathbin{\underline{\therefore}} \underline{1} \mathbin{\underline{\therefore}} \mathsf{T})^{\mathrm{I}} \mathbin{\therefore} \mathsf{x}]$$

as a *tagged map* where $[\underline{0} \mathbin{\underline{\therefore}} \underline{1} \mathbin{\underline{\therefore}} \mathsf{T}]$ is the tag and $[\mathsf{x}]$ is the map itself.

The set of *tagged trees* is the smallest set which contains $[\mathsf{T}]$ and all tagged cardinals, which contains $[\mathsf{x} \mathbin{::} \mathsf{y}]$ whenever $[\mathsf{x}]$ and $[\mathsf{y}]$ are tagged trees, and which contains $[(\underline{0} \mathbin{\underline{\therefore}} \underline{1} \mathbin{\underline{\therefore}} \mathsf{T})^{\mathrm{I}} \mathbin{\therefore} \mathsf{x}]$ for all maps $[\mathsf{x}]$.

3.5.4 The retract for tagged trees

We define the retract for tagged trees thus:

$$[\mathsf{x}^{\mathrm{M}} \overset{\mathrm{val}}{\Rightarrow} \mathrm{if}(\mathsf{x}, \mathsf{T}, \mathrm{if}(\mathsf{x}^{\mathrm{H}}, \mathsf{T} \mathbin{\underline{\therefore}} \mathsf{x}^{\mathrm{TC}}, \mathrm{if}(\mathsf{x}^{\mathrm{HTH}}, \mathsf{x}^{\mathrm{THM}} \mathbin{\underline{::}} \mathsf{x}^{\mathrm{TTM}}, \mathcal{M}(\mathsf{x}^{\mathrm{T}}))))]$$

We define the constructor operation for tagged pairs such that it normalises its arguments and, hence, is guaranteed to return $[\bot]$ or a tagged tree:

$$[\mathsf{x} :: \mathsf{y} \overset{\mathrm{val}}{\Rightarrow} \mathsf{x}^{\mathrm{M}} \mathbin{\underline{::}} \mathsf{y}^{\mathrm{M}}]$$

Operations that retract their arguments allow certain optimizations to occur behind the scene (c.f. Section 3.6).

3.5.5 Predicates on tagged trees

The following predicate tests for being data (as opposed to being a tagged map):

$$[\mathsf{x}^{\mathrm{d}} \overset{\mathrm{val}}{\Rightarrow} \mathsf{x}^{\mathrm{MHTHB}}]$$

The following predicate tests for being a cardinal:

$$[\mathsf{x}^{\mathrm{c}} \overset{\mathrm{val}}{\Rightarrow} \mathrm{if}(\mathsf{x}, \mathsf{F}, \mathsf{x}^{\mathrm{MHB}})]$$

We shall say that [0] is *singular* among tagged cardinals, that [T] is singular among tagged peano trees, and that $[\mathcal{M}(\mathsf{T})]$ is singular among tagged maps. Non-singular entities are called *regular*. The following predicate tests for singularity:

$$[\mathsf{x}^{\mathrm{s}} \overset{\mathrm{val}}{\Rightarrow} \mathsf{x}^{\mathrm{MTB}}]$$

3.5.6 Operations on tagged trees

We define the head of a tagged cardinal [x] to be [T] if [x] is even and [F] otherwise. All operations on tagged trees are defined such that they retract their arguments before operating on them. The head operation is defined thus:

$$[\mathsf{x}^{\mathrm{h}} \overset{\mathrm{val}}{\Rightarrow} \mathrm{if}(\mathsf{x}^{\mathrm{d}}, \mathsf{x}^{\mathrm{MTH}}, \mathsf{T})]$$

We define the tail of a tagged cardinal [x] to be [x] integer divided by two (i.e. divided by two and rounded to the nearest cardinal in the direction of minus infinity). The tail of a tagged pair is the second component of the pair. The tail of a tagged map or [T] equals [T]:

$$[\mathsf{x}^{\mathrm{t}} \overset{\mathrm{val}}{\Rightarrow} \mathrm{if}(\mathsf{x}^{\mathrm{d}}, \mathrm{if}(\mathsf{x}^{\mathrm{c}}, \mathsf{T} \mathbin{\dot{.}\,.} \mathsf{x}^{\mathrm{MTT}}, \mathsf{x}^{\mathrm{MTT}}), \mathsf{T})]$$

Accessors for the zeroth to ninth element of a list [x] read $[\mathsf{x}^{0} \overset{\mathrm{val}}{\rightarrow} \mathsf{x}^{\mathrm{h}}]$[17], $[\mathsf{x}^{1} \overset{\mathrm{val}}{\rightarrow} \mathsf{x}^{\mathrm{t}0}]$[18], $[\mathsf{x}^{2} \overset{\mathrm{val}}{\rightarrow} \mathsf{x}^{\mathrm{t}1}]$[19], $[\mathsf{x}^{3} \overset{\mathrm{val}}{\rightarrow} \mathsf{x}^{\mathrm{t}2}]$[20], $[\mathsf{x}^{4} \overset{\mathrm{val}}{\rightarrow} \mathsf{x}^{\mathrm{t}3}]$[21], $[\mathsf{x}^{5} \overset{\mathrm{val}}{\rightarrow} \mathsf{x}^{\mathrm{t}4}]$[22], $[\mathsf{x}^{6} \overset{\mathrm{val}}{\rightarrow} \mathsf{x}^{\mathrm{t}5}]$[23], $[\mathsf{x}^{7} \overset{\mathrm{val}}{\rightarrow} \mathsf{x}^{\mathrm{t}6}]$[24], $[\mathsf{x}^{8} \overset{\mathrm{val}}{\rightarrow} \mathsf{x}^{\mathrm{t}7}]$[25], and $[\mathsf{x}^{9} \overset{\mathrm{val}}{\rightarrow} \mathsf{x}^{\mathrm{t}8}]$[26].

[17] $[\mathsf{x}^{0} \overset{\mathrm{pyk}}{=} \text{“∗ zeroth”}]$
[18] $[\mathsf{x}^{1} \overset{\mathrm{pyk}}{=} \text{“∗ first”}]$
[19] $[\mathsf{x}^{2} \overset{\mathrm{pyk}}{=} \text{“∗ second”}]$
[20] $[\mathsf{x}^{3} \overset{\mathrm{pyk}}{=} \text{“∗ third”}]$
[21] $[\mathsf{x}^{4} \overset{\mathrm{pyk}}{=} \text{“∗ fourth”}]$
[22] $[\mathsf{x}^{5} \overset{\mathrm{pyk}}{=} \text{“∗ fifth”}]$
[23] $[\mathsf{x}^{6} \overset{\mathrm{pyk}}{=} \text{“∗ sixth”}]$
[24] $[\mathsf{x}^{7} \overset{\mathrm{pyk}}{=} \text{“∗ seventh”}]$
[25] $[\mathsf{x}^{8} \overset{\mathrm{pyk}}{=} \text{“∗ eighth”}]$
[26] $[\mathsf{x}^{9} \overset{\mathrm{pyk}}{=} \text{“∗ ninth”}]$

3.5.7 Selection

The following operation is a retracting if-then-else construct which plays an important role in connection with optimizations and fitness and strictness analysis that occur behind the scene:

$$[\mathrm{If}(\mathsf{x},\mathsf{y},\mathsf{z}) \overset{\mathrm{val}}{\Rightarrow} \mathrm{if}(\mathsf{x}^{\mathrm{M}},\mathsf{y}^{\mathrm{M}},\mathsf{z}^{\mathrm{M}})]$$

We shall use selection

$$[\mathsf{x} \left\{ \begin{array}{l} \mathsf{y} \\ \mathsf{z} \end{array} \right. \overset{\mathrm{val}}{\Rightarrow} \mathrm{If}(\mathsf{x},\mathsf{y},\mathsf{z})]$$

as an alternative for $[\mathrm{If}(\mathsf{x},\mathsf{y},\mathsf{z})]$. It would be very natural to macro define selection, but macro definitions depend on the Logiweb self-interpretter which in turn depends on the definitions presented on the present page. To keep the present page simple and to the point, we omit introducing the notion of macro definitions and, hence, avoid macro defining selection.

3.5.8 Semitagged maps

We introduce a *semitagged* representation of maps as follows:

1. The canonical representation of a map $[\mathsf{m}]$ is $[\mathcal{M}(\mathsf{m})]$.
2. Any tagged tree $[\mathsf{x}]$ liberally represents the map $[\mathrm{If}(\mathsf{x}^{\mathrm{d}},\mathsf{x},\mathsf{x}^{\mathrm{T}})]$.

$[\mathcal{U}(\mathsf{x})]$ returns the map represented by the semitagged $[\mathsf{x}]$. The untagger $[\mathcal{U}(\mathsf{x})]$ leaves tagged data untouched and untags tagged maps:

$$[\tilde{\mathcal{U}}(\mathsf{x}) \overset{\mathrm{val}}{\rightarrow} \mathrm{if}(\mathsf{x}^{\mathrm{d}},\mathsf{x},\mathsf{x}^{\mathrm{T}})]$$

$$[\mathcal{U}(\mathsf{x}) \overset{\mathrm{val}}{\rightarrow} \tilde{\mathcal{U}}(\mathsf{x}^{\mathrm{M}})]$$

The construct $[\mathbf{apply}(\mathsf{f},\mathsf{x})]$ applies a tagged map $[\mathsf{f}]$ to a semitagged map $[\mathsf{x}]$ and returns the result as a tagged map. If $[\mathsf{f}]$ is tagged data then $[\mathbf{apply}(\mathsf{f},\mathsf{x})]$ equals $[\mathsf{f}]$:

$$[\mathbf{apply}(\mathsf{f},\mathsf{x}) \overset{\mathrm{val}}{\Rightarrow} \mathbf{apply}_1(\mathsf{f}^{\mathrm{M}},\mathsf{x}^{\mathrm{M}})]$$

$$[\mathbf{apply}_1(\mathsf{f},\mathsf{x}) \overset{\mathrm{val}}{\rightarrow} \mathsf{f}^{\mathrm{d}} \left\{ \begin{array}{l} \mathrm{If}(\mathsf{x}^{\mathrm{d}},\mathsf{f},\mathsf{f}) \\ \mathrm{If}(\mathsf{x}^{\mathrm{d}},\mathcal{M}(\mathsf{f}^{\mathrm{T}}\,{}^{\backprime}\mathsf{x}),\mathcal{M}(\mathsf{f}^{\mathrm{T}}\,{}^{\backprime}(\mathsf{x}^{\mathrm{T}})^{\mathrm{I}})) \end{array} \right.]$$

A short version of $[\mathbf{apply}(\mathsf{f},\mathsf{x})]$ reads:

$$[\mathsf{f}\,{}^{\backprime}\mathsf{x} \overset{\mathrm{val}}{\rightarrow} \mathbf{apply}(\mathsf{f},\mathsf{x})]$$

We shall also introduce a tagged version of lambda abstraction. Logiweb, however, does not support value definitions of new binding constructs. This is convenient for designers of proof systems who can rely on lambda being the only binding construct in Logiweb.

In Logiweb it is the intension that new binding constructs should be macro defined so that binding constructs are expanded into lambdas before proof checking. At the present stage of the bootstrap of Logiweb, however, we do not yet have macro definitions.

For that reason we introduce a unary operator $[\Lambda \mathsf{x}]$ and use $[\Lambda\lambda \mathsf{x.y}]$ to denote tagged lambda abstraction. At a later stage we introduce a construct that macro expands into $[\Lambda\lambda \mathsf{x.y}]$. We define $[\Lambda \mathsf{x}]$ thus:

$$[\Lambda \mathsf{x} \stackrel{\text{val}}{\Rightarrow} \mathcal{M}(\lambda \mathsf{u}.\mathcal{U}(\mathsf{x}\,\mathsf{'}\,\mathcal{M}(\mathsf{u})))]$$

$[\mathcal{U}^{\mathrm{M}}(\mathsf{x})]$ is a version of $[\mathcal{U}(\mathsf{x})]$ which untags and then normalizes a semitagged map.

$$[\mathcal{U}^{\mathrm{M}}(\mathsf{x}) \stackrel{\text{val}}{\Rightarrow} \mathcal{U}(\mathsf{x})^{\mathrm{M}}]$$

3.6 Optimization

3.6.1 On-stage and off-stage semantics

The Logiweb computing engine maintains an *on-stage semantics* and *off-stage semantics*. The on-stage semantics concerns the input-output relation of operations. The off-stage semantics concerns the run time and memory usage of operations.

One needs to know the on-stage semantics to make correct programs and one needs to know the off-stage semantics to make efficient programs.

The on-stage semantics of operations must be the same on all implementations of Logiweb. The off-stage semantics may vary considerably. For that reason, it is only possible to say something definite about the on-stage semantics. The remarks on off-stage semantics below concern one, particular implementation of the Logiweb computing engine which we shall refer to as the *first edition engine*.

The on-stage semantics of $[\mathsf{x}^{\mathrm{h}}]$ is that the Logiweb computing engine reduces $[\mathsf{x}^{\mathrm{h}}]$ to $[\mathsf{x}^{\mathrm{MTH}}]$ because of the definition $[\mathsf{x}^{\mathrm{h}} \stackrel{\text{val}}{\Rightarrow} \mathsf{x}^{\mathrm{MTH}}]$.

Logiweb has three kinds of *revelation* constructs that connect syntactic constructs with semantic concepts. The three kinds of revelations are called proclamations, definititions, and introductions. The revelation of $[\mathsf{x}^{\mathrm{h}}]$ above is an introduction rather than a definition. One can see that by looking at the equal sign which deviates from a normal equal sign.

The first edition engine maintains a list of constructs that it can compute using hardcoded functions. Whenever the first edition engine sees an introduction, it runs through its list of known constructs to see, if it can recognize the construct being introduced. When doing so, the first edition engine is insensitive to names of bound variables, names of parameters, and names of auxiliary

functions, but apart from that the engine merely recognises an introduction if its right hand side is identical to something the engine knows. If the first edition engine cannot recognize an introduction, it prints a warning and treats the introduction as if it were a definition.

The notion of introduction is proclaimed: $[[\mathsf{y} \overset{\mathsf{x}}{\Rightarrow} \mathsf{z}] \bowtie \text{“introduce”}]$.

On the first-edition engine, the off-stage semantics of $[\mathsf{x}^{\mathrm{h}}]$ is that, behind the scenes, the engine converts the argument of $[\mathsf{x}^{\mathrm{h}}]$ to an internal, efficient representation, then manipulates that internal representation, and then converts the internal representation back to maps.

The conversion back and forth takes time, but the manipulation of the internal representation is fast. When computing a single head operation, it does not pay off to round the internal representation. But when performing a long sequence of head and other, hardcoded operations, the engine merely has to convert back and forth once at the beginning and end of the computation.

3.6.2 Optimized bottom

The revelation $[\bot \overset{\mathrm{val}}{\Rightarrow} (\lambda \mathsf{x}.\mathsf{x}\,\text{'}\,\mathsf{x})^{\mathrm{I}}\,\text{'}\,(\lambda \mathsf{x}.\mathsf{x}\,\text{'}\,\mathsf{x})^{\mathrm{I}}]$ of bottom is an introduction

Bottom has the property that evaluation of it never returns a value. Genuine bottom $[(\lambda \mathsf{x}.\mathsf{x}\text{'}\mathsf{x})^{\mathrm{I}}\text{'}(\lambda \mathsf{x}.\mathsf{x}\text{'}\mathsf{x})^{\mathrm{I}}]$ never returns a value because it makes the computer run forever.

As any other optimized construct, optimized bottom must have exactly the same behavior as genuine bottom and, hence, optimized bottom must avoid returning a value. When evaluated on the first edition engine, optimized bottom signals an irrecoverable error and, hence, satisfies the requirements.

Another valid implementation of optimized bottom would be to let it decrease the priority of the current operating system process so much that the process received no more cpu-time.

The Logiweb machine described later is multitasking, so the latter implementation of optimized bottom would be reasonable there.

Please never do “bessermachen” by introducing a construct that can test for optimized bottomness. That would ruin the applicative behaviour of the Logiweb reduction system. If a bottomness test seems desirable to have in some situation, consider using exceptions and exception handling instead; and don't try to solve Turings halting problem.

3.6.3 Strictness

Define

$$[\mathsf{x} \wedge \mathsf{y} \overset{\mathrm{val}}{\rightarrow} \mathsf{x} \left\{ \begin{array}{l} \mathrm{If}(\mathsf{y}, \mathsf{T}, \mathsf{F}) \\ \mathrm{If}(\mathsf{y}, \mathsf{F}, \mathsf{F}) \end{array} \right.]$$

We shall say that $[\mathsf{x} \wedge \mathsf{y}]$ is *strict* in its first argument because

$$[\bot \wedge \mathsf{y} = \bot]$$

When an equation contains a free variable, it is understood that the equation holds for all values of the free variable. Hence, above, it is understood that the equation holds for all [y].

We shall say that $[x \wedge y]$ is strict in its second argument because

$$[x \wedge \bot = \bot]$$

We shall say that $[x \wedge y]$ is strict because it is strict in all its arguments.

3.6.4 Types

We shall say that that the return value of $[x \wedge y]$ is of *type* tagged tree because
$[(x \wedge y)^{IM} = x \wedge y]$

We shall say that the first argument of $[x \wedge y]$ is of type tagged tree because
$[x^{M} \wedge y = x \wedge y]$

Likewise, we shall say that the second argument of $[x \wedge y]$ is of type tagged tree because
$[x \wedge y^{M} = x \wedge y]$

Summing up, we shall say that $[x \wedge y]$ is an *operation over* tagged trees because its return value and all its arguments are of type tagged tree.

Being an operation over tagged trees does not exclude being an operation over other types as well.

3.6.5 User defined strict operations

We shall say that an operation is *fit for optimization* if the first edition engine can prove that the operation is a strict operation over tagged trees. The first edition engine translates operations that are fit for optimization in a particularly efficient way.

Users of the first edition engine need some understanding of fitness for optimization in order to make definitions that the first edition engine can compute efficiently.

We shall refer to $[T]$, $[0]$, $[x^{M}]$, $[x +2* y]$, $[x :: y]$, $[x^{h}]$, $[x^{t}]$, $[x^{s}]$, $[x^{c}]$, and $[If(x, y, z)]$ as the *basic tagged tree operations*.

The first edition engine knows that the basic tagged tree operations are operations over tagged trees and that all but $[If(x, y, z)]$ are strict.

From this the first edition engine can deduce that all operations defined from these constructs are operations over tagged trees. This includes operations that are defined by recursion or mutual recursion. There is one pathological exception, though. A projection operation like $[f(x, y) = x]$ may be defined using the functions above zero times but is not an operation over tagged trees.

3.6.6 Fitness analysis

Operations defined from the basic tagged tree operations need not be strict because $[If(x, y, z)]$ is non-strict. But the first edition engine performs fitness analysis as follows:

The engine classifies all operations as "fit", "unfit", or "possibly fit" for optimization. As a fourth classification, the engine classifies [If(x, y, z)] as a "conditional".

Among the four elementary, computable constructs, the engine classifies [T] as fit and [λx.y], [x ' y], and [if(x, y, z)] as unfit.

Among the basic tagged tree operations the engine classifies [If(x, y, z)] as a conditional and the other ones as fit.

The engine classifies all other operations as possibly fit, except that it classifies projection operations as unfit. Then the engine reclassifies possibly fit operations as unfit according to certain rules described in the following. And when no further operations can be reclassified as unfit, all remaining possibly fit operations are reclassified as fit.

In other words, operations are considered fit unless they are provably unfit. This happens to be completely safe.

Possibly fit operations are reclassified as unfit according to two rules:

A possibly fit operation is reclassified as unfit if its definition contains an unfit operation.

A possibly fit operation if reclassified as unfit if some variable one the left hand side of the definition does not occur strict in the right hand side where "occurs strict" is defined in the next section.

3.6.7 Strict variables

Possibly fit operations are reclassified as unfit according to two rules. The first is rather simple:

A possibly fit operation is reclassified as unfit if its definition contains an unfit operation.

The other rule is more complex and makes reference to the notion of a *strict variable*. The notion of a strict variable is similar to the notion of a free variable. In fact, the strict variables of a term comprise a subset of the free variables of the term.

The notion of strict variables of a term [$\mathcal{A}$] is defined by structural induction in [$\mathcal{A}$] by the following three cases:

If the term [$\mathcal{A}$] is the variable [x], then the strict variables of [$\mathcal{A}$] comprises [x].

If the term [$\mathcal{A}$] is an operation applied to subtrees and the operation is not classified as conditional, then the free variables of [$\mathcal{A}$] comprises the strict variables of the subtrees.

If the term [$\mathcal{A}$] is a conditional operation applied to three subtrees like [If(x, y, z)], then the strict variables of [$\mathcal{A}$] comprises the strict variables of [x] plus all variables that occur strict in both [y] and [z].

As an example, [x] and [y] are the strict variables of [If(x, y, If(y, z, y))].

3.6.8 Fitness of conjunction

The first edition engine classifies

$$[\mathsf{x} \wedge \mathsf{y} \overset{\text{val}}{\rightarrow} \mathsf{x} \left\{ \begin{array}{l} \mathrm{If}(\mathsf{y}, \mathsf{T}, \mathsf{F}) \\ \mathrm{If}(\mathsf{y}, \mathsf{F}, \mathsf{F}) \end{array} \right.]$$

as fit for optimization because conjunction is not the conditional construct and because the engine does not classify conjunction as unfit. The engine does not classify conjunction as unfit because the right hand side of the definition above contains no unfit operations and because both variables, [x] and [y], occur strict in the right hand side.

3.6.9 Logical connectives

Define

$$[\neg \mathsf{x} \overset{\text{val}}{\rightarrow} \mathrm{If}(\mathsf{x}, \mathsf{F}, \mathsf{T})]$$

$$[\mathsf{x} \vee \mathsf{y} \overset{\text{val}}{\rightarrow} \mathsf{x} \left\{ \begin{array}{l} \mathrm{If}(\mathsf{y}, \mathsf{T}, \mathsf{T}) \\ \mathrm{If}(\mathsf{y}, \mathsf{T}, \mathsf{F}) \end{array} \right.]$$

The operations above are fit for optimization.

Sometimes, when programming, it is useful to have lazy versions of the connectives that compute the first argument before they decide whether or not to compute the second, but we shall not need them here.

If such lazy connectives are needed, one should consider to macro define them rather than value defining them. If lazy connectives are value defined, they will be classified as unfit for optimization. On the contrary, if they are macro defined they will be macro expanded away before fitness analysis and, hence, they can be used for defining operations that are fit for optimization.

3.6.10 Auxiliary operations

Parentheses cannot occur in definitions that are fit for optimization since $[(\mathsf{x})^{\mathrm{I}}]$ does not normalize its argument.

Later, we macro define parentheses. Once parentheses are macro defined, they disappear from terms before fitness analysis. Hence, once parentheses are macro defined, they can be used in optimizable definitions. Until we macro define parentheses, we shall use a normalizing version of parentheses:

$$[(\mathsf{x})^{\mathrm{M}} \overset{\text{val}}{\rightarrow} \mathsf{x}^{\mathrm{M}}]$$

The following normalizing guard operation can be used in definitions for making the defined function strict:

$$[\mathsf{x}!\mathsf{y} \overset{\text{val}}{\Rightarrow} \mathrm{If}(\mathsf{x}, \mathsf{y}, \mathsf{y})]$$

3.6.11 Equality

Define

$$[\mathsf{x} \approx \mathsf{y} \overset{\mathrm{val}}{\Rightarrow} \mathsf{x}^{\mathrm{d}} \left\{ \begin{array}{l} \mathrm{If}(\mathsf{y}^{\mathrm{d}}, \mathsf{x} \overset{\mathrm{D}}{\approx} \mathsf{y}, \mathsf{F}) \\ \mathrm{If}(\mathsf{y}^{\mathrm{d}}, \mathsf{F}, \mathsf{T}) \end{array} \right.]$$

$$[\mathsf{x} \overset{\mathrm{D}}{\approx} \mathsf{y} \overset{\mathrm{val}}{\rightarrow} \mathsf{x}^{\mathrm{c}} \left\{ \begin{array}{l} \mathrm{If}(\mathsf{y}^{\mathrm{c}}, \mathsf{x} \overset{\mathrm{C}}{\approx} \mathsf{y}, \mathsf{F}) \\ \mathrm{If}(\mathsf{y}^{\mathrm{c}}, \mathsf{F}, \mathsf{x} \overset{\mathrm{P}}{\approx} \mathsf{y}) \end{array} \right.]$$

$$[\mathsf{x} \overset{\mathrm{C}}{\approx} \mathsf{y} \overset{\mathrm{val}}{\rightarrow} \mathsf{x}^{\mathrm{s}} \left\{ \begin{array}{l} \mathrm{If}(\mathsf{y}^{\mathrm{s}}, \mathsf{T}, \mathsf{F}) \\ \mathrm{If}(\mathsf{y}^{\mathrm{s}}, \mathsf{F}, \mathsf{x}^{\mathrm{h}} \overset{\mathrm{B}}{\approx} \mathsf{y}^{\mathrm{h}} \wedge \mathsf{x}^{\mathrm{t}} \overset{\mathrm{C}}{\approx} \mathsf{y}^{\mathrm{t}}) \end{array} \right.]$$

$$[\mathsf{x} \overset{\mathrm{B}}{\approx} \mathsf{y} \overset{\mathrm{val}}{\rightarrow} \mathsf{x} \left\{ \begin{array}{l} \mathrm{If}(\mathsf{y}, \mathsf{T}, \mathsf{F}) \\ \mathrm{If}(\mathsf{y}, \mathsf{F}, \mathsf{T}) \end{array} \right.]$$

$$[\mathsf{x} \overset{\mathrm{P}}{\approx} \mathsf{y} \overset{\mathrm{val}}{\rightarrow} \mathsf{x}^{\mathrm{s}} \left\{ \begin{array}{l} \mathrm{If}(\mathsf{y}^{\mathrm{s}}, \mathsf{T}, \mathsf{F}) \\ \mathrm{If}(\mathsf{y}^{\mathrm{s}}, \mathsf{F}, \mathsf{x}^{\mathrm{h}} \approx \mathsf{y}^{\mathrm{h}} \wedge \mathsf{x}^{\mathrm{t}} \approx \mathsf{y}^{\mathrm{t}}) \end{array} \right.]$$

$[\mathsf{x} \approx \mathsf{y}]$ is an example of an operation defined by mutual recursion which is fit for optimization.

3.7 Arithmetic

3.7.1 Addition

Addition of cardinals is defined thus:

$$[\mathsf{x} + \mathsf{y} \doteq \mathrm{If}(\mathsf{x}^{\mathrm{c}} \wedge \mathsf{y}^{\mathrm{c}}, \mathsf{x} +_0 \mathsf{y}, \mathsf{T})]^{27}$$

$$[\mathsf{x} +_0 \mathsf{y} \doteq \mathsf{x}^{\mathrm{s}} \left\{ \begin{array}{l} \mathsf{y} \\ \mathsf{y}^{\mathrm{s}} \left\{ \begin{array}{l} \mathsf{x} \\ \mathsf{x}^{\mathrm{h}} \left\{ \begin{array}{l} \mathsf{y}^{\mathrm{h}} \left\{ \begin{array}{l} \mathsf{T} +2* \mathsf{x}^{\mathrm{t}} +_0 \mathsf{y}^{\mathrm{t}} \\ \mathsf{F} +2* \mathsf{x}^{\mathrm{t}} +_0 \mathsf{y}^{\mathrm{t}} \end{array} \right. \\ \mathsf{y}^{\mathrm{h}} \left\{ \begin{array}{l} \mathsf{F} +2* \mathsf{x}^{\mathrm{t}} +_0 \mathsf{y}^{\mathrm{t}} \\ \mathsf{T} +2* \mathsf{x}^{\mathrm{t}} +_1 \mathsf{y}^{\mathrm{t}} \end{array} \right. \end{array} \right. \end{array} \right. \end{array} \right.]^{28}$$

$$[\mathsf{x} +_1 \mathsf{y} \doteq \mathsf{x}^{\mathrm{s}} \left\{ \begin{array}{l} \mathsf{y} +_0 1 \\ \mathsf{y}^{\mathrm{s}} \left\{ \begin{array}{l} \mathsf{x} +_0 1 \\ \mathsf{x}^{\mathrm{h}} \left\{ \begin{array}{l} \mathsf{y}^{\mathrm{h}} \left\{ \begin{array}{l} \mathsf{F} +2* \mathsf{x}^{\mathrm{t}} +_0 \mathsf{y}^{\mathrm{t}} \\ \mathsf{T} +2* \mathsf{x}^{\mathrm{t}} +_1 \mathsf{y}^{\mathrm{t}} \end{array} \right. \\ \mathsf{y}^{\mathrm{h}} \left\{ \begin{array}{l} \mathsf{T} +2* \mathsf{x}^{\mathrm{t}} +_1 \mathsf{y}^{\mathrm{t}} \\ \mathsf{F} +2* \mathsf{x}^{\mathrm{t}} +_1 \mathsf{y}^{\mathrm{t}} \end{array} \right. \end{array} \right. \end{array} \right. \end{array} \right.]^{29}$$

3.7.2 Comparison

Comparison of cardinals is defined thus:

[27] $[\mathsf{x} + \mathsf{y} \overset{\mathrm{pyk}}{=}$ "$*$ plus $*$"$]$

[28] $[\mathsf{x} +_0 \mathsf{y} \overset{\mathrm{pyk}}{=}$ "$*$ plus zero $*$"$]$

[29] $[\mathsf{x} +_1 \mathsf{y} \overset{\mathrm{pyk}}{=}$ "$*$ plus one $*$"$]$

$$[\mathsf{x} < \mathsf{y} \doteq \mathrm{If}(\mathsf{x}^{\mathrm{c}} \wedge \mathsf{y}^{\mathrm{c}}, \mathsf{x} <' \mathsf{y}, \mathsf{F})]^{30}$$

$$[\mathsf{x} <' \mathsf{y} \doteq \mathsf{y}^{\mathrm{s}} \begin{cases} \mathsf{x}!\mathsf{F} \\ \mathsf{x}^{\mathrm{s}} \begin{cases} \mathsf{T} \\ \mathsf{x}^{\mathrm{h}} \begin{cases} \mathsf{y}^{\mathrm{h}} \begin{cases} \mathsf{x}^{\mathrm{t}} <' \mathsf{y}^{\mathrm{t}} \\ \mathsf{x}^{\mathrm{t}} \leq' \mathsf{y}^{\mathrm{t}} \end{cases} \\ \mathsf{y}^{\mathrm{h}} \begin{cases} \mathsf{x}^{\mathrm{t}} <' \mathsf{y}^{\mathrm{t}} \\ \mathsf{x}^{\mathrm{t}} <' \mathsf{y}^{\mathrm{t}} \end{cases} \end{cases} \end{cases} \end{cases}]^{31}$$

$$[\mathsf{x} \leq' \mathsf{y} \doteq \mathsf{x}^{\mathrm{s}} \begin{cases} \mathsf{y}!\mathsf{T} \\ \mathsf{y}^{\mathrm{s}} \begin{cases} \mathsf{F} \\ \mathsf{x}^{\mathrm{h}} \begin{cases} \mathsf{y}^{\mathrm{h}} \begin{cases} \mathsf{x}^{\mathrm{t}} \leq' \mathsf{y}^{\mathrm{t}} \\ \mathsf{x}^{\mathrm{t}} \leq' \mathsf{y}^{\mathrm{t}} \end{cases} \\ \mathsf{y}^{\mathrm{h}} \begin{cases} \mathsf{x}^{\mathrm{t}} <' \mathsf{y}^{\mathrm{t}} \\ \mathsf{x}^{\mathrm{t}} \leq' \mathsf{y}^{\mathrm{t}} \end{cases} \end{cases} \end{cases} \end{cases}]^{32}$$

3.7.3 Subtraction

Subtraction of cardinals is defined below. Subtraction yields zero when a large number is subtracted from a small.

$$[\mathsf{x} - \mathsf{y} \doteq \mathrm{If}(\mathsf{x}^{\mathrm{c}} \wedge \mathsf{y}^{\mathrm{c}}, \mathrm{If}(\mathsf{x} < \mathsf{y}, 0, \mathsf{x} -_0 \mathsf{y}), \mathsf{T})]^{33}$$

$$[\mathsf{x} -_0 \mathsf{y} \doteq \mathsf{y}^{\mathrm{s}} \begin{cases} \mathsf{x} \\ \mathsf{x}^{\mathrm{h}} \begin{cases} \mathsf{y}^{\mathrm{h}} \begin{cases} \mathsf{T} +2* \mathsf{x}^{\mathrm{t}} -_0 \mathsf{y}^{\mathrm{t}} \\ \mathsf{F} +2* \mathsf{x}^{\mathrm{t}} -_1 \mathsf{y}^{\mathrm{t}} \end{cases} \\ \mathsf{y}^{\mathrm{h}} \begin{cases} \mathsf{F} +2* \mathsf{x}^{\mathrm{t}} -_0 \mathsf{y}^{\mathrm{t}} \\ \mathsf{T} +2* \mathsf{x}^{\mathrm{t}} -_0 \mathsf{y}^{\mathrm{t}} \end{cases} \end{cases} \end{cases}]^{34}$$

$$[\mathsf{x} -_1 \mathsf{y} \doteq \mathsf{y}^{\mathrm{s}} \begin{cases} \mathsf{x} -_0 1 \\ \mathsf{x}^{\mathrm{h}} \begin{cases} \mathsf{y}^{\mathrm{h}} \begin{cases} \mathsf{F} +2* \mathsf{x}^{\mathrm{t}} -_1 \mathsf{y}^{\mathrm{t}} \\ \mathsf{T} +2* \mathsf{x}^{\mathrm{t}} -_1 \mathsf{y}^{\mathrm{t}} \end{cases} \\ \mathsf{y}^{\mathrm{h}} \begin{cases} \mathsf{T} +2* \mathsf{x}^{\mathrm{t}} -_0 \mathsf{y}^{\mathrm{t}} \\ \mathsf{F} +2* \mathsf{x}^{\mathrm{t}} -_1 \mathsf{y}^{\mathrm{t}} \end{cases} \end{cases} \end{cases}]^{35}$$

3.7.4 Multiplication

Multiplication of cardinals is defined thus:

$$[\mathsf{x} \cdot \mathsf{y} \doteq \mathrm{If}(\mathsf{x}^{\mathrm{c}} \wedge \mathsf{y}^{\mathrm{c}}, \mathsf{x} \cdot_0 \mathsf{y}, \mathsf{T})]^{36}$$

---

[30] $[\mathsf{x} < \mathsf{y} \overset{\mathrm{pyk}}{=}$ "$*$ less $*$"$]$

[31] $[\mathsf{x} <' \mathsf{y} \overset{\mathrm{pyk}}{=}$ "$*$ less zero $*$"$]$

[32] $[\mathsf{x} \leq' \mathsf{y} \overset{\mathrm{pyk}}{=}$ "$*$ less one $*$"$]$

[33] $[\mathsf{x} - \mathsf{y} \overset{\mathrm{pyk}}{=}$ "$*$ minus $*$"$]$

[34] $[\mathsf{x} -_0 \mathsf{y} \overset{\mathrm{pyk}}{=}$ "$*$ minus zero $*$"$]$

[35] $[\mathsf{x} -_1 \mathsf{y} \overset{\mathrm{pyk}}{=}$ "$*$ minus one $*$"$]$

[36] $[\mathsf{x} \cdot \mathsf{y} \overset{\mathrm{pyk}}{=}$ "$*$ times $*$"$]$

$$[\mathsf{x} \cdot_0 \mathsf{y} \doteq \mathsf{y}^{\mathsf{s}} \left\{ \begin{array}{ll} \mathsf{x}!0 & \\ \mathsf{y}^{\mathsf{h}} \left\{ \begin{array}{ll} \mathsf{T} +2* \mathsf{x} \cdot_0 \mathsf{y}^{\mathsf{t}} & \\ (\mathsf{T} +2* \mathsf{x} \cdot_0 \mathsf{y}^{\mathsf{t}})^{\mathrm{M}} +_0 \mathsf{x} & \end{array} \right. & \end{array} \right.]$$ [37]

3.7.5 Bit access

$[\mathrm{bit}(\mathsf{x}, \mathsf{y})]$ is true if bit number $[\mathsf{x}]$ of the cardinal $[\mathsf{y}]$ is zero. The least significant bit is bit number zero.

$$[\mathrm{bit}(\mathsf{x}, \mathsf{y}) \doteq \mathrm{If}(\mathsf{x}^{\mathsf{c}} \wedge \mathsf{y}^{\mathsf{c}}, \mathrm{bit}_1(\mathsf{x}, \mathsf{y}), \mathsf{T})]$$ [38]

$$[\mathrm{bit}_1(\mathsf{x}, \mathsf{y}) \doteq \mathrm{If}(\mathsf{x}^{\mathsf{s}}, \mathsf{y}^{\mathsf{h}}, \mathrm{bit}(\mathsf{x} - 1, \mathsf{y}^{\mathsf{t}}))]$$ [39]

3.8 Quoting

3.8.1 Terms

Terms have form $[(\mathsf{r} :: \mathsf{i} :: \mathsf{d})^{\mathrm{I}} :: \mathsf{a}]$ where $[\mathsf{r}]$ and $[\mathsf{i}]$ are the reference and identifier, respectively, of the root symbol of the term and the argument list $[\mathsf{a}]$ is a list of terms. The debugging information $[\mathsf{d}]$ indicates the location of the root symbol on the page and allows a browser to highlight the root symbol even if the root symbol has been moved around by such processes as macro expansion.

Accessors for the reference, identifier, and debug information of a term $[\mathsf{x}]$ read $[\mathsf{x}^{\mathsf{r}} \overset{\mathrm{val}}{\to} \mathsf{x}^{\mathrm{hh}}]$[40], $[\mathsf{x}^{\mathsf{i}} \overset{\mathrm{val}}{\to} \mathsf{x}^{\mathrm{hth}}]$[41], and $[\mathsf{x}^{\mathsf{d}} \overset{\mathrm{val}}{\to} \mathsf{x}^{\mathrm{htt}}]$[42], respectively.

$[\mathsf{x}^{\mathrm{R}} \overset{\mathrm{val}}{\to} \mathsf{x}^{\mathsf{r}} :: \mathsf{x}^{\mathsf{i}} :: \mathsf{T}]$[43] returns the root of the term $[\mathsf{x}]$ without debugging information.

$[\mathsf{x} \overset{\mathsf{r}}{=} \mathsf{y}]$[44] is true if the roots of the terms $[\mathsf{x}]$ and $[\mathsf{y}]$ are equal except for debugging information:

$$[\mathsf{x} \overset{\mathsf{r}}{=} \mathsf{y} \overset{\mathrm{val}}{\to} \mathrm{If}(\mathsf{x}^{\mathsf{r}} \approx \mathsf{y}^{\mathsf{r}}, \mathsf{x}^{\mathsf{i}} \approx \mathsf{y}^{\mathsf{i}}, \mathsf{F})]$$

When testing two terms for identity, one should disregard debugging information. $[\mathsf{x} \overset{\mathsf{t}}{=} \mathsf{y}]$ tests two terms $[\mathsf{x}]$ and $[\mathsf{y}]$ for identity disregarding debugging information:

$$[\mathsf{x} \overset{\mathsf{t}}{=} \mathsf{y} \overset{\mathrm{val}}{\to} \mathrm{If}(\mathsf{x} \overset{\mathsf{r}}{=} \mathsf{y}, \mathsf{x}^{\mathsf{t}} \overset{\mathsf{t}^*}{=} \mathsf{y}^{\mathsf{t}}, \mathsf{F})]$$

---

[37] $[\mathsf{x} \cdot_0 \mathsf{y} \overset{\mathrm{pyk}}{=} \text{“}* \text{ times zero } *\text{”}]$

[38] $[\mathrm{bit}(\mathsf{x}, \mathsf{y}) \overset{\mathrm{pyk}}{=} \text{“bit } * \text{ of } * \text{ end bit”}]$

[39] $[\mathrm{bit}_1(\mathsf{x}, \mathsf{y}) \overset{\mathrm{pyk}}{=} \text{“bit one } * \text{ of } * \text{ end bit”}]$

[40] $[\mathsf{x}^{\mathsf{r}} \overset{\mathrm{pyk}}{=} \text{“}* \text{ ref”}]$

[41] $[\mathsf{x}^{\mathsf{i}} \overset{\mathrm{pyk}}{=} \text{“}* \text{ id”}]$

[42] $[\mathsf{x}^{\mathsf{d}} \overset{\mathrm{pyk}}{=} \text{“}* \text{ debug”}]$

[43] $[\mathsf{x}^{\mathrm{R}} \overset{\mathrm{pyk}}{=} \text{“}* \text{ root”}]$

[44] $[\mathsf{x} \overset{\mathsf{r}}{=} \mathsf{y} \overset{\mathrm{pyk}}{=} \text{“}* \text{ term root equal } *\text{”}]$

$[\mathsf{x} \overset{\mathsf{t}^*}{=} \mathsf{y}]$ tests two lists $[\mathsf{x}]$ and $[\mathsf{y}]$ of terms for identity disregarding debugging information:

$$[\mathsf{x} \overset{\mathsf{t}^*}{=} \mathsf{y} \overset{\mathrm{val}}{\rightarrow} \mathsf{x}^{\mathrm{a}} \begin{cases} \mathrm{If}(\mathsf{y}^{\mathrm{a}}, \mathsf{T}, \mathsf{F}) \\ \mathrm{If}(\mathsf{y}^{\mathrm{a}}, \mathsf{F}, \mathrm{If}(\mathsf{x}^{\mathrm{h}} \overset{\mathsf{t}}{=} \mathsf{y}^{\mathrm{h}}, \mathsf{x}^{\mathrm{t}} \overset{\mathsf{t}^*}{=} \mathsf{y}^{\mathrm{t}}, \mathsf{F})) \end{cases}]$$

3.8.2 Gödel trees

In addition to the fundamental, computable constructs $[\mathsf{T}]$, $[\lambda\mathsf{x}.\mathsf{y}]$, $[\mathsf{f}\ '\ \mathsf{x}]$, and $[\mathrm{if}(\mathsf{x}, \mathsf{y}, \mathsf{z})]$ we introduce the *quote* construct $[\lceil\mathcal{A}\rceil]$ for pragmatic reasons.

The quote construct is computable but does not add to the computational power of the Logiweb engine. If it were omitted from Logiweb, one could replace it with a macro defined construct.

The quote construct resembles "quote" in Lisp [9] and Gödel numbers as introduced in [3].

The reason for introducing quoting as a fundamental construct rather than macro defining it is that a macro defined quote would expand into rather large terms. In other words, quotes are introduced to enhance the efficiency of the macro facility. End users will probably prefer "backquote" like macros that are based on the quote construct rather than using quote raw.

The quote construct is proclaimed thus:

$$[\lceil\mathsf{x}\rceil \bowtie \text{“quote”}]$$

For a term $[\mathcal{A}]$, $[\lceil\mathcal{A}\rceil]$ denotes the tagged tree that represents $[\mathcal{A}]$. As an example,

$$[\lceil\mathsf{x} \approx \mathsf{y}\rceil]$$

is the tagged tree that represents the term $[\mathsf{x} \approx \mathsf{y}]$. We have

$$\left[\lceil\mathsf{x} \approx \mathsf{y}\rceil = (\mathsf{r} :: \mathsf{i} :: \mathsf{d} :: \mathsf{T})^{\mathrm{I}} :: \lceil\mathsf{x}\rceil :: \lceil\mathsf{y}\rceil :: \mathsf{T}\right]$$

except for debugging information present in each occurrence of $[\mathsf{x}]$ and $[\mathsf{y}]$.

The $[\mathsf{r}]$ and $[\mathsf{i}]$ above are the reference and identifier, respectively, of the $[* \approx *]$ symbol. The reference $[\mathsf{r}]$ is a large cardinal (around 200 bits long) which identifies the home page of the $[* \approx *]$ symbol, i.e. the page on which the $[* \approx *]$ symbol is introduced. The identifier [i] identifies the $[* \approx *]$ symbol among all symbols introduced on that page.

The $[\mathsf{d}]$ above is debugging information that indicates the location of the particular instance of the $[* \approx *]$ symbol. The value of $[\mathsf{d}]$ has form $[\mathsf{p}_1 :: \mathsf{p}_2 :: \cdots :: \mathsf{p}_{\mathsf{n}} :: \mathsf{q} :: \mathsf{T}]$ which indicates that the particular instance of the $[* \approx *]$ symbol occurs as the p_1's subtree of the p_2's subtree of the p_3's subtree etc. of page $[\mathsf{q}]$. Since the particular instance of the $[* \approx *]$ symbol occurs on the present page we have that $[\mathsf{q}]$ equals the reference of the present page.

In general, $[\lceil\mathcal{A}\rceil^{\mathrm{t}}]$ is the list of subterms of the term $[\mathcal{A}]$ and $[\lceil\mathcal{A}\rceil^{\mathrm{h}}]$ represents the identity and location of the root symbol of the term. As an example,

$$[\lceil\mathsf{x} \approx \mathsf{y}\rceil^{\mathrm{r}}]$$

and

$$\left[\ulcorner x \approx y \urcorner^{i}\right]$$

are the reference and identifier, respectively, of the equality construct. As another example, we have

$$\left[\ulcorner \text{ijcar base} \urcorner^{i} = 0\right]$$

because the identifier of the page symbol [ijcar base] is zero.

4 The macro expansion facility

This section describes the macro expansion facility of Logiweb[6]. The macro facility allows expressions that are easy to read for humans to expand into more machine readable formats before computer consumption.

4.1 Logiweb identifiers

4.1.1 Representation scheme

Occasionally, we shall represent strings of characters by cardinals. To translate a string

$$[c_1 c_2 \cdots c_m]$$

of characters to a cardinal, we proceed as follows. First, we mirror the string to get:

$$[c_m \cdots c_2 c_1]$$

Then we translate each character $[c_i]$ into its Unicode representation $[u_i]$, yielding a list of cardinals:

$$[u_m \cdots u_2 u_1]$$

Then we express each cardinal $[u_i]$ as a sequence of digits

$$[d_{in_i} \cdots d_{i1} d_{i0}]$$

in radix $[128]$ such that $[0 \leq d_{ij} < 128]$ and such that

$$u_i = \left[\sum_{j=0}^{n_i} d_{ij} 128^j\right]$$

In addition, we require $[d_{in_i} \neq 0]$ except when $[u_i = 0]$ in which case we require $[n_i = 1]$ instead.

Next, we add 128 do all digits except leading ones, i.e. we define

$$\left[d'_{ij} = \begin{cases} d_{ij} + 128 & \text{if } 0 \leq j < n_i \\ d_{ij} & \text{if } j = n_i \end{cases} \right]$$

This gives rise to a sequence

$$\begin{array}{lllll} [& d'_{mn_m} & \cdots & d'_{m0} & \\ & \cdots & & & \\ & d'_{2n_2} & \cdots & d'_{20} & \\ & d'_{1n_1} & \cdots & d'_{10} &] \end{array}$$

of cardinals in the range from [0] to [255]. Renumbering the sequence above gives the sequence

$$[d''_p \cdots d''_1 d''_0]$$

which we then interpret as a sequence of digits in radix 256, so that we represent the string

$$[c_1 c_2 \cdots c_m]$$

by the cardinal

$$c = \left[\sum_{k=0}^{p} d''_k 256^k \right]$$

4.1.2 Representation of ASCII strings

Logiweb ASCII is a character encoding that is very close to the American Standard Code for Information Interchange (ASCII). The differences are:

In Logiweb ASCII, character codes in the ranges [0..9] and [11..31] as well as character code [127] are *dead* character codes in the sense that if they occur in input, they should be discarded before they make their way into strings.

In Logiweb ASCII, character code [10], nothing but character code [10], and always character code [10], represents the newline character.

Logiweb ASCII is similar to ASCII in that no characters have codes outside [0..127].

Logiweb ASCII has two formatting character: newline (code [10]) and space (code [32]). All characters in the range [33..126] are printable glyphs.

When running Logiweb under some host operating system, the convention that character [10] is the newline character may equate or differ from the convention of the host operating system. In the very few cases where strings are exchanged between Logiweb and surrounding systems, one should translate between the newline conventions of Logiweb and the surroundings during the exchange.

Translation from ASCII strings to cardinals and vice versa is particularly simple. To translate e.g. the string “abc”, do as follows. First, write the string backwards: “cba”. Then convert each character to an eight bit byte:

$$[0110\ 0011 \quad 0110\ 0010 \quad 0110\ 0001]$$

Finally, interpret the result as a binary number.

Needless to say (so we say it for safety's sake), the above representation of ASCII strings by cardinals is quite memory efficient.

4.1.3 Constructors for binary numbers

To express binary numbers easily, we introduce the following three constructs:

$$[\mathsf{x}0 \overset{\text{val}}{\to} \mathsf{T} +2* \mathsf{x}]$$

$$[\mathsf{x}1 \overset{\text{val}}{\to} \mathsf{F} +2* \mathsf{x}]$$

$$[0\mathsf{b} \overset{\text{val}}{\to} 0]$$

4.1.4 Representation of Logiweb identifiers

We shall refer to a particular, short, finite list of strings as *Logiweb indentifiers*. The list of identifiers, and the semantics of each identifier, is at the heart of the Logiweb standard. At the time of writing I, the author of the present paper, control that list.

Each Logiweb indentifier has the property that it only includes Unicode characters from 97 to 122 (small letter a to small letter z). Now define

$$[\text{identifier}(\mathsf{x}) \overset{\text{val}}{\to} \text{If}(\mathsf{x}^{\mathsf{t}}, 0, \text{identifier}_1(\mathsf{x}^{\mathsf{i}}, \text{identifier}(\mathsf{x}^1)))]$$

[45]

$$[\text{identifier}_1(\mathsf{x}, \mathsf{y}) \overset{\text{val}}{\to} \text{If}(\mathsf{x}^6, \mathsf{y}, \mathsf{x}^0 +2* \mathsf{x}^1 +2* \mathsf{x}^2 +2* \mathsf{x}^3 +2* \mathsf{x}^4 +2* \mathsf{x}^5 +2* \mathsf{F} +2* \mathsf{T} +2* \mathsf{y})]$$

[46]

We have that e.g. [identifier(⌈“code”⌉)] is the string [“code”] translated to the cardinal specified in Section 4.1.1:

$$[\text{identifier}(\ulcorner \text{“code”} \urcorner) \approx 0\text{b}11001010110010001101111011000 11]\cdot$$

In general, [identifier(⌈$\mathcal{A}$⌉)] converts the string [$\mathcal{A}$] correctly under the following conditions:

[45] $[\text{identifier}(\mathsf{x}) \overset{\text{pyk}}{=} \text{“identifier} * \text{end identifier”}]$

[46] $[\text{identifier}_1(\mathsf{x}, \mathsf{y}) \overset{\text{pyk}}{=} \text{“identifier one} * \text{plus id} * \text{end identifier”}]$

- All non-blind characters of the string are between 64 and 126
- All blind characters of the string are between 0 and 9 or between 11 and 31.
- All characters are represented by unary operators except the end-of-text character, which must be nulary and blind.

4.2 Associative structures

4.2.1 Associations trees

We define the set of *associations* as the set of pairs $[\mathsf{k} :: \mathsf{v}]$ where $[\mathsf{k}]$ is a cardinal and $[\mathsf{v}]$ is a tagged tree. For an association $[\mathsf{k} :: \mathsf{v}]$ we shall refer to $[\mathsf{k}]$ and $[\mathsf{v}]$ as the *key* and *value* of the association, respectively.

We define the set of *association trees* as the smallest set which contains $[\mathsf{T}]$, which contains all associations, and which contains $[\mathsf{a} :: \mathsf{b}]$ for all association trees $[\mathsf{a}]$ and $[\mathsf{b}]$. Each association tree represents a finite set of associations.

If all keys of an association tree $[\mathsf{a}]$ are distinct, then $[\mathsf{a}]$ represents the function that maps $[\mathsf{k}]$ to $[\mathsf{v}]$ whenever $[\mathsf{k} :: \mathsf{v}]$ belongs to $[\mathsf{a}]$ and maps $[\mathsf{k}]$ to $[\mathsf{T}]$ whenever $[\mathsf{k}]$ is a cardinal which does not occur as a key in $[\mathsf{a}]$.

In particular, the association tree $[\mathsf{T}]$ represents the function that maps all cardinals to $[\mathsf{T}]$.

4.2.2 Finite functions

We shall say that a function $[\mathsf{f}]$ is a *finite function* if it maps cardinals to tagged trees and, furthermore, maps at most finitely many cardinals to tagged trees different from $[\mathsf{T}]$.

One may represent finite functions by association trees.

4.2.3 Addresses

Each association in an association tree $[\mathsf{a}]$ can be accessed using a sequence of head and tail operations. As an example, if $[\mathsf{a}]$ is an association tree and $[\mathsf{a}^{\mathrm{hthh}}]$ is an association, then the association can be addressed by a head-tail-head-head sequence.

We now represent "head" by binary zero and "tail" by binary one. As an example, if $[\mathsf{a}]$ is an association tree and $[\mathsf{a}^{\mathrm{hthh}}]$ is an association then we say that the association occurs at address $[0, 1, 0, 0]$.

4.2.4 Arrays

A cardinal $[\mathsf{k}]$ may be written on form

$$\left[\sum_{\mathrm{n}=0}^{\infty} \mathrm{b}_{\mathrm{n}} 2^{\mathrm{n}}\right]$$

where $[b_0, b_1, b_2, \ldots]$ are bits (i.e. "binary terms", i.e. zero or one). We shall refer to $[b_0, b_1, b_2, \ldots]$ as the little endian, infinite binary representation of $[k]$.

We shall say that an association tree $[a]$ is an *array* if it satisfies the following two conditions:

1. For every association $[k :: v]$ in $[a]$, the address of the association is a prefix of the little endian, infinite binary representation of $[k]$.
2. No substructure of $[a]$ has form $[T :: p]$ or $[p :: T]$ where $[p]$ is an association.

The first condition makes it fast and easy to look up a key $[k]$ in an array. The second condition together with the first ensures that each finite function is represented by exactly one array.

4.2.5 Array access

We shall say that a data structure is *atomic* if it is not a pair. $[x^a]$ is true if $[x]$ is atomic:

$$[x^a \xrightarrow{val} \neg x^d \vee x^c \vee x^s]$$

$[a[k]]$ looks up the value associated to the key $[k]$ in the array $[a]$:

$$[a[k] \xrightarrow{val} \mathbf{assoc}_1(a, k, k)]$$

$$[\mathbf{assoc}_1(a, d, i) \xrightarrow{val} a^a \left\{ \begin{array}{l} d!i!T \\ \\ a^{hc} \left\{ \begin{array}{l} i \approx a^h \left\{ \begin{array}{l} d!a^t \\ d!T \end{array} \right. \\ \\ d^h \left\{ \begin{array}{l} \mathbf{assoc}_1(a^h, d^t, i) \\ \mathbf{assoc}_1(a^t, d^t, i) \end{array} \right. \end{array} \right. \end{array} \right. \quad]$$

The above representation of arrays allows array access and update to occur reasonably efficiently. If even greater efficiency is needed, one may implement arrays as hash tables behind the scenes. The first edition engine (the only Logiweb engine in existence at the time of writing) does not treat arrays specially behind the scenes. But arrays are still defined the way they are in order to allow improvements in the future.

As a matter of terminology, we shall refer to keys of arrays as *incices*. So an array maps indices to values.

4.2.6 Racks

Formally, we shall use "*rack*" as a synonym for "array". Informally, we shall use "array" and "rack" for homogeneous and heterogeneous structures, respectively. So an array is an association list whose values are of homogeneous format and a rack is an association list with heterogeneous values.

As a matter of terminology, we shall refer to keys of racks as *hooks*. So a rack maps hooks to values.

4.2.7 Multidimensional arrays

We shall refer to arbitrary values as *zero dimensional arrays* and to arrays of [n]-dimensional arrays as [n + 1]-dimensional arrays. As an example, if [a] is a three-dimensional array and [u], [v], and [w] are cardinals, then

[a[u][v][w]]

is the value indexed by [u], [v], and [w].

4.2.8 Array assignment

[a[i→v]] equals the array [a] except that [a[i→v]] maps the index [i] to the value [v].

$$[\mathsf{a}[\mathsf{i}{\rightarrow}\mathsf{v}] \doteq \mathsf{i}^{\mathrm{c}} \left\{ \begin{array}{l} \mathsf{v} \left\{ \begin{array}{l} \text{array-remove}(\mathsf{i},\mathsf{a},0) \\ \text{array-put}(\mathsf{i},\mathsf{v},\mathsf{a},0) \end{array} \right. \\ \mathsf{v}!\mathsf{a} \end{array} \right.]^{47}$$

$$[\text{array-plus}(\mathsf{x},\mathsf{y}) \doteq \mathsf{x}^{\mathrm{a}} \left\{ \begin{array}{l} \mathsf{y}^{\mathrm{a}} \left\{ \begin{array}{l} \mathsf{T} \\ \mathsf{y}^{\mathrm{hc}} \left\{ \begin{array}{l} \mathsf{y} \\ \mathsf{x} :: \mathsf{y} \end{array} \right. \\ \mathsf{x}^{\mathrm{hc}} \left\{ \begin{array}{l} \mathsf{x} \\ \mathsf{x} :: \mathsf{y} \end{array} \right. \end{array} \right. \\ \mathsf{y}^{\mathrm{a}} \left\{ \begin{array}{l} \\ \mathsf{x} :: \mathsf{y} \end{array} \right. \end{array} \right.]^{48}$$

$$[\text{array-remove}(\mathsf{i},\mathsf{a},\mathsf{l}) \doteq \mathsf{i}!\mathsf{l}!\mathsf{a}^{\mathrm{a}} \left\{ \begin{array}{l} \mathsf{T} \\ \mathsf{a}^{\mathrm{hc}} \left\{ \begin{array}{l} \mathsf{a}^{\mathrm{h}} \approx \mathsf{i} \left\{ \begin{array}{l} \mathsf{T} \\ \mathsf{a} \end{array} \right. \\ \text{bit}(\mathsf{l},\mathsf{i}) \left\{ \begin{array}{l} \text{array-plus}(\text{array-remove}(\mathsf{i},\mathsf{a}^{\mathrm{h}},\mathsf{l}+1),\mathsf{a}^{\mathrm{t}}) \\ \text{array-plus}(\mathsf{a}^{\mathrm{h}},\text{array-remove}(\mathsf{i},\mathsf{a}^{\mathrm{t}},\mathsf{l}+1)) \end{array} \right. \end{array} \right. \end{array} \right.]^{49}$$

$$[\text{array-put}(\mathsf{i},\mathsf{v},\mathsf{a},\mathsf{l}) \doteq \mathsf{l}!\mathsf{a}^{\mathrm{a}} \left\{ \begin{array}{l} \mathsf{i} :: \mathsf{v} \\ \mathsf{a}^{\mathrm{hc}} \left\{ \begin{array}{l} \mathsf{a}^{\mathrm{h}} \approx \mathsf{i} \left\{ \begin{array}{l} \mathsf{i} :: \mathsf{v} \\ \text{array-add}(\mathsf{i},\mathsf{v},\mathsf{a}^{\mathrm{h}},\mathsf{a}^{\mathrm{t}},\mathsf{l}) \end{array} \right. \\ \text{bit}(\mathsf{l},\mathsf{i}) \left\{ \begin{array}{l} \text{array-put}(\mathsf{i},\mathsf{v},\mathsf{a}^{\mathrm{h}},\mathsf{l}+1) :: \mathsf{a}^{\mathrm{t}} \\ \mathsf{a}^{\mathrm{h}} :: \text{array-put}(\mathsf{i},\mathsf{v},\mathsf{a}^{\mathrm{t}},\mathsf{l}+1) \end{array} \right. \end{array} \right. \end{array} \right.]^{50}$$

[47] [a[i→v]] $\stackrel{\text{pyk}}{=}$ "∗ set ∗ to ∗ end set"]

[48] [array-plus(x, y)] $\stackrel{\text{pyk}}{=}$ "array plus ∗ and ∗ end plus"]

[49] [array-remove(i, a, l)] $\stackrel{\text{pyk}}{=}$ "array remove ∗ array ∗ level ∗ end remove"]

[50] [array-put(i, v, a, l)] $\stackrel{\text{pyk}}{=}$ "array put ∗ value ∗ array ∗ level ∗ end put"]

$$[\text{array-add}(\mathsf{i},\mathsf{v},\mathsf{i}',\mathsf{v}',\mathsf{l}) \doteq \text{bit}(\mathsf{l},\mathsf{i}) \begin{cases} \text{bit}(\mathsf{l},\mathsf{i}') \begin{cases} \text{array-add}(\mathsf{i},\mathsf{v},\mathsf{i}',\mathsf{v}',\mathsf{l}+1) :: \mathsf{T} \\ (\mathsf{i} :: \mathsf{v})^{\mathrm{M}} :: (\mathsf{i}' :: \mathsf{v}')^{\mathrm{M}} \end{cases} \\ \text{bit}(\mathsf{l},\mathsf{i}') \begin{cases} (\mathsf{i}' :: \mathsf{v}')^{\mathrm{M}} :: (\mathsf{i} :: \mathsf{v})^{\mathrm{M}} \\ \mathsf{T} :: \text{array-add}(\mathsf{i},\mathsf{v},\mathsf{i}',\mathsf{v}',\mathsf{l}+1) \end{cases} \end{cases}]$$ [51]

4.2.9 Multidimensional assignment

[a[i⇒v]] equals the multidimensional array [a] except that [a[i⇒v]] maps the list [i] of indices to the value [v]. As an example, [T[1 :: 2 :: 3 :: T⇒4][1][2][3]] equals [4].

$$[\mathsf{a}[\mathsf{i}\Rightarrow\mathsf{v}] \doteq \mathsf{i}^{\mathrm{a}} \begin{cases} \mathsf{a}!\mathsf{v} \\ \mathsf{a}[\mathsf{i}^{\mathrm{h}}\rightarrow\mathsf{a}[\mathsf{i}^{\mathrm{h}}][\mathsf{i}^{\mathrm{t}}\Rightarrow\mathsf{v}]] \end{cases}]$$ [52]

4.2.10 Variables

The term [λx.x] contains the variable [x].

In Logiweb, any term may serve as a variable. Whether or not a given term is interpreted as a variable depends on context.

A lambda construct forces its first argument to be a variable. This forces the first occurrence of [x] in [λx.x] to be a variable.

The lambda does not force the second occurrence of [x] in [λx.x] to be a variable. But the second occurrence is a variable anyway because of another rule which says that any term whose root symbol has no value definition is a variable.

As a more bizarre example,

$$[\lambda 2.2]$$

is a valid term. The lambda forces the first occurrence of [2] to be a variable. The second occurrence of [2] is no variable, however, since nothing forces the second [2] to be a variable and [2] has a value definition. Renaming of the bound variable [2] to [x] shows that [λ2.2] equals [λx.2]:

$$[\lambda 2.2 = \lambda \mathsf{x}.2]$$

We now introduce the binary construct [x_y], but we assign no value to it. For that reason, any term with the [x_y] construct in the root will be a variable. As an example of use,

$$[\lambda \mathsf{x}_1.\lambda \mathsf{x}_2.\mathsf{x}_1 :: \mathsf{x}_2]$$

equals

$$[\lambda \mathsf{u}.\lambda \mathsf{v}.\mathsf{u} :: \mathsf{v}]$$

[51] [array-add(i, v, i′, v′, l) $\stackrel{\text{pyk}}{=}$ "array add ∗ value ∗ index ∗ value ∗ level ∗ end add"]

[52] [a[i⇒v] $\stackrel{\text{pyk}}{=}$ "∗ set multi ∗ to ∗ end set"]

To boost the number of available variables even further, we introduce a unary prime operator $[\mathsf{x}']$[53] which allows to use $[\mathsf{x}']$, $[\mathsf{y}']$, $[\mathsf{z}''']$, and so on as variables.

Two variables are equal if they are identical terms (disregarding debugging information). For that reason, even if we defined $[\mathsf{x}+\mathsf{y}]$ such that $[2+2]$ equaled $[4]$, $[\mathsf{x}_{2+2}]$ and $[\mathsf{x}_4]$ would still be distinct variables.

4.2.11 Stacks

We shall refer to dynamic environments as *stacks*. A stack is an association list from variables to values.

We cannot use $[\mathsf{s}[\mathsf{v}]]$ to look up the variable $[\mathsf{v}]$ in the stack $[\mathsf{s}]$ because we should disregard debugging information present in $[\mathsf{v}]$. For that reason, we use the following construct instead:

$$[\mathbf{lookup}(\mathsf{v},\mathsf{s},\mathsf{d}) \overset{\text{val}}{\rightarrow} \mathsf{v}!\mathsf{d}!\mathrm{If}(\mathsf{s},\mathsf{d},\mathrm{If}(\mathsf{v} \overset{\text{t}}{=} \mathsf{s}^{\text{hh}},\mathsf{s}^{\text{ht}},\mathbf{lookup}(\mathsf{v},\mathsf{s}^{\text{t}},\mathsf{d})))]$$

$[\mathbf{lookup}(\mathsf{v},\mathsf{s},\mathsf{d})]$ looks up the variable $[\mathsf{v}]$ in the stack $[\mathsf{s}]$. If the variable is not found, $[\mathbf{lookup}(\mathsf{v},\mathsf{s},\mathsf{d})]$ returns the default value $[\mathsf{d}]$ instead.

The $[\mathbf{zip}(\mathsf{p},\mathsf{a})]$ construct zips two lists into a list of pairs and is suited for forming stacks:

$$[\mathbf{zip}(\mathsf{p},\mathsf{a}) \overset{\text{val}}{\rightarrow} \mathsf{a}!\mathrm{If}(\mathsf{p},\mathsf{T},(\mathsf{p}^{\text{h}} :: \mathsf{a}^{\text{h}})^{\text{M}} :: \mathbf{zip}(\mathsf{p}^{\text{t}},\mathsf{a}^{\text{t}}))]$$

4.3 Static semantics

4.3.1 The structure of Logiweb pages

As far as Logiweb is concerned, a Logiweb page consists of a *bibliography*, a *dictionary*, and a *body*. What you read right now is a rendering of the body of a Logiweb page.

The Logiweb bibliography of a page is a list of Logiweb references. The first reference (reference number zero) is the reference of the page itself, and the other references are references of other Logiweb pages.

The bibliography at the end of the present page is a BibTeX bibliography which is part of the body of the page. The BibTeX bibliography is completely unrelated to the Logiweb bibliography.

The Logiweb dictionary of a page is an array which maps identifiers to arities. If the dictionary of the page with reference $[\mathsf{r}]$ maps the cardinal $[\mathsf{i}]$ to $[\mathsf{a}]$ then, by definition, there exists a Logiweb symbol with reference $[\mathsf{r}]$ and identifier $[\mathsf{i}]$, and that symbol has arity $[\mathsf{a}]$.

The Logiweb body of a page is one, big term.

[53] $[\mathsf{x}' \overset{\text{pyk}}{=} \text{“}* \text{ prime”}]$

4.3.2 Transitive bibliographies

As mentioned, the first entry of the Logiweb bibliography of any page is the reference of the page itself. Apart from this, the bibliographic references of Logiweb form a non-cyclic graph.

Given a Logiweb page [p], we shall define the *transitive bibliography* of [p] as the set of references of all pages reachable following bibliographic links starting at [p]. The transitive bibliography of a page includes the reference of the page itself.

We define the transitive, irreflexive bibliography of a Logiweb page as the transitive bibliography with the reference of the page itself removed.

4.3.3 Loading, referencing, and verbatim copying

Logiweb is able to *load* Logiweb pages. Loading a Logiweb page involves a number of activities such as locating, retrieving, unpacking, and codifying the page. To display a page the system has to load and then render the page.

Loading a page is cascading in that loading a page requires loading of all pages in the transitive bibliography of the page.

A page cannot be loaded unless all pages in the transitive bibliography can be found. Hence, deleting a page from Logiweb is a serious thing since it ruins all pages that refer, directly or indirectly, to the deleted page. Fortunately, Logiweb allows duplication of pages so that a single Logiweb page may reside several places in the world. The author of a Logiweb page should secure a local copy of all pages in the transitive bibliography of the page unless the author trusts other people very much.

Referencing, indexing, and verbatim copying are core activities of Logiweb; if one does not allow referencing, indexing, and verbatim copying of a page, one should not submit it to Logiweb. In other words, the author of a page silently permits referencing, indexing, and verbatim copying by submitting it.

Authors may of course still claim copyright to a Logiweb page. That copyright may prevent other people from e.g. modifying the page, publishing the page outside Logiweb, or other activities other than the core activities of Logiweb.

4.3.4 The cache of a page

Loading a page results in the construction of a *cache* and a *rack* for the page. The cache and rack of a page represent the static semantics of the page.

The cache [c] of a page [p] is an array which maps the references in the transitive bibliography of [p] to the rack of the associated page. Hence, [c[r]] is the rack of the page referenced by [r]. In particular, the cache of a page contains the rack of the page itself.

The cache [c] of a page is an almost homogeneous array. The only exception is that [c[0]] equals the reference of the page. As a consequence, if [c] is the cache of a page then [c[c[0]]] is the rack of the page.

4.3.5 The value of the page construct

A *page construct* (the construct of a page whose id is zero) represents the page. As an example, the pyk and tex aspects of a page construct effectively become the name of the page expressed in pyk and TEX, respectively. The page construct is always nulary.

Evaluation of a page construct always yields the Logiweb cache of the associated page. In this way, the evaluator has a clean way to access its environment. Logiweb ignores value definitions of page constructs.

From the point of view of mathematical reasoning, page constructs are constants whose values happen to be the cache of their associated pages.

[ijcar base] is the page construct of the present page. Hence, the value of [ijcar base] is the cache of the present page.

According to Section 4.3.4, [ijcar base[0]] is the reference of the present page. If we take any construct introduced on the present page, say [7], then the reference of the root of that construct must equal [ijcar base[0]]:

$$[\text{ijcar base}[0] \approx \lceil 7 \rceil^{\mathrm{r}}]^{\cdot}$$

In section 4.5.2 we macro define [self] so that [self] macro expands to the page construct of the page on which the construct occurs. Hence, on the present page, [self] expands to [ijcar base]. On other pages, [self] expands to the page construct of those other pages.

We could use [self] in place of [ijcar base] above. Logiweb does not care about the order of definitions, and it is completely valid to use macros before they are defined. But beware of circularities, they can lead to intractable errors.

4.3.6 The rack of a page

According to Section 4.3.4, [rack $\doteq$ ijcar base[ijcar base[0]]][54] is the rack of the present page.

The rack of a page maps various, predefined hooks to various values. The hooks of the rack are Logiweb identifiers (c.f. Section 4.1.4). At the time of writing, the rack [a] of a page has the following hooks:

- ["vector" $\overset{\text{val}}{\rightarrow}$ identifier($\lceil$"vector"$\rceil$)][55] The vector of the page, i.e. the sequence of bytes that constitute the page when it is stored on disc or transmitted over a network. As an example, [rack["vector"][h]] is the first byte of the vector that represents the present page. That byte happens to be the major version number so, since the present page is encoded using Logiweb version [1] we have [rack["vector"][h] $\approx$ 1]$^{\cdot}$.

- ["bibliography" $\overset{\text{val}}{\rightarrow}$ identifier($\lceil$"bibliography"$\rceil$)][56] The bibliography of the page, i.e. a list of references where the first reference is the reference of

[54] [rack $\overset{\text{pyk}}{=}$ "example rack"]

[55] ["vector" $\overset{\text{pyk}}{=}$ "vector hook"]

[56] ["bibliography" $\overset{\text{pyk}}{=}$ "bibliography hook"]

the page itself and the other references are references of other pages. The present page is a base page in the sense that it references no other pages: $[\text{rack}[\text{"bibliography"}] \approx \lceil 7 \rceil^{\text{r}} :: \mathsf{T}]^{\cdot}$.

- $[\text{"dictionary"} \overset{\text{val}}{\to} \text{identifier}(\lceil \text{“dictionary”} \rceil)]$[57] The dictionary of the page, i.e. an array that maps the identifiers of the symbols introduced on the page to their arities. As an example, the arity of $[\mathsf{x} :: \mathsf{y}]$ is two:

  $$[\text{rack}[\text{"dictionary"}][\lceil \mathsf{x} :: \mathsf{y} \rceil^{\text{i}}] \approx 2]^{\cdot}$$

  We say that a *symbol exists* if the dictionary of the page pointed out by the reference of the symbol maps the identifier of the symbol to a value different from $[\mathsf{T}]$.

- $[\text{"body"} \overset{\text{val}}{\to} \text{identifier}(\lceil \text{“body”} \rceil)]$[58] The body of the page, i.e. the term that one should typeset when viewing the page. On a base page, the root of the body necessarily comes from the page itself: $[\text{rack}[\text{"body"}]^{\text{r}} \approx \lceil 7 \rceil^{\text{r}}]^{\cdot}$.

- $[\text{"codex"} \overset{\text{val}}{\to} \text{identifier}(\lceil \text{“codex”} \rceil)]$[59] An array that contains all revelations (i.e. proclamations, definitons, and introductions) of the page as described in Section 4.3.7.

- $[\text{"expansion"} \overset{\text{val}}{\to} \text{identifier}(\lceil \text{“expansion”} \rceil)]$[60] The macro expanded version of the body of the page. The expanded version is always a syntactically valid term, i.e. a term in which all symbols exist and have correct arities as specified in the dictionaries. Logiweb ensures this syntactic validity in a roundabout way: The macro expansion facility is Turing complete and can produce any value as an output. Macro expansion may even loop indefinitely, in which case the author of the page should consider to correct the page before submitting it to Logiweb. If macro expansion does produce a value, then Logiweb checks the result for syntactical validity, and whenever it finds an invalid symbol or a symbol whose arity does not match the number of subtrees, Logiweb replaces the tree rooted at the invalid symbol with the page construct of the present page.

- $[\text{"code"} \overset{\text{val}}{\to} \text{identifier}(\lceil \text{“code”} \rceil)]$[61] An array that contains compiled versions of all value revelations of a page. With two exceptions, the compiled versions are tagged, curried functions: $[\mathcal{U}(\text{rack}[\text{"code"}][\lceil \mathsf{x} :: \mathsf{y} \rceil^{\text{i}}] \text{ ‘ } 2 \text{ ‘ } 3) \approx 2 :: 3]^{\cdot}$. The first exceptions is lambda abstraction which, rather arbitrarily, is represented by a zero: $[\text{rack}[\text{"code"}][\lceil \lambda \mathsf{x}.\mathsf{y} \rceil^{\text{i}}] \approx 0]^{\cdot}$. The other exception is quoting which is represented by a one: $[\text{rack}[\text{"code"}][\lceil \lceil \mathsf{x} \rceil \rceil^{\text{i}}] \approx 1]^{\cdot}$.

[57] $[\text{"dictionary"} \overset{\text{pyk}}{=} \text{“dictionary hook”}]$

[58] $[\text{"body"} \overset{\text{pyk}}{=} \text{“body hook”}]$

[59] $[\text{"codex"} \overset{\text{pyk}}{=} \text{“codex hook”}]$

[60] $[\text{"expansion"} \overset{\text{pyk}}{=} \text{“expansion hook”}]$

[61] $[\text{"code"} \overset{\text{pyk}}{=} \text{“code hook”}]$

- ["cache" $\overset{\text{val}}{\to}$ identifier(⌜"cache"⌝)][62] An array which maps all references in the transitive, irreflexive bibliography of the page [p] to the caches of the associated pages. The present page is a base page in the sense that it references no other pages, so the cache that hangs on the cache hook is empty: [rack["cache"] ≈ T]˙. If the present page did reference a page with reference [r] then [rack["cache"][r]] would be the cache of page [r]. In that case, [ijcar base[r]] as well as [rack["cache"][r][r]] would be the rack of page [r]. In general, the racks and caches of referenced pages can be addressed several ways.

- ["diagnose" $\overset{\text{val}}{\to}$ identifier(⌜"diagnose"⌝)][63] If the value that hangs on the diagnose hook is [T] then the page is correct. Otherwise, the value is a term which, when typeset, is supposed to explain what is wrong with the page. In other words, the diagnose hook is where the checking machinery hangs its complaints. The diagnose is pruned to a term the same way as the expansion is. The diagnose is added after checking. During checking the value that hangs on the diagnose hook is always [T].

Constants (i.e. nulary constructs) like the ones above are computed lazily and the result is kept. For that reason, constants are computed at most once when a Logiweb page is loaded. Computation of e.g. [identifier(⌜"diagnose"⌝)] is not particularly time and memory efficient but that is irrelevant when using ["diagnose"] since it is only evaluated once.

4.3.7 The codex of a page

The codex [c] of a page is a four dimensional array that hangs on the ["codex"] hook of the rack of the page.

A page may contain revelations (i.e. proclamations, definitions, and introductions), and each revelation defines some aspect of some symbol.

If $[\mathsf{r_s}]$ and $[\mathsf{i_s}]$ are the reference and identifier, respectively, of a symbol and if $[\mathsf{r_a}]$ and $[\mathsf{i_a}]$ are the reference and identifier, respectively, of an aspect, then

$$[\mathsf{c}[\mathsf{r_s}][\mathsf{i_s}][\mathsf{r_a}][\mathsf{i_a}]]$$

is the revelation (if any) of the given aspect of the given symbol.

If the revelation is a definition or introduction, the thing stored in the codex is the entire term that constitutes the revelation. In this case, the first argument of the definition is the aspect (with possible parameters), the second argument is the left hand side of the definition (with possible parameters), and the third argument is the right hand side. The root of the term is a symbol that is proclaimed to denote the "definition" or "introduction" concept.

If the revelation is a proclamation, the thing stored in the codex has form $[(0 :: \mathsf{i} :: \mathsf{T}) :: \mathsf{T}]$ where [i] where the cardinal [i] identifies the proclamation as explained in Section 4.3.12.

[62] ["cache" $\overset{\text{pyk}}{=}$ "cache hook"]

[63] ["diagnose" $\overset{\text{pyk}}{=}$ "diagnose hook"]

4.3.8 Aspects

An aspect may be predefined or user defined.

User defined aspects are introduced using a mechanism described later. The reference and identifier of a user defined aspect always identify an existing Logiweb symbol.

The reference of a predefined aspect equals zero and the identifier is a Logiweb identifier. At the time of writing, the following Logiweb identifiers denote predefined aspects:

- ["value" $\overset{\text{val}}{\to}$ identifier(⌜"value"⌝)][64]
- ["pyk" $\overset{\text{val}}{\to}$ identifier(⌜"pyk"⌝)][65]
- ["tex" $\overset{\text{val}}{\to}$ identifier(⌜"tex"⌝)][66]
- ["texname" $\overset{\text{val}}{\to}$ identifier(⌜"texname"⌝)][67]
- ["message" $\overset{\text{val}}{\to}$ identifier(⌜"message"⌝)][68]
- ["macro" $\overset{\text{val}}{\to}$ identifier(⌜"macro"⌝)][69]
- ["definition" $\overset{\text{val}}{\to}$ identifier(⌜"definition"⌝)][70]
- ["unpack" $\overset{\text{val}}{\to}$ identifier(⌜"unpack"⌝)][71]
- ["claim" $\overset{\text{val}}{\to}$ identifier(⌜"claim"⌝)][72]
- ["priority" $\overset{\text{val}}{\to}$ identifier(⌜"priority"⌝)][73]

As an example, if [c] is the codex of a page and if the page makes a value definition of a symbol with reference [r] and identifier [i] then

[c[r][i][0]["value"]]

will equal that definition.

---

[64] ["value" $\overset{\text{pyk}}{=}$ "value aspect"]
[65] ["pyk" $\overset{\text{pyk}}{=}$ "pyk aspect"]
[66] ["tex" $\overset{\text{pyk}}{=}$ "tex aspect"]
[67] ["texname" $\overset{\text{pyk}}{=}$ "texname aspect"]
[68] ["message" $\overset{\text{pyk}}{=}$ "message aspect"]
[69] ["macro" $\overset{\text{pyk}}{=}$ "macro aspect"]
[70] ["definition" $\overset{\text{pyk}}{=}$ "definition aspect"]
[71] ["unpack" $\overset{\text{pyk}}{=}$ "unpack aspect"]
[72] ["claim" $\overset{\text{pyk}}{=}$ "claim aspect"]
[73] ["priority" $\overset{\text{pyk}}{=}$ "priority aspect"]

4.3.9 Domestic and foreign definitions

If a page with reference [r] has cache [c] then e.g.

$$[\mathsf{c}[\mathsf{r}]["\text{codex}"][\mathsf{r_s}][\mathsf{i_s}][0]["\text{value}"]]$$

denotes the value aspect of the symbol with reference $[\mathsf{r_s}]$ and identifier $[\mathsf{i_s}]$ as defined on page [r].

We shall say that a definition is *domestic* if $[\mathsf{r} = \mathsf{r_s}]$ and *foreign* otherwise. In other words, a definition is domestic if it occurs on the home page of the symbol being defined.

Logiweb ignores foreign value definitions but store them in the codex for the record. In general, Logiweb ignores foreign definitions of any predefined aspect.

One could decide that foreign definitions could shadow domestic ones in certain situations. Actually, shadowing of pyk, tex, and tex name aspects has been considered for Logiweb, but has been rejected to keep core Logiweb simple. Authors who define their own user aspects may decide if and how foreign definitions may shadow domestic ones for their aspects.

4.3.10 A codex accessor

$[\mathbf{aspect}(\mathsf{a},\mathsf{c})]$[74] looks up the the aspect [a] in the *subcodex* [c]. The subcodex is assumed to be the result of accessing a codex be the reference and identifier of a symbol. The subcodex is what is called a *property list* in Lisp [9, 11].

If [a] is a cardinal then it is interpreted as the Logiweb identifier of a predefined aspect. Otherwise, [a] is interpreted as a term whose root symbol denotes a user defined aspect.

$$[\mathbf{aspect}(\mathsf{a},\mathsf{c}) \overset{\text{val}}{\to} \mathsf{a}^{\mathsf{c}} \begin{cases} \mathsf{c}[0][\mathsf{a}] \\ \mathsf{c}[\mathsf{a}^{\mathsf{r}}][\mathsf{a}^{\mathsf{i}}] \end{cases}]$$

$[\mathbf{aspect}(\mathsf{a},\mathsf{t},\mathsf{c})]$[75] looks up the domestic definition of the root of the term [t] for the aspect [a] in the cache [c]. If [a] is a cardinal then it is interpreted as the Logiweb identifier of a predefined aspect. Otherwise, [a] is interpreted as a term whose root symbol denotes a user defined aspect.

$$[\mathbf{aspect}(\mathsf{a},\mathsf{t},\mathsf{c}) \overset{\text{val}}{\to} \mathbf{aspect}(\mathsf{a},\mathsf{c}[\mathsf{t}^{\mathsf{r}}]["\text{codex}"][\mathsf{t}^{\mathsf{r}}][\mathsf{t}^{\mathsf{i}}])]$$

4.3.11 Value proclamations

Proclaiming a construct to denote "apply", "lambda", "true", "if", or "quote" affects the value aspect of the construct. For that reason we shall refer to proclamations with one of these strings in the right hand side as *value proclamations*.

The strings that make sense in the right hand side of proclamations form a subset of the strings that define Logiweb identifiers. The Logiweb identifiers associated to the five value proclamations are defined thus:

---

[74] $[\mathbf{aspect}(\mathsf{a},\mathsf{c}) \overset{\text{pyk}}{=}$ "aspect $*$ subcodex $*$ end aspect"]

[75] $[\mathbf{aspect}(\mathsf{a},\mathsf{t},\mathsf{c}) \overset{\text{pyk}}{=}$ "aspect $*$ term $*$ cache $*$ end aspect"]

- $[\texttt{"lambda"} \overset{\text{val}}{\rightarrow} \text{identifier}(\lceil \text{“lambda”} \rceil)]$[76]
- $[\texttt{"apply"} \overset{\text{val}}{\rightarrow} \text{identifier}(\lceil \text{“apply”} \rceil)]$[77]
- $[\texttt{"true"} \overset{\text{val}}{\rightarrow} \text{identifier}(\lceil \text{“true”} \rceil)]$[78]
- $[\texttt{"if"} \overset{\text{val}}{\rightarrow} \text{identifier}(\lceil \text{“if”} \rceil)]$[79]
- $[\texttt{"quote"} \overset{\text{val}}{\rightarrow} \text{identifier}(\lceil \text{“quote”} \rceil)]$[80]

4.3.12 Codification of revelations

Logiweb *codifies* a definition like

$$[\mathsf{x}^{\mathrm{H}} \overset{\text{val}}{\rightarrow} \mathsf{x}\,\text{'}\,\mathsf{T}]$$

simply by placing the entire definition in the codex:

$$[\textbf{aspect}(\texttt{"value"}, \lceil \mathsf{x}^{\mathrm{H}} \rceil, \text{ijcar base}) \overset{\text{t}}{=} \lceil [\mathsf{x}^{\mathrm{H}} \overset{\text{val}}{\rightarrow} \mathsf{x}\,\text{'}\,\mathsf{T}] \rceil]^{\cdot}$$

Having that definition, one can gain access not only to the right hand side of the definition, but also e.g. the parameter list of the left hand side. If one decides to have parameterized aspects, then one will also have access to the parameter list of the aspect in the codex.

Introductions are treated exactly the same way. An introduction like

$$[\mathsf{y} \mathrel{\therefore} \mathsf{z} \overset{\text{val}}{\Rightarrow} \lambda\mathsf{x}.\text{if}(\mathsf{x}, \mathsf{y}, \mathsf{z})]$$

results in the following entry in the codex:

$$[\textbf{aspect}(\texttt{"value"}, \lceil \mathsf{y} \mathrel{\therefore} \mathsf{z} \rceil, \text{ijcar base}) \overset{\text{t}}{=} \lceil [\mathsf{y} \mathrel{\therefore} \mathsf{z} \overset{\text{val}}{\Rightarrow} \lambda\mathsf{x}.\text{if}(\mathsf{x}, \mathsf{y}, \mathsf{z})] \rceil]^{\cdot}$$

In contrast, a proclamation like

$$[\lambda\mathsf{x}.\mathsf{y} \bowtie \text{“lambda”}]$$

results in an entry like

$$[\textbf{aspect}(\texttt{"value"}, \lceil \lambda\mathsf{x}.\mathsf{y} \rceil, \text{ijcar base}) \approx (0 :: \texttt{"lambda"} :: \mathsf{T})^{\mathrm{I}} :: \mathsf{T}]^{\cdot}$$

In general, a proclamation results in an entry of form $[(0 :: \mathsf{i} :: \mathsf{T})^{\mathrm{I}} :: \mathsf{T}]$ where [i] is the Logiweb identifier that corresponds to the string in the right hand side of the proclamation.

---

[76] $[\texttt{"lambda"} \overset{\text{pyk}}{=} \text{“lambda identifier”}]$

[77] $[\texttt{"apply"} \overset{\text{pyk}}{=} \text{“apply identifier”}]$

[78] $[\texttt{"true"} \overset{\text{pyk}}{=} \text{“true identifier”}]$

[79] $[\texttt{"if"} \overset{\text{pyk}}{=} \text{“if identifier”}]$

[80] $[\texttt{"quote"} \overset{\text{pyk}}{=} \text{“quote identifier”}]$

4.3.13 Message proclamations

A construct denotes an aspect if the construct has a "message" aspect.

Proclaiming a construct to denote "pyk", "tex", "texname", "value", "message", "macro", "unpack", "claim", or "priority" affects the message aspect of the construct. For that reason we shall refer to proclamations with one of these strings in the right hand side as *message proclamations*.

The aspects mentioned above are said to be "predefined". User defined aspects are treated later.

In addition to the predefined aspects mentioned above, Logiweb recognizes one more predefined aspect: the "definition" aspect, which is unproclaimable and which is treated in Section 4.3.14.

As an example of use,

$$[\mathbf{aspect}(\texttt{"message"}, \lceil \text{pyk} \rceil, \text{ijcar base}) \approx (0 :: \texttt{"pyk"} :: \mathsf{T})^{\mathrm{I}} :: \mathsf{T}]^{\cdot}$$

4.3.14 Definition proclamation

A construct denotes a revelation if the construct has a "definition" aspect.

Proclaiming a construct to denote "proclaim", "define", "introduce", or "hide" affects the revelation aspect of the construct. For that reason we shall refer to proclamations with one of these strings in the right hand side as *definition proclamations*. The Logiweb identifiers associated to the definition proclamations are defined thus:

- $[\texttt{"proclaim"} \overset{\text{val}}{\rightarrow} \text{identifier}(\lceil \text{“proclaim”} \rceil)]$[81]
- $[\texttt{"define"} \overset{\text{val}}{\rightarrow} \text{identifier}(\lceil \text{“define”} \rceil)]$[82]
- $[\texttt{"introduce"} \overset{\text{val}}{\rightarrow} \text{identifier}(\lceil \text{“introduce”} \rceil)]$[83]
- $[\texttt{"hide"} \overset{\text{val}}{\rightarrow} \text{identifier}(\lceil \text{“hide”} \rceil)]$[84]

Constructs proclaimed to "hide" disable all proclamations in their subtrees to have effect as described in Section 4.5.3.

As an example of use of the definitions aspect we have

$$[\mathbf{aspect}(\texttt{"definition"}, \lceil [\mathsf{x}^{\mathrm{H}} \overset{\text{val}}{\rightarrow} \mathsf{x} \text{ ' } \mathsf{T}] \rceil, \text{ijcar base}) \approx (0 :: \texttt{"define"} :: \mathsf{T})^{\mathrm{I}} :: \mathsf{T}]^{\cdot}$$

4.3.15 Priority proclamations

Proclaiming a construct to denote "pre" or "post" affects the priority aspect of the construct. For that reason we shall refer to proclamations with one of these

[81] $[\texttt{"proclaim"} \overset{\text{pyk}}{=} \text{“proclaim identifier”}]$

[82] $[\texttt{"define"} \overset{\text{pyk}}{=} \text{“define identifier”}]$

[83] $[\texttt{"introduce"} \overset{\text{pyk}}{=} \text{“introduce identifier”}]$

[84] $[\texttt{"hide"} \overset{\text{pyk}}{=} \text{“hide identifier”}]$

strings in the right hand side as *priority proclamations*. The Logiweb identifiers associated to the priority proclamations are defined thus:

- $[\texttt{"pre"} \xrightarrow{\text{val}} \text{identifier}(\lceil \text{“pre”} \rceil)]$[85]
- $[\texttt{"post"} \xrightarrow{\text{val}} \text{identifier}(\lceil \text{“post”} \rceil)]$[86]

As an example,

$$[\textbf{aspect}(\texttt{"priority"}, \lceil \textbf{Preassociative} \mathsf{x}; \mathsf{y} \rceil, \text{ijcar base}) \approx (0 :: \texttt{"pre"} :: \mathsf{T})^{\mathrm{I}} :: \mathsf{T}]^{\cdot}$$

4.3.16 The code of a page

The code of a page is a one-dimensional array that hangs on the ["code"] hook of the rack of the page.

If [c] is the cache of a page and [r] belongs to the transitive bibliography of the page, then

$$[\mathsf{c}[\mathsf{r}][\texttt{"code"}][\mathsf{i}]]$$

is a tagged, Curried version of

$$[\mathsf{c}[\mathsf{r}][\texttt{"codex"}][\mathsf{r}][\mathsf{i}][0][\texttt{"value"}]]$$

As an example, if [r] and [i] are the reference and identifier, respectively, of the $[\mathsf{x} :: \mathsf{y}]$ construct, then

$$[\mathsf{c}[\mathsf{r}][\texttt{"code"}][\mathsf{i}] = \mathcal{M}(\lambda\mathsf{x}.\lambda\mathsf{y}.\mathsf{x} :: \mathsf{y})]$$

The first edition engine performs compilation backstage so that the code of a page contains a compiled versions of the value aspects of the codex.

If [r] and [i] are the reference and identifier, respectively, of a symbol that is proclaimed to denote lambda abstraction, then $[\mathsf{c}[\mathsf{r}][\texttt{"code"}][\mathsf{i}] = 0]$. If [r] and [i] are the reference and identifier, respectively, of a symbol that is proclaimed to denote quoting, then $[\mathsf{c}[\mathsf{r}][\texttt{"code"}][\mathsf{i}] = 1]$.

4.4 Macro expansion

4.4.1 A self interpreter

Before we can define a macro expander we need a self interpreter. We define a self interpreter in the following.

The result of evaluating a term is a map, but the evaluator returns a semitagged map for efficiency reasons. To get the map itself one has to untag the return value.

[85] $[\texttt{"pre"} \overset{\text{pyk}}{=} \text{“pre identifier”}]$

[86] $[\texttt{"post"} \overset{\text{pyk}}{=} \text{“post identifier”}]$

The self interpreter $[\mathcal{E}(\mathsf{t},\mathsf{s},\mathsf{c})]$ evaluates the term $[\mathsf{t}]$ for the stack $[\mathsf{s}]$ and the cache $[\mathsf{c}]$ and returns the result as a semitagged map. $[\mathcal{U}(\mathcal{E}(\mathsf{t},\mathsf{s},\mathsf{c}))]$ is the map itself.

The self interpreter computes the reference $[\mathsf{t}^{\mathrm{r}}]$ and identifier $[\mathsf{t}^{\mathrm{i}}]$ of the root of the term and passes control to an auxiliary function:

$$[\mathcal{E}(\mathsf{t},\mathsf{s},\mathsf{c}) \overset{\mathrm{val}}{\to} \mathcal{E}_2(\mathsf{t},\mathsf{t}^{\mathrm{r}},\mathsf{t}^{\mathrm{i}},\mathsf{s},\mathsf{c})]$$

$[\mathcal{E}_2(\mathsf{t},\mathsf{r},\mathsf{i},\mathsf{s},\mathsf{c})]$ evaluates the term $[\mathsf{t}]$ whose root symbol has reference $[\mathsf{r}]$ and identifier $[\mathsf{i}]$ for the stack $[\mathsf{s}]$ and the cache $[\mathsf{c}]$. If the root symbol is a page symbol, it returns the associated cache. Otherwise, it looks up the code of the root symbol:

$$[\mathcal{E}_2(\mathsf{t},\mathsf{r},\mathsf{i},\mathsf{s},\mathsf{c}) \overset{\mathrm{val}}{\to} \mathsf{i}^{\mathrm{s}} \left\{ \begin{array}{l} \mathsf{t!s!If}(\mathsf{r} \approx \mathsf{c}[0],\mathsf{c},\mathsf{c}[\mathsf{c}[0]][\texttt{"cache"}][\mathsf{r}]) \\ \mathcal{E}_3(\mathsf{t},\mathsf{c}[\mathsf{r}][\texttt{"code"}][\mathsf{i}],\mathsf{s},\mathsf{c}) \end{array} \right.]$$

$[\mathcal{E}_3(\mathsf{t},\mathsf{f},\mathsf{s},\mathsf{c})]$ evaluates the term $[\mathsf{t}]$ whose root symbol has code $[\mathsf{f}]$ for the stack $[\mathsf{s}]$ and the cache $[\mathsf{c}]$. If the code equals $[\mathsf{T}]$ then the root symbol has no value definition and, hence, the term $[\mathsf{t}]$ is treated a variable. If the code equals zero then the root symbol denotes lambda abstraction and is treated accordingly. If the code equals one then the root symbol denotes the quote construct and is treated accordingly. Otherwise, the code is a tagged, Curried function to be applied to the argument list $[\mathsf{t}^{\mathrm{t}}]$ of the term $[\mathsf{t}]$:

$$[\mathcal{E}_3(\mathsf{t},\mathsf{f},\mathsf{s},\mathsf{c}) \overset{\mathrm{val}}{\to} \mathsf{f}^{\mathrm{c}} \left\{ \begin{array}{ll} \mathsf{f}^{\mathrm{s}} \left\{ \begin{array}{l} \mathbf{abstract}(\mathsf{t}^1,\mathsf{t}^2,\mathsf{s},\mathsf{c}) \\ \mathsf{c!s!t}^1 \end{array} \right. & \\ \mathsf{f} \left\{ \begin{array}{l} \mathsf{c!}\mathbf{lookup}(\mathsf{t},\mathsf{s},\mathsf{T}) \\ \mathcal{E}_4(\mathsf{f},\mathsf{t}^{\mathrm{t}},\mathsf{s},\mathsf{c}) \end{array} \right. & \end{array} \right.]$$

$[\mathcal{E}_4(\mathsf{f},\mathsf{a},\mathsf{s},\mathsf{c})]$ applies the function $[\mathsf{f}]$ to the arguments $[\mathsf{a}]$ evaluated for the stack $[\mathsf{s}]$ and the cache $[\mathsf{c}]$:

$$[\mathcal{E}_4(\mathsf{f},\mathsf{a},\mathsf{s},\mathsf{c}) \overset{\mathrm{val}}{\to} \mathrm{If}(\mathsf{a},\mathsf{s!c!f},\mathcal{E}_4(\mathbf{apply}(\mathsf{f},\mathcal{E}(\mathsf{a}^{\mathrm{h}},\mathsf{s},\mathsf{c})),\mathsf{a}^{\mathrm{t}},\mathsf{s},\mathsf{c}))]$$

$[\mathbf{abstract}(\mathsf{v},\mathsf{t},\mathsf{s},\mathsf{c})]$ abstracts the variable $[\mathsf{v}]$ in the term $[\mathsf{t}]$ for the stack $[\mathsf{s}]$ and the cache $[\mathsf{c}]$:

$$[\mathbf{abstract}(\mathsf{v},\mathsf{t},\mathsf{s},\mathsf{c}) \overset{\mathrm{val}}{\to} \mathsf{v!t!s!c!}\Lambda\lambda\mathsf{x}.\mathcal{E}(\mathsf{t},(\mathsf{v} :: \mathsf{x})^{\mathrm{M}} :: \mathsf{s},\mathsf{c})]$$

4.4.2 A macro expander

Macro definitions are based on definitions of the *macro aspect* of symbols. Definitions of macro aspects will be referred to as *macro definitions*. Other kinds of definitions treated so far include value, pyk, TEX, and priority definitions. To make macro definitions we must proclaim [macro] to denote the macro aspect:

$$[\mathrm{macro} \bowtie \text{“macro”}]$$

The macro expander $[\mathcal{M}(\mathsf{t},\mathsf{s},\mathsf{c})]$ macro expands the term $[\mathsf{t}]$ for the *macro state* $[\mathsf{s}]$ and the cache $[\mathsf{c}]$ and returns the result as a semitagged map. The untagged version $[\mathcal{U}(\mathcal{M}(\mathsf{t},\mathsf{s},\mathsf{c}))]$ is the expansion itself.

As we shall see, macros typically use the macro state to define what should happen after one macro expansion. The initial macro state defined in Section 4.4.3 just specifies that the macro expander should be reinvoked which ensures that macro expansion proceeds until all macros are expanded. But macros may decide not to reinvoke the macro expander. Or they may decide to reinvoke it but with a modified macro state. Or they may decide to invoke an entirely different macro expander.

If the term to be expanded is a page construct, the macro expander returns the term unchanged (page constructs should not be macro expanded for reasons explained later). Otherwise, the macro expander looks up the macro definition of the root of the term and passes control to an auxiliary function:

$$[\mathcal{M}(\mathsf{t},\mathsf{s},\mathsf{c}) \overset{\text{val}}{\to} \mathsf{s!c!If}(\mathsf{t}^{\text{is}},\mathsf{t},\mathcal{M}_2(\mathsf{t},\textbf{aspect}(\texttt{"macro"},\mathsf{t},\mathsf{c}),\mathsf{s},\mathsf{c}))]$$

$[\mathcal{M}_2(\mathsf{t},\mathsf{d},\mathsf{s},\mathsf{c})]$ macro expands the term $[\mathsf{t}]$ whose root symbol has macro definition $[\mathsf{d}]$ for the macro state $[\mathsf{s}]$ and the cache $[\mathsf{c}]$. If the macro definition equals $[\mathsf{T}]$ then the root symbol has no macro definition. In this case, the subterms of the term are macro expanded. Otherwise, $[\mathcal{M}_2(\mathsf{t},\mathsf{d},\mathsf{s},\mathsf{c})]$ evaluates the right hand side $[\mathsf{d}^3]$ of the definition (for completeness: $[\mathsf{d}^0]$ is a symbol that denotes the "definition" concept, $[\mathsf{d}^1]$ is a term that represents the macro aspect, $[\mathsf{d}^2]$ is the left hand side and $[\mathsf{d}^3]$ is the right hand side of the definition). Finally, the evaluated right hand side is applied to the term $[\mathsf{t}]$, the macro state $[\mathsf{s}]$, and the cache $[\mathsf{c}]$ so that the user defined macro $[\mathsf{d}]$ gets access to term, state, and cache (at expansion time, the cache will be the cache of the page on which the term being macro expanded resides).

$$[\mathcal{M}_2(\mathsf{t},\mathsf{d},\mathsf{s},\mathsf{c}) \overset{\text{val}}{\to} \mathsf{d}\left\{\begin{array}{l}\mathsf{t}^{\text{h}} :: \mathcal{M}^*(\mathsf{t}^{\text{t}},\mathsf{s},\mathsf{c})\\ \mathcal{U}^{\text{M}}(\mathcal{E}(\mathsf{d}^3,\mathsf{T},\mathsf{c})\text{`}\,\mathsf{t}\,\text{`}\,\mathsf{s}\,\text{`}\,\mathsf{c})\end{array}\right.]$$

The $[\mathcal{M}^*(\mathsf{a},\mathsf{s},\mathsf{c})]$ construct macro expands the elements of the list $[\mathsf{a}]$.

$$[\mathcal{M}^*(\mathsf{a},\mathsf{s},\mathsf{c}) \overset{\text{val}}{\to} \mathsf{s!c!If}(\mathsf{a},\mathsf{T},\mathcal{M}(\mathsf{a}^{\text{h}},\mathsf{s},\mathsf{c}) :: \mathcal{M}^*(\mathsf{a}^{\text{t}},\mathsf{s},\mathsf{c}))]$$

4.4.3 The initial macro state

The *initial macro state* $[\mathsf{s}_0]$ is useful for passing as the second parameter to $[\mathcal{M}(\mathsf{t},\mathsf{s},\mathsf{c})]$. $[\mathsf{s}_0]$ is a pair whose head is the macro expander itself and whose tail is left blank. The definition of $[\mathsf{s}_0]$ reads:

$$[\mathsf{s}_0 \overset{\text{val}}{\to} \mathcal{M}(\lambda\mathsf{t}.\lambda\mathsf{s}.\lambda\mathsf{c}.\mathcal{M}(\mathsf{t},\mathsf{s},\mathsf{c})) :: \mathsf{T}]$$

In the definition of $[\mathsf{s}_0]$ note that the first $\mathcal{M}$ is a unary "map tag" operation that converts the lambda abstraction into a tagged map whereas the second $\mathcal{M}$ is the ternary macro expansion operation.

Advanced users may write macros that pass down a macro state different from $[\mathsf{s}_0]$. Such users should pass down a macro state of form $[\mathsf{m} :: \mathsf{p}]$ where $[\mathsf{m}]$ is a ternary macro expander and $[\mathsf{p}]$ is a parameter that can contain arbitrary data. The default macro expander $[\mathcal{M}(\mathsf{t}, \mathsf{s}, \mathsf{c})]$ makes no use of the parameter and for that reason the parameter is set to $[\mathsf{T}]$ in $[\mathsf{s}_0]$.

4.4.4 Pruning

In connection with macro expansion, Logiweb uses a hardwired retract on Gödel trees which we shall refer to as *pruning*.

Given the cache of a page and an arbitrary data structure, the pruning operation scans the data structure as if it were a Gödel tree. Whenever the pruning operation encounters a branch that cannot be a Gödel tree, it replaces the branch by the page construct of the given page.

The pruning operation first applies the retract for tagged trees to the given data structure. Then, for each branch of the Gödel tree, it checks that the reference and identifier are cardinals. Then it looks up the arity of the associated symbol in the proper dictionary inside the cache of the given page. If no arity is found, the symbol is invalid and the branch is pruned. Otherwise, the pruning operation checks that the number of subtrees agree with the arity and descend recursively into the subtrees.

If no arity is found for a given symbol it either means that the symbol does not exist on Logiweb or that it exists, but not on a page transitively referenced by the given page.

4.4.5 Potentially inherited page aspects

Macro expansion of a page is governed by the "potentially inherited macro aspect" of the page. The potentially inherited macro aspect is an example of a *potentially inherited page aspect*. Let us take "potentially inherited page aspect" one word at a time.

Aspects of the page construct of a page are "page aspects" for that page.

An "inherited page aspect" of a page $[\mathsf{p}]$ is a page aspect of the *page bed* $[\mathsf{b}]$ of page $[\mathsf{p}]$. The bed $[\mathsf{b}]$ of a page $[\mathsf{p}]$ is the first page referenced in the bibliography of page $[\mathsf{p}]$. If page $[\mathsf{p}]$ is a base page, i.e. if page $[\mathsf{p}]$ references no other pages, then the bed of page $[\mathsf{p}]$ is page $[\mathsf{p}]$ itself. The bed $[\mathsf{b}]$ of a page $[\mathsf{p}]$ is the page on which page $[\mathsf{p}]$ rests.

Note for programmers: The first entry of the bibliography of a page is the one that comes after the zeroth entry. The zeroth entry references the page itself. A bibliography is a list of references, and the head of the list is the zeroth entry.

But back to "potentially inherited page aspects". A potentially inherited page aspect of a page $[\mathsf{p}]$ is taken from the page construct of $[\mathsf{p}]$ if that aspect is available, and is taken from the page construct of the bed of $[\mathsf{p}]$ otherwise. As an example, the potentially inherited macro aspect of page $[\mathsf{p}]$ is the macro

aspect of the page construct of page [p] if such an aspect is defined, and else the macro aspect of the page construct of the bed of [p] if such an aspect is defined.

At the time of writing, Logiweb makes use of two potentially inherited page aspects: the macro aspect described in the following, and the claim aspect described in Section 5. At a later stage, Logiweb will also make use of a potentially inherited value aspect which defines things to be computed once and for all when pages are loaded and an unconditionally inherited unpack aspect which may decompress and/or decrypt Logiweb vectors when loaded.

4.4.6 Installing the macro expander

We shall refer to the Gödel tree that constitutes the macro expanded version of a page as the *expansion* of the page. We define the *expander* of a page to be the potentially inherited macro aspect of the page.

Logiweb expands a page by applying the expander to the body and cache of the page. If no expander is defined then Logiweb does no macro expansion and uses the body unchanged as expansion.

Now define

$$[\text{ijcar base} \overset{\text{macro}}{\rightarrow} \lambda\mathsf{t}.\lambda\mathsf{c}.\mathcal{M}(\mathsf{t}, \mathrm{s}_0, \mathsf{c})]$$

Because of the definition above, the expansion of the present page is the pruned version of $[\mathcal{M}(\mathsf{t}, \mathrm{s}_0, \mathsf{c})]$ where [t] and [c] are the body and cache, respectively, of the page.

Another page that uses the present page as bed and defines no expander of its own also has expansion $[\mathcal{M}(\mathsf{t}, \mathrm{s}_0, \mathsf{c})]$ but then [t] and [c] are the body and cache, respectively, of that other page.

Recall that the macro expander $[\mathcal{M}(\mathsf{t}, \mathsf{s}, \mathsf{c})]$ does not macro expand page constructs. One reason for that is that the macro aspect of the page construct defines an expander rather than a macro.

4.4.7 Iterated macro expansion

The description of macro expansion above is part of the story rather than the whole story.

As descripted in Section 2.1.1, Logiweb "loads" a page by "resolving" its reference into a Uniform Resource Locator, then "retrieves" the page as a vector of bytes, then "unpacks" the vector into bibliography, dictionary, and body, then "codifies" the page into codex and expansion, and finally "verifies" the page.

Codification comprises macro expansion and "harvesting". Harvesting is done by scanning the expansion of a page for revelations, i.e. proclamations, definitions, and introductions. The process of harvesting converts the expansion, which is a Gödel tree, into a codex.

Codification proceeds thus: First, with the exception stated in Section 2.1.5, Logiweb sets the codex of the page to be empty. Then Logiweb macro expands the body of the page using the empty codex for the page in question plus all the codicies of referenced pages. Then Logiweb sets the codex of the page to

the result of harvesting the expansion, and then Logiweb macro expands the body of the page once more. Logiweb proceeds this way until the codex does not change anymore (if ever). This is what was meant by "reading the page over and over again" in Section 2.1.1 and 2.1.5.

4.5 Macro definitions

4.5.1 Protection

[(x)$^{\mathsf{p}}$] is a particularly important macro; it protects its argument against macro expansion. During macro expansion, the protection construct itself disappears. The definition of protection is not completely trivial:

$$([(\mathsf{x})^{\mathsf{p}} \overset{\text{macro}}{\rightarrow} \lambda\mathsf{t}.\lambda\mathsf{s}.\lambda\mathsf{c}.\mathsf{t}^{1}])^{\mathsf{p}}$$

Note that the definition has two occurrences of the protection construct: an outermost and an innermost.

At first reading of the page, no expander has been defined yet, the definition construct has not yet been proclaimed, and Logiweb has not yet understood what the protection construct is supposed to do.

After a couple of readings, Logiweb has still not understood what the protection construct is supposed to do, but has reached a level where it can understand that the formula above defines the macro aspect of the protection construct.

At the next reading of the page, the protection construct is operational, and for that reason, the outermost protection construct protects the innermost protection construct from macro expansion. Without the outermost protection construct, the innermost protection construct would disappear so that it was [x] that was macro defined to denote protection, which leads to disaster.

It should be noted that base pages as large as the present one are not particularly easy to write. There are many things that can go wrong in intractable ways. Authors who insist on writing their own base page are encouraged to build them up in small steps, starting with a base page as simple as possible.

4.5.2 Self references

We macro define [self] such that each instance of [self] expands to the page construct of the page on which the instance occurs.

$$([\text{self} \overset{\text{macro}}{\rightarrow} \lambda\mathsf{t}.\lambda\mathsf{s}.\lambda\mathsf{c}.(\mathsf{c}[0] :: 0 :: \mathsf{t}^{\mathrm{d}})^{\mathrm{I}} :: \mathsf{T}])^{\mathsf{p}}$$

Alternatively, one could define [self] to expand into an invalid tree and let pruning insert a page construct. As an example, one could replace the entire right hand side above by [T]. But the definition above ensures that resulting page sysmbol enherits the debugging information present in the [self] construct.

4.5.3 Avoid macro expansion and harvesting of strings

Good Logiweb pages contain loads of text. Actually, the formal mathematics present on well written Logiweb pages only constitute a minor fraction of the page. Since macro expansion takes time, we arrange that strings are not macro expanded.

The pyk compiler puts a unicode start and end of text construct at the root and leaf of strings, respectively, with the individual characters of the string in between as a row of perls (or, rather, a column of perls; in the string itself, the characters are like a row of perls). We can protect strings against macro expansion by a proper definition of the macro aspect of the unicode start of text construct:

$$[\text{“x”} \overset{\text{macro}}{\rightarrow} \lambda\mathsf{t}.\lambda\mathsf{s}.\lambda\mathsf{c}.\mathsf{t}]$$

Once Logiweb has expanded the body of a page, it does *harvesting* of definitions: it scans the expansion and collects all definitions found. To avoid spending time on harvesting inside strings, we proclaim the unicode start of text construct to protect against harvesting:

$$[\text{“x”} \bowtie \text{“hide”}]$$

4.5.4 Macro definitions

$([\mathsf{x} \doteq \mathsf{y}])^{\mathbf{P}}$ defines $[\mathsf{x}]$ as shorthand for $[\mathsf{y}]$. Hence, $([\mathsf{x} \doteq \mathsf{y}])^{\mathbf{P}}$ makes ordinary macro definitions. The definition of $([\mathsf{x} \doteq \mathsf{y}])^{\mathbf{P}}$ is slightly involved since $([\mathsf{x} \doteq \mathsf{y}])^{\mathbf{P}}$ is defined such that it macro expands into a general macro definition.

$$([[\mathsf{x} \doteq \mathsf{y}] \overset{\text{macro}}{\rightarrow} \lambda\mathsf{t}.\lambda\mathsf{s}.\lambda\mathsf{c}.\tilde{\mathcal{M}}_3(\mathsf{t})])^{\mathbf{P}}$$

$[\tilde{\mathcal{M}}(\mathsf{t},\mathsf{s},\mathsf{c})]$ macro expands the term $[\mathsf{t}]$ using the macro expander embedded in the macro state $[\mathsf{s}]$ using the cache $[\mathsf{c}]$.

$$[\tilde{\mathcal{M}}(\mathsf{t},\mathsf{s},\mathsf{c}) \overset{\text{val}}{\rightarrow} \mathcal{U}(\mathsf{s}^{\mathrm{h}} \text{ ‘ } \mathsf{t} \text{ ‘ } \mathsf{s} \text{ ‘ } \mathsf{c})]$$

$[\tilde{\mathcal{M}}_1]$ is a template that defines what $([\mathsf{x} \doteq \mathsf{y}])^{\mathbf{P}}$ expands into:

$$[\tilde{\mathcal{M}}_1 \overset{\text{val}}{\rightarrow} \lceil[\mathsf{x} \overset{\text{macro}}{\rightarrow} \lambda\mathsf{t}.\lambda\mathsf{s}.\lambda\mathsf{c}.\tilde{\mathcal{M}}_4(\mathsf{t},\mathsf{s},\mathsf{c},\lceil\mathsf{d}\rceil)]\rceil]$$

If $[\mathsf{t}]$ is a term of form $([\mathsf{x} \doteq \mathsf{y}])^{\mathbf{P}}$ then $[\tilde{\mathcal{M}}_2(\mathsf{t})]$ is a stack that maps the variable $[\mathsf{x}]$ to the left hand side of the macro definition and maps the variable $[\mathsf{d}]$ to the entire macro definition.

$$[\tilde{\mathcal{M}}_2(\mathsf{t}) \overset{\text{val}}{\rightarrow} (\lceil\mathsf{x}\rceil :: \mathsf{t}^1)^{\mathrm{M}} :: (\lceil\mathsf{d}\rceil :: \mathsf{t})^{\mathrm{M}} :: \mathsf{T}]$$

$[\tilde{\mathcal{M}}_3(\mathsf{t})]$ translates a term $[\mathsf{t}]$ of form $([\mathsf{x} \doteq \mathsf{y}])^{\mathbf{P}}$ into a general macro definition. This is done by replacing $[\mathsf{x}]$ and $[\mathsf{d}]$ in the template $[\tilde{\mathcal{M}}_1]$ by the left hand side of $[\mathsf{t}]$ and all of $[\mathsf{t}]$, respectively.

$$[\tilde{\mathcal{M}}_3(\mathsf{t}) \overset{\text{val}}{\to} \tilde{\mathcal{Q}}(\mathsf{t}, \tilde{\mathcal{M}}_1, \tilde{\mathcal{M}}_2(\mathsf{t}))]$$

$[\tilde{\mathcal{M}}_4(\mathsf{t},\mathsf{s},\mathsf{c},\mathsf{d})]$ performs the macro expansion defined by $([\mathsf{x} \ddot{=} \mathsf{y}])^{\mathbf{p}}$. The function above macro expands $([\mathsf{x} \ddot{=} \mathsf{y}])^{\mathbf{p}}$. The function below expands the macro defined by $([\mathsf{x} \ddot{=} \mathsf{y}])^{\mathbf{p}}$.

$$[\tilde{\mathcal{M}}_4(\mathsf{t},\mathsf{s},\mathsf{c},\mathsf{d}) \overset{\text{val}}{\to} \tilde{\mathcal{M}}(\tilde{\mathcal{Q}}(\mathsf{t},\mathsf{d}^2,\mathbf{zip}(\mathsf{d}^{1\,\mathsf{t}},\mathsf{t}^{\mathsf{t}})),\mathsf{s},\mathsf{c})]$$

$[\tilde{\mathcal{Q}}(\mathsf{r},\mathsf{t},\mathsf{s})]$ performs the substitutions defined by the stack $[\mathsf{s}]$ in the term $[\mathsf{t}]$. Set the debugging information of all nodes taken from $[\mathsf{t}]$ to the debugging information of the root of the term $[\mathsf{r}]$.

$$[\tilde{\mathcal{Q}}(\mathsf{r},\mathsf{t},\mathsf{s}) \overset{\text{val}}{\to} \tilde{\mathcal{Q}}_2(\mathsf{r}^{\mathsf{d}},\mathsf{t},\mathsf{s})]$$

$[\tilde{\mathcal{Q}}_2(\mathsf{r},\mathsf{t},\mathsf{s})]$ performs the substitutions defined by the stack $[\mathsf{s}]$ in the term $[\mathsf{t}]$. Set the debugging information of all nodes taken from $[\mathsf{t}]$ to the debugging information $[\mathsf{r}]$.

$$[\tilde{\mathcal{Q}}_2(\mathsf{r},\mathsf{t},\mathsf{s}) \overset{\text{val}}{\to} \tilde{\mathcal{Q}}_3(\mathsf{r},\mathsf{t},\mathsf{s},\mathbf{lookup}(\mathsf{t},\mathsf{s},\mathsf{T}))]$$

$[\tilde{\mathcal{Q}}_3(\mathsf{r},\mathsf{t},\mathsf{s},\mathsf{v})]$ performs the substitutions defined by the stack $[\mathsf{s}]$ in the term $[\mathsf{t}]$. Set the debugging information of all nodes taken from $[\mathsf{t}]$ to the debugging information $[\mathsf{r}]$. If $[\mathsf{v}]$ differs from $[\mathsf{T}]$ then $[\mathsf{t}]$ is supposed to occur in $[\mathsf{s}]$ and $[\mathsf{v}]$ is supposed to be the associated value. Hence, if $[\mathsf{v}]$ differs from $[\mathsf{T}]$ then $[\mathsf{v}]$ is returned. Otherwise, the root of $[\mathsf{t}]$ is merged properly with the debugging information $[\mathsf{r}]$ and substitutions are performed in all subterms of $[\mathsf{t}]$.

$$[\tilde{\mathcal{Q}}_3(\mathsf{r},\mathsf{t},\mathsf{s},\mathsf{v}) \overset{\text{val}}{\to} \mathsf{v} \begin{cases} (\mathsf{t}^{\mathsf{r}} :: \mathsf{t}^{\mathsf{i}} :: \mathsf{r})^{\mathrm{M}} :: \tilde{\mathcal{Q}}^*(\mathsf{r},\mathsf{t}^{\mathsf{t}},\mathsf{s}) \\ \mathsf{r!t!s!v} \end{cases}]$$

$[\tilde{\mathcal{Q}}^*(\mathsf{r},\mathsf{t},\mathsf{s})]$ performs the substitutions defined by the stack $[\mathsf{s}]$ in the list $[\mathsf{t}]$ of terms. Set the debugging information of all nodes taken from $[\mathsf{t}]$ to the debugging information $[\mathsf{r}]$.

$$[\tilde{\mathcal{Q}}^*(\mathsf{r},\mathsf{t},\mathsf{s}) \overset{\text{val}}{\to} \mathsf{t} \begin{cases} \mathsf{r!t!s!T} \\ \tilde{\mathcal{Q}}_2(\mathsf{r},\mathsf{t}^{\mathsf{h}},\mathsf{s}) :: \tilde{\mathcal{Q}}^*(\mathsf{r},\mathsf{t}^{\mathsf{t}},\mathsf{s}) \end{cases}]$$

4.5.5 Parentheses

$[(\mathsf{x})]$ is probably the most important among all macros, at least from the point of view of the working mathematician. Parentheses simply disappear when macro expanded:

$$[(\mathsf{x}) \ddot{=} \mathsf{x}]$$

The following construct uses `\left(...\right)` to render parentheses. Beware that the construct disables line breaking.

$$[(\mathsf{x}) \ddot{=} \mathsf{x}]$$[87]

[87] $[(\mathsf{x}) \overset{\text{pyk}}{=}$ "big parenthesis $*$ end parenthesis"$]$

4.6 Tagged lambda

The following macro definition allows to state tagged lambdas in a readable way:

$$[\Lambda x.y \doteqdot \Lambda\lambda x.y]^{88}$$

4.6.1 Local abbreviations

The following construct allows to override macros locally:

$$([\mathbf{let}\ x \doteqdot y\ \ \mathrm{in}\ \ z \overset{\mathrm{macro}}{\rightarrow} \lambda t.\lambda s.\lambda c.\\ \tilde{\mathcal{M}}(t^3, s, c[t^{1r} :: \text{"codex"} :: t^{1r} :: t^{1i} :: 0 :: \text{"macro"} :: \mathsf{T}\Rightarrow\tilde{\mathcal{M}}_3(t)])])^{\mathbf{p}}{}^{89}$$

As a somewhat constructed example of use,

$$[\mathbf{let}\ (x) \doteqdot x :: x\ \ \mathrm{in}\ \ (3)]$$

macro expands to

$$[3 :: 3]$$

4.6.2 Other kinds of definitions

The following constructs allow to define particular aspects of a left hand side to be the given right hand side.

The following construct is useful for value definitions.

$$[[x \doteq y] \doteqdot [(x)^{\mathbf{p}} \overset{\mathrm{val}}{\rightarrow} y]]$$

Use of the construct above for value definitions on which the definition of the macro expander depends may lead to intractable errors. For that reason, the present page is organised such that the construct above is only used from Section 5 and on and such that the macro expander only depends on definitions stated before 5. Logiweb does not care about the order of definitions. The avoidance of vicious circles by dividing at the start of Section 5 is just for the sake of the author.

The following construct is suitable for value introductions, i.e. for value definitions of constructs for which the Logiweb system in use is expected to contain a hardcoded version. As with the value definition, it is not used before Section 5.

$$[[x \doteq y] \doteqdot [(x)^{\mathbf{p}} \overset{\mathrm{val}}{\Rightarrow} y]]$$

---

[88] $[\Lambda x.y \overset{\mathrm{pyk}}{=}$ "tagged lambda $*$ dot $*$"]

[89] $[\mathbf{let}\ x \doteqdot y\ \ \mathrm{in}\ \ z \overset{\mathrm{pyk}}{=}$ "let $*$ abbreviate $*$ in $*$"]

The following constructs are convenient for defining [pyk], [tex], and [name] definitions. The constructs protect the left hand side against macro expansion and ensures that the right hand side is typeset as a string. The three constructs below are used uncritically on the present page since rendering is done after codification and, hence, cannot interfere with codification.

$$[[\mathsf{x} \overset{\text{pyk}}{=} \mathsf{y}] \doteqdot [(\mathsf{x})^{\mathbf{p}} \overset{\text{pyk}}{\to} \mathsf{y}]]$$

$$[[\mathsf{x} \overset{\text{tex}}{=} \mathsf{y}] \doteqdot [(\mathsf{x})^{\mathbf{p}} \overset{\text{tex}}{\to} \mathsf{y}]]$$

$$[[\mathsf{x} \overset{\text{name}}{=} \mathsf{y}] \doteqdot [(\mathsf{x})^{\mathbf{p}} \overset{\text{name}}{\to} \mathsf{y}]]$$

Finally, the following construct is convenient for defining priority tables.

$$[\mathbf{Priority\ table}[\mathsf{x}] \doteqdot [\mathsf{self} \overset{\text{prio}}{\to} (\mathsf{x})^{\mathbf{p}}]]$$[90]

4.6.3 Tuples

We shall use $[\langle \mathsf{x} \rangle]$[91] and $[\mathsf{x}, \mathsf{y}]$[92] to express tuples. $[\langle \mathsf{x} \rangle]$ will be macro defined such that e.g. $[\langle \mathsf{x}, \mathsf{y}, \mathsf{z} \rangle]$ macro expands into $[\mathsf{x} :: \mathsf{y} :: \mathsf{z} :: \mathsf{T}]$.

$$([\langle \mathsf{x} \rangle \overset{\text{macro}}{\to} \lambda \mathsf{t}.\lambda \mathsf{s}.\lambda \mathsf{c}.\tilde{\mathcal{M}}(\mathbf{tuple}_1(\mathsf{t}), \mathsf{s}, \mathsf{c})])^{\mathbf{p}}$$

$[\mathbf{tuple}_1(\mathsf{t})]$[93] expands the root of $[\langle \mathsf{x} \rangle]$:

$$[\mathbf{tuple}_1(\mathsf{t}) \overset{\text{val}}{\to} \mathsf{t}^1 \overset{\text{r}}{=} \lceil \mathsf{x}, \mathsf{y} \rceil \begin{cases} \tilde{\mathcal{Q}}(\mathsf{t}, \lceil \mathsf{x} :: (\langle \mathsf{y} \rangle)^{\mathbf{p}} \rceil, \mathbf{tuple}_2(\mathsf{t}^1)) \\ \tilde{\mathcal{Q}}(\mathsf{t}, \lceil \mathsf{x} :: \mathsf{T} \rceil, (\lceil \mathsf{x} \rceil :: \mathsf{t}^1) :: \mathsf{T}) \end{cases}]$$

$[\mathbf{tuple}_2(\mathsf{t})]$[94] forms the stack used when expanding $[\langle \mathsf{x}, \mathsf{y} \rangle]$:

$$[\mathbf{tuple}_2(\mathsf{t}) \overset{\text{val}}{\to} (\lceil \mathsf{x} \rceil :: \mathsf{t}^1) :: (\lceil \mathsf{y} \rceil :: \mathsf{t}^2) :: \mathsf{T}]$$

4.6.4 Typography of the expansion

The author of a Logiweb page should ensure that the tex aspect of the body looks good when typeset by TEX. Among other, the author should avoid overfull and underfull TEX boxes in the body.

When it comes to the expansion of the page, matters are a bit different. Firstly, the aesthetics of the expansion is less important since the expansion does not address the audience to the same degree as the body does. Secondly, precise control of the typography of the expansion is more intricate.

---

[90] $[\mathbf{Priority\ table}[\mathsf{x}] \overset{\text{pyk}}{=}$ "priority table ∗ end table"]

[91] $[\langle \mathsf{x} \rangle \overset{\text{pyk}}{=}$ "tuple ∗ end tuple"]

[92] $[\mathsf{x}, \mathsf{y} \overset{\text{pyk}}{=}$ "∗ comma ∗"]

[93] $[\mathbf{tuple}_1(\mathsf{t}) \overset{\text{pyk}}{=}$ "tuple one ∗ end tuple"]

[94] $[\mathbf{tuple}_2(\mathsf{t}) \overset{\text{pyk}}{=}$ "tuple two ∗ end tuple"]

For these reasons it is convenient to typeset the expansion in a *ragged right* style. To do so we first introduce a contruct [ragged right][95] which produces a `\raggedright` command when typeset using its tex aspect.

Second, we introduce a construct which is invisible when typeset using its tex aspect but which macro expands into the construct above:

[ragged right expansion $\ddot{=}$ ragged right][96]

Including the latter construct above near the start of a Logiweb page, the expansion of the page will be typeset ragged right without affecting the supposedly splendid typography the body.

4.7 Programming aids

4.7.1 Newlines in definitions

Traditional mathematical formulas typically fit on a single line. Formally, a computer program is nothing but a mathematical formula, but computer programs have a tendency to span more than one line. This is so even if one breaks up a computer program in individual definitions of computable functions and consider each definition at a time.

For the sake of readablity, it is often convenient to control where line breaks occur in definitions of computable functions. To allow such control we introduce a newline construct whose tex name aspect is "newline" but whose tex aspect simply forces the formula following it to begin on a new line.

The newline construct is prefix and should be put in front of the formula to begin a new line.

[newline $\mathsf{x} \doteq \mathsf{x}^{\mathrm{M}}$][97]

The newline construct is value defined such that it retracts which allows it to occur in functions that are fit for optimization.

The newline construct can be value or macro defined. Above, it has been value defined so that it also affects the typography of the expansion of the page. That is convenient for debugging. In formal reasoning about a computable function it is more convenient to have a newline construct that macro expands to nothingness:

[macro newline $\mathsf{x} \ddot{=} \mathsf{x}$][98]

[95] [ragged right $\stackrel{\text{pyk}}{=}$ "ragged right"]

[96] [ragged right expansion $\stackrel{\text{pyk}}{=}$ "ragged right expansion"]

[97] [newline $\mathsf{x} \stackrel{\text{pyk}}{=}$ "newline $*$"]

[98] [macro newline $\mathsf{x} \stackrel{\text{pyk}}{=}$ "macro newline $*$"]

4.7.2 The visibility operator

We now introduce a *visibility* operator $[(\mathsf{x})^{\mathbf{v}} \doteq \mathsf{x}]$[99] which forces its argument to be typeset using the tex name aspect and which macro expands to nothingness like parentheses do. The construct is useful in situations where e.g. the right hand side of a definition contains operators whose tex aspect make them invisible.

When typeset in the tex aspect, the visibility operator does not exhibit itself, it just affects how its argument is typeset. When typeset in the tex name aspect, the visibility operator reads $[(\mathsf{x})^{\mathbf{v}}]$.

The bracket $[(\mathsf{x})^{\mathbf{v}}]$ contains two visibility operators. The outermost operator makes the innermost visible.

The visibility operator $[(\mathsf{x})^{\mathbf{v}}]$ is almost identical to the text operator $[(\mathsf{x})^{\mathbf{t}}]$. The difference is that the visibility operator macro expands into nothingness whereas the text operator is value defined such that it is self-evaluating.

The bracket $[(\mathsf{x})^{\mathbf{t}}]$ contains a visibility and a text operator. The visibility operator makes the text operator visible.

4.7.3 Self-evaluation

The juxtaposition operator $[\mathsf{x}\ \mathsf{then}\ \mathsf{y}]$ has a tex name aspect that reads $[\mathsf{x}\ \mathsf{then}\ \mathsf{y}]$ and a tex aspect that reads $[\mathsf{xy}]$. The tex aspect of the juxtaposition operator allows to put two typographic entities back-to-back. In the following we silently use the $[(\mathsf{x})^{\mathbf{v}}]$ operator to make juxtaposition visible.

We make juxtaposition *self-evaluating*:

$$[\mathsf{x}\ \mathsf{then}\ \mathsf{y} \doteq \langle \lceil * \ \mathsf{then}\ * \rceil^{\mathrm{R}}, \mathsf{x}, \mathsf{y} \rangle]$$

The definition above has the effect that e.g. $[\lceil \mathsf{x}\ \mathsf{then}\ \mathsf{y} \rceil]$ denotes the same term as $[\lceil \mathsf{x} \rceil\ \mathsf{then}\ \lceil \mathsf{y} \rceil]$ except for debugging information present in the terms. The root of the latter is "clean" in the sense that its debugging information equals $[\mathsf{T}]$.

We also make the "text" construct self-evaluating:

$$[(\mathsf{x})^{\mathbf{t}} \doteq \langle \lceil (*)^{\mathbf{t}} \rceil^{\mathrm{R}}, \mathsf{x} \rangle]$$

4.7.4 Open if

The *open if* construct is defined thus:

$$[\mathbf{if}\ \mathsf{x}\ \mathbf{then}\ \mathsf{y}\ \mathbf{else}\ \mathsf{z} \doteq \mathrm{If}(\mathsf{x}, \mathsf{y}, \mathsf{z})]$$[100]

The open if construct is useful when writing programs with long chains of if-constructs.

---

[99] $[(\mathsf{x})^{\mathbf{v}} \stackrel{\text{pyk}}{=}$ "make visible $*$ end visible"]

[100] $[\mathbf{if}\ \mathsf{x}\ \mathbf{then}\ \mathsf{y}\ \mathbf{else}\ \mathsf{z} \stackrel{\text{pyk}}{=}$ "open if $*$ then $*$ else $*$"]

4.7.5 The "let" construct

We define eager, retracting functional application $[\mathrm{let}_1(\mathsf{f},\mathsf{y})]$ and the *let* construct $[\mathbf{let}\ \mathsf{x}=\mathsf{y}\ \mathbf{in}\ \mathsf{z}]$ as follows:

$$[\mathbf{let}\ \mathsf{x}=\mathsf{y}\ \mathbf{in}\ \mathsf{z} \mathrel{\ddot{=}} \mathrm{let}_1(\lambda\mathsf{x}.\mathsf{z},\mathsf{y})]^{101}$$

$$[\mathrm{let}_1(\mathsf{f},\mathsf{y}) \mathrel{\dot{=}} \mathrm{let}_2(\mathsf{f},\mathsf{y}^{\mathrm{M}})^{\mathrm{M}}]^{102}$$

$$[\mathrm{let}_2(\mathsf{f},\mathsf{y}) \mathrel{\dot{=}} (\mathsf{y}!\mathsf{f}\ '\ \mathsf{y})^{\mathrm{I}}]^{103}$$

The initial Logiweb engine translates all terms of form $[\mathrm{let}_1(\lambda\mathsf{x}.\mathsf{z},\mathsf{y})]$ to an efficient let-construct internally.

4.7.6 Macro defined connectives

The connectives $[\mathsf{x}\ \ddot{\wedge}\ \mathsf{y}]$, $[\mathsf{x}\ \ddot{\vee}\ \mathsf{y}]$, and $[\mathsf{x} \mathrel{\ddot{\Rightarrow}} \mathsf{y}]$ macro expand into constructs that only compute $[\mathsf{y}]$ when needed. Furthermore, the constructs return the value of $[\mathsf{y}]$ whenever $[\mathsf{y}]$ is computed.

Such a left-to-right computing behaviour can also be achieved using lazy constructs, but the macro defined connectives above have the benefit that, when used with caution, they can occur in definitions that are fit for optimization.

$$[\mathsf{x}\ \ddot{\wedge}\ \mathsf{y} \mathrel{\ddot{=}} \mathrm{If}(\mathsf{x},\mathsf{y},\mathsf{F})]^{104}$$

$$[\mathsf{x}\ \ddot{\vee}\ \mathsf{y} \mathrel{\ddot{=}} \mathrm{If}(\mathsf{x},\mathsf{T},\mathsf{y})]^{105}$$

$$[\mathsf{x} \mathrel{\ddot{\Rightarrow}} \mathsf{y} \mathrel{\ddot{=}} \mathrm{If}(\mathsf{x},\mathsf{y},\mathsf{T})]^{106}$$

4.7.7 Display construct

$[\mathrm{display}(\mathsf{x})]^{107}$ typesets its argument as a paragraph whose left margin equals the indentation of a displayed equation. Space is added before and after the paragraph and the first paragraph after a display is unindented.

$[\mathrm{statement}(\mathsf{x})]^{108}$ does the same except that the left margin flushes the left margin of the document.

---

[101] $[\mathbf{let}\ \mathsf{x}=\mathsf{y}\ \mathbf{in}\ \mathsf{z} \overset{\mathrm{pyk}}{=}$ "let $*$ be $*$ in $*$"]

[102] $[\mathrm{let}_1(\mathsf{f},\mathsf{y}) \overset{\mathrm{pyk}}{=}$ "let one $*$ apply $*$ end let"]

[103] $[\mathrm{let}_2(\mathsf{f},\mathsf{y}) \overset{\mathrm{pyk}}{=}$ "let two $*$ apply $*$ end let"]

[104] $[\mathsf{x}\ \ddot{\wedge}\ \mathsf{y} \overset{\mathrm{pyk}}{=}$ "$*$ macro and $*$"]

[105] $[\mathsf{x}\ \ddot{\vee}\ \mathsf{y} \overset{\mathrm{pyk}}{=}$ "$*$ macro or $*$"]

[106] $[\mathsf{x} \mathrel{\ddot{\Rightarrow}} \mathsf{y} \overset{\mathrm{pyk}}{=}$ "$*$ macro imply $*$"]

[107] $[\mathrm{display}(\mathsf{x}) \overset{\mathrm{pyk}}{=}$ "display $*$ end display"]

[108] $[\mathrm{statement}(\mathsf{x}) \overset{\mathrm{pyk}}{=}$ "statement $*$ end statement"]

4.7.8 Introduction of new constructs

The following constructs allow to introduce a new construct ("introduce" in the normal sense of the word, not in the sense of a Logiweb revelation). The constructs allow to define the pyk aspect [p] and tex aspect [t] of a new construct [x] and also makes two entries in the index. The first of the constructs below adds the text [i] in front of one of the index entries.

[intro(x, i, p, t) ≔ $[x $\overset{\text{pyk}}{=}$ p]$ $[x $\overset{\text{tex}}{=}$ t]$][109]

[intro(x, p, t) ≔ $[x $\overset{\text{pyk}}{=}$ p]$ $[x $\overset{\text{tex}}{=}$ t]$][110]

4.7.9 Further intro constructs

The following four constructs all help introducing new constructs. Each of them expand into pyk and tex and, optionally, tex name definitions. But they also render the pyk definition as a footnote, defer the tex and tex name definitions to an appendix, and make two entries in the index. One of the entries has form "pyk: (pyk name) (tex rendering)". The other entry has form "(index): (tex rendering) (pyk name)". If no index is given, then the latter entry has form "(tex rendering) (pyk name)" and is placed in front of other entries in the index. The four constructs are defined thus:

[x/intro(y, p, t) ≔ x $[y $\overset{\text{pyk}}{=}$ p]$ $[y $\overset{\text{tex}}{=}$ t]$][111]

[x/indexintro(y, i, p, t) ≔ x $[y $\overset{\text{pyk}}{=}$ p]$ $[y $\overset{\text{tex}}{=}$ t]$][112]

[x/nameintro(y, p, t, n) ≔ x $[y $\overset{\text{pyk}}{=}$ p]$ $[y $\overset{\text{tex}}{=}$ t]$ $[y $\overset{\text{name}}{=}$ n]$][113]

[x/bothintro(y, i, p, t, n) ≔ x $[y $\overset{\text{pyk}}{=}$ p]$ $[y $\overset{\text{tex}}{=}$ t]$ $[y $\overset{\text{name}}{=}$ n]$][114]

5 Claims

5.1 The claim aspect

We define the *claim* of a page to be the potentially inherited claim aspect of the page (c.f. 4.4.5). Logiweb checks the correctness of a Logiweb page by applying the claim of the page to the expansion and cache of the page. If the result is [T] then the page is considered correct.

---

[109] [intro(x, i, p, t) $\overset{\text{pyk}}{=}$ "intro ∗ index ∗ pyk ∗ tex ∗ end intro"]

[110] [intro(x, p, t) $\overset{\text{pyk}}{=}$ "intro ∗ pyk ∗ tex ∗ end intro"]

[111] [x/intro(y, p, t) $\overset{\text{pyk}}{=}$ "∗ intro ∗ pyk ∗ tex ∗ end intro"]

[112] [x/indexintro(y, i, p, t) $\overset{\text{pyk}}{=}$ "∗ intro ∗ index ∗ pyk ∗ tex ∗ end intro"]

[113] [x/nameintro(y, p, t, n) $\overset{\text{pyk}}{=}$ "∗ intro ∗ pyk ∗ tex ∗ name ∗ end intro"]

[114] [x/bothintro(y, i, p, t, n) $\overset{\text{pyk}}{=}$ "∗ intro ∗ index ∗ pyk ∗ tex ∗ name ∗ end intro"]

If the page has no potentially inherited claim aspect then the page is said to make no claim and is considered trivially correct.

We shall refer to the result of applying the claim of a page to the expansion and cache of the page as the *correctness* of the page. The correctness equals $[\mathsf{T}]$ for pages that are correct, including pages that are trivially correct.

Pages are in error if their correctness differs from $[\mathsf{T}]$. In that case Logiweb prunes the correctness (c.f. Section 4.4.4) and hangs the result on the *diagnose* hook (c.f. Section 4.3.6) of the page.

The proclaimation of the "claim" aspect reads:

$$[\text{claim} \bowtie \text{“claim”}]$$

For convenience, we define a construct for making claim definitions:

$$[[\mathsf{x} \overset{\text{claim}}{=} \mathsf{y}] \doteq [\mathsf{x} \overset{\text{claim}}{\rightarrow} \mathsf{y}]]^{115}$$

5.2 Conjunction of claims

Logiweb merely allows a page to make one claim. But that claim can be arbitrarily complex. As we shall see, the claim of the present page is the conjuctions of a "checker" and a "verifier".

The "checker" scans the expansion of the page for tests like $[\mathsf{F} +2* 3 \approx 7]^{\cdot}$ and checks the correctness of them all. In contrast, the "verifier" scans the codex for proofs and verifies all proofs found.

Claims may be combined into a conjunction using $[\mathsf{x} \wedge_{\mathrm{c}} \mathsf{y}]$:

$$[\mathsf{x} \mathbin{\tilde{\wedge}} \mathsf{y} \doteq \mathrm{if}(\mathsf{x}, \mathsf{y}, \mathsf{x})]^{116}$$

$$[\mathsf{x} \wedge_{\mathrm{c}} \mathsf{y} \doteq \lambda\mathsf{t}.\lambda\mathsf{c}.\mathsf{x}\,\text{'}\,\mathsf{t}\,\text{'}\,\mathsf{c} \mathbin{\tilde{\wedge}} \mathsf{y}\,\text{'}\,\mathsf{t}\,\text{'}\,\mathsf{c}]^{117}$$

5.3 The checker

The *checker* checks the correctness of all test statements on a page:

$$[\text{checker} \doteq \lambda\mathsf{t}.\lambda\mathsf{c}.\mathbf{check}(\mathsf{t}, \mathsf{c})]^{118}$$

The construct $[\mathbf{check}(\mathsf{t}, \mathsf{c})]^{119}$ allows to check that all claims in the term $[\mathsf{t}]$ evaluate to $[\mathsf{T}]$.

If some claim is false then $[\mathbf{check}(\mathsf{t}, \mathsf{c})]$ returns the value of the first false claim found. Such a non-nil value is a Logiweb *diagnose*. To be useful, a diagnose should be a Gödel tree that can be typeset and which gives a clue what went wrong.

$[\mathbf{check}(\mathsf{t}, \mathsf{c})]$ skips all page constructs when checking $[\mathsf{t}]$.

---

[115] $[[\mathsf{x} \overset{\text{claim}}{=} \mathsf{y}] \overset{\text{pyk}}{=} \text{“claim define} * \text{as} * \text{end define”}]$

[116] $[\mathsf{x} \mathbin{\tilde{\wedge}} \mathsf{y} \overset{\text{pyk}}{=} \text{“}* \text{simple and} * \text{”}]$

[117] $[\mathsf{x} \wedge_{\mathrm{c}} \mathsf{y} \overset{\text{pyk}}{=} \text{“}* \text{claim and} * \text{”}]$

[118] $[\text{checker} \overset{\text{pyk}}{=} \text{“checker”}]$

[119] $[\mathbf{check}(\mathsf{t}, \mathsf{c}) \overset{\text{pyk}}{=} \text{“check} * \text{cache} * \text{end check”}]$

$$[\mathbf{check}(\mathsf{t},\mathsf{c}) \doteq \mathrm{If}(\mathsf{t}^{\mathrm{is}},\mathsf{c}!\mathsf{T},\mathbf{check}_2(\mathsf{t},\mathsf{c},\mathbf{aspect}(\texttt{"claim"},\mathsf{t},\mathsf{c})))]$$

The construct $[\mathbf{check}_2(\mathsf{t},\mathsf{c},\mathsf{d})]$[120] allows to check that all claims in $[\mathsf{t}]$ evaluate to $[\mathsf{T}]$ where $[\mathsf{d}]$ is the definition of the claim aspect of the root of $[\mathsf{t}]$.

$$[\mathbf{check}_2(\mathsf{t},\mathsf{c},\mathsf{d}) \doteq \mathsf{d}\left\{\begin{array}{l}\mathbf{check}_3(\mathsf{t},\mathsf{c},\mathbf{aspect}(\texttt{"definition"},\mathsf{t},\mathsf{c}))\\ \mathcal{U}^{\mathrm{M}}(\mathcal{E}(\mathsf{d}^3,\mathsf{T},\mathsf{c})\text{`}\,\mathsf{t}\,\text{`}\,\mathsf{c})\end{array}\right.]$$

The construct $[\mathbf{check}_3(\mathsf{t},\mathsf{c},\mathsf{d})]$[121] allows to check that all claims in $[\mathsf{t}]$ evaluate to $[\mathsf{T}]$. $[\mathbf{check}_3(\mathsf{t},\mathsf{c},\mathsf{d})]$ does not descend into trees whose root has a definition aspect $[\mathsf{d}]$. For that reason, checking does not occur inside definitions, introductions, proclamations, and trees with a "hide" construct in its root. In particular, no checking is done inside strings since strings have a "hide" construct in their root. This saves a lot of cpu-time. Furthermore, no checking is done inside definitions. This is convenient since otherwise one could not pyk, TEX, macro, and priority define test constructs (c.f. Section 5.5 ff) without getting spurious errors from the occurrences of the test constructs inside the definitions of the test constructs themselves.

$$[\mathbf{check}_3(\mathsf{t},\mathsf{c},\mathsf{d}) \doteq \mathrm{If}(\mathsf{d},\mathbf{check}^*(\mathsf{t}^{\mathrm{t}},\mathsf{c}),\mathsf{t}!\mathsf{c}!\mathsf{T})]$$

The construct $[\mathbf{check}^*(\mathsf{t},\mathsf{c})]$[122] allows to check that all claims in the list $[\mathsf{t}]$ of terms evaluate to $[\mathsf{T}]$.

$$[\mathbf{check}^*(\mathsf{t},\mathsf{c}) \doteq \mathrm{If}(\mathsf{t},\mathsf{c}!\mathsf{T},\mathbf{check}_2^*(\mathsf{t}^{\mathrm{t}},\mathsf{c},\mathbf{check}(\mathsf{t}^{\mathrm{h}},\mathsf{c})))]$$

The construct $[\mathbf{check}_2^*(\mathsf{t},\mathsf{c},\mathsf{v})]$[123] also allows to check that all claims in the list $[\mathsf{t}]$ of terms evaluate to $[\mathsf{T}]$ except that it returns $[\mathsf{v}]$ if $[\mathsf{v}]$ is false.

$$[\mathbf{check}_2^*(\mathsf{t},\mathsf{c},\mathsf{v}) \doteq \mathrm{If}(\neg\mathsf{v},\mathsf{t}!\mathsf{c}!\mathsf{v},\mathbf{check}^*(\mathsf{t},\mathsf{c}))]$$

5.4 Installing the checker

We now define

$$[\mathrm{self} \overset{\mathrm{claim}}{=} \mathrm{checker} \wedge_{\mathrm{c}} \mathrm{verifier}]$$

Because of the definition above, the present page claims that $[\mathbf{check}(\mathsf{t},\mathsf{c})]$ is true where $[\mathsf{t}]$ and $[\mathsf{c}]$ are the expansion and cache, respectively, of the present page (the claim above also invokes [verifier] which is defined later).

Another page that uses the present page as bed and makes no claim of its own also claims that $[\mathbf{check}(\mathsf{t},\mathsf{c})]$ is true but then $[\mathsf{t}]$ and $[\mathsf{c}]$ are the expansion and cache, respectively, of that other page.

---

[120] $[\mathbf{check}_2(\mathsf{t},\mathsf{c},\mathsf{d}) \overset{\mathrm{pyk}}{=}$ "check two $*$ cache $*$ def $*$ end check"]

[121] $[\mathbf{check}_3(\mathsf{t},\mathsf{c},\mathsf{d}) \overset{\mathrm{pyk}}{=}$ "check three $*$ cache $*$ def $*$ end check"]

[122] $[\mathbf{check}^*(\mathsf{t},\mathsf{c}) \overset{\mathrm{pyk}}{=}$ "check list $*$ cache $*$ end check"]

[123] $[\mathbf{check}_2^*(\mathsf{t},\mathsf{c},\mathsf{v}) \overset{\mathrm{pyk}}{=}$ "check list two $*$ cache $*$ value $*$ end check"]

5.5 Test for truth

The construct $[\mathsf{t}]^{\cdot}$[124] allows to claim that a term $[\mathsf{t}]$ evaluates to $[\mathsf{T}]$.

$$[[\mathsf{t}]^{\cdot} \overset{\text{claim}}{=} \lambda\mathsf{t}.\lambda\mathsf{c}.\mathrm{if}(\mathcal{U}(\mathcal{E}(\mathsf{t}^1, \mathsf{T}, \mathsf{c})), \mathsf{T}, \mathsf{t})]$$

As an example, one may claim that $[\mathsf{F} +2* 3]$ equals seven:

$$[\mathsf{F} +2* 3 \approx 7]^{\cdot}$$

$[\mathcal{A}]^{\cdot}$ returns the Gödel tree of $[\mathcal{A}]^{\cdot}$ as the diagnose if the claim fails. The overall claim defined in Section 5.4 returns the value of the first failing claim (if any). Hence, if $[\mathsf{F} +2* 3 \approx 7]^{\cdot}$ failed and was the first failing claim found, then the value of the claim of the page would be the Gödel tree of $[\mathsf{F} +2* 3 \approx 7]^{\cdot}$. Reasonable Logiweb compilers typeset the diagnose so that the failing claim is easy to identify.

Note that macro expansion occurs before checking, so the Gödel tree of a failing claim is the macro expanded version of the claim. The macro expansion facility defined on the present page, however, carefully carries debugging information around when macro expanding, so the debugging information in the root of the expanded Gödel tree indicates the exact location of the failing claim in the unexpanded page.

Note that $[\mathsf{t}]^{\cdot}$ does not fail; all unbound variables evaluate to $[\mathsf{T}]$.

5.6 Test for falsehood

The construct

$$[[\mathsf{x}]^{-} \overset{\text{claim}}{=} \lambda\mathsf{t}.\lambda\mathsf{c}.\mathrm{if}(\mathcal{U}(\mathcal{E}(\mathsf{t}^1, \mathsf{T}, \mathsf{c})), \mathsf{t}, \mathsf{T})]$$[125]

tests that its argument is false. As an example of use, one may claim that $[2 \approx 3]$ is false:

$$[2 \approx 3]^{-}$$

5.7 Raw test

Like $[\mathsf{t}]^{\cdot}$, the construct $[\mathsf{t}]^{\circ}$[126] allows to claim that a term $[\mathsf{t}]$ evaluates to $[\mathsf{T}]$. Contrary to $[\mathsf{x}]^{\cdot}$, $[\mathsf{x}]^{\circ}$ just returns the value of $[\mathsf{x}]$. This is useful for constructs that generate their own error messages.

$$[[\mathsf{t}]^{\circ} \overset{\text{claim}}{=} \lambda\mathsf{t}.\lambda\mathsf{c}.\mathcal{U}(\mathcal{E}(\mathsf{t}^1, \mathsf{T}, \mathsf{c}))]$$

[124] $[[\mathsf{t}]^{\cdot} \overset{\text{pyk}}{=}$ "test $*$ end test"$]$

[125] $[[\mathsf{x}]^{-} \overset{\text{pyk}}{=}$ "false test $*$ end test"$]$

[126] $[[\mathsf{t}]^{\circ} \overset{\text{pyk}}{=}$ "raw test $*$ end test"$]$

5.8 Spy construct

[x spy y $\overset{\text{val}}{\Rightarrow}$ x!y][127] evaluates [x], discards the value, evaluates [y], and returns the value of [y]. However, behind the scene, [x spy y] also stores the value of [x] in the Lisp variable *spy*.

[[x]$^{\cdot}$ $\ddot{=}$ [⌈x⌉ spy x]$^{\cdot}$][128] is exactly like an ordinary test except that it calls the spy function (the spying nature is emphasized by the fact that spying and non-spying tests are rendered identically:-). [[x]$^{-}$ $\ddot{=}$ [⌈x⌉ spy x]$^{-}$][129] is a spying test for falsehood.

Spying tests are good for debugging when some test loops indefinitely. If a test case loops indefinitely, then *spy* will typically contain the last spying test that was executed before the infinite loop.

6 Logiweb sequent calculus

The *Logiweb sequent calculus* is the low level theory (or "metatheory") that comes with Logiweb. Users may use Logiweb sequent calculus and its associated proof checker as they are or use them as a model for constructing their own proof checkers.

The Logiweb sequent calculus corresponds to the assembly language of computing machines. It is possible in principle to use it as it is, but it is much more efficient to use higher level theories built on top of the calculus. When building theories on top of the calculus, the calculus ensures correctness whereas the high level theories provide user friendliness. Programming bugs in the high level theories cannot compromise correctness since correctness is ensured by the low level sequent calculus. Only bugs in the implementation of the sequent calculus and in the levels below can compromise correctness.

In contrast, *Map Theory* described later is the high level theory that comes with Logiweb. Again, users may use Map Theory as it is or use it as a model for constructing their own theories. Map Theory is a high level theory implemented on top of the Logiweb sequent calculus.

Map Theory is a classical theory with power as classical ZFC set theory. Contrary to Set Theory, Map Theory builds on lambda calculus. More specifically, Map Theory is the result of adding quantifiers to the Logiweb computing engine. For that reason, Map Theory is suited both for classical mathematics and for reasoning about programs in general and programs for the Logiweb engine in particular.

Logiweb sequent calculus is intuitionistic and has much less power than Map Theory.

Now let [$\mathcal{M}$] be the intuitionistic conjunction of all axioms, axioms schemes, and inference rules of Map Theory and let [$\mathcal{L}$] be a lemma of Map Theory.

[127] [x spy y $\overset{\text{pyk}}{=}$ "$*$ spy $*$"]

[128] [[x]$^{\cdot}$ $\overset{\text{pyk}}{=}$ "spying test $*$ end test"]

[129] [[x]$^{-}$ $\overset{\text{pyk}}{=}$ "false spying test $*$ end test"]

In general, $[\mathcal{L}]$ will not be intuitionistically valid and, hence, $[\mathcal{L}]$ will not be a lemma of Logiweb sequent calculus. But the statement that $[\mathcal{M}]$ infers $[\mathcal{L}]$, written $[\mathcal{M} \vdash \mathcal{L}]$, will be intuitionistically valid, and that is the lemma to be proved from the point of view of the sequent calculus.

Hence, if the user proves a lemma $[\mathcal{L}]$ in Map Theory, then the lemma will be translated into $[\mathcal{M} \vdash \mathcal{L}]$ and the proof will be translated into a proof of that statement in the sequent calculus.

Now let $[\mathcal{Z}]$ be the intuitionistic conjunction of all axioms, axiom schemes, and inference rules of classical ZFC set theory. At the time of writing, a rather large, formal proof of $[\mathcal{M} \vdash \mathcal{Z}]$ exists, and that particular proof is going to be the main test case for checkout of the Logiweb system. The lemma $[\mathcal{M} \vdash \mathcal{Z}]$ says that Map Theory can "simulate" ZFC set theory and that all developments possible in ZFC set theory are also possible in Map Theory.

6.1 Statements

6.1.1 Messages

Logiweb comes with a number of predefined aspects like [pyk], [tex], [val], and [macro]. Logiweb also has a facility for defining new aspects. New aspects are introduced using the *message* aspect. The message aspect is a predefined one and, hence, has to be proclaimed:

$$[\mathsf{msg} \bowtie \text{“message”}]^{130}$$

For convenience, we define a construct for making message definitions:

$$[[\mathsf{x} \overset{\mathsf{msg}}{=} \mathsf{y}] \doteq [(\mathsf{x})^{\mathbf{p}} \overset{\mathsf{msg}}{\rightarrow} \mathsf{y}]]^{131}$$

Just like a construct may have e.g. a value aspect, a construct may have a message aspect. If a construct $[\mathcal{A}]$ has a message aspect $[\mathcal{B}]$, then $[\mathcal{A}]$ denotes the aspect $[\mathcal{B}]$. Hence, $[\mathcal{B}]$ becomes an aspect by occurring in the right hand side of a message definition.

If two constructs $[\mathcal{A}_1]$ and $[\mathcal{A}_2]$ both have message aspect $[\mathcal{B}]$ then $[\mathcal{A}_1]$ and $[\mathcal{A}_2]$ denote the same aspect. The ability to have several constructs that denote the same aspect allows different users to use different notation for the same aspect. This is important because of the notational freedom of mathematics.

If a construct $[\mathcal{A}]$ has a message aspect $[\mathcal{B}]$ then both $[\mathcal{A}]$ and $[\mathcal{B}]$ may have parameters. Parameters of $[\mathcal{A}]$ allow to define parameterised aspects. Parameters of $[\mathcal{B}]$ have no effect since an aspect is identified by the reference and identifier of the root of $[\mathcal{B}]$.

Aspects of Logiweb have a slight resemblance with messages in object-oriented programming, and for that reason, we use "aspect" and "message" as synonyms here. But why use "message" when "message" is just used as a synonym for "aspect"? The answer is that "message aspect" sounds better than "aspect aspect".

---

[130] $[\mathsf{msg} \overset{\mathsf{pyk}}{=} \text{“message”}]$

[131] $[[\mathsf{x} \overset{\mathsf{msg}}{=} \mathsf{y}] \overset{\mathsf{pyk}}{=} \text{“message define} * \text{as} * \text{end define”}]$

6.1.2 The statement aspect

We shall refer to entities like axioms and inference rules as *statements*. To define such entities it is convenient to have a *statement aspect*. To introduce a statement aspect, we first introduce [<stmt>][132] to represent it.

[<stmt>] has a [pyk] definition so that it can be referred to in pyk source files and a [tex] definition so that it can be rendered in TEX output from Logiweb, and it evaluates to itself:

$$[\text{<stmt>} \doteq \lceil\text{<stmt>}\rceil]$$

Apart from that [<stmt>] is inert. Its purpose is to occur in the right hand side of a message definition where it identifies the statement aspect.

We shall use [stmt] to denote the statement aspect:

$$[\text{stmt} \overset{\text{msg}}{=} \text{<stmt>}]$$[133]

For convenience, we define a construct for making statement definitions:

$$[[\mathsf{x} \overset{\text{stmt}}{=} \mathsf{y}] \doteq [(\mathsf{x})^{\mathbf{p}} \overset{\text{stmt}}{\rightarrow} \mathsf{y}]]$$[134]

6.1.3 Axioms

An *axiom* is a statement that somebody accepts as true without proof. As an example, one could declare $[\mathsf{T}^{\mathrm{h}} = \mathsf{T}]$ as an axiom. The definition $[\text{HeadNil}' \doteq \mathsf{T}^{\mathrm{h}} = \mathsf{T}]$[135] allows to use [HeadNil′] as shorthand for the given axiom.

6.1.4 Axiom schemes

An *axiom scheme* is a systematic collection of axioms. As an example, one could declare $[\forall\mathcal{A}{:}\,\forall\mathcal{B}{:}\,(\mathcal{A} :: \mathcal{B})^{\mathrm{h}} = \mathcal{A}]$ as an axiom scheme. The axiom scheme says that $[(\mathcal{A} :: \mathcal{B})^{\mathrm{h}} = \mathcal{A}]$ for all terms [$\mathcal{A}$] and [$\mathcal{B}$]. The definition $[\text{HeadPair}' \doteq \forall\mathcal{A}{:}\,\forall\mathcal{B}{:}\,(\mathcal{A} :: \mathcal{B})^{\mathrm{h}} = \mathcal{A}]$[136] allows to use [HeadPair′] as shorthand for the given axiom scheme.

The variables [$\mathcal{A}$] and [$\mathcal{B}$] are *metavariables*. Metavariables range over terms as opposed to *ordinary variables* which range over values. For more on metavariables see Section 6.2.

---

[132] $[\text{<stmt>} \overset{\text{pyk}}{=} \text{“the statement aspect”}]$

[133] $[\text{stmt} \overset{\text{pyk}}{=} \text{“statement”}]$

[134] $[[\mathsf{x} \overset{\text{stmt}}{=} \mathsf{y}] \overset{\text{pyk}}{=} \text{“statement define} * \text{as} * \text{end define”}]$

[135] $[\text{HeadNil}' \overset{\text{pyk}}{=} \text{“example axiom”}]$

[136] $[\text{HeadPair}' \overset{\text{pyk}}{=} \text{“example scheme”}]$

6.1.5 Inference rules

An *inference rule* is a rule which states that a certain *conclusion* follows from one or more *premisses*. An inference rule states that the conclusion is provable if all the premisses are provable.

As an example, one could declare $[\forall\mathcal{A}\colon\forall\mathcal{B}\colon\forall\mathcal{C}\colon \mathcal{A}=\mathcal{B}\vdash\mathcal{A}=\mathcal{C}\vdash\mathcal{B}=\mathcal{C}]$ as an inference rule. The inference rule says that if $[\mathcal{A}]$, $[\mathcal{B}]$, and $[\mathcal{C}]$ are arbitrary terms, and if $[\mathcal{A}=\mathcal{B}]$ and $[\mathcal{A}=\mathcal{C}]$ are provable, then $[\mathcal{B}=\mathcal{C}]$ is provable.

The definition $[\text{Transitivity}' \doteqdot \forall\mathcal{A}\colon\forall\mathcal{B}\colon\forall\mathcal{C}\colon \mathcal{A}=\mathcal{B}\vdash\mathcal{A}=\mathcal{C}\vdash\mathcal{B}=\mathcal{C}]$[137] allows to use [Transitivity′] as shorthand for the given inference rule.

6.1.6 Contradictions

We make the convention that $[\perp\!\!\!\perp]$ represents absurdity.

A Logiweb *contradiction* is a statement that somebody accepts as false without proof. As an example, one could declare $[\mathsf{T}::\mathsf{T}=\mathsf{T}]$ as a contradiction. The contradiction $[\mathsf{T}::\mathsf{T}=\mathsf{T}]$ may be expressed as the statement $[\mathsf{T}::\mathsf{T}=\mathsf{T}\vdash\perp\!\!\!\perp]$.

Contradictions were called "Logiweb contradictions" first time they were mentioned above. That was done to emphasize that the word "contradiction" is used here in a way that is not necessarily in line with the litterature.

As we shall see, contradictions have two uses. Firstly, they may be used to refute a conjecture by proving that the conjecture implies a contradiction. Secondly, they may be used to refute a theory as inconsistent by proving that a contradiction is provable within the theory.

The definition $[\text{Contra}' \doteqdot \mathsf{T}::\mathsf{T}=\mathsf{T}\vdash\perp\!\!\!\perp]$[138] allows to use [Contra′] as shorthand for the given contradiction.

6.1.7 Theories

We shall refer to axioms, axiom schemes, and inference rules as *rules*.

A Logiweb *theory* is a structure that encodes a finite set of rules and contradictions. As an example, one could declare the rules [HeadNil′], [HeadPair′], and [Transitivity′] and the contradiction [Contra′] to form a theory.

The definition

$$[\mathrm{T}'_{\mathrm{E}} \overset{\text{stmt}}{=} \text{HeadNil}' \oplus \text{HeadPair}' \oplus \text{Transitivity}' \oplus \text{Contra}']$$[139]

allows to refer to the example theory as $[\mathrm{T}'_{\mathrm{E}}]$ ("T" for "Theory", "E" for "Example", $[\mathrm{T}'_{\mathrm{E}}]$ for "example theory").

The notion of a "Logiweb theory" is close to the notion of an "axiotic theory" in the litterature. Some branches of the litterature use the word "theory" in a quite different sense, namely to denote the set of all statements provable from an axiomatic theory.

---

[137] [Transitivity′ $\overset{\text{pyk}}{=}$ "example rule"]

[138] [Contra′ $\overset{\text{pyk}}{=}$ "contraexample"]

[139] [T'_{E} $\overset{\text{pyk}}{=}$ "example theory primed"]

The notion of a "Logiweb theory" differs from the notion of an "axiomatic theory" in three ways:

Firstly, the notion of a "Logiweb theory" has a precise, technical meaning inside the Logiweb framework. The notion of an "axiomatic theory" is independent of any framework but is more vague.

Secondly, a Logiweb theory may include Logiweb contradictions. Axiomatic theories merely contain rules.

Finally, a Logiweb theory contains finitely many rules whereas an axiomatic theory may contain infinitely many. This is because Logiweb theories and axiomatic theories count rules differently.

Axiomatic theories count rules in a very peculiar way: (1) inference rules do not count. (2) axioms that are axioms of first order predicate calculus do not count. (3) axiom schemes count as infinitely many rules. A theory like ZFC set theory [10] has the property that it is "not finitely axiomatizable". That result depends on the counting used in axiomatic theories. Using Logiweb counting, ZFC has finitely many rules. All axiomatic theories can be expressed as Logiweb theories using finitely many rules.

6.1.8 Conjectures, lemmas, and antilemmas

A Logiweb *conjecture* is a statement of form $[\mathcal{T} \vdash \mathcal{C}]$ where $[\mathcal{T}]$ is a theory and $[\mathcal{C}]$ is a statement that is expected to follow from the theory. As an example, one could declare $[\mathrm{T}'_{\mathrm{E}} \vdash \forall\mathcal{A}\colon\forall\mathcal{B}\colon \mathcal{A} = \mathcal{B} \vdash \mathcal{B} = \mathcal{A}]$ as a conjecture.

The definition $[\mathrm{L}_1 \stackrel{\mathrm{stmt}}{=} \mathrm{T}'_{\mathrm{E}} \vdash \forall\mathcal{A}\colon\forall\mathcal{B}\colon \mathcal{A} = \mathcal{B} \vdash \mathcal{B} = \mathcal{A}]$[140] allows to refer to the given conjecture as $[\mathrm{L}_1]$.

Later, we introduce the "Logiweb proof checker" and the notion of a "Logiweb proof". Once a proof of a conjecture is written, accepted by the proof checker, and published on Logiweb, one may refer to the conjecture as a Logiweb *lemma*.

Furthermore, if $[\mathcal{T} \vdash \mathcal{C} \vdash \perp\!\!\!\perp]$, i.e. $[\mathcal{T} \vdash (\mathcal{C} \vdash \perp\!\!\!\perp)]$ is a Logiweb lemma then one may refer to the conjecture $[\mathcal{T} \vdash \mathcal{C}]$ as a Logiweb *antilemma*.

6.1.9 Statement constructors

We shall refer to a term as a *statement* if we care about its mathematical truth. Hence, rules, contradictions, theories, conjectures, lemmas, and antilemmas are statements.

The constructors $[\mathsf{x} \vdash \mathsf{y}]$[141], $[\mathsf{x} \Vdash \mathsf{y}]$[142], $[\forall\mathsf{x}\colon \mathsf{y}]$[143], $[\perp\!\!\!\perp]$[144], and $[\mathsf{x} \oplus \mathsf{y}]$[145] allow to form statements from arbitrary terms and statements:

---

[140] $[\mathrm{L}_1 \stackrel{\mathrm{pyk}}{=}$ "example lemma"$]$

[141] $[\mathsf{x} \vdash \mathsf{y} \stackrel{\mathrm{pyk}}{=}$ "$*$ infer $*$"$]$

[142] $[\mathsf{x} \Vdash \mathsf{y} \stackrel{\mathrm{pyk}}{=}$ "$*$ endorse $*$"$]$

[143] $[\forall\mathsf{x}\colon \mathsf{y} \stackrel{\mathrm{pyk}}{=}$ "all $*$ indeed $*$"$]$

[144] $[\perp\!\!\!\perp \stackrel{\mathrm{pyk}}{=}$ "absurdity"$]$

[145] $[\mathsf{x} \oplus \mathsf{y} \stackrel{\mathrm{pyk}}{=}$ "$*$ rule plus $*$"$]$

- $[\mathcal{A} \vdash \mathcal{B}]$ reads $[\mathcal{A}]$ *infers* $[\mathcal{B}]$ and states that if $[\mathcal{A}]$ is provable then $[\mathcal{B}]$ is provable.
- $[\mathcal{A} \Vdash \mathcal{B}]$ reads $[\mathcal{A}]$ *endorses* $[\mathcal{B}]$ and states that if $[\mathcal{A}]$ evaluates to $[\mathsf{T}]$ then $[\mathcal{B}]$ is provable.
- $[\forall \mathcal{A}\colon \mathcal{B}]$ reads *all* $[\mathcal{A}]$ *indeed* $[\mathcal{B}]$ and states that $[\mathcal{B}]$ is provable for all terms $[\mathcal{A}]$.
- $[\perp\!\!\!\perp]$ reads *absurdity*. One may claim that a term $[\mathcal{A}]$ is disprovable by the statement $[\mathcal{A} \vdash \perp\!\!\!\perp]$.
- $[\mathcal{A} \oplus \mathcal{B}]$ reads $[\mathcal{A}]$ *rule plus* $[\mathcal{B}]$ and states that $[\mathcal{A}]$ and $[\mathcal{B}]$ are provable.

All the constructors above except endorsement have been used in examples. According to the Unicode standard, $[\Vdash]$ reads "forses", but we use $[\mathcal{A} \Vdash \mathcal{B}]$ for endorsement rather than enforcement.

$[\mathcal{A} \Vdash \mathcal{B}]$ states that the side condition $[\mathcal{A}]$ endorses the conclusion $[\mathcal{B}]$. The construct can be used for statements like "$[\mathcal{B}]$ is free for $[\mathcal{X}]$ in $[\mathcal{A}]$ endorses $[(\lambda\mathcal{X}.\mathcal{A})\,'\,\mathcal{B} = \langle \mathcal{A}\,|\mathcal{X}{:=}\mathcal{B}\rangle]$.

To give a meaningful example of the use of endorsement, however, one has to program some side condition like "is free for" first. Examples of use of endorsement will be given later.

6.1.10 Self-evaluation

We define the statement constructors such that they are "self-evaluating". As an example, the definition of $\mathsf{x} \vdash \mathsf{y}$ is such that e.g. $[\lceil\mathcal{A}\rceil \vdash \lceil\mathcal{B}\rceil]$ and $[\lceil\mathcal{A} \vdash \mathcal{B}\rceil]$ are equal terms (equal except for debugging information present in the terms which indicates exactly where the terms occur on the Logiweb page). The following definitions make the constructors self-evaluating:

$[\mathsf{x} \vdash \mathsf{y} \doteq \langle\lceil\mathsf{x} \vdash \mathsf{y}\rceil^{\mathrm{R}}, \mathsf{x}, \mathsf{y}\rangle]$
$[\mathsf{x} \Vdash \mathsf{y} \doteq \langle\lceil\mathsf{x} \Vdash \mathsf{y}\rceil^{\mathrm{R}}, \mathsf{x}, \mathsf{y}\rangle]$
$[\forall\mathsf{x}\colon \mathsf{y} \doteq \langle\lceil\forall\mathsf{x}\colon \mathsf{y}\rceil^{\mathrm{R}}, \mathsf{x}, \mathsf{y}\rangle]$
$[\perp\!\!\!\perp \doteq \langle\lceil\perp\!\!\!\perp\rceil^{\mathrm{R}}\rangle]$
$[\mathsf{x} \oplus \mathsf{y} \doteq \langle\lceil\mathsf{x} \oplus \mathsf{y}\rceil^{\mathrm{R}}, \mathsf{x}, \mathsf{y}\rangle]$
$[\mathsf{x}^{\mathrm{I}} \doteq \langle\lceil\mathsf{x}^{\mathrm{I}}\rceil^{\mathrm{R}}, \mathsf{x}\rangle]$
$[\mathsf{x}^{\triangleright} \doteq \langle\lceil\mathsf{x}^{\triangleright}\rceil^{\mathrm{R}}, \mathsf{x}\rangle]$
$[\mathsf{x}^{\mathrm{V}} \doteq \langle\lceil\mathsf{x}^{\mathrm{V}}\rceil^{\mathrm{R}}, \mathsf{x}\rangle]$
$[\mathsf{x}^{+} \doteq \langle\lceil\mathsf{x}^{+}\rceil^{\mathrm{R}}, \mathsf{x}\rangle]$
$[\mathsf{x}^{-} \doteq \langle\lceil\mathsf{x}^{-}\rceil^{\mathrm{R}}, \mathsf{x}\rangle]$
$[\mathsf{x}^{*} \doteq \langle\lceil\mathsf{x}^{*}\rceil^{\mathrm{R}}, \mathsf{x}\rangle]$
$[\mathsf{x}\,@\,\mathsf{y} \doteq \langle\lceil\mathsf{x}\,@\,\mathsf{y}\rceil^{\mathrm{R}}, \mathsf{x}, \mathsf{y}\rangle]$
$[\mathsf{x}\ \text{i.e.}\ \mathsf{y} \doteq \langle\lceil\mathsf{x}\ \text{i.e.}\ \mathsf{y}\rceil^{\mathrm{R}}, \mathsf{x}, \mathsf{y}\rangle]$
$[\mathsf{x};\mathsf{y} \doteq \langle\lceil\mathsf{x};\mathsf{y}\rceil^{\mathrm{R}}, \mathsf{x}, \mathsf{y}\rangle]$

6.2 Metavariables

6.2.1 Definition of metavariables

As mentioned in Section 6.1.4, *metavariables* range over terms as opposed to ordinary variables which range over values. In the sequent calculus presented in Section 6.3 we need to distinguish metavariables from other variables.

To do so, we simply introduce the unary operator $[\underline{\mathsf{x}}]$[146] and make the convention that a term is a metavariable iff it has the $[\underline{*}]$ construct in its root.

Having this convention in place, we introduce the metavariables $[\mathcal{A} \doteq \underline{\mathsf{a}}]$[147], $[\mathcal{B} \doteq \underline{\mathsf{b}}]$[148], $[\mathcal{C} \doteq \underline{\mathsf{c}}]$[149], $[\mathcal{D} \doteq \underline{\mathsf{d}}]$[150], $[\mathcal{E} \doteq \underline{\mathsf{e}}]$[151], $[\mathcal{F} \doteq \underline{\mathsf{f}}]$[152], $[\mathcal{G} \doteq \underline{\mathsf{g}}]$[153], $[\mathcal{H} \doteq \underline{\mathsf{h}}]$[154], $[\mathcal{I} \doteq \underline{\mathsf{i}}]$[155], $[\mathcal{J} \doteq \underline{\mathsf{j}}]$[156], $[\mathcal{K} \doteq \underline{\mathsf{k}}]$[157], $[\mathcal{L} \doteq \underline{\mathsf{l}}]$[158], $[\mathcal{M} \doteq \underline{\mathsf{m}}]$[159], $[\mathcal{N} \doteq \underline{\mathsf{n}}]$[160], $[\mathcal{O} \doteq \underline{\mathsf{o}}]$[161], $[\mathcal{P} \doteq \underline{\mathsf{p}}]$[162], $[\mathcal{Q} \doteq \underline{\mathsf{q}}]$[163], $[\mathcal{R} \doteq \underline{\mathsf{r}}]$[164], $[\mathcal{S} \doteq \underline{\mathsf{s}}]$[165], $[\mathcal{T} \doteq \underline{\mathsf{t}}]$[166], $[\mathcal{U} \doteq \underline{\mathsf{u}}]$[167], $[\mathcal{V} \doteq \underline{\mathsf{v}}]$[168], $[\mathcal{W} \doteq \underline{\mathsf{w}}]$[169], $[\mathcal{X} \doteq \underline{\mathsf{x}}]$[170], $[\mathcal{Y} \doteq \underline{\mathsf{y}}]$[171], and $[\mathcal{Z} \doteq \underline{\mathsf{z}}]$[172].

---

146 $[\underline{\mathsf{x}} \overset{\text{pyk}}{=}$ "metavar $*$ end metavar"$]$

147 $[\mathcal{A} \overset{\text{pyk}}{=}$ "meta a"$]$

148 $[\mathcal{B} \overset{\text{pyk}}{=}$ "meta b"$]$

149 $[\mathcal{C} \overset{\text{pyk}}{=}$ "meta c"$]$

150 $[\mathcal{D} \overset{\text{pyk}}{=}$ "meta d"$]$

151 $[\mathcal{E} \overset{\text{pyk}}{=}$ "meta e"$]$

152 $[\mathcal{F} \overset{\text{pyk}}{=}$ "meta f"$]$

153 $[\mathcal{G} \overset{\text{pyk}}{=}$ "meta g"$]$

154 $[\mathcal{H} \overset{\text{pyk}}{=}$ "meta h"$]$

155 $[\mathcal{I} \overset{\text{pyk}}{=}$ "meta i"$]$

156 $[\mathcal{J} \overset{\text{pyk}}{=}$ "meta j"$]$

157 $[\mathcal{K} \overset{\text{pyk}}{=}$ "meta k"$]$

158 $[\mathcal{L} \overset{\text{pyk}}{=}$ "meta l"$]$

159 $[\mathcal{M} \overset{\text{pyk}}{=}$ "meta m"$]$

160 $[\mathcal{N} \overset{\text{pyk}}{=}$ "meta n"$]$

161 $[\mathcal{O} \overset{\text{pyk}}{=}$ "meta o"$]$

162 $[\mathcal{P} \overset{\text{pyk}}{=}$ "meta p"$]$

163 $[\mathcal{Q} \overset{\text{pyk}}{=}$ "meta q"$]$

164 $[\mathcal{R} \overset{\text{pyk}}{=}$ "meta r"$]$

165 $[\mathcal{S} \overset{\text{pyk}}{=}$ "meta s"$]$

166 $[\mathcal{T} \overset{\text{pyk}}{=}$ "meta t"$]$

167 $[\mathcal{U} \overset{\text{pyk}}{=}$ "meta u"$]$

168 $[\mathcal{V} \overset{\text{pyk}}{=}$ "meta v"$]$

169 $[\mathcal{W} \overset{\text{pyk}}{=}$ "meta w"$]$

170 $[\mathcal{X} \overset{\text{pyk}}{=}$ "meta x"$]$

171 $[\mathcal{Y} \overset{\text{pyk}}{=}$ "meta y"$]$

172 $[\mathcal{Z} \overset{\text{pyk}}{=}$ "meta z"$]$

6.2.2 Recognition of metavariables

$[\mathsf{t}^{\mathcal{V}}]$[173] is true if $[\mathsf{t}]$ is a metavariable:

$$[\mathsf{t}^{\mathcal{V}} \doteq \mathsf{t} \overset{\mathrm{r}}{=} \lceil \mathcal{A} \rceil]$$

6.2.3 Closedness

$[\mathsf{t}^{\mathcal{C}}]$[174] is true if the term $[\mathsf{t}]$ contains no metavariables and $[\mathsf{t}^{\mathcal{C}^*}]$[175] is true if the list $[\mathsf{t}]$ of terms contains no metavariables:

$$[\mathsf{t}^{\mathcal{C}} \doteq \mathrm{If}(\mathsf{t}^{\mathcal{V}}, \mathsf{F}, \mathsf{t}^{\mathrm{t}\mathcal{C}^*})]$$

$$[\mathsf{t}^{\mathcal{C}^*} \doteq \mathrm{If}(\mathsf{t}, \mathsf{T}, \mathrm{If}(\mathsf{t}^{\mathrm{h}\mathcal{C}}, \mathsf{t}^{\mathrm{t}\mathcal{C}^*}, \mathsf{F}))]$$

6.2.4 The "free in" predicate

$[\mathsf{v}$ free in $\mathsf{t}]$[176] is true if the metavariable $[\mathsf{v}]$ is occurs free in the term $[\mathsf{t}]$. Furthermore, $[\mathsf{v}$ free in* $\mathsf{t}]$[177] is true if the metavariable $[\mathsf{v}]$ is occurs free in the list $[\mathsf{t}]$ of terms.

$[\mathsf{v}$ free in $\mathsf{t} \doteq$
if $\mathsf{v} \overset{\mathrm{t}}{=} \mathsf{t}$ **then** T **else**
if $\neg\mathsf{t} \overset{\mathrm{r}}{=} \lceil \forall *\colon * \rceil$ **then** v free in* t^{t} **else**
if $\mathsf{v} \overset{\mathrm{t}}{=} \mathsf{t}^1$ **then** F **else** v free in $\mathsf{t}^2]$

$$[\mathsf{v} \text{ free in}^* \mathsf{t} \doteq \mathrm{If}(\mathsf{t}, \mathsf{v}!\mathsf{F}, \mathrm{If}(\mathsf{v} \text{ free in } \mathsf{t}^{\mathrm{h}}, \mathsf{T}, \mathsf{v} \text{ free in}^* \mathsf{t}^{\mathrm{t}}))]$$

6.2.5 The "free for" predicate

$[\mathsf{a}$ free for x in $\mathsf{b}]$[178] is true if the term $[\mathsf{a}]$ is free for the metavariable $[\mathsf{x}]$ in the term $[\mathsf{b}]$. Furthermore, $[\mathsf{a}$ free for* $*$ in $\mathsf{b}]$[179] is true if the term $[\mathsf{a}]$ is free for the metavariable $[\mathsf{x}]$ in the list $[\mathsf{b}]$ of terms.

$[\mathsf{a}$ free for x in $\mathsf{b} \doteq \mathsf{a}!\mathsf{x}!$
if $\mathsf{b}^{\mathcal{V}}$ **then** T **else**
if $\neg\mathsf{b} \overset{\mathrm{r}}{=} \lceil \forall *\colon * \rceil$ **then** a free for* x in b^{t} **else**
if $\mathsf{x} \overset{\mathrm{t}}{=} \mathsf{b}^1$ **then** T **else**
if $\neg\mathsf{x}$ free in b^2 **then** T **else**
if b^1 free in a **then** F **else**
a free for x in $\mathsf{b}^2]$

---

[173] $[\mathsf{t}^{\mathcal{V}} \overset{\mathrm{pyk}}{=}$ "$*$ is metavar"$]$
[174] $[\mathsf{t}^{\mathcal{C}} \overset{\mathrm{pyk}}{=}$ "$*$ is metaclosed"$]$
[175] $[\mathsf{t}^{\mathcal{C}^*} \overset{\mathrm{pyk}}{=}$ "$*$ is metaclosed star"$]$
[176] $[\mathsf{v}$ free in $\mathsf{t} \overset{\mathrm{pyk}}{=}$ "$*$ free in $*$"$]$
[177] $[\mathsf{v}$ free in* $\mathsf{t} \overset{\mathrm{pyk}}{=}$ "$*$ free in star $*$"$]$
[178] $[\mathsf{a}$ free for x in $\mathsf{b} \overset{\mathrm{pyk}}{=}$ "$*$ free for $*$ in $*$"$]$
[179] $[\mathsf{a}$ free for* $*$ in $\mathsf{b} \overset{\mathrm{pyk}}{=}$ "$*$ free for star $*$ in $*$"$]$

$[\mathsf{a}\text{ free for}^*\ \mathsf{x}\text{ in }\mathsf{b} \doteq$
if b **then** $\mathsf{a}!\mathsf{x}!\mathsf{T}$ **else**
if a free for x in b^{h} **then** a free for* x in b^{t} **else** $\mathsf{F}]$

6.2.6 Metavariable substitution

$[\langle \mathsf{a} \,|\mathsf{x}{:=}\, \mathsf{b}\rangle]$[180] replaces all free occurrences of the metavariable $[\mathsf{x}]$ by the term $[\mathsf{b}]$ in the term $[\mathsf{a}]$. The operation does no renaming of bound variables, so variable clashes will occur if $[\mathsf{b}]$ is not free for $[\mathsf{x}]$ in $[\mathsf{a}]$. Furthermore, $[\langle {}^*\mathsf{a} \,|\mathsf{x}{:=}\, \mathsf{b}\rangle]$[181] replaces all free occurrences of the metavariable $[\mathsf{x}]$ by the term $[\mathsf{b}]$ in the list $[\mathsf{a}]$ of terms.

$[\langle \mathsf{a} \,|\mathsf{x}{:=}\, \mathsf{b}\rangle \doteq \mathsf{x}!\mathsf{b}!$
if $\mathsf{a}^{\mathcal{V}}$ **then** $\mathrm{If}(\mathsf{a} \stackrel{\mathrm{t}}{=} \mathsf{x}, \mathsf{b}, \mathsf{a})$ **else**
if $\neg\mathsf{a} \stackrel{\mathrm{r}}{=} \lceil \forall *\colon * \rceil$ **then** $\mathsf{a}^{\mathrm{h}} :: \langle {}^*\mathsf{a}^{\mathrm{t}} \,|\mathsf{x}{:=}\, \mathsf{b}\rangle$ **else**
if $\mathsf{a}^1 \stackrel{\mathrm{t}}{=} \mathsf{x}$ **then** a **else**
$\langle \mathsf{a}^0, \mathsf{a}^1, \langle \mathsf{a}^2 \,|\mathsf{x}{:=}\, \mathsf{b}\rangle\rangle]$

$[\langle {}^*\mathsf{a} \,|\mathsf{x}{:=}\, \mathsf{b}\rangle \doteq \mathsf{x}!\mathsf{b}!\mathrm{If}(\mathsf{a}, \mathsf{T}, \langle \mathsf{a}^{\mathrm{h}} \,|\mathsf{x}{:=}\, \mathsf{b}\rangle :: \langle {}^*\mathsf{a}^{\mathrm{t}} \,|\mathsf{x}{:=}\, \mathsf{b}\rangle)]$

6.3 Sequent calculus

6.3.1 Term sets

We shall represent sets of terms as lists without repetitions. We require the lists to be without repetitions in the sense that a term $[\mathsf{x}]$ is not allowed to occur more than once and, furthermore, if two distinct terms $[\mathsf{x}]$ and $[\mathsf{y}]$ satisfy $[\mathsf{x} \stackrel{\mathrm{t}}{=} \mathsf{y}]$ then the lists are not allowed to contain them both.

The membership operation $[\mathsf{x} \in_{\mathrm{t}} \mathsf{y}]$[182] is defined thus:

$$[\mathsf{x} \in_{\mathrm{t}} \mathsf{y} \doteq \mathrm{If}(\mathsf{y}^{\mathrm{a}}, \mathsf{x}!\mathsf{F}, \mathrm{If}(\mathsf{x} \stackrel{\mathrm{t}}{=} \mathsf{y}^{\mathrm{h}}, \mathsf{T}, \mathsf{x} \in_{\mathrm{t}} \mathsf{y}^{\mathrm{t}}))]$$

The subset operation $[\mathsf{x} \subseteq_{\mathrm{T}} \mathsf{y}]$[183] is defined thus:

$$[\mathsf{x} \subseteq_{\mathrm{T}} \mathsf{y} \doteq \mathrm{If}(\mathsf{x}^{\mathrm{a}}, \mathsf{y}!\mathsf{T}, \mathrm{If}(\mathsf{x}^{\mathrm{h}} \in_{\mathrm{t}} \mathsf{y}, \mathsf{x}^{\mathrm{t}} \subseteq_{\mathrm{T}} \mathsf{y}, \mathsf{F}))]$$

The set equality operation $[\mathsf{x} \stackrel{\mathrm{T}}{=} \mathsf{y}]$[184] is defined thus:

$$[\mathsf{x} \stackrel{\mathrm{T}}{=} \mathsf{y} \doteq \mathrm{If}(\mathsf{x} \subseteq_{\mathrm{T}} \mathsf{y}, \mathsf{y} \subseteq_{\mathrm{T}} \mathsf{x}, \mathsf{F})]$$

The empty set $[\emptyset]$[185] is defined thus:

---

[180] $[\langle \mathsf{a} \,|\mathsf{x}{:=}\, \mathsf{b}\rangle \stackrel{\mathrm{pyk}}{=}$ "sub $*$ set $*$ to $*$ end sub"]

[181] $[\langle {}^*\mathsf{a} \,|\mathsf{x}{:=}\, \mathsf{b}\rangle \stackrel{\mathrm{pyk}}{=}$ "sub star $*$ set $*$ to $*$ end sub"]

[182] $[\mathsf{x} \in_{\mathrm{t}} \mathsf{y} \stackrel{\mathrm{pyk}}{=}$ "$*$ term in $*$"]

[183] $[\mathsf{x} \subseteq_{\mathrm{T}} \mathsf{y} \stackrel{\mathrm{pyk}}{=}$ "$*$ term subset $*$"]

[184] $[\mathsf{x} \stackrel{\mathrm{T}}{=} \mathsf{y} \stackrel{\mathrm{pyk}}{=}$ "$*$ term set equal $*$"]

[185] $[\emptyset \stackrel{\mathrm{pyk}}{=}$ "the empty set"]

$$[\emptyset \doteq \mathsf{T}]$$

The operation $[\mathsf{x} \cup \{\mathsf{y}\}]$[186] adds the term $[\mathsf{y}]$ to the term set $[\mathsf{x}]$:

$$[\mathsf{x} \cup \{\mathsf{y}\} \doteq \mathrm{If}(\mathsf{y} \in_{\mathrm{t}} \mathsf{x}, \mathsf{x}, \mathsf{y} :: \mathsf{x})]$$

The operation $[\mathsf{x}\backslash\{\mathsf{y}\}]$[187] removes the term $[\mathsf{y}]$ from the term set $[\mathsf{x}]$:

$$[\mathsf{x}\backslash\{\mathsf{y}\} \doteq \mathrm{If}(\mathsf{x}^{\mathrm{a}}, \mathsf{y}!\emptyset, \mathrm{If}(\mathsf{y} \stackrel{\mathrm{t}}{=} \mathsf{x}^{\mathrm{h}}, \mathsf{x}^{\mathrm{t}}, \mathsf{x}^{\mathrm{h}} :: \mathsf{x}^{\mathrm{t}}\backslash\{\mathsf{y}\}))]$$

The operation $[\mathsf{x} \cup \mathsf{y}]$[188] computes the union of the term sets $[\mathsf{x}]$ and $[\mathsf{y}]$:

$$[\mathsf{x} \cup \mathsf{y} \doteq \mathrm{If}(\mathsf{x}^{\mathrm{a}}, \mathsf{y}, \mathsf{x}^{\mathrm{t}} \cup \mathsf{y} \cup \{\mathsf{x}^{\mathrm{h}}\})]$$

6.3.2 Sequents

A Logiweb *sequent* is a triple $[\langle \mathsf{p}, \mathsf{s}, \mathsf{c} \rangle]$ where $[\mathsf{p}]$ and $[\mathsf{s}]$ are term sets and $[\mathsf{c}]$ is a term. In a sequent $[\langle \mathsf{p}, \mathsf{s}, \mathsf{c} \rangle]$ we shall refer to $[\mathsf{p}]$, $[\mathsf{c}]$, and $[\mathsf{c}]$ as *premise*, *side condition*, and *conclusion*, respectively, of the sequent.

A sequent $[\langle \mathsf{p}, \mathsf{s}, \mathsf{c} \rangle]$ represents the statement that if all members of $[\mathsf{p}]$ are provable and if all members of $[\mathsf{s}]$ evaluate to $[\mathsf{T}]$ then $[\mathsf{c}]$ is provable.

Sequents allow to express the same thing in many ways. As an example, the following five sequents all state that if $[\mathcal{A}]$ and $[\mathcal{B}]$ are provable then $[\mathcal{A} \oplus \mathcal{B}]$ is provable:

$$\langle \emptyset, \emptyset, \mathcal{A} \vdash \mathcal{B} \vdash \mathcal{A} \oplus \mathcal{B} \rangle$$
$$\langle \emptyset, \emptyset, \mathcal{B} \vdash \mathcal{A} \vdash \mathcal{A} \oplus \mathcal{B} \rangle$$
$$\langle \emptyset \cup \{\mathcal{A}\}, \emptyset, \mathcal{B} \vdash \mathcal{A} \oplus \mathcal{B} \rangle$$
$$\langle \emptyset \cup \{\mathcal{B}\}, \emptyset, \mathcal{A} \vdash \mathcal{A} \oplus \mathcal{B} \rangle$$
$$\langle \emptyset \cup \{\mathcal{A}\} \cup \{\mathcal{B}\}, \emptyset, \mathcal{A} \oplus \mathcal{B} \rangle$$

The sequent $[\langle \emptyset \cup \{\mathcal{B}\} \cup \{\mathcal{A}\}, \emptyset, \mathcal{A} \oplus \mathcal{B} \rangle]$ is considered equal to the last of the sequents above since the order of elements in term sets is considered unimportant.

From an operational point of view one may think of a sequent $[\langle \mathsf{p}, \mathsf{s}, \mathsf{c} \rangle]$ as a tool makers workshop where $[\mathsf{c}]$ is the anvil where lemmas are forged. $[\mathsf{p}]$ may be seen as a storage room for premises, in which case part of the tool makers work is to bring premises back and forth between anvil and storage room. Alternatively, $[\mathsf{p}]$ may be seen as a backlog of proof obligations. $[\mathsf{s}]$ is like $[\mathsf{p}]$ except that it stores side conditions.

As we shall see, the storage room $[\mathsf{p}]$ will be used for several different things. As an example, when proving $[\mathcal{A} = \mathcal{B} \vdash \mathcal{B} = \mathcal{A}]$ in $[\mathrm{T}'_{\mathrm{E}}]$, $[\mathsf{p}]$ will be used to contain the axiomatic theory $[\mathrm{T}'_{\mathrm{E}}]$ itself, it will be used to contain the premise $[\mathcal{A} = \mathcal{B}]$, and it will be used to contain conclusions of individual lines of the proof.

---

[186] $[\mathsf{x} \cup \{\mathsf{y}\} \stackrel{\mathrm{pyk}}{=}$ "$*$ term plus $*$ end plus"$]$

[187] $[\mathsf{x}\backslash\{\mathsf{y}\} \stackrel{\mathrm{pyk}}{=}$ "$*$ term minus $*$ end minus"$]$

[188] $[\mathsf{x} \cup \mathsf{y} \stackrel{\mathrm{pyk}}{=}$ "$*$ term union $*$"$]$

6.3.3 Sequent equality

The sequent equality operation $[\mathsf{x} \overset{\mathrm{s}}{=} \mathsf{y}]$[189] is defined thus:

$$[\mathsf{x} \overset{\mathrm{s}}{=} \mathsf{y} \doteq \mathrm{If}(\neg \mathsf{x}^2 \overset{\mathrm{t}}{=} \mathsf{y}^2, \mathsf{F}, \mathrm{If}(\mathsf{x}^0 \overset{\mathrm{T}}{=} \mathsf{y}^0, \mathsf{x}^1 \overset{\mathrm{T}}{=} \mathsf{y}^1, \mathsf{F}))]$$

6.3.4 Sequent operations

Logiweb sequent calculus comprises twelve *sequent operations*. We shall refer to a term that is built up from these twelve operations as a *sequent proof*. We shall say that a term $[\mathsf{c}]$ is a *sequent lemma* if there exists a sequent proof which evaluates to $[\langle \emptyset, \emptyset, \mathsf{c} \rangle]$. In the tool makers analogy, the sequent $[\langle \emptyset, \emptyset, \mathsf{c} \rangle]$ represents the situation where $[\mathsf{c}]$ has been forged and the backlogs of remaining work are empty.

We shall refer to a sequent operation as sequent-nulary, sequent-unary, or sequent-binary if it depends on zero, one, or two sequents, respectively. As an example, a binary operation that depends on one sequent and one non-sequent is sequent-unary. Logiweb sequent calculus comprises one sequent-nulary, ten sequent-unary, and one sequent-binary operation. The sole sequent-binary operation of Logiweb sequent calculus is the "cut" operation.

In the tool makers analogy, each sequent-unary operation corresponds to a work process such as moving a premise from anvil to storage room. The sequent-nulary operation is the starting point of any work process, and the sequent-binary operation allows to use the outcome of one work process as a tool in another process.

6.3.5 Proof initiation

The sequent-nulary operation $[\mathsf{x}^{\mathrm{I}}]$[190] performs the following operation:

$$[\mathsf{a}^{\mathrm{I}}] \text{ evaluates to } [\langle \emptyset, \emptyset, \mathsf{a} \vdash \mathsf{a} \rangle]$$

In section 6.5 we introduce a "proof evaluator" which, given a sequent proof, computes the value of the proof. It is the proof evaluator that is going to evaluate $[\mathsf{a}^{\mathrm{I}}]$ as shown above.

As an example of use, $[(\mathcal{A} \oplus \mathcal{B}) \vdash (\mathcal{A} \oplus \mathcal{B})]$ is a sequent lemma because $[(\mathcal{A} \oplus \mathcal{B})^{\mathrm{I}}]$ evaluates to $[\langle \emptyset, \emptyset, (\mathcal{A} \oplus \mathcal{B}) \vdash (\mathcal{A} \oplus \mathcal{B}) \rangle]$. In other words, $[(\mathcal{A} \oplus \mathcal{B})^{\mathrm{I}}$ proves $(\mathcal{A} \oplus \mathcal{B}) \vdash (\mathcal{A} \oplus \mathcal{B})]^{\circ}$, where the "proves" predicate will be defined in Section 6.5.5.

The statement that

$$[(\mathcal{A} \oplus \mathcal{B})^{\mathrm{I}}] \text{ proves } [(\mathcal{A} \oplus \mathcal{B}) \vdash (\mathcal{A} \oplus \mathcal{B})]$$

is just an informal statement that the given sequent proof proves the given term. In contrast, the statement that

[189] $[\mathsf{x} \overset{\mathrm{s}}{=} \mathsf{y} \overset{\mathrm{pyk}}{=}$ "$*$ sequent equal $*$"]

[190] $[\mathsf{x}^{\mathrm{I}} \overset{\mathrm{pyk}}{=}$ "$*$ init"]

$[(\mathcal{A} \oplus \mathcal{B})^{\mathrm{I}}$ proves $(\mathcal{A} \oplus \mathcal{B}) \vdash (\mathcal{A} \oplus \mathcal{B})]^{\circ}$

is a formal one that uses the formally defined [x proves y] relation defined in Section 6.5.5. The latter statement makes Logiweb verify the claim formally.

The statement that

$[(\mathcal{A} \oplus \mathcal{B})^{\mathrm{I}}$ proves $(\mathcal{A} \oplus \mathcal{B}) \vdash (\mathcal{A} \oplus \mathcal{B})]^{\cdot}$

also forces Logiweb to verify the claim formally, but if the claim turned out to fail, the user would merely get the information that that particular claim failed. If [x] does not prove [y] then [x proves y] returns a term which, when typeset, explains why not. For that reason,

$[(\mathcal{A} \oplus \mathcal{B})^{\mathrm{I}}$ proves $(\mathcal{A} \oplus \mathcal{B}) \vdash (\mathcal{A} \oplus \mathcal{B})]^{\circ}$

has the benefit that if the claim fails then the generated error message will be more precise.

6.3.6 Inference introduction

We overload $[\mathsf{x} \vdash \mathsf{y}]$ to denote both an operation on statements and a sequent operation. Ambiguity cannot occur since the proof evaluator will decide what $[\mathsf{x} \vdash \mathsf{y}]$ means on a case by case basis.

$[\mathsf{a} \vdash \mathsf{q}]$ is sequent-unary; it takes a premise [a] and a sequent [q] and moves the premise from storage room to anvil:

$[\mathsf{a} \vdash \langle \mathsf{p}, \mathsf{s}, \mathsf{c} \rangle]$ evaluates to $[\langle \mathsf{p} \backslash \{\mathsf{a}\}, \mathsf{s}, \mathsf{a} \vdash \mathsf{c} \rangle]$

Note that a premise can be moved from storage room to anvil even if there the premise was not present in the storage room to begin with. A premise may be seen as work to be done, and moving a premise that did not exist before corresponds to creation of work to be done from nothing.

As an example of use, $[\mathcal{A} \vdash \mathcal{B}^{\mathrm{I}}$ proves $\mathcal{A} \vdash \mathcal{B} \vdash \mathcal{B}]^{\circ}$.

6.3.7 Endorsement and quantifier introduction

We also overload $[\mathsf{x} \Vdash \mathsf{y}]$ and $[\forall \mathsf{x}\colon \mathsf{y}]$:

$[\mathsf{a} \Vdash \langle \mathsf{p}, \mathsf{s}, \mathsf{c} \rangle]$ evaluates to $[\langle \mathsf{p}, \mathsf{s} \backslash \{\mathsf{a}\}, \mathsf{a} \Vdash \mathsf{c} \rangle]$

$[\forall \mathsf{x}\colon \langle \mathsf{p}, \mathsf{s}, \mathsf{c} \rangle]$ evaluates to $[\langle \mathsf{p}, \mathsf{s}, \forall \mathsf{x}\colon \mathsf{c} \rangle]$ if [x] is a metavariable and is not free in [p] and [s]

As an example of use, $[\forall \mathcal{A}\colon \forall \mathcal{B}\colon \mathcal{A} \vdash \mathcal{B}^{\mathrm{I}}$ proves $\forall \mathcal{A}\colon \forall \mathcal{B}\colon \mathcal{A} \vdash \mathcal{B} \vdash \mathcal{B}]^{\circ}$.

6.3.8 Inference and endorsement elimination

The operation $[\mathsf{x}^{\triangleright}]$[191] eliminates an inference or endorsement:

$[\langle \mathsf{p}, \mathsf{s}, \mathsf{a} \vdash \mathsf{c} \rangle^{\triangleright}]$ evaluates to $[\langle \mathsf{p} \cup \{\mathsf{a}\}, \mathsf{s}, \mathsf{c} \rangle]$

$[\langle \mathsf{p}, \mathsf{s}, \mathsf{a} \Vdash \mathsf{c} \rangle^{\triangleright}]$ evaluates to $[\langle \mathsf{p}, \mathsf{s} \cup \{\mathsf{a}\}, \mathsf{c} \rangle]$

As an example of use, $[\mathcal{A} \vdash \mathcal{B} \vdash \mathcal{A}^{\mathrm{I}\triangleright}$ proves $\mathcal{A} \vdash \mathcal{B} \vdash \mathcal{A}]^{\circ}$.

We shall refer to $[\mathsf{x}^{\triangleright}]$ as the *modus* operation. As we shall see, the inference rule of *modus ponens* will be implemented by the modus operation together with the cut operation described later: the modus operation is used for preparation whereas a cut is used for the actual execution.

Inferences can be eliminated by modus ponens whereas endorsements can be eliminated by either *modus probans* or *verification*. Modus ponens says that if $[\mathsf{x} \vdash \mathsf{y}]$ and if we can prove $[\mathsf{x}]$ then we positivily have $[\mathsf{y}]$. Verification says that if $[\mathsf{x} \Vdash \mathsf{y}]$ and if $[\mathsf{x}]$ evaluates to $[\mathsf{T}]$ then $[\mathsf{y}]$ is verified. Modus probans says that if $[\mathsf{x} \Vdash \mathsf{y}]$ and if we happen to know or assume that $[\mathsf{x}]$ evaluates to $[\mathsf{T}]$ then this knowledge approves $[\mathsf{y}]$. The modus operation implements half of modus ponens or modus probans. Verification is treated in Section 6.3.10.

6.3.9 Quantifier elimination

The operation $[\mathsf{x} \mathbin{@} \mathsf{y}]$[192] eliminates a quantifer:

$[\langle \mathsf{p}, \mathsf{s}, \forall \mathsf{x}\colon \mathsf{c} \rangle \mathbin{@} \mathsf{a}]$ evaluates to $[\langle \mathsf{p}, \mathsf{s}, \langle \mathsf{c} \,|\mathsf{x}{:=}\, \mathsf{a} \rangle \rangle]$ if $[\mathsf{x}]$ is a metavariable and $[\mathsf{a}]$ is free for $[\mathsf{x}]$ in $[\mathsf{c}]$

As an example of use, $[(\forall \mathcal{A}\colon \mathcal{A} = \mathcal{A}) \vdash (\forall \mathcal{A}\colon \mathcal{A} = \mathcal{A})^{\mathrm{I}\triangleright} \mathbin{@} 0$ proves $(\forall \mathcal{A}\colon \mathcal{A} = \mathcal{A}) \vdash 0 = 0]^{\circ}$.

6.3.10 Verification

The operation $[\mathsf{x}^{\mathrm{V}}]$[193]eliminates a side condition by evaluating it and seeing that it is true:

$[\langle \mathsf{p}, \mathsf{s}, \mathsf{a} \Vdash \mathsf{b} \rangle^{\mathrm{V}}]$ evaluates to $[\langle \mathsf{p}, \mathsf{s}, \mathsf{b} \rangle]$ if $[\mathcal{U}^{\mathrm{M}}(\mathcal{E}(\mathsf{a}, \mathsf{T}, \mathsf{c}) \mathbin{`} \mathsf{c})]$ evaluates to $[\mathsf{T}]$ where $[\mathsf{c}]$ is the cache of the page on which the verification operation occurs

As an example of use, $[(\forall \mathcal{A}\colon \mathcal{T}(\mathcal{A}) \Vdash \mathcal{A}) \vdash ((\forall \mathcal{A}\colon \mathcal{T}(\mathcal{A}) \Vdash \mathcal{A})^{\mathrm{I}\triangleright} \mathbin{@} (\mathsf{F} +2{*}\ 2 \approx 5))^{\mathrm{V}}$ proves $(\forall \mathcal{A}\colon \mathcal{T}(\mathcal{A}) \Vdash \mathcal{A}) \vdash \mathsf{F} +2{*}\ 2 \approx 5]^{\circ}$ where

$[\mathcal{T}(\mathsf{x}) \doteq \lambda \mathsf{c}.\mathcal{U}^{\mathrm{M}}(\mathcal{E}(\lceil \mathsf{x} \rceil, \mathsf{T}, \mathsf{c}))]$[194]

In the example above, $[\forall \mathcal{A}\colon \mathcal{T}(\mathcal{A}) \Vdash \mathcal{A}]$ is an axiom scheme that says that any term that evaluates to $[\mathsf{T}]$ is provable.

---

[191] $[\mathsf{x}^{\triangleright} \overset{\text{pyk}}{=}$ "$*$ modus"$]$

[192] $[\mathsf{x} \mathbin{@} \mathsf{y} \overset{\text{pyk}}{=}$ "$*$ at $*$"$]$

[193] $[\mathsf{x}^{\mathrm{V}} \overset{\text{pyk}}{=}$ "$*$ verify"$]$

[194] $[\mathcal{T}(\mathsf{x}) \overset{\text{pyk}}{=}$ "computably true $*$ end true"$]$

6.3.11 Currying

The operations $[\mathsf{x}^+]$[195] and $[\mathsf{x}^-]$[196] add and remove a $[\oplus]$, respectively:

$[\langle \mathsf{p}, \mathsf{s}, \mathsf{a} \vdash \mathsf{b} \vdash \mathsf{c} \rangle^+]$ evaluates to $[\langle \mathsf{p}, \mathsf{s}, (\mathsf{a} \oplus \mathsf{b}) \vdash \mathsf{c} \rangle]$

$[\langle \mathsf{p}, \mathsf{s}, (\mathsf{a} \oplus \mathsf{b}) \vdash \mathsf{c} \rangle^-]$ evaluates to $[\langle \mathsf{p}, \mathsf{s}, \mathsf{a} \vdash \mathsf{b} \vdash \mathsf{c} \rangle]$

$[\mathsf{x}^-]$ and $[\mathsf{x}^+]$ are known as *currying* and *decurrying*, respectively.
As an example of use, $[(\mathcal{A} \vdash \mathcal{B}^{\mathrm{I}})^+$ proves $(\mathcal{A} \oplus \mathcal{B}) \vdash \mathcal{B}]^\circ$.
As another example, $[(\mathcal{A} \oplus \mathcal{B})^{\mathrm{I}-}$ proves $\mathcal{A} \vdash \mathcal{B} \vdash (\mathcal{A} \oplus \mathcal{B})]^\circ$.

6.3.12 Referencing and dereferencing

The operations [x i.e. y][197] and $[\mathsf{x}^*]$[198] change from term to name and vice versa, respectively:

$[\langle \mathsf{p}, \mathsf{s}, \mathsf{c} \rangle$ i.e. $\mathsf{n}]$ evaluates to $[\langle \mathsf{p}, \mathsf{s}, \mathsf{n} \rangle]$ if $[\mathsf{c}]$ is the statement aspect of $[\mathsf{n}]$

$[\langle \mathsf{p}, \mathsf{s}, \mathsf{n} \rangle^*]$ evaluates to $[\langle \mathsf{p}, \mathsf{s}, \mathsf{c} \rangle]$ where $[\mathsf{c}]$ is the statement aspect of $[\mathsf{n}]$

We shall refer to [x i.e. y] and $[\mathsf{x}^*]$ as *referencing* and *dereferencing*, respectively. The asterisk in $[\mathsf{x}^*]$ is inspired by the C programming language.

As an example of use, $[\mathrm{T}'_{\mathrm{E}} \vdash \mathrm{T}'^{\mathrm{I}\triangleright *}_{\mathrm{E}}$ proves $\mathrm{T}'_{\mathrm{E}} \vdash (\mathrm{HeadNil}' \oplus \mathrm{HeadPair}' \oplus \mathrm{Transitivity}' \oplus \mathrm{Contra}')]^\circ$.

As another example, $[(\mathrm{HeadNil}' \oplus \mathrm{HeadPair}' \oplus \mathrm{Transitivity}' \oplus \mathrm{Contra}') \vdash (\mathrm{HeadNil}' \oplus \mathrm{HeadPair}' \oplus \mathrm{Transitivity}' \oplus \mathrm{Contra}')^{\mathrm{I}\triangleright}$ i.e. T'_{E} proves $(\mathrm{HeadNil}' \oplus \mathrm{HeadPair}' \oplus \mathrm{Transitivity}' \oplus \mathrm{Contra}') \vdash \mathrm{T}'_{\mathrm{E}}]^\circ$.

6.3.13 The cut operation

The sequent-binary operation $[\mathsf{x}; \mathsf{y}]$[199] is the well-known *cut* operation. It satisfies:

$[\langle \mathsf{p}_1, \mathsf{s}_1, \mathsf{c}_1 \rangle; \langle \mathsf{p}_2, \mathsf{s}_2, \mathsf{c}_2 \rangle]$ evaluates to $[\langle \mathsf{p}_1 \cup (\mathsf{p}_2 \backslash \{\mathsf{c}_1\}), \mathsf{s}_1 \cup \mathsf{s}_2, \mathsf{c}_2 \rangle]$

As an example of use, let [Remainder][200] denote all of $[\mathrm{T}'_{\mathrm{E}}]$ except [HeadNil′]:

$[\mathrm{Remainder} \doteqdot \mathrm{HeadPair}' \oplus \mathrm{Transitivity}' \oplus \mathrm{Contra}']$

We have that

---

[195] $[\mathsf{x}^+ \overset{\mathrm{pyk}}{=}$ "∗ curry plus"]
[196] $[\mathsf{x}^- \overset{\mathrm{pyk}}{=}$ "∗ curry minus"]
[197] [x i.e. y $\overset{\mathrm{pyk}}{=}$ "∗ id est ∗"]
[198] $[\mathsf{x}^* \overset{\mathrm{pyk}}{=}$ "∗ dereference"]
[199] $[\mathsf{x}; \mathsf{y} \overset{\mathrm{pyk}}{=}$ "∗ cut ∗"]
[200] [Remainder $\overset{\mathrm{pyk}}{=}$ "example remainder"]

$[\mathrm{T}'_{\mathrm{E}}{}^{\mathrm{I}\triangleright}*]$

evaluates to

$[\langle \mathrm{T}'_{\mathrm{E}}, \emptyset, \mathrm{HeadNil}' \oplus \mathrm{Remainder}\rangle]$

and that

$[(\mathrm{HeadNil}' \vdash \mathrm{Remainder} \vdash \mathrm{HeadNil}'^{\mathrm{I}})^{+\triangleright}]$

evaluates to

$[\langle \mathrm{HeadNil}' \oplus \mathrm{Remainder}, \emptyset, \mathrm{HeadNil}'\rangle]$

so

$[\mathrm{T}'_{\mathrm{E}} \vdash (\mathrm{T}'_{\mathrm{E}}{}^{\mathrm{I}\triangleright}*; (\mathrm{HeadNil}' \vdash \mathrm{Remainder} \vdash \mathrm{HeadNil}'^{\mathrm{I}\triangleright})^{+\triangleright})$ proves $\mathrm{T}'_{\mathrm{E}} \vdash \mathrm{HeadNil}']^{\circ}$.

The lemma says that [HeadNil′] is provable in $[\mathrm{T}'_{\mathrm{E}}]$. In general, as one should expect, all rules of a theory are provable in the theory.

6.3.14 Rule lemmas

For a rule $[\mathcal{R}]$ of a theory $[\mathcal{T}]$ we shall refer to $[\mathcal{T} \vdash \mathcal{R}]$ as the *rule lemma* of $[\mathcal{R}]$.

In proofs, one constantly needs rule lemmas. Hence, as soon as one has defined a theory it is convenient to prove the rule lemma of each rule. Once that is done one can forget about the rules and use the rule lemmas instead.

For that reason, one should not waste good names on rules; one should reserve good names for rule lemmas.

6.4 Implementation of the twelve sequent operations

Sequent evaluators for the twelve individual sequent operations of Logiweb sequent calculus are defined in the following sections. See Section 6.3 for explanations of what each operation does.

6.4.1 Evaluation of the init operation

$[\mathcal{S}^{\mathrm{I}}(\mathsf{c},\mathsf{t}) \doteq \mathsf{c}!\langle \emptyset, \emptyset, \mathsf{t}\text{-color}(\mathsf{t}^1 \vdash \mathsf{t}^1)\rangle]$[201]

6.4.2 Evaluation of the modus operation

$[\mathcal{S}^{\triangleright}(\mathsf{c},\mathsf{t}) \doteq \mathcal{S}_1^{\triangleright}(\mathsf{c},\mathsf{t},\mathcal{S}(\mathsf{c},\mathsf{t}^1))]$[202]

[201] $[\mathcal{S}^{\mathrm{I}}(\mathsf{x},\mathsf{y}) \overset{\mathrm{pyk}}{=}$ "seqeval init ∗ term ∗ end eval"]

[202] $[\mathcal{S}^{\triangleright}(\mathsf{x},\mathsf{y}) \overset{\mathrm{pyk}}{=}$ "seqeval modus ∗ term ∗ end eval"]

$[\mathcal{S}_1^{\triangleright}(\mathsf{c},\mathsf{t},\mathsf{q}) \doteq \mathsf{c!t}!$
if q^{E} **then** q **else**
if $\mathsf{q}^2 \overset{\mathrm{r}}{=} \lceil * \vdash * \rceil$ **then** $\langle \mathsf{q}^0 \cup \{\mathsf{q}^{21}\}, \mathsf{q}^1, \mathsf{q}^{22}\rangle$ **else**
if $\mathsf{q}^2 \overset{\mathrm{r}}{=} \lceil * \Vdash * \rceil$ **then** $\langle \mathsf{q}^0, \mathsf{q}^1 \cup \{\mathsf{q}^{21}\}, \mathsf{q}^{22}\rangle$ **else**
error("The modus operation requires the conclusion of its argument to be an inference or an endorsement", t)][203]

6.4.3 Evaluation of the verify operation

$[\mathcal{S}^{\mathrm{E}}(\mathsf{c},\mathsf{t}) \doteq \mathcal{S}_1^{\mathrm{E}}(\mathsf{c},\mathsf{t},\mathcal{S}(\mathsf{c},\mathsf{t}^1))][204]$

$[\mathcal{S}_1^{\mathrm{E}}(\mathsf{c},\mathsf{t},\mathsf{q}) \doteq \mathsf{c!t}!$
if q^{E} **then** q **else**
if $\neg\mathsf{q}^2 \overset{\mathrm{r}}{=} \lceil * \Vdash * \rceil$ **then** error("The verify operation requires the conclusion of its argument to be an endorsement:", t) **else**
if $\mathcal{U}^{\mathrm{M}}(\mathcal{E}(\mathsf{q}^{21},\mathsf{T},\mathsf{c})\,\text{`}\,\mathsf{c})$ **then** $\langle \mathsf{q}^0,\mathsf{q}^1,\mathsf{q}^{22}\rangle$ **else**
error("False side condition:", t)][205]

6.4.4 Evaluation of the decurrying operation

$[\mathcal{S}^{+}(\mathsf{c},\mathsf{t}) \doteq \mathcal{S}_1^{+}(\mathsf{c},\mathsf{t},\mathcal{S}(\mathsf{c},\mathsf{t}^1))][206]$

$[\mathcal{S}_1^{+}(\mathsf{c},\mathsf{t},\mathsf{q}) \doteq \mathsf{c!t}!$
if q^{E} **then** q **else**
if $\mathsf{q}^2 \overset{\mathrm{r}}{=} \lceil * \vdash * \rceil \wedge \mathsf{q}^{22} \overset{\mathrm{r}}{=} \lceil * \vdash * \rceil$ **then**
$\langle \mathsf{q}^0, \mathsf{q}^1, \mathsf{t\text{-}color}((\mathsf{q}^{21} \oplus \mathsf{q}^{221}) \vdash \mathsf{q}^{222})\rangle$ **else**
error("Term; conclusion not fit for decurrying:", t; q^2)][207]

6.4.5 Evaluation of the currying operation

$[\mathcal{S}^{-}(\mathsf{c},\mathsf{t}) \doteq \mathcal{S}_1^{-}(\mathsf{c},\mathsf{t},\mathcal{S}(\mathsf{c},\mathsf{t}^1))][208]$

$[\mathcal{S}_1^{-}(\mathsf{c},\mathsf{t},\mathsf{q}) \doteq \mathsf{c!t}!$
if q^{E} **then** q **else**
if $\mathsf{q}^2 \overset{\mathrm{r}}{=} \lceil * \vdash * \rceil \wedge \mathsf{q}^{21} \overset{\mathrm{r}}{=} \lceil * \oplus * \rceil$ **then**
$\langle \mathsf{q}^0, \mathsf{q}^1, \mathsf{t\text{-}color}(\mathsf{q}^{211} \vdash \mathsf{q}^{212} \vdash \mathsf{q}^{22})\rangle$ **else**
error("Term; conclusion not fit for decurrying:", t; q^2)][209]

6.4.6 Evaluation of the dereferencing operation

$[\mathcal{S}^{*}(\mathsf{c},\mathsf{t}) \doteq \mathcal{S}_1^{*}(\mathsf{c},\mathsf{t},\mathcal{S}(\mathsf{c},\mathsf{t}^1))][210]$

---

[203] $[\mathcal{S}_1^{\triangleright}(\mathsf{x},\mathsf{y},\mathsf{z}) \overset{\mathrm{pyk}}{=}$ "seqeval modus one $*$ term $*$ sequent $*$ end eval"]
[204] $[\mathcal{S}^{\mathrm{E}}(\mathsf{x},\mathsf{y}) \overset{\mathrm{pyk}}{=}$ "seqeval verify $*$ term $*$ end eval"]
[205] $[\mathcal{S}_1^{\mathrm{E}}(\mathsf{x},\mathsf{y},\mathsf{z}) \overset{\mathrm{pyk}}{=}$ "seqeval verify one $*$ term $*$ sequent $*$ end eval"]
[206] $[\mathcal{S}^{+}(\mathsf{x},\mathsf{y}) \overset{\mathrm{pyk}}{=}$ "sequent eval plus $*$ term $*$ end eval"]
[207] $[\mathcal{S}_1^{+}(\mathsf{x},\mathsf{y},\mathsf{z}) \overset{\mathrm{pyk}}{=}$ "seqeval plus one $*$ term $*$ sequent $*$ end eval"]
[208] $[\mathcal{S}^{-}(\mathsf{x},\mathsf{y}) \overset{\mathrm{pyk}}{=}$ "seqeval minus $*$ term $*$ end eval"]
[209] $[\mathcal{S}_1^{-}(\mathsf{x},\mathsf{y},\mathsf{z}) \overset{\mathrm{pyk}}{=}$ "seqeval minus one $*$ term $*$ sequent $*$ end eval"]
[210] $[\mathcal{S}^{*}(\mathsf{x},\mathsf{y}) \overset{\mathrm{pyk}}{=}$ "seqeval deref $*$ term $*$ end eval"]

$[\mathcal{S}_1^*(\mathsf{c},\mathsf{t},\mathsf{q}) \doteq \mathsf{c}!\mathsf{t}!$
if q^{E} **then** q **else**
$\mathcal{S}_2^*(\mathsf{c},\mathsf{t},\mathsf{q},\mathbf{aspect}(<\mathrm{stmt}>,\mathsf{q}^2,\mathsf{c}))]$[211]

$[\mathcal{S}_2^*(\mathsf{c},\mathsf{t},\mathsf{q},\mathsf{d}) \doteq \mathsf{c}!\mathsf{t}!\mathsf{q}!$
if d **then** error("Dereferencing construct that has no statement def:", t) **else**
$\langle \mathsf{q}^0, \mathsf{q}^1, \mathsf{d}^3 \rangle]$[212]

6.4.7 Evaluation of quantifier elimination

$[\mathcal{S}^{@}(\mathsf{c},\mathsf{t}) \doteq \mathcal{S}_1^{@}(\mathsf{c},\mathsf{t},\mathcal{S}(\mathsf{c},\mathsf{t}^1))]$[213]

$[\mathcal{S}_1^{@}(\mathsf{c},\mathsf{t},\mathsf{q}) \doteq \mathsf{c}!\mathsf{t}!$
if q^{E} **then** q **else**
if $\neg\mathsf{q}^2 \stackrel{\mathrm{r}}{=} \lceil \forall *: * \rceil$ **then** error("Quantifier elimination requires the conclusion of its argument to be a quantifier:", t) **else**
if $\neg\mathsf{t}^2$ free for q^{21} in q^{22} **then** error("Quantifier elimination leads to variable clash:", t) **else**
$\langle \mathsf{q}^0, \mathsf{q}^1, \langle \mathsf{q}^{22} \,|\mathsf{q}^{21} := \mathsf{t}^2 \rangle \rangle]$[214]

6.4.8 Evaluation of inference introduction

$[\mathcal{S}^{\vdash}(\mathsf{c},\mathsf{t}) \doteq \mathcal{S}_1^{\vdash}(\mathsf{c},\mathsf{t},\mathsf{t}^1,\mathcal{S}(\mathsf{c},\mathsf{t}^2))]$[215]

$[\mathcal{S}_1^{\vdash}(\mathsf{c},\mathsf{t},\mathsf{p},\mathsf{q}) \doteq \mathsf{c}!\mathsf{t}!\mathsf{p}!$
if q^{E} **then** q **else**
$\langle \mathsf{q}^0 \backslash \{\mathsf{p}\}, \mathsf{q}^1, \mathsf{t}\text{-color}(\mathsf{p} \vdash \mathsf{q}^2) \rangle]$[216]

6.4.9 Evaluation of endorsement introduction

$[\mathcal{S}^{\Vdash}(\mathsf{c},\mathsf{t}) \doteq \mathcal{S}_1^{\Vdash}(\mathsf{c},\mathsf{t},\mathsf{t}^1,\mathcal{S}(\mathsf{c},\mathsf{t}^2))]$[217]

$[\mathcal{S}_1^{\Vdash}(\mathsf{c},\mathsf{t},\mathsf{p},\mathsf{q}) \doteq \mathsf{c}!\mathsf{t}!\mathsf{p}!$
if q^{E} **then** q **else**
$\langle \mathsf{q}^0, \mathsf{q}^1 \backslash \{\mathsf{p}\}, \mathsf{t}\text{-color}(\mathsf{p} \Vdash \mathsf{q}^2) \rangle]$[218]

6.4.10 Evaluation of the referencing operation

$[\mathcal{S}^{\mathrm{i.e.}}(\mathsf{c},\mathsf{t}) \doteq \mathcal{S}_1^{\mathrm{i.e.}}(\mathsf{c},\mathsf{t},\mathsf{t}^2,\mathcal{S}(\mathsf{c},\mathsf{t}^1))]$[219]

---

[211] $[\mathcal{S}_1^*(\mathsf{x},\mathsf{y},\mathsf{z}) \stackrel{\mathrm{pyk}}{=}$ "seqeval deref one * term * sequent * end eval"]

[212] $[\mathcal{S}_2^*(\mathsf{c},\mathsf{t},\mathsf{q},\mathsf{d}) \stackrel{\mathrm{pyk}}{=}$ "seqeval deref two * term * sequent * def * end eval"]

[213] $[\mathcal{S}^{@}(\mathsf{x},\mathsf{y}) \stackrel{\mathrm{pyk}}{=}$ "seqeval at * term * end eval"]

[214] $[\mathcal{S}_1^{@}(\mathsf{c},\mathsf{t},\mathsf{q}) \stackrel{\mathrm{pyk}}{=}$ "seqeval at one * term * sequent * end eval"]

[215] $[\mathcal{S}^{\vdash}(\mathsf{x},\mathsf{y}) \stackrel{\mathrm{pyk}}{=}$ "seqeval infer * term * end eval"]

[216] $[\mathcal{S}_1^{\vdash}(\mathsf{x},\mathsf{y},\mathsf{z},\mathsf{u}) \stackrel{\mathrm{pyk}}{=}$ "seqeval infer one * term * premise * sequent * end eval"]

[217] $[\mathcal{S}^{\Vdash}(\mathsf{x},\mathsf{y}) \stackrel{\mathrm{pyk}}{=}$ "seqeval endorse * term * end eval"]

[218] $[\mathcal{S}_1^{\Vdash}(\mathsf{x},\mathsf{y},\mathsf{z},\mathsf{u}) \stackrel{\mathrm{pyk}}{=}$ "seqeval endorse one * term * side * sequent * end eval"]

[219] $[\mathcal{S}^{\mathrm{i.e.}}(\mathsf{x},\mathsf{y}) \stackrel{\mathrm{pyk}}{=}$ "seqeval est * term * end eval"]

$[\mathcal{S}_1^{\text{i.e.}\cdot}(\mathsf{c},\mathsf{t},\mathsf{a},\mathsf{q}) \doteq \mathsf{c}!\mathsf{t}!\mathsf{a}!$
$\text{If}(\mathsf{q}^{\text{E}},\mathsf{q},\mathcal{S}_2^{\text{i.e.}\cdot}(\mathsf{c},\mathsf{t},\mathsf{a},\mathsf{q},\textbf{aspect}(<\text{stmt}>,\mathsf{a},\mathsf{c})))]$[220]

$[\mathcal{S}_2^{\text{i.e.}\cdot}(\mathsf{c},\mathsf{t},\mathsf{a},\mathsf{q},\mathsf{d}) \doteq \mathsf{c}!\mathsf{t}!\mathsf{a}!\mathsf{q}!$
if d **then** error("Referencing construct that has no statement def:", t) **else**
if $\neg\mathsf{d}^3 \overset{\mathsf{t}}{=} \mathsf{q}^2$ **then** error("Reference; conclusion do not match:", $\mathsf{a};\mathsf{q}^2$) **else**
$\langle\mathsf{q}^0,\mathsf{q}^1,\mathsf{a}\rangle]$[221]

6.4.11 Evaluation of quantifier introduction

$[\mathcal{S}^{\forall}(\mathsf{c},\mathsf{t}) \doteq \mathcal{S}_1^{\forall}(\mathsf{c},\mathsf{t},\mathsf{t}^1,\mathcal{S}(\mathsf{c},\mathsf{t}^2))]$[222]

$[\mathcal{S}_1^{\forall}(\mathsf{c},\mathsf{t},\mathsf{v},\mathsf{q}) \doteq \mathsf{c}!\mathsf{t}!\mathsf{v}!$
if q^{E} **then** q **else**
if $\neg\mathsf{v}^{\mathcal{V}}$ **then** error("Metageneralization over non-metavariable:", t) **else**
if v free in q^0 **then** error("Metageneralization over metavariable that occurs free in some premise:", t) **else**
if v free in q^1 **then** error("Metageneralization over metavariable that occurs free in some side condition:", t) **else**
$\langle\mathsf{q}^0,\mathsf{q}^1,\text{t-color}(\forall\mathsf{v}{:}\,\mathsf{q}^2)\rangle]$[223]

6.4.12 Evaluation of the cut operation

$[\mathcal{S}^{:}(\mathsf{c},\mathsf{t}) \doteq \mathcal{S}_1^{:}(\mathsf{c},\mathsf{t},\mathcal{S}(\mathsf{c},\mathsf{t}^1))]$[224]

$[\mathcal{S}_1^{:}(\mathsf{c},\mathsf{t},\mathsf{p}) \doteq \mathsf{c}!\mathsf{t}!$
$\text{If}(\mathsf{p}^{\text{E}},\mathsf{p},\mathcal{S}_2^{:}(\mathsf{c},\mathsf{t},\mathsf{p},\mathcal{S}(\mathsf{c},\mathsf{t}^2)))]$[225]

$[\mathcal{S}_2^{:}(\mathsf{c},\mathsf{t},\mathsf{p},\mathsf{q}) \doteq \mathsf{c}!\mathsf{t}!\mathsf{p}!$
$\text{If}(\mathsf{q}^{\text{E}},\mathsf{q},\langle\mathsf{p}^0 \cup (\mathsf{q}^0 \backslash \{\mathsf{p}^2\}),\mathsf{p}^1 \cup \mathsf{q}^1,\mathsf{q}^2\rangle)]$[226]

6.5 The proof evaluator

6.5.1 Coloring

$[\mathsf{x} \vdash \mathsf{y}]$ is self-evaluating. For that reason, $[\lceil\mathsf{x}\rceil \vdash \lceil\mathsf{y}\rceil]$ and $[\lceil\mathsf{x} \vdash \mathsf{y}\rceil]$ are equal terms except for debugging information.

The debugging information of the root of $[\lceil\mathsf{x}\rceil \vdash \lceil\mathsf{y}\rceil]$ equals $[\mathsf{T}]$ which represents no debugging information. For that reason we shall call the root *uncolored*.

---

[220] $[\mathcal{S}_1^{\text{i.e.}\cdot}(\mathsf{x},\mathsf{y},\mathsf{z},\mathsf{u}) \overset{\text{pyk}}{=}$ "seqeval est one * term * name * sequent * end eval"]

[221] $[\mathcal{S}_2^{\text{i.e.}\cdot}(\mathsf{c},\mathsf{t},\mathsf{a},\mathsf{q},\mathsf{d}) \overset{\text{pyk}}{=}$ "seqeval est two * term * name * sequent * def * end eval"]

[222] $[\mathcal{S}^{\forall}(\mathsf{x},\mathsf{y}) \overset{\text{pyk}}{=}$ "seqeval all * term * end eval"]

[223] $[\mathcal{S}_1^{\forall}(\mathsf{c},\mathsf{t},\mathsf{v},\mathsf{q}) \overset{\text{pyk}}{=}$ "seqeval all one * term * variable * sequent * end eval"]

[224] $[\mathcal{S}^{:}(\mathsf{x},\mathsf{y}) \overset{\text{pyk}}{=}$ "seqeval cut * term * end eval"]

[225] $[\mathcal{S}_1^{:}(\mathsf{c},\mathsf{t},\mathsf{p}) \overset{\text{pyk}}{=}$ "seqeval cut one * term * forerunner * end eval"]

[226] $[\mathcal{S}_2^{:}(\mathsf{x},\mathsf{y},\mathsf{z},\mathsf{u}) \overset{\text{pyk}}{=}$ "seqeval cut two * term * forerunner * sequent * end eval"]

Proper debugging information comprises a list of cardinals where the last cardinal in the list is the reference of the page on which the given node occurs. Hence, proper debugging information always differs from [T]. We shall refer to nodes whose debugging information differs from [T] as *colored*.

Whenever terms are generated using self-evaluating constructs, there will be uncolored nodes near the root of the generated term. The binary [a-color(t)] operation returns the term [t] but copies the debugging information from the root of [a] into uncolored nodes of [t]. We shall refer to this artistic touch as *coloring*.

The coloring operation merely descends into the tree until it meets nodes with proper debugging information, so the operation will miss uncolored nodes hidden deeply inside the tree.

The coloring operation is defined thus:

$$[\mathsf{a\text{-}color}(\mathsf{t}) \doteq \mathsf{t}^{\mathrm{d}} \begin{cases} (\mathsf{t}^{\mathrm{r}} :: \mathsf{t}^{\mathrm{i}} :: \mathsf{a}^{\mathrm{d}}) :: \mathsf{a\text{-}color}^*(\mathsf{t}^{\mathrm{t}}) \\ \mathsf{a!t} \end{cases}]^{227}$$

$$[\mathsf{a\text{-}color}^*(\mathsf{t}) \doteq \mathsf{t} \begin{cases} \mathsf{a!T} \\ \mathsf{a\text{-}color}(\mathsf{t}^{\mathrm{h}}) :: \mathsf{a\text{-}color}^*(\mathsf{t}^{\mathrm{t}}) \end{cases}]^{228}$$

6.5.2 Error message generation

The following macro expands into a construct that returns an error message. The first argument is supposed to be a string and the value of the second is supposed to be a term.

When used, the construct generates an error message consisting of the string followed by a newline followed by the term, which is of course rather limited. Whenever it is convenient to include more than one term in a message we take the rather crude approach to combine the messages by the cut operation! The cut operation is self-evaluating and, when typeset, puts a semicolon between its arguments, which is close to tolerable.

The error generation macro simply quotes the string argument and leaves the actual generation of the error message to an auxiliary function:

$$[\mathsf{error}(\mathsf{m},\mathsf{t}) \doteq\doteq \mathsf{error}_2(\lceil \mathsf{m} \rceil, \mathsf{t})]^{229}$$

The following function generates an error message from a message [m] and a term [t].

[m] is supposed to be a unicode string with a unicode start of text in the root. That start of text character is removed by computing $[\mathsf{m}^1]$ to drop the "double quotes" from the string, but a $[(\mathsf{x})^{\mathsf{t}}]$ ensures that the unquoted string is still typeset as a string eventually.

---

[227] $[\mathsf{x\text{-}color}(\mathsf{y}) \overset{\text{pyk}}{=}$ "$*$ color $*$ end color"]

[228] $[\mathsf{x\text{-}color}^*(\mathsf{y}) \overset{\text{pyk}}{=}$ "$*$ color star $*$ end color"]

[229] $[\mathsf{error}(\mathsf{m},\mathsf{t}) \overset{\text{pyk}}{=}$ "error $*$ term $*$ end error"]

The text is glued together with a string that merely contains a newline character and which is treated similarly. Furthermore, the text plus newline character are glued together with the term [t].

The text, newline character, and term are glued together with the [x then y]. The $[(\mathsf{x})^{\mathbf{t}}]$ and [x then y] are invisible when typeset using their tex aspects and, hence, do not exhibit themselves in the final error message. These two constructs are visible in the right hand side of the definition below and in several places above because they are made visible using $[(\mathsf{x})^{\mathbf{v}}]$.

A [t-color(x)] copies the debugging information in the root of [t] to nodes in the error message that are generated by $[(\mathsf{x})^{\mathbf{t}}]$ and [x then y] and have no debugging information of their own. The definition reads:

$[\mathrm{error}_2(\mathsf{m},\mathsf{t}) \doteq \mathsf{t}\text{-color}((\mathsf{m}^1)^{\mathbf{t}} \text{ then } (\lceil \text{“} \\ \text{”} \rceil^1)^{\mathbf{t}} \text{ then } \mathsf{t})]$[230]

6.5.3 Error recognition

The sequent evaluator returns a sequent or an error message. For obvious reasons, we shall need the ability to tell whether a return value is the one or the other.

Error messages are terms that can be typeset by the Logiweb system. When recognizing error messages, we assume that they are generated by the error message generator of the previous section or similar means that put a [x then y] operator in the root of the message. That particular operator is harmless to have in the root of error messages because typesetting of [x then y] results in [x] and [y] being typeset and then concatenated. Note that [x then y] above reads $[(\mathsf{x} \text{ then } \mathsf{y})^{\mathbf{v}}]$ in the source of the present page, otherwise the “then” would have been invisible as in [xy].

The predicate below is true if the argument is an error message and false if the argument is a sequent. It is also false in many other cases as we shall benefit from later.

$[\mathsf{x}^{\mathrm{E}} \doteq \mathsf{x} \stackrel{\mathrm{r}}{=} \lceil \mathsf{x} \text{ then } \mathsf{y} \rceil]$[231]

6.5.4 Sequent evaluation

The sequent evaluation function $[\mathcal{S}(\mathsf{c},\mathsf{t})]$ evaluates the sequent proof [t] in the context defined by the cache [c]. The result is a sequent or an error message. If it is a sequent, then the proof is valid and the sequent is the conclusion of the proof.

The sequent evaluation function recognises the twelve sequent operations of Logiweb sequent calculus and turns against any other operations in [t].

$[\mathcal{S}(\mathsf{c},\mathsf{t}) \doteq \mathsf{c}!$
if t^{E} **then** t **else**

[230] $[\mathrm{error}_2(\mathsf{m},\mathsf{t}) \stackrel{\mathrm{pyk}}{=}$ “error two $*$ term $*$ end error”]

[231] $[\mathsf{x}^{\mathrm{E}} \stackrel{\mathrm{pyk}}{=}$ “$*$ is error”]

if $\mathsf{t} \stackrel{\mathrm{r}}{=} \lceil *^{\mathrm{I}} \rceil$ **then** $\mathcal{S}^{\mathrm{I}}(\mathsf{c},\mathsf{t})$ **else**
if $\mathsf{t} \stackrel{\mathrm{r}}{=} \lceil *^{\triangleright} \rceil$ **then** $\mathcal{S}^{\triangleright}(\mathsf{c},\mathsf{t})$ **else**
if $\mathsf{t} \stackrel{\mathrm{r}}{=} \lceil *^{\mathrm{V}} \rceil$ **then** $\mathcal{S}^{\mathrm{E}}(\mathsf{c},\mathsf{t})$ **else**
if $\mathsf{t} \stackrel{\mathrm{r}}{=} \lceil *^{+} \rceil$ **then** $\mathcal{S}^{+}(\mathsf{c},\mathsf{t})$ **else**
if $\mathsf{t} \stackrel{\mathrm{r}}{=} \lceil *^{-} \rceil$ **then** $\mathcal{S}^{-}(\mathsf{c},\mathsf{t})$ **else**
if $\mathsf{t} \stackrel{\mathrm{r}}{=} \lceil *^{*} \rceil$ **then** $\mathcal{S}^{*}(\mathsf{c},\mathsf{t})$ **else**
if $\mathsf{t} \stackrel{\mathrm{r}}{=} \lceil * \mathbin{@} * \rceil$ **then** $\mathcal{S}^{@}(\mathsf{c},\mathsf{t})$ **else**
if $\mathsf{t} \stackrel{\mathrm{r}}{=} \lceil * \vdash * \rceil$ **then** $\mathcal{S}^{\vdash}(\mathsf{c},\mathsf{t})$ **else**
if $\mathsf{t} \stackrel{\mathrm{r}}{=} \lceil * \Vdash * \rceil$ **then** $\mathcal{S}^{\Vdash}(\mathsf{c},\mathsf{t})$ **else**
if $\mathsf{t} \stackrel{\mathrm{r}}{=} \lceil * \text{ i.e. } * \rceil$ **then** $\mathcal{S}^{\text{i.e.}}(\mathsf{c},\mathsf{t})$ **else**
if $\mathsf{t} \stackrel{\mathrm{r}}{=} \lceil \forall * : * \rceil$ **then** $\mathcal{S}^{\forall}(\mathsf{c},\mathsf{t})$ **else**
if $\mathsf{t} \stackrel{\mathrm{r}}{=} \lceil *;* \rceil$ **then** $\mathcal{S}^{;}(\mathsf{c},\mathsf{t})$ **else**
error("Unknown sequent operator:", t)[232]

6.5.5 Lemma verification

The macro [p proves t] claims that [p] proves [t]:

[p proves t $\ddot{=}$ proof($\lceil \mathsf{p} \rceil, \lceil \mathsf{t} \rceil$, self)][233]

As an example, the following verifies $[2 \vdash 2]$:

$[2^{\mathrm{I}}$ proves $2 \vdash 2]^{\circ}$

Perturbing the conclusion to e.g. $[2 \vdash 3]$ provokes the diagnose aspect of the page symbol to be set to a term which, if typeset using the tex aspect, generates an appropriate error message.

[proof(p, t, c)] equals [⊤] if the proof [p] proves the sequent lemma [t] in the context defined by the cache [c]. Otherwise, [proof(p, t, c)] returns an error message:

[proof(p, t, c) $\doteq$ proof$_2(\mathcal{S}(\mathsf{c},\mathsf{p}),\mathsf{t})$][234]

[proof$_2$(q, t)] equals [⊤] if [q] is a sequent which proves the sequent lemma [t]. Otherwise, [proof$_2$(q, t)] returns an error message:

[proof$_2(\mathsf{q},\mathsf{t}) \doteq \mathsf{t}!$
if q^{E} **then** q **else**
if $\neg\mathsf{q}^{0}$ **then** error("Proof has at least one unresolved premise. Lemma; premise reads:", $\mathsf{t};\mathsf{q}^{0\mathrm{h}}$) **else**
if $\neg\mathsf{q}^{1}$ **then** error("Proof has at least one unresolved side condition. Lemma; condition reads:", $\mathsf{t};\mathsf{q}^{1\mathrm{h}}$) **else**
if $\mathsf{q}^{2} \stackrel{\mathrm{t}}{=} \mathsf{t}$ **then** ⊤ **else**

[232] $[\mathcal{S}(\mathsf{x},\mathsf{y}) \stackrel{\mathrm{pyk}}{=}$ "sequent eval ∗ term ∗ end eval"]
[233] $[\mathsf{x}$ proves $\mathsf{y} \stackrel{\mathrm{pyk}}{=}$ "∗ proves ∗"]
[234] $[\mathrm{proof}(\mathsf{p},\mathsf{t},\mathsf{c}) \stackrel{\mathrm{pyk}}{=}$ "proof ∗ term ∗ cache ∗ end proof"]

error(“Lemma does not match conclusion. Lemma; conclusion reads:”, $\mathsf{t}; \mathsf{q}^2$)][235]

6.6 The verifier

6.6.1 Conjunction membership

The verifier allows proofs to refer to previously proved lemmas. Such previously proved lemmas may occur on the same page as the proof or in transitively referenced pages.

When a lemma on another page is referenced, the verifier needs to ensure that that other page has been properly checked. The verifier does so by looking up the diagnose aspect of the referenced page to see that the referenced page is correct. Furthermore, the verifier looks up the claim aspect of the referenced page to see that the claim is a conjunction that includes the verifier itself.

The above verification check for referenced pages is fast because the verifier itself is not invoked on the page. Rather, the verifier just ensures that the verifier was invoked successfully when the page was loaded.

Furthermore, the verification check is flexible in that the verifier only requires referenced pages to be checked by the verifier if those other pages actually contribute to proofs. This allows pages that contain proofs to reference pages that are unrelated to proofs such as pages that define computer programs, formatting constructs, fonts, or whatever.

The verifier is also flexible in that it allows referenced pages to be checked by any number of other checkers alongside the verifier itself. That allows the verifier to coexist with other proof checkers and all sorts of other checkers. As an example, one may decide to impose strong typing on a page using some type checker without affecting the ability to check proofs.

To implement the above verification check we need a relation $[\mathsf{x} \in_\mathsf{c} \mathsf{y}]$ which is true iff the term $[\mathsf{x}]$ belongs to the conjunction $[\mathsf{y}]$:

$$[\mathsf{x} \in_\mathsf{c} \mathsf{y} \doteq \mathsf{y} \overset{\mathsf{r}}{=} \lceil \mathsf{x} \wedge_\mathsf{c} \mathsf{y} \rceil \begin{cases} \mathrm{If}(\mathsf{x} \in_\mathsf{c} \mathsf{y}^1, \mathsf{T}, \mathsf{x} \in_\mathsf{c} \mathsf{y}^2) \\ \mathsf{x} \overset{\mathsf{t}}{=} \mathsf{y} \end{cases}][236]$$

Furthermore, we need a function $[\mathrm{claims}(\mathsf{t}, \mathsf{c}, \mathsf{r})]$ which is true if the term $[\mathsf{t}]$ belongs to the potentially inherited claim of page $[\mathsf{r}]$ according to the cache $[\mathsf{c}]$:

$$[\mathrm{claims}(\mathsf{t}, \mathsf{c}, \mathsf{r}) \doteq \mathrm{If}(\mathrm{claims}_2(\mathsf{t}, \mathsf{c}, \mathsf{r}), \mathsf{T}, \mathrm{claims}_2(\mathsf{t}, \mathsf{c}, \mathsf{c}[\mathsf{r}][\text{"bibliography"}]^1))][237]$$

$$[\mathrm{claims}_2(\mathsf{t}, \mathsf{c}, \mathsf{r}) \doteq \mathrm{If}(\neg \mathsf{r}^\mathsf{c}, \mathsf{t!c!F}, \mathsf{t} \in_\mathsf{c} \mathsf{c}[\mathsf{r}][\text{"codex"}][\mathsf{r}][0][0][\text{"claim"}]^3)][238]$$

[235] $[\mathrm{proof}_2(\mathsf{q}, \mathsf{t}) \overset{\text{pyk}}{=} \text{“proof two} * \text{term} * \text{end proof”}]$

[236] $[\mathsf{x} \in_\mathsf{c} \mathsf{y} \overset{\text{pyk}}{=} \text{“}* \text{ claim in } *\text{”}]$

[237] $[\mathrm{claims}(\mathsf{t}, \mathsf{c}, \mathsf{r}) \overset{\text{pyk}}{=} \text{“claims} * \text{cache} * \text{ref} * \text{end claims”}]$

[238] $[\mathrm{claims}_2(\mathsf{t}, \mathsf{c}, \mathsf{r}) \overset{\text{pyk}}{=} \text{“claims two} * \text{cache} * \text{ref} * \text{end claims”}]$

6.6.2 The proof aspect

The "proves" predicate in Section 6.5.5 is useful for testing the validity of a single, stand-alone proof which proves some lemma from scratch without reference to any auxilliary lemmas.

For more complex proofs, we introduce a *proof aspect* and make the convention that whenever a construct has a proof aspect, then the construct should also have a statement aspect and, furthermore, the proof aspect should be a proof of the statement aspect. We make the proof aspect self-evaluating and use [proof] to denote it:

$$[\text{<proof>} \doteq \lceil \text{<proof>} \rceil]^{239}$$

$$[\text{proof} \overset{\text{msg}}{=} \text{<proof>}]^{240}$$

From the point of view of Logiweb, lemmas and proofs are just definitions:

$$[[\textbf{Lemma}\ \mathsf{x}\colon \mathsf{y}] \doteq [\mathsf{x} \overset{\text{stmt}}{\longrightarrow} \mathsf{y}]]^{241}$$

$$[[\textbf{Proof of}\ \mathsf{x}\colon \mathsf{y}] \doteq [\mathsf{x} \overset{\text{proof}}{\longrightarrow} \mathsf{y}]]^{242}$$

The following construct provides a reader friendly way of stating that a statement [z] is provable in a theory [x] and giving the conjecture the name [y].

$$[[\mathsf{x}\ \textbf{lemma}\ \mathsf{y}\colon \mathsf{z}] \doteq [\mathsf{y} \overset{\text{stmt}}{=} \mathsf{x} \vdash \mathsf{z}]]^{243}$$

As an example of use,

$$[\mathsf{T}'_{\mathrm{E}}\ \textbf{lemma}\ \text{HeadNil}''\colon \mathsf{T}^{\mathrm{h}} = \mathsf{T}]^{244}$$

conjectures that $[\mathsf{T}^{\mathrm{h}} = \mathsf{T}]$ is provable in $[\mathsf{T}'_{\mathrm{E}}]$.

The following construct provides a reader friendly way of stating that a statement [z] is disprovable in a theory [x] and giving the anticonjecture the name [y].

$$[[\mathsf{x}\ \textbf{antilemma}\ \mathsf{y}\colon \mathsf{z}] \doteq [\mathsf{x}\ \textbf{lemma}\ \mathsf{y}\colon \mathsf{z} \vdash \perp\!\!\!\perp]]^{245}$$

As an example of use,

$$[\mathsf{T}'_{\mathrm{E}}\ \textbf{antilemma}\ \text{Contra}''\colon \mathsf{T} :: \mathsf{T} = \mathsf{T}]^{246}$$

conjectures that $[\mathsf{T} :: \mathsf{T} = \mathsf{T}]$ is disprovable in $[\mathsf{T}'_{\mathrm{E}}]$.

---

[239] [<proof> $\overset{\text{pyk}}{=}$ "the proof aspect"]

[240] [proof $\overset{\text{pyk}}{=}$ "proof"]

[241] [[**Lemma** x: y] $\overset{\text{pyk}}{=}$ "lemma ∗ says ∗ end lemma"]

[242] [[**Proof of** x: y] $\overset{\text{pyk}}{=}$ "proof of ∗ reads ∗ end proof"]

[243] [[x **lemma** y: z] $\overset{\text{pyk}}{=}$ "in theory ∗ lemma ∗ says ∗ end lemma"]

[244] [HeadNil″ $\overset{\text{pyk}}{=}$ "example axiom lemma primed"]

[245] [[x **antilemma** y: z] $\overset{\text{pyk}}{=}$ "in theory ∗ antilemma ∗ says ∗ end antilemma"]

[246] [Contra″ $\overset{\text{pyk}}{=}$ "contraexample lemma primed"]

6.6.3 Verifier

The [verifier] checks the correctness of all proofs on a page. It is suited to appear in a conjunction that makes up the claim of a page. The verifier ignores the macro expanded tree [t] and merely uses the cache [c].

$$[\text{verifier} \doteq \lambda t.\lambda c.\mathcal{V}_1(c)]^{247}$$

$[\mathcal{V}_1(c)]$ extracts the reference [r] of the page to be checked from the cache [c] of the page. Then it extracts the subcodex [x] of all domestic definition of the page and uses $[\mathcal{V}_2(c,x)]$ to evaluate the array of proofs into an array [p] of sequents. Then it uses $[\mathcal{V}_3(c,r,p,d)]$ to verify the correctness of all proofs.

$$\begin{array}{l}
[\mathcal{V}_1(c) \doteq \\
\textbf{let } r = c[0] \textbf{ in} \\
\textbf{let } x = c[r]["\text{codex}"][r] \textbf{ in} \\
\textbf{let } p = \mathcal{V}_2(c,x) \textbf{ in} \\
\textbf{let } d = \mathcal{V}_3(c,r,p,\mathsf{T}) \textbf{ in} \\
\textbf{if } \neg d \textbf{ then } d \textbf{ else} \\
\textbf{let } i = \mathcal{V}_5(c,r,p,p) \textbf{ in} \\
\textbf{if } \neg i^c \textbf{ then } \mathsf{T} \textbf{ else} \\
\text{error}(\text{“Circular proof. Circle includes:”}, p[i]^{0h})]^{248}
\end{array}$$

$[\mathcal{V}_2(c,p)]$ evaluates all proofs in the subcodex [p] and returns the result as a one-dimensional array of sequents (indexed by the identifier of each proof). $[\mathcal{V}_2(c,p)]$ is not particularly efficient since it continues to evaluate all proofs even if one of the proofs returns an error message.

For each symbol in the subcodex, $[\mathcal{V}_2(c,p)]$ extracts the proof definition [d] (i.e. the right hand side of the [<proof>] aspect) of the symbol. If the proof aspect exists (i.e. differs from [T]) then $[\mathcal{V}_2(c,p)]$ uses the evaluator $[\mathcal{E}(d,\mathsf{T},c)]$ to compute the value of [d], applies the result to the cache [c] in order to give access to the cache from proof tactics inside [d], and then uses the sequent evaluator $[\mathcal{S}(c,t)]$ to evaluate the proof into a sequent.

$$\begin{array}{l}
[\mathcal{V}_2(c,p) \doteq c! \\
\textbf{if } p \textbf{ then } \mathsf{T} \textbf{ else} \\
\textbf{if } \neg p^{hc} \textbf{ then } \mathcal{V}_2(c,p^h) :: \mathcal{V}_2(c,p^t) \textbf{ else } p^h :: \\
\textbf{let } d = \textbf{aspect}(\text{<proof>}, p^t) \textbf{ in} \\
\textbf{if } d \textbf{ then } \mathsf{T} \textbf{ else} \\
\textbf{let } r = \mathcal{S}(c,\mathcal{U}^M(\mathcal{E}(d^3,\mathsf{T},c)\text{ ‘ }c\text{ ‘ }p)) \textbf{ in} \\
\textbf{if } r^E \textbf{ then } \text{error}(\text{“Error in proof of”}, d^2\lceil\text{“} \\
\text{”}\rceil^1 r) \textbf{ else } r]^{249}
\end{array}$$

---

[247] [verifier $\stackrel{\text{pyk}}{=}$ “verifier”]

[248] [$\mathcal{V}_1(c)$ $\stackrel{\text{pyk}}{=}$ “verify one ∗ end verify”]

[249] [$\mathcal{V}_2(c,p)$ $\stackrel{\text{pyk}}{=}$ “verify two ∗ proofs ∗ end verify”]

$[\mathcal{V}_3(\mathsf{c},\mathsf{r},\mathsf{p},\mathsf{d})]$ returns the diagnose $[\mathsf{d}]$ if $[\mathsf{d}]$ differs from $[\mathsf{T}]$. Otherwise, it traverses the array $[\mathsf{p}]$ of sequents and checks each sequent $[\mathsf{q}]$ for correctness. During the check, $[\mathsf{i}]$ is bound to the identifier of the proof being checked and $[\mathsf{d}]$ is bound to the lemma being checked.

$[\mathcal{V}_3(\mathsf{c},\mathsf{r},\mathsf{p},\mathsf{d}) \doteq \mathsf{c!r!p!}$
if $\neg\mathsf{d}$ **then** d **else**
if p **then** T **else**
if $\neg\mathsf{p}^{\mathrm{hc}}$ **then** $\mathcal{V}_3(\mathsf{c},\mathsf{r},\mathsf{p}^{\mathrm{t}},\mathcal{V}_3(\mathsf{c},\mathsf{r},\mathsf{p}^{\mathrm{h}},\mathsf{T}))$ **else**
let $\mathsf{i} = \mathsf{p}^{\mathrm{h}}$ **in**
let $\mathsf{q} = \mathsf{p}^{\mathrm{t}}$ **in**
if q **then** T **else**
if q^{E} **then** q **else**
if $\neg\mathsf{q}^{1}$ **then** error("Unchecked sidecondition:", $\mathsf{q}^{1\mathrm{h}}$) **else**
let $\mathsf{d} =$ **aspect**(<stmt>, c[r]["codex"][r][i]) **in**
if d **then** error("Proof of non-existent lemma:", q^2) **else**
if $\neg\mathsf{q}^2 \overset{\mathrm{t}}{=} \mathsf{d}^3$ **then** error("Lemma/proof mismatch:", $\mathsf{d}^2;\mathsf{q}^2$) **else**
$\mathcal{V}_4(\mathsf{c},\mathsf{q}^0)]$[250]

$[\mathcal{V}_4(\mathsf{c},\mathsf{p})]$ checks that the list $[\mathsf{p}]$ of premises consists of proved lemmas. That is done by verifying that each member of $[\mathsf{p}]$ has a proof and belongs to a correct page that lists the verifier among its claims. The page being verified is considered correct during the check since the diagnose aspect is not yet set while checking the page. The only thing $[\mathcal{V}_4(\mathsf{c},\mathsf{p})]$ does not check for is circular proofs (e.g. proofs that make use of the lemma they prove). Checking for circularity is done elsewhere.

$[\mathcal{V}_4(\mathsf{c},\mathsf{p}) \doteq \mathsf{c!}$
if p **then** T **else**
let $\mathsf{d} = \mathcal{V}_4(\mathsf{c},\mathsf{p}^{\mathrm{t}})$ **in**
if $\neg\mathsf{d}$ **then** d **else**
let $\mathsf{p} = \mathsf{p}^{\mathrm{h}}$ **in**
let $\mathsf{r} = \mathsf{p}^{\mathrm{r}}$ **in**
let $\mathsf{i} = \mathsf{p}^{\mathrm{i}}$ **in**
if $\neg\mathsf{c}$[r]["diagnose"] **then**
error("Reference to erroneous page", p) **else**
if $\neg$claims($\lceil$verifier$\rceil$, c, r) **then**
error("Reference to unchecked lemma", p) **else**
if aspect(<proof>, p, c) **then**
error("Reference to unproved lemma", p) **else** T][251]

$[\mathcal{V}_7(\mathsf{c},\mathsf{r},\mathsf{i},\mathsf{q})]$ takes as input an array $[\mathsf{q}]$ that contains the conclusions of all proofs on the present page. These conclusions are all sequents and they are indexed by the identifiers of the symbols they belong to.

[250] $[\mathcal{V}_3(\mathsf{c},\mathsf{r},\mathsf{p},\mathsf{d}) \overset{\mathrm{pyk}}{=}$ "verify three ∗ ref ∗ sequents ∗ diagnose ∗ end verify"]

[251] $[\mathcal{V}_4(\mathsf{c},\mathsf{p}) \overset{\mathrm{pyk}}{=}$ "verify four ∗ premises ∗ end verify"]

The sequents are allowed to have unresolved premises provided the premises are proved elsewhere. Furthermore, the directed graph with sequents as nodes and premises as edges is required to be non-cyclic. $[\mathcal{V}_7(\mathsf{c},\mathsf{r},\mathsf{i},\mathsf{q})]$ performs a search for cycles.

Logiweb pages and Logiweb bibliographic references form a non-cyclic graph, so whenever a premise refers to a lemma proved on another page, that premise cannot be part of a cycle. For that reason, $[\mathcal{V}_7(\mathsf{c},\mathsf{r},\mathsf{i},\mathsf{q})]$ ignores premises whose reference differs from the reference $[\mathsf{r}]$ of the page being checked.

$[\mathcal{V}_7(\mathsf{c},\mathsf{r},\mathsf{i},\mathsf{q})]$ checks all nodes and edges reachable from the node with identifier $[\mathsf{i}]$ for cycles. If a cycle is found, the identifier of one of the nodes in the cycle is returned. Hence, a return value which is a cardinal indicates that a cycle is found. Otherwise, $[\mathcal{V}_7(\mathsf{c},\mathsf{r},\mathsf{i},\mathsf{q})]$ replaces all reachable nodes in $[\mathsf{q}]$ by the cardinal $[1]$ and returns the modified $[\mathsf{q}]$.

When $[\mathcal{V}_7(\mathsf{c},\mathsf{r},\mathsf{i},\mathsf{q})]$ reaches a node that is set to $[1]$ then it assumes that that node has already been checked for cycles.

During the search for cycles, $[\mathcal{V}_7(\mathsf{c},\mathsf{r},\mathsf{i},\mathsf{q})]$ temporarily sets all nodes being considered to $[0]$. When $[\mathcal{V}_7(\mathsf{c},\mathsf{r},\mathsf{i},\mathsf{q})]$ reaches a node that is set to $[0]$ it assumes that a cycle has been found and returns the identifier of the node.

$$
\begin{array}{l}
[\mathcal{V}_7(\mathsf{c},\mathsf{r},\mathsf{i},\mathsf{q}) \doteq \mathsf{c}!\mathsf{r}! \\
\textbf{let } \mathsf{v} = \mathsf{q}[\mathsf{i}] \textbf{ in} \\
\textbf{if } \mathsf{v} \textbf{ then } \mathsf{q} \textbf{ else} \\
\textbf{if } \mathsf{v} \approx 0 \textbf{ then } \mathsf{i} \textbf{ else} \\
\textbf{if } \mathsf{v} \approx 1 \textbf{ then } \mathsf{q} \textbf{ else} \\
\textbf{let } \mathsf{q} = \mathcal{V}_6(\mathsf{c},\mathsf{r},\mathsf{v}^0,\mathsf{q}[\mathsf{i}{\to}0]) \textbf{ in} \\
\textbf{if } \mathsf{q}^{\mathrm{c}} \textbf{ then } \mathsf{q} \textbf{ else } \mathsf{q}[\mathsf{i}{\to}1]]
\end{array}
\tag{252}
$$

$[\mathcal{V}_6(\mathsf{c},\mathsf{r},\mathsf{p},\mathsf{q})]$ checks all indices in the list $[\mathsf{p}]$ if premises for circularity in the array $[\mathsf{q}]$ of sequents.

$$
\begin{array}{l}
[\mathcal{V}_6(\mathsf{c},\mathsf{r},\mathsf{p},\mathsf{q}) \doteq \mathsf{c}!\mathsf{r}!\mathsf{p}! \\
\textbf{if } \mathsf{q}^{\mathrm{c}} \textbf{ then } \mathsf{q} \textbf{ else} \\
\textbf{if } \mathsf{p} \textbf{ then } \mathsf{q} \textbf{ else} \\
\textbf{let } \mathsf{q} = \mathcal{V}_6(\mathsf{c},\mathsf{r},\mathsf{p}^{\mathrm{t}},\mathsf{q}) \textbf{ in} \\
\textbf{if } \mathsf{q}^{\mathrm{c}} \textbf{ then } \mathsf{q} \textbf{ else} \\
\textbf{if } \neg\mathsf{r} \approx \mathsf{p}^{\mathrm{hr}} \textbf{ then } \mathsf{q} \textbf{ else} \\
\mathcal{V}_7(\mathsf{c},\mathsf{r},\mathsf{p}^{\mathrm{hi}},\mathsf{q})]
\end{array}
\tag{253}
$$

$[\mathcal{V}_5(\mathsf{c},\mathsf{r},\mathsf{a},\mathsf{q})]$ checks all indices in the array $[\mathsf{a}]$ for circularity in the array $[\mathsf{q}]$ of sequents.

$$
\begin{array}{l}
[\mathcal{V}_5(\mathsf{c},\mathsf{r},\mathsf{a},\mathsf{q}) \doteq \mathsf{c}!\mathsf{r}!\mathsf{a}! \\
\textbf{if } \mathsf{q}^{\mathrm{c}} \textbf{ then } \mathsf{q} \textbf{ else} \\
\textbf{if } \mathsf{a} \textbf{ then } \mathsf{q} \textbf{ else}
\end{array}
$$

[252] $[\mathcal{V}_7(\mathsf{c},\mathsf{r},\mathsf{i},\mathsf{q}) \stackrel{\text{pyk}}{=}$ "verify seven $*$ ref $*$ id $*$ sequents $*$ end verify"]

[253] $[\mathcal{V}_6(\mathsf{c},\mathsf{r},\mathsf{p},\mathsf{q}) \stackrel{\text{pyk}}{=}$ "verify six $*$ ref $*$ list $*$ sequents $*$ end verify"]

if $\neg \mathsf{a}^{\mathrm{hc}}$ **then** $\mathcal{V}_5(\mathsf{c}, \mathsf{r}, \mathsf{a}^{\mathrm{t}}, \mathcal{V}_5(\mathsf{c}, \mathsf{r}, \mathsf{a}^{\mathrm{h}}, \mathsf{q}))$ **else**
$\mathcal{V}_7(\mathsf{c}, \mathsf{r}, \mathsf{a}^{\mathrm{h}}, \mathsf{q})]$[254]

6.6.4 The rule lemma tactic

In Section 6.6.2 we stated two lemmas:

[T'_{E} **lemma** HeadNil″: $\mathsf{T}^{\mathrm{h}} = \mathsf{T}$]

[T'_{E} **antilemma** Contra″: $\mathsf{T} :: \mathsf{T} = \mathsf{T}$]

Both lemmas are trivial; they are both instances of the general fact that all rules of a theory are provable in the theory. It is rather straightforward to prove both lemmas using suitable sequent operations but instead of writing a handmade proof for each lemma we shall write a proof tactic which, given a theory and a rule of the theory, generates a proof that the rule follows from the theory. We shall refer to the tactic as the [Rule tactic], and we shall arrange that the following become valid proofs of the above lemmas:

[**Proof of** HeadNil″: Rule tactic]

[**Proof of** Contra″: Rule tactic]

We define the rule tactic thus:

[Rule tactic $\doteq \lambda \mathsf{c}.\lambda \mathsf{p}.\mathrm{rule}(\mathsf{c}, \mathsf{p})$][255]

When invoked from the verifier, the tactic is applied to the cache [c] of the current page and a pair [p] whose first component is the identifier of the lemma to be proved and whose second component is the subcodex containing all aspects of that lemma. The rule tactic passes control to the function below.

[$\mathrm{rule}(\mathsf{c}, \mathsf{p}) \doteq \mathsf{c}!$
let $\mathsf{s} =$ **aspect**(<stmt>, $\mathsf{p}^{\mathrm{t}})^3$ **in**
if s **then** ⌈"Rule has no statement aspect"⌉ **else**
if $\neg \mathsf{s} \overset{\mathrm{r}}{=} \lceil \mathsf{x} \vdash \mathsf{y} \rceil$ **then** error("Rule has invalid statement aspect", s) **else**
let $\mathsf{t} =$ **aspect**(<stmt>, $\mathsf{s}^1, \mathsf{c})^3$ **in**
if t **then** ⌈"Theory has no statement aspect"⌉ **else**
let $\mathsf{r} = \mathrm{rule}_1(\mathsf{s}^2, \mathsf{t})$ **in**
if r^{c} **then** error("The theory does not assert the given rule", $\mathsf{s}; \mathsf{t}$) **else**
$(\mathsf{s}^1 \vdash \mathrm{Cut}(\mathsf{s}^{1\mathrm{I}\triangleright *}, \mathsf{r}))$][256]

The function above finds out what the rule tactic is supposed to prove and then passes control to the function below for constructing the proofs.

---

[254] $[\mathcal{V}_5(\mathsf{c}, \mathsf{r}, \mathsf{a}, \mathsf{q}) \overset{\mathrm{pyk}}{=}$ "verify five $*$ ref $*$ array $*$ sequents $*$ end verify"]

[255] [Rule tactic $\overset{\mathrm{pyk}}{=}$ "rule tactic"]

[256] $[\mathrm{rule}(\mathsf{c}, \mathsf{p}) \overset{\mathrm{pyk}}{=}$ "rule $*$ subcodex $*$ end rule"]

$[\mathrm{rule}_1(\mathsf{s},\mathsf{t}) \doteq$
if $\mathsf{s} \overset{\mathrm{t}}{=} \mathsf{t}$ **then** T **else**
if $\neg\mathsf{t} \overset{\mathrm{r}}{=} \lceil \mathsf{x} \oplus \mathsf{y} \rceil$ **then** 0 **else**
let $\mathsf{p} = \mathrm{rule}_1(\mathsf{s},\mathsf{t}^1)$ **in**
if $\neg\mathsf{p}^{\mathrm{c}}$ **then** $\mathrm{Cut}(\mathrm{Head}_{\oplus}(\mathsf{t}),\mathsf{p})$ **else**
let $\mathsf{p} = \mathrm{rule}_1(\mathsf{s},\mathsf{t}^2)$ **in**
if $\neg\mathsf{p}^{\mathrm{c}}$ **then** $\mathrm{Cut}(\mathrm{Tail}_{\oplus}(\mathsf{t}),\mathsf{p})$ **else** $0]$[257]

The function above searches for the given rule in the given theory. When found, it constructs a proof using the three proof constructors below.

$[\mathrm{Cut}(\mathsf{a},\mathsf{b}) \doteq \mathrm{If}(\mathsf{b},\mathsf{a},\mathsf{a};\mathsf{b})]$[258]

$[\mathrm{Head}_{\oplus}(\mathsf{s}) \doteq (\mathsf{s}^1 \vdash \mathsf{s}^2 \vdash \mathsf{s}^{1\mathrm{I}\triangleright})^{+\triangleright}]$[259]

$[\mathrm{Tail}_{\oplus}(\mathsf{s}) \doteq (\mathsf{s}^1 \vdash \mathsf{s}^{2\mathrm{I}})^{+\triangleright}]$[260]

6.6.5 Stating rules

The rule tactic of the previous section allows to prove that all rules of a theory are provable in the theory. The following macros allow to express lemma and proof in one go:

$[[\mathsf{x}\ \textbf{rule}\ \mathsf{y}\colon\mathsf{z}] \doteqdot [\mathsf{x}\ \textbf{lemma}\ \mathsf{y}\colon\mathsf{z}][\textbf{Proof of}\ \mathsf{y}\colon\mathrm{Rule\ tactic}]]$[261]

$[[\mathsf{x}\ \textbf{antirule}\ \mathsf{y}\colon\mathsf{z}] \doteqdot [\mathsf{x}\ \textbf{rule}\ \mathsf{y}\colon\mathsf{z} \vdash \perp\!\!\!\perp]]$[262]

Having these constructs it is easy to state that all four rules of $[\mathrm{T}'_{\mathrm{E}}]$ are provable in $[\mathrm{T}'_{\mathrm{E}}]$:

$[\mathrm{T}'_{\mathrm{E}}\ \textbf{rule}\ \mathrm{HeadNil}''\colon \mathsf{T}^{\mathrm{h}} = \mathsf{T}]$

$[\mathrm{T}'_{\mathrm{E}}\ \textbf{rule}\ \mathrm{HeadPair}''\colon \forall\mathcal{A}\colon\forall\mathcal{B}\colon(\mathcal{A} :: \mathcal{B})^{\mathrm{h}} = \mathcal{A}]$[263]

$[\mathrm{T}'_{\mathrm{E}}\ \textbf{rule}\ \mathrm{Transitivity}''\colon \forall\mathcal{A}\colon\forall\mathcal{B}\colon\forall\mathcal{C}\colon \mathcal{A} = \mathcal{B} \vdash \mathcal{A} = \mathcal{C} \vdash \mathcal{B} = \mathcal{C}]$[264]

$[\mathrm{T}'_{\mathrm{E}}\ \textbf{antirule}\ \mathrm{Contra}''\colon \mathsf{T} :: \mathsf{T} = \mathsf{T}]$

---

[257] $[\mathrm{rule}_1(\mathsf{s},\mathsf{t}) \overset{\mathrm{pyk}}{=}$ "rule one $*$ theory $*$ end rule"]
[258] $[\mathrm{Cut}(\mathsf{a},\mathsf{b}) \overset{\mathrm{pyk}}{=}$ "cut $*$ and $*$ end cut"]
[259] $[\mathrm{Head}_{\oplus}(\mathsf{s}) \overset{\mathrm{pyk}}{=}$ "head $*$ end head"]
[260] $[\mathrm{Tail}_{\oplus}(\mathsf{s}) \overset{\mathrm{pyk}}{=}$ "tail $*$ end tail"]
[261] $[[\mathsf{x}\ \textbf{rule}\ \mathsf{y}\colon\mathsf{z}] \overset{\mathrm{pyk}}{=}$ "in theory $*$ rule $*$ says $*$ end rule"]
[262] $[[\mathsf{x}\ \textbf{antirule}\ \mathsf{y}\colon\mathsf{z}] \overset{\mathrm{pyk}}{=}$ "in theory $*$ antirule $*$ says $*$ end antirule"]
[263] $[\mathrm{HeadPair}'' \overset{\mathrm{pyk}}{=}$ "example scheme lemma primed"]
[264] $[\mathrm{Transitivity}'' \overset{\mathrm{pyk}}{=}$ "example rule lemma primed"]

6.6.6 Stating theories

We have now defined the theory $[\mathrm{T}'_{\mathrm{E}}]$ and stated the four rule lemmas there are for that theory, namely one for each rule of the theory. Once the rules are stated, the definition if $[\mathrm{T}'_{\mathrm{E}}]$ is a bit redundant since the four rules contain all information about the theory.

We now define the theory and the four rule lemmas once more, but in a more elegant way:

$[\mathrm{T}_{\mathrm{E}}$ **rule** HeadNil: $\mathsf{T}^{\mathrm{h}} = \mathsf{T}]$[265]

$[\mathrm{T}_{\mathrm{E}}$ **rule** HeadPair: $\forall\mathcal{A}\colon\forall\mathcal{B}\colon(\mathcal{A} :: \mathcal{B})^{\mathrm{h}} = \mathcal{A}]$[266]

$[\mathrm{T}_{\mathrm{E}}$ **rule** Transitivity: $\forall\mathcal{A}\colon\forall\mathcal{B}\colon\forall\mathcal{C}\colon\mathcal{A} = \mathcal{B} \vdash \mathcal{A} = \mathcal{C} \vdash \mathcal{B} = \mathcal{C}]$[267]

$[\mathrm{T}_{\mathrm{E}}$ **antirule** Contra: $\mathsf{T} :: \mathsf{T} = \mathsf{T}]$[268]

[**Theory** T_{E}][269]

In the next section we define $([\textbf{Theory } \mathsf{x}])^{\mathbf{p}}$ such that $([\textbf{Theory } \mathrm{T}_{\mathrm{E}}])^{\mathbf{p}}$ macro expands into a definition which defines the statement aspect of $[\mathrm{T}_{\mathrm{E}}]$ to be a conjunction of $[\mathsf{T}^{\mathrm{h}} = \mathsf{T}]$, $[\forall\mathcal{A}\colon\forall\mathcal{B}\colon(\mathcal{A} :: \mathcal{B})^{\mathrm{h}} = \mathcal{A}]$, $[\forall\mathcal{A}\colon\forall\mathcal{B}\colon\forall\mathcal{C}\colon\mathcal{A} = \mathcal{B} \vdash \mathcal{A} = \mathcal{C} \vdash \mathcal{B} = \mathcal{C}]$, and $[\mathsf{T} :: \mathsf{T} = \mathsf{T} \vdash \perp\!\!\!\perp]$, in some, arbitrary order.

When $([\textbf{Theory } \mathsf{x}])^{\mathbf{p}}$ is macro expanded, it scans the codex of the page it occurs on for rules that belong to the theory $[\mathsf{x}]$, then forms a conjunction of the rules found, and then constructs a suitable definition for $[\mathsf{x}]$.

6.6.7 Rule collection

Here is the macro for collecting all rules of a theory from the codex:

$([[\textbf{Theory } \mathsf{n}] \overset{\text{macro}}{\rightarrow} \lambda\mathsf{t}.\lambda\mathsf{s}.\lambda\mathsf{c}.\mathrm{theory}_2(\mathsf{t},\mathsf{c})])^{\mathbf{p}}$[270]

$[\mathrm{theory}_2(\mathsf{t},\mathsf{c}) \doteq$
let $\mathsf{n} = \mathsf{t}^1$ **in**
let $\mathsf{s} = \langle\lceil\mathsf{n}\rceil :: \mathsf{n}, \lceil\mathsf{x}\rceil :: \mathrm{theory}_3(\mathsf{c},\mathsf{n})\rangle$ **in**
$\tilde{\mathcal{Q}}(\mathsf{t}, \lceil[\mathsf{n} \overset{\text{stmt}}{=} \mathsf{x}]\rceil, \mathsf{s})]$[271]

---

[265] [HeadNil $\overset{\text{pyk}}{=}$ "example axiom lemma"]

[266] [HeadPair $\overset{\text{pyk}}{=}$ "example scheme lemma"]

[267] [Transitivity $\overset{\text{pyk}}{=}$ "example rule lemma"]

[268] [Contra $\overset{\text{pyk}}{=}$ "contraexample lemma"]

[269] [T_{E} $\overset{\text{pyk}}{=}$ "example theory"]

[270] [[**Theory** n] $\overset{\text{pyk}}{=}$ "theory $*$ end theory"]

[271] [$\mathrm{theory}_2(\mathsf{t},\mathsf{c})$ $\overset{\text{pyk}}{=}$ "theory two $*$ cache $*$ end theory"]

$[\text{theory}_3(\mathsf{c},\mathsf{n}) \doteq \mathsf{n}!$
let $\mathsf{r} = \mathsf{c}[0]$ **in**
$\text{theory}_4(\mathsf{c}[\mathsf{r}][\text{"codex"}][\mathsf{r}],\mathsf{n},\mathsf{T})]$[272]

$[\text{theory}_4(\mathsf{c},\mathsf{n},\mathsf{s}) \doteq \mathsf{n}!$
if c **then** s **else**
if $\neg\mathsf{c}^{\mathrm{hc}}$ **then** $\text{theory}_4(\mathsf{c}^{\mathrm{t}},\mathsf{n},\text{theory}_4(\mathsf{c}^{\mathrm{h}},\mathsf{n},\mathsf{s}))$ **else**
if $\neg$**aspect**$(<\text{proof}>,\mathsf{c}^{\mathrm{t}})^3 \stackrel{\mathrm{t}}{=} \lceil\text{Rule tactic}\rceil$ **then** s **else**
let $\mathsf{d} =$ **aspect**$(<\text{stmt}>,\mathsf{c}^{\mathrm{t}})^3$ **in**
if $\neg\mathsf{d}^1 \stackrel{\mathrm{t}}{=} \mathsf{n}$ **then** s **else**
$\text{Plus}(\mathsf{d}^2,\mathsf{s})]$[273]

$[\text{Plus}(\mathsf{a},\mathsf{b}) \doteq \text{If}(\mathsf{b},\mathsf{a},\mathsf{a}\oplus\mathsf{b})]$[274]

6.6.8 Example lemmas

One may think of the Logiweb sequent calculus as an assembly language for expressing proofs. Proofs directly expressed in the calculus are somewhat obscure to read and write, but before we solve that problem in Section 6.8, we state and prove two lemmas the hard way. The first lemma proves reflexivity in the $[\mathrm{T_E}]$ theory:

$[\mathrm{T_E}$ **lemma** Reflexivity: $\forall\mathcal{A}\colon \mathcal{A} = \mathcal{A}]$[275]

[**Proof of** Reflexivity: $\lceil\mathrm{T_E} \vdash \forall\mathcal{A}\colon ($
$\text{HeadPair}^{\mathrm{I}\rhd *\rhd}\ @\,\mathcal{A}\,@\,\mathcal{A};$
$(\text{Transitivity}^{\mathrm{I}\rhd *\rhd}\ @(\mathcal{A}::\mathcal{A})^{\mathrm{h}}\,@\,\mathcal{A}\,@\,\mathcal{A})^{\rhd\rhd})\rceil]$

Now that we have reflexivity available, we may use it to prove commutativity in $[\mathrm{T_E}]$:

$[\mathrm{T_E}$ **lemma** Commutativity: $\forall\mathcal{A}\colon\forall\mathcal{B}\colon \mathcal{A} = \mathcal{B} \vdash \mathcal{B} = \mathcal{A}]$[276]

[**Proof of** Commutativity: $\lceil\mathrm{T_E} \vdash \forall\mathcal{A}\colon\forall\mathcal{B}\colon \mathcal{A} = \mathcal{B} \vdash ($
$\text{Reflexivity}^{\mathrm{I}\rhd *\rhd}\ @\,\mathcal{A};$
$(\text{Transitivity}^{\mathrm{I}\rhd *\rhd}\ @\,\mathcal{A}\,@\,\mathcal{B}\,@\,\mathcal{A})^{\rhd\rhd})\rceil]$

As can be seen on the diagnose hook of the present page, the proofs above are correct according the the machine check made by the verifier.

---

[272] $[\text{theory}_3(\mathsf{c},\mathsf{n}) \stackrel{\text{pyk}}{=}$ "theory three $*$ name $*$ end theory"]

[273] $[\text{theory}_4(\mathsf{c},\mathsf{n},\mathsf{s}) \stackrel{\text{pyk}}{=}$ "theory four $*$ name $*$ sum $*$ end theory"]

[274] $[\text{Plus}(\mathsf{a},\mathsf{b}) \stackrel{\text{pyk}}{=}$ "plus $*$ and $*$ end plus"]

[275] [Reflexivity $\stackrel{\text{pyk}}{=}$ "sequent reflexivity"]

[276] [Commutativity $\stackrel{\text{pyk}}{=}$ "sequent commutativity"]

6.7 Unification

The facilities defined in Section 6.8 for expressing proofs in a human-tolerant style makes use of unification. Section 6.7 implements unification.

6.7.1 Parameter terms

We shall refer to terms in which bound metavariables are replaced by cardinals as *parameter terms*. The function $[\mathrm{parm}(\mathsf{t},\mathsf{s},\mathsf{n})]$ converts an ordinary term $[\mathsf{t}]$ into a parameter term in which numbers are constructed from $[\mathsf{n}]$ using $[\mathsf{b}+2*\mathsf{n}]$ iteratively. When calling $[\mathrm{parm}(\mathsf{t},\mathsf{s},\mathsf{n})]$, $[\mathsf{s}]$ should be $[\mathsf{T}]$.

$$[\mathrm{parm}(\mathsf{t},\mathsf{s},\mathsf{n}) \doteq \mathsf{n}!\\ \textbf{if } \mathsf{t} \overset{\mathrm{r}}{=} \lceil\forall \mathsf{x}\colon \mathsf{y}\rceil \textbf{ then } \forall \mathsf{n}\colon \mathrm{parm}(\mathsf{t}^2,(\mathsf{t}^1::\mathsf{n})::\mathsf{s},\mathsf{T}+2*\mathsf{n}) \textbf{ else}\\ \textbf{let } \mathsf{m}=\textbf{lookup}(\mathsf{t},\mathsf{s},\mathsf{T}) \textbf{ in}\\ \textbf{if } \neg\mathsf{m} \textbf{ then } \mathsf{m} \textbf{ else } \mathsf{t}^{\mathrm{R}}::\mathrm{parm}^*(\mathsf{t}^{\mathrm{t}},\mathsf{s},\mathsf{n})]^{277}$$

$$[\mathrm{parm}^*(\mathsf{t},\mathsf{s},\mathsf{n}) \doteq \mathsf{s}!\mathsf{n}!\mathrm{If}(\mathsf{t}^{\mathrm{a}},\mathsf{T},\mathrm{parm}(\mathsf{t}^{\mathrm{h}},\mathsf{s},\mathsf{n})::\mathrm{parm}^*(\mathsf{t}^{\mathrm{t}},\mathsf{s},\mathsf{n}))]^{278}$$

6.7.2 Substitutions

We shall refer to an array of parameter terms as a substitution. The following function instantiates a parameter term $[\mathsf{t}]$ using a substitution $[\mathsf{s}]$:

$$[\mathrm{inst}(\mathsf{t},\mathsf{s}) \doteq \mathrm{If}(\mathsf{t}^{\mathrm{c}},\mathrm{inst}(\mathsf{s}[\mathsf{t}],\mathsf{s}),\mathsf{t}^{\mathrm{R}}::\mathrm{inst}^*(\mathsf{t}^{\mathrm{t}},\mathsf{s}))]^{279}$$

$$[\mathrm{inst}^*(\mathsf{t},\mathsf{s}) \doteq \mathsf{s}!\mathrm{If}(\mathsf{t}^{\mathrm{a}},\mathsf{T},\mathrm{inst}(\mathsf{t}^{\mathrm{h}},\mathsf{s})::\mathrm{inst}^*(\mathsf{t}^{\mathrm{t}},\mathsf{s}))]^{280}$$

Instantiation may loop indefinitely. As an example, a substitution which maps $[\mathcal{A}]$ to $[\mathcal{A}::\mathcal{A}]$ will keep expanding $[\mathcal{A}]$ forever. We shall say that a substitution $[\mathsf{s}]$ is *circular* if there exists a term $[\mathsf{t}]$ for which $[\mathrm{inst}(\mathsf{t},\mathsf{s})]$ loops indefinitely.

6.7.3 Occurrence

$[\mathrm{occur}(\mathsf{t},\mathsf{u},\mathsf{s})]$ is true if the parameter $[\mathsf{t}]$ occurs in $[\mathrm{inst}(\mathsf{u},\mathsf{s})]$. $[\mathrm{occur}(\mathsf{t},\mathsf{u},\mathsf{s})]$ may loop indefinitely if the substitution $[\mathsf{s}]$ is circular.

$$[\mathrm{occur}(\mathsf{t},\mathsf{u},\mathsf{s}) \doteq \mathsf{s}!\mathrm{If}(\mathsf{u}^{\mathrm{c}},\mathsf{t}\approx\mathsf{u} \mathbin{\ddot{\vee}} \mathrm{occur}(\mathsf{t},\mathsf{s}[\mathsf{u}],\mathsf{s}),\mathrm{occur}^*(\mathsf{t},\mathsf{u}^{\mathrm{t}},\mathsf{s}))]^{281}$$

$$[\mathrm{occur}^*(\mathsf{t},\mathsf{u},\mathsf{s}) \doteq \mathsf{t}!\mathsf{s}!\mathrm{If}(\mathsf{u}^{\mathrm{a}},\mathsf{F},\mathrm{occur}(\mathsf{t},\mathsf{u}^{\mathrm{h}},\mathsf{s}) \mathbin{\ddot{\vee}} \mathrm{occur}^*(\mathsf{t},\mathsf{u}^{\mathrm{t}},\mathsf{s}))]^{282}$$

---

[277] $[\mathrm{parm}(\mathsf{t},\mathsf{s},\mathsf{n}) \overset{\mathrm{pyk}}{=}$ "parameter term $*$ stack $*$ seed $*$ end parameter"]

[278] $[\mathrm{parm}^*(\mathsf{t},\mathsf{s},\mathsf{n}) \overset{\mathrm{pyk}}{=}$ "parameter term star $*$ stack $*$ seed $*$ end parameter"]

[279] $[\mathrm{inst}(\mathsf{t},\mathsf{s}) \overset{\mathrm{pyk}}{=}$ "instantiate $*$ with $*$ end instantiate"]

[280] $[\mathrm{inst}^*(\mathsf{t},\mathsf{s}) \overset{\mathrm{pyk}}{=}$ "instantiate star $*$ with $*$ end instantiate"]

[281] $[\mathrm{occur}(\mathsf{t},\mathsf{u},\mathsf{s}) \overset{\mathrm{pyk}}{=}$ "occur $*$ in $*$ substitution $*$ end occur"]

[282] $[\mathrm{occur}^*(\mathsf{t},\mathsf{u},\mathsf{s}) \overset{\mathrm{pyk}}{=}$ "occur star $*$ in $*$ substitution $*$ end occur"]

6.7.4 Unifications

We shall refer to the result of applying a substitution to a parameter term as an *instance* of the term. We shall refer to a common instance of two parameter terms as a *unification* of the terms. As an example, $[\mathcal{A} :: \mathsf{F}]$ and $[\mathsf{T} :: \mathcal{B}]$ (where $[\mathcal{A}]$ and $[\mathcal{B}]$ denote numbers) have exactly one unification, namely $[\mathsf{T} :: \mathsf{F}]$. We shall say that two terms are *compatible* if they have at least one unification and *incompatible* otherwise.

A substitution which yields the same result when applied to two terms $[\mathsf{u}]$ and $[\mathsf{v}]$ is said to *unify* the terms. As an example, the substitution which maps $[\mathcal{A}]$ to $[\mathsf{T}]$ and $[\mathcal{B}]$ to $[\mathsf{F}]$ unifies $[\mathcal{A} :: \mathsf{F}]$ and $[\mathsf{T} :: \mathcal{B}]$.

The *unification algorithm* presented in the following takes two terms as input and returns a unifying substitution if the terms are compatible. As an example, when applied to $[\mathcal{A} :: \mathsf{F}]$ and $[\mathsf{T} :: \mathcal{B}]$, the unification algorithm returns the substitution which maps $[\mathcal{A}]$ to $[\mathsf{T}]$ and $[\mathcal{B}]$ to $[\mathsf{F}]$.

There is more than one substitution which unifies $[\mathcal{A} :: \mathsf{F}]$ and $[\mathsf{T} :: \mathcal{B}]$. As an example the substitution that maps $[\mathcal{A}]$ to $[\mathsf{T}]$, $[\mathcal{B}]$ to $[\mathsf{F}]$, and $[\mathcal{C}]$ to some term $[\mathsf{x}]$ also unifies $[\mathcal{A} :: \mathsf{F}]$ and $[\mathsf{T} :: \mathcal{B}]$.

6.7.5 Unification algorithm

The original unification algorithm by Robinson takes a set of equations (i.e. a set of pairs of parameter terms) as input and unifies all the pairs. In contrast, $[\mathrm{unify}(\mathsf{t} = \mathsf{u}, \mathsf{s})]$ defined below adds the result of unifying $[\mathsf{t}]$ and $[\mathsf{u}]$ to the substitution $[\mathsf{s}]$. To unify a set of equations one should start with the empty substitution $[\mathsf{T}]$ and call $[\mathrm{unify}(\mathsf{t} = \mathsf{u}, \mathsf{s})]$ once for each equation.

$[\mathrm{unify}(\mathsf{t} = \mathsf{u}, \mathsf{s})]$ extends the substitution $[\mathsf{s}]$ to a substitution that unifies $[\mathsf{t}]$ with $[\mathsf{u}]$ if such a substitution exists and returns $[0]$ otherwise. In particular, $[\mathrm{unify}(\mathsf{t} = \mathsf{u}, \mathsf{s})]$ returns zero if $[\mathsf{s}]$ equals zero since in that case there is no substitution to extend.

$[\mathrm{unify}(\mathsf{t} = \mathsf{u}, \mathsf{s})]$ calls to auxiliary functions, $[\mathrm{unify}_2(\mathsf{t} = \mathsf{u}, \mathsf{s})]$ and $[\mathrm{unify}^*(\mathsf{t} = \mathsf{u}, \mathsf{s})]$. $[\mathrm{unify}_2(\mathsf{t} = \mathsf{u}, \mathsf{s})]$ does the same as $[\mathrm{unify}(\mathsf{t} = \mathsf{u}, \mathsf{s})]$ but only covers the special case where $[\mathsf{t}]$ is a parameter. $[\mathrm{unify}^*(\mathsf{t} = \mathsf{u}, \mathsf{s})]$ unifies two lists $[\mathsf{t}]$ and $[\mathsf{u}]$ of parameter terms.

$[\mathrm{unify}(\mathsf{t} = \mathsf{u}, \mathsf{s})]$ errors out if $[\mathsf{s} \approx 0]$, calls $[\mathrm{unify}_2(* = *, *)]$ if $[\mathsf{t}]$ or $[\mathsf{u}]$ happens to be a parameter, and calls $[\mathrm{unify}^*(* = *, *)]$ otherwise:

$[\mathrm{unify}(\mathsf{t} = \mathsf{u}, \mathsf{s}) \doteq \mathsf{t}!\mathsf{u}!$
if s^{c} **then** s **else**
if t^{c} **then** $\mathrm{unify}_2(\mathsf{t} = \mathsf{u}, \mathsf{s})$ **else**
if u^{c} **then** $\mathrm{unify}_2(\mathsf{u} = \mathsf{t}, \mathsf{s})$ **else**
if $\mathsf{t} \overset{\mathrm{r}}{=} \mathsf{u}$ **then** $\mathrm{unify}^*(\mathsf{t}^{\mathsf{t}} = \mathsf{u}^{\mathsf{t}}, \mathsf{s})$ **else** $0]$[283]

$[\mathrm{unify}^*(\mathsf{t} = \mathsf{u}, \mathsf{s})]$ iterates $[\mathrm{unify}(\mathsf{t} = \mathsf{u}, \mathsf{s})]$; it depends on the fact that $[\mathrm{unify}(\mathsf{t} = \mathsf{u}, \mathsf{s})]$ equals zero if $[\mathsf{s}]$ equals zero so that failure to find a unification propagates.

[283] $[\mathrm{unify}(\mathsf{t} = \mathsf{u}, \mathsf{s}) \overset{\mathrm{pyk}}{=}$ "unify $*$ with $*$ substitution $*$ end unify"]

$[\text{unify}^*(\mathsf{t} = \mathsf{u}, \mathsf{s}) \doteq \mathsf{u}!\text{If}(\mathsf{t}^{\text{a}}, \mathsf{s}, \text{unify}^*(\mathsf{t}^{\text{t}} = \mathsf{u}^{\text{t}}, \text{unify}(\mathsf{t}^{\text{h}} = \mathsf{u}^{\text{h}}, \mathsf{s}))))]$[284]

$[\text{unify}_2(\mathsf{t} = \mathsf{u}, \mathsf{s})]$ extends the substitution $[\mathsf{s}]$ with the result of unifying the parameter $[\mathsf{t}]$ with the parameter term $[\mathsf{u}]$. $[\mathsf{s}]$ is assumed to be a genuine substituion (i.e. not zero). Furthermore, $[\mathsf{s}]$ is assumed to be non-circular.

$[\text{unify}_2(\mathsf{t} = \mathsf{u}, \mathsf{s})]$ first tests $[\mathsf{t}]$ and $[\mathsf{u}]$ for equality. If they are equal, they are already unified and $[\mathsf{s}]$ is returned. Otherwise, if $[\mathsf{s}]$ already associates the parameter $[\mathsf{t}]$ with a term, then $[\text{unify}_2(\mathsf{t} = \mathsf{u}, \mathsf{s})]$ unifies $[\mathsf{u}]$ with that term. Otherwise, $[\text{unify}_2(\mathsf{t} = \mathsf{u}, \mathsf{s})]$ extends the substitution $[\mathsf{s}]$ with an association from $[\mathsf{t}]$ to $[\mathsf{u}]$. If $[\text{occur}(\mathsf{t}, \mathsf{u}, \mathsf{s})]$ is true, however, such an association would create a circular substitution. In that case no unification exists and $[\text{unify}_2(\mathsf{t} = \mathsf{u}, \mathsf{s})]$ returns zero:

$[\text{unify}_2(\mathsf{t} = \mathsf{u}, \mathsf{s}) \doteq$
if $\mathsf{t} \approx \mathsf{u}$ **then** s **else**
let $\mathsf{t}' = \mathsf{s}[\mathsf{t}]$ **in**
if $\neg\mathsf{t}'$ **then** $\text{unify}(\mathsf{t}' = \mathsf{u}, \mathsf{s})$ **else**
if $\text{occur}(\mathsf{t}, \mathsf{u}, \mathsf{s})$ **then** 0 **else** $\mathsf{s}[\mathsf{t} \rightarrow \mathsf{u}]]$[285]

6.8 Proof generation

6.8.1 Proof tactics

Counted in sequent operators, proofs typically comprise a small amount of original thought and a lot of trivial derivations. Frequently, trivial derivations can be generated by computer programs. Such programs are typically called *tactics* or *proof tactics* [8].

To support proof tactics, we introduce a *tactic aspect* for defining them and a *proof expander* for evaluating them. The proof evaluator evaluates proof tactics and generates sequent proofs, which the proof evaluator may then evaluate to sequents.

The proof expander is itself a tactic since it generates proofs. The proof expander is a value defined tactic like the rule lemma tactic defined previously. But the proof expander is a particularly general tactic which brings life to tactics that are defined using the "tactic aspect" which is introduced later.

6.8.2 Medium level proofs

The proof tactics and the proof expander defined later can translate medium level proofs like those below to Logiweb sequent calculus proofs like those in Section 6.6.8. Actually, the proofs below exactly translate to the proofs in Section 6.6.8. We start out proving reflexivity:

$[\text{T}_{\text{E}}$ **lemma** $\text{Reflexivity}_1 \colon \forall \mathcal{A} \colon \mathcal{A} = \mathcal{A}]$[286]

[284] $[\text{unify}^*(\mathsf{t} = \mathsf{u}, \mathsf{s}) \overset{\text{pyk}}{=}$ "unify star $*$ with $*$ substitution $*$ end unify"$]$

[285] $[\text{unify}_2(\mathsf{t} = \mathsf{u}, \mathsf{s}) \overset{\text{pyk}}{=}$ "unify two $*$ with $*$ substitution $*$ end unify"$]$

[286] $[\text{Reflexivity}_1 \overset{\text{pyk}}{=}$ "tactic reflexivity"$]$

$\mathrm{T_E}$ **proof of** $\mathrm{Reflexivity_1}$:

L01:	Arbitrary $\gg$	$\mathcal{A}$	;
L02:	HeadPair $\gg$	$(\mathcal{A} :: \mathcal{A})^{\mathrm{h}} = \mathcal{A}$	;
L03:	Transitivity $\triangleright$ L02 $\triangleright$ L02 $\gg$	$\mathcal{A} = \mathcal{A}$	$\square$

In the proof above, the first line declares that $[\mathcal{A}]$ denotes an arbitrary term.

The second line uses [HeadPair] to prove $[(\mathcal{A} :: \mathcal{A})^{\mathrm{h}} = \mathcal{A}]$. In Section 6.6.8, that statement is proved by $[\mathrm{HeadPair}^{\mathrm{I}\triangleright * \triangleright} \mathbin{@} \mathcal{A} \mathbin{@} \mathcal{A}]$. As we shall see, the tactic introduced in Section 6.8.6 adds the $[*^{\mathrm{I}\triangleright * \triangleright}]$ formula for referencing lemmas. That tactic also adds the $[\cdots \mathbin{@} \mathcal{A} \mathbin{@} \mathcal{A}]$ for instantiating the meta-quantifiers in [HeadPair]. The tactic uses unification and $[\cdots \gg (\mathcal{A} :: \mathcal{A})^{\mathrm{h}} = \mathcal{A}]$ to decide that both meta-quantifiers should be instantiated to an $[\mathcal{A}]$.

The third line uses [Transitivity] and the meta-modus-ponens operator $[* \triangleright *]$ to prove $[\mathcal{A} = \mathcal{A}]$. The tactic defined in Section 6.8.6 translates a meta-modus-ponens into a modus operation followed by a cut (c.f. the expanded proof in Section 6.6.8).

The tactic can expand modus ponens $[* \triangleright *]$ and modus probans $[* \mathbin{|\!\triangleright} *]$. Modus ponens says that if $[\mathcal{A}]$ is proved and $[\mathcal{A}]$ infers $[\mathcal{B}]$ then we may conclude $[\mathcal{B}]$. Modus probans ("probans" for "approve") says that if $[\mathcal{A}]$ is known to evaluate to $[\mathsf{T}]$ and the side condition $[\mathcal{A}]$ endorses $[\mathcal{B}]$ then we may conclude $[\mathcal{B}]$.

The tactic eliminates side conditions using modus probans when told to do so using $[* \mathbin{|\!\triangleright} *]$ and otherwise eliminates them using the $[*^{\mathrm{V}}]$ sequent operation which actually evaluates the side condition. Modus probans is useful in situations where a lemma assumes some side condition to hold. Assumed side-conditions are introduced in proofs using a Side-condition operation which is similar to the Premise used in Line 3 of the proof of $[\mathrm{Commutativity_1}]$ stated in a moment.

Line 3 above contains two references to Line 2. Each reference expands into the conclusion of Line 2 at macro expansion time. Hence, Line 3 reads $[\mathrm{Transitivity} \triangleright (\mathcal{A} :: \mathcal{A})^{\mathrm{h}} = \mathcal{A} \triangleright (\mathcal{A} :: \mathcal{A})^{\mathrm{h}} = \mathcal{A} \gg \mathcal{A} = \mathcal{A}]$ when the proof tactic is invoked.

Now let us turn to the proof of $[\mathrm{Commutativity_1}]$:

$[\mathrm{T_E}$ **lemma** $\mathrm{Commutativity_1}$: $\forall \mathcal{A}$: $\forall \mathcal{B}$: $\mathcal{A} = \mathcal{B} \vdash \mathcal{B} = \mathcal{A}]$[287]

$\mathrm{T_E}$ **proof of** $\mathrm{Commutativity_1}$:

L01:	Arbitrary $\gg$	$\mathcal{A}$	;
L02:	Arbitrary $\gg$	$\mathcal{B}$	;
L03:	Premise $\gg$	$\mathcal{A} = \mathcal{B}$	;
L04:	$\mathrm{Reflexivity_1} \gg$	$\mathcal{A} = \mathcal{A}$	;
L05:	Transitivity $\triangleright$ L03 $\triangleright$ L04 $\gg$	$\mathcal{B} = \mathcal{A}$	$\square$

The proof above only contains one new thing: Line 3 assumes $[\mathcal{A} = \mathcal{B}]$.

[287] $[\mathrm{Commutativity_1} \overset{\mathrm{pyk}}{=}$ "tactic commutativity"$]$

6.8.3 The "tactic" aspect

We now return to the actual implementation of proof tactics and the proof expander. We make the [<tactic>] aspect self-evaluating and use [tactic] to denote it:

$$[\text{<tactic>} \doteq \lceil \text{<tactic>} \rceil]^{288}$$

$$[\text{tactic} \overset{\text{msg}}{=} \text{<tactic>}]^{289}$$

For convenience, we define a construct for making tactic definitions:

$$[[\mathsf{x} \overset{\text{tactic}}{=} \mathsf{y}] \doteq [(\mathsf{x})^{\mathbf{p}} \overset{\text{tactic}}{\rightarrow} \mathsf{y}]]^{290}$$

6.8.4 The proof expander

The proof expander $[\mathcal{P}(\mathsf{t},\mathsf{s},\mathsf{c})]$ proof expands the term [t] for the *proof state* [s] and the cache [c] and returns the result as a semitagged map. The untagged version $[\mathcal{U}(\mathcal{P}(\mathsf{t},\mathsf{s},\mathsf{c}))]$ is the expansion itself.

The proof expander $[\mathcal{P}(\mathsf{t},\mathsf{s},\mathsf{c})]$ differs from the macro expander $[\mathcal{M}(\mathsf{t},\mathsf{s},\mathsf{c})]$ in that it has no special treatment for page symbols and in that it uses the [tactic] aspect instead of the [macro] aspect.

When proofs are expanded, they are first macro expanded and then proof expanded. Macro expansion is done iteratively until the codex reaches a fixed point whereas proof expansion is done only once. Furthermore, the result of macro expansion is kept in the codex whereas the result of proof expansion is discarded as soon as a proof is checked. Hence, proof expansion is a momentary burden to the computers memory whereas macro expansion is chronic.

The definition of $[\mathcal{P}(\mathsf{t},\mathsf{s},\mathsf{c})]$ is almost identical to that of $[\mathcal{M}(\mathsf{t},\mathsf{s},\mathsf{c})]$. But it is easier to express since we can use macros like the 'let' macro when defining proof expansion. For obvious reasons, macros cannot be used when defining the notion of macro expansion. The definition of $[\mathcal{P}(\mathsf{t},\mathsf{s},\mathsf{c})]$ reads:

$$[\mathcal{P}(\mathsf{t},\mathsf{s},\mathsf{c}) \doteq \mathsf{s}!$$
$$\mathbf{let}\ \mathsf{d} = \mathbf{aspect}(\text{<tactic>},\mathsf{t},\mathsf{c})\ \mathbf{in}$$
$$\mathbf{if}\ \mathsf{d}\ \mathbf{then}\ \mathsf{t}^{\mathrm{h}} :: \mathcal{P}^*(\mathsf{t}^{\mathrm{t}},\mathsf{s},\mathsf{c})\ \mathbf{else}$$
$$\mathcal{U}^{\mathrm{M}}(\mathcal{E}(\mathsf{d}^3,\mathsf{T},\mathsf{c})\ \text{`}\ \mathsf{t}\ \text{`}\ \mathsf{s}\ \text{`}\ \mathsf{c})]^{291}$$

$$[\mathcal{P}^*(\mathsf{t},\mathsf{s},\mathsf{c}) \doteq \mathsf{s}!\mathsf{c}!\mathrm{If}(\mathsf{t},\mathsf{T},\mathcal{P}(\mathsf{t}^{\mathrm{h}},\mathsf{s},\mathsf{c}) :: \mathcal{P}^*(\mathsf{t}^{\mathrm{t}},\mathsf{s},\mathsf{c}))]^{292}$$

---

[288] [<tactic> $\overset{\text{pyk}}{=}$ "the tactic aspect"]

[289] [tactic $\overset{\text{pyk}}{=}$ "tactic"]

[290] [[x $\overset{\text{tactic}}{=}$ y] $\overset{\text{pyk}}{=}$ "tactic define * as * end define"]

[291] $[\mathcal{P}(\mathsf{t},\mathsf{s},\mathsf{c}) \overset{\text{pyk}}{=}$ "proof expand * state * cache * end expand"]

[292] $[\mathcal{P}^*(\mathsf{t},\mathsf{s},\mathsf{c}) \overset{\text{pyk}}{=}$ "proof expand list * state * cache * end expand"]

6.8.5 The initial proof state

Proof states have exactly the same format as macro states.

The *initial proof state* $[\mathsf{p}_0]$ is useful for passing as the second parameter to $[\mathcal{P}(\mathsf{t},\mathsf{s},\mathsf{c})]$. $[\mathsf{p}_0]$ is a pair whose head is the proof expander itself and whose tail is left blank. The definition of $[\mathsf{p}_0]$ reads:

$$[\mathsf{p}_0 \overset{\text{val}}{\rightarrow} \mathcal{M}(\lambda\mathsf{t}.\lambda\mathsf{s}.\lambda\mathsf{c}.\mathcal{P}(\mathsf{t},\mathsf{s},\mathsf{c})) :: \mathsf{T}]^{293}$$

The function $[\tilde{\mathcal{M}}(\mathsf{t},\mathsf{s},\mathsf{c})]$ defined previously macro expands the term $[\mathsf{t}]$ using the macro expander embedded in the macro state $[\mathsf{s}]$ using the cache $[\mathsf{c}]$. Since proof states have exactly the same syntax and semantics as macro states, we shall take the liberty to use $[\tilde{\mathcal{M}}(\mathsf{t},\mathsf{s},\mathsf{c})]$ for proof states also.

6.8.6 The conclusion tactic

The *conclusion tactic* $[\mathsf{x} \gg \mathsf{y}]$ constructs a proof of $[\mathsf{y}]$ from the partial proof $[\mathsf{x}]$ as described in Section 6.8.2.

Among other, the tactic tactic expands modus ponens $[\mathsf{x} \triangleright \mathsf{y}]$ and modus probans $[\mathsf{x} \mathbb{\triangleright} \mathsf{y}]$ which we make self-evaluating:

$$[\mathsf{x} \triangleright \mathsf{y} \doteq \langle\lceil \mathsf{x} \triangleright \mathsf{y} \rceil^{\mathrm{R}}, \mathsf{x}, \mathsf{y}\rangle]^{294}$$

$$[\mathsf{x} \mathbb{\triangleright} \mathsf{y} \doteq \langle\lceil \mathsf{x} \mathbb{\triangleright} \mathsf{y} \rceil^{\mathrm{R}}, \mathsf{x}, \mathsf{y}\rangle]^{295}$$

The conclusion tactic is defined thus:

$$[\mathsf{x} \gg \mathsf{y} \overset{\text{tactic}}{=} \lambda\mathsf{t}.\lambda\mathsf{s}.\lambda\mathsf{c}.\mathrm{conclude}_1(\mathsf{t},\mathsf{c})]^{296}$$

$$[\mathrm{conclude}_1(\mathsf{t},\mathsf{c}) \doteq$$
$$\textbf{let } \mathsf{r} = \mathrm{conclude}_2(\mathsf{t}^1,\mathsf{t}^2,\mathsf{c}) \textbf{ in}$$
$$\textbf{if } \mathsf{r}^{\mathrm{c}} \textbf{ then } \mathrm{error}(\text{“Unification failed”},\mathsf{t}) \textbf{ else } \mathsf{r}]^{297}$$

$$[\mathrm{conclude}_2(\mathsf{a},\mathsf{t},\mathsf{c}) \doteq \mathsf{t}!$$
$$\textbf{if } \mathsf{a} \overset{\mathrm{r}}{=} \lceil \mathsf{x} \triangleright \mathsf{y} \rceil \textbf{ then } \mathrm{conclude}_2(\mathsf{a}^1, \mathsf{a}\text{-color}(\mathsf{t} \triangleright \mathsf{a}^2),\mathsf{c}) \textbf{ else}$$
$$\textbf{if } \mathsf{a} \overset{\mathrm{r}}{=} \lceil \mathsf{x} \mathbb{\triangleright} \mathsf{y} \rceil \textbf{ then } \mathrm{conclude}_2(\mathsf{a}^1, \mathsf{a}\text{-color}(\mathsf{t} \mathbb{\triangleright} \mathsf{a}^2),\mathsf{c}) \textbf{ else}$$
$$\textbf{if } \mathsf{a} \overset{\mathrm{r}}{=} \lceil \mathsf{x} @ \mathsf{y} \rceil \textbf{ then } \mathrm{conclude}_2(\mathsf{a}^1, \mathsf{a}\text{-color}(\mathsf{t} @ \mathsf{a}^2),\mathsf{c}) \textbf{ else}$$
$$\textbf{if aspect}(\text{<proof>},\mathsf{a},\mathsf{c}) \textbf{ then } \mathrm{error}(\text{“Lemma expected”},\mathsf{a}) \textbf{ else}$$
$$\textbf{let } \mathsf{d} = \textbf{aspect}(\text{<stmt>},\mathsf{a},\mathsf{c}) \textbf{ in}$$
$$\mathrm{conclude}_3(\mathsf{a}\text{-color}(\mathrm{conclude}_4(\mathsf{a}^{\mathrm{I}\triangleright * \triangleright}, \mathsf{d}^{32})),\mathsf{t},\mathrm{parm}(\mathsf{d}^{32},\mathsf{T},1),\mathsf{T})]^{298}$$

---

[293] $[\mathsf{p}_0 \overset{\text{pyk}}{=} \text{“proof state”}]$

[294] $[\mathsf{x} \triangleright \mathsf{y} \overset{\text{pyk}}{=} \text{“} * \text{ modus ponens } * \text{”}]$

[295] $[\mathsf{x} \mathbb{\triangleright} \mathsf{y} \overset{\text{pyk}}{=} \text{“} * \text{ modus probans } * \text{”}]$

[296] $[\mathsf{x} \gg \mathsf{y} \overset{\text{pyk}}{=} \text{“} * \text{ conclude } * \text{”}]$

[297] $[\mathrm{conclude}_1(\mathsf{t},\mathsf{c}) \overset{\text{pyk}}{=} \text{“conclude one } * \text{ cache } * \text{ end conclude”}]$

[298] $[\mathrm{conclude}_2(\mathsf{a},\mathsf{t},\mathsf{c}) \overset{\text{pyk}}{=} \text{“conclude two } * \text{ proves } * \text{ cache } * \text{ end conclude”}]$

$$
\begin{array}{l}
[\mathrm{conclude}_3(\mathsf{a},\mathsf{t},\mathsf{l},\mathsf{s}) \doteq \mathsf{a}!\mathsf{t}!\mathsf{l}!\mathsf{s}! \\
\textbf{if } \mathsf{l} \stackrel{\mathrm{r}}{=} \lceil \mathsf{x} \vdash \mathsf{y} \rceil \textbf{ then} \\
\mathsf{t} \stackrel{\mathrm{r}}{=} \lceil \mathsf{x} \rhd \mathsf{y} \rceil \left\{ \begin{array}{l} \mathrm{conclude}_3(\mathsf{a}^{\rhd}, \mathsf{t}^1, \mathsf{l}^2, \mathrm{unify}(\mathsf{l}^1 = \mathsf{t}^2, \mathsf{s})) \\ \mathrm{conclude}_3(\mathsf{a}^{\rhd}, \mathsf{t}, \mathsf{l}^2, \mathsf{s}) \end{array} \right. \textbf{else} \\
\textbf{if } \mathsf{l} \stackrel{\mathrm{r}}{=} \lceil \mathsf{x} \Vdash \mathsf{y} \rceil \textbf{ then} \\
\mathsf{t} \stackrel{\mathrm{r}}{=} \lceil \mathsf{x} \Rrightarrow \mathsf{y} \rceil \left\{ \begin{array}{l} \mathrm{conclude}_3(\mathsf{a}^{\rhd}, \mathsf{t}^1, \mathsf{l}^2, \mathrm{unify}(\mathsf{l}^1 = \mathsf{t}^2, \mathsf{s})) \\ \mathrm{conclude}_3(\mathsf{a}^{\mathrm{V}}, \mathsf{t}, \mathsf{l}^2, \mathsf{s}) \end{array} \right. \textbf{else} \\
\textbf{if } \mathsf{l} \stackrel{\mathrm{r}}{=} \lceil \forall \mathsf{x}\colon \mathsf{y} \rceil \textbf{ then} \\
\mathsf{t} \stackrel{\mathrm{r}}{=} \lceil \mathsf{x} @ \mathsf{y} \rceil \left\{ \begin{array}{l} \mathrm{conclude}_3(\mathsf{a} @ \mathsf{t}^2, \mathsf{t}^1, \mathsf{l}^2, \mathrm{unify}(\mathsf{l}^1 = \mathsf{t}^2, \mathsf{s})) \\ \mathrm{conclude}_3(\mathsf{a} @ \mathsf{l}^1, \mathsf{t}, \mathsf{l}^2, \mathsf{s}) \end{array} \right. \textbf{else} \\
\textbf{let } \mathsf{s} = \mathrm{unify}(\mathsf{l} = \mathsf{t}, \mathsf{s}) \textbf{ in} \\
\textbf{if } \mathsf{s}^{\mathrm{c}} \textbf{ then } \mathsf{s} \textbf{ else} \\
\mathrm{inst}(\mathsf{a},\mathsf{s})]^{299}
\end{array}
$$

$$
\begin{array}{l}
[\mathrm{conclude}_4(\mathsf{a},\mathsf{l}) \doteq \mathsf{a}!\mathsf{l}! \\
\textbf{if } \neg \mathsf{l} \stackrel{\mathrm{r}}{=} \lceil \forall \mathsf{x}\colon \mathsf{y} \rceil \textbf{ then } \mathsf{a} \textbf{ else} \\
\textbf{let } \mathsf{v} = \langle \lceil \underline{*} \rceil^{\mathrm{R}}, \mathsf{l}^1 \rangle \textbf{ in } \forall \mathsf{v}\colon \mathrm{conclude}_4(\mathsf{a} @ \mathsf{v}, \mathsf{l}^2)]^{300}
\end{array}
$$

6.8.7 Proof constructors

The following macros make it easy to construct medium level proofs:

$$[\mathsf{t} \textbf{ proof of } \mathsf{s}\colon \mathsf{p} \doteq [\textbf{Proof of } \mathsf{s}\colon \lambda \mathsf{c}.\lambda \mathsf{x}.\mathcal{P}(\lceil \mathsf{t} \vdash \mathsf{p} \rceil, \mathsf{p}_0, \mathsf{c})]]^{301}$$

$$[\mathrm{Line}\, \mathsf{l} : \mathsf{a} \gg \mathsf{i}; \mathsf{p} \doteq (\mathsf{a} \gg \mathsf{i}; \textbf{let } \mathsf{l} \doteq \mathsf{i} \ \textbf{ in } \ \mathsf{p})]^{302}$$

$$[\mathrm{Last\ line}\, \mathsf{a} \gg \mathsf{i}\, \Box \doteq (\mathsf{a} \gg \mathsf{i})]^{303}$$

$$[\mathrm{Line}\, \mathsf{l} : \mathrm{Premise} \gg \mathsf{i}; \mathsf{p} \doteq (\mathsf{i} \vdash \textbf{let } \mathsf{l} \doteq \mathsf{i} \ \textbf{ in } \ \mathsf{p})]^{304}$$

$$[\mathrm{Line}\, \mathsf{l} : \text{Side-condition} \gg \mathsf{i}; \mathsf{p} \doteq (\mathsf{i} \Vdash \textbf{let } \mathsf{l} \doteq \mathsf{i} \ \textbf{ in } \ \mathsf{p})]^{305}$$

$$[\mathrm{Arbitrary} \gg \mathsf{i}; \mathsf{p} \doteq (\forall \mathsf{i}\colon \mathsf{p})]^{306}$$

$$[\mathrm{Local} \gg \mathsf{a} = \mathsf{i}; \mathsf{p} \doteq (\textbf{let } \mathsf{a} \doteq \mathsf{i} \ \textbf{ in } \ \mathsf{p})]^{307}$$

---

[299] $[\mathrm{conclude}_3(\mathsf{a},\mathsf{t},\mathsf{l},\mathsf{s}) \stackrel{\mathrm{pyk}}{=}$ "conclude three $*$ proves $*$ lemma $*$ substitution $*$ end conclude"]

[300] $[\mathrm{conclude}_4(\mathsf{a},\mathsf{l}) \stackrel{\mathrm{pyk}}{=}$ "conclude four $*$ lemma $*$ end conclude"]

[301] $[\mathsf{t} \textbf{ proof of } \mathsf{l} : \mathsf{p} \stackrel{\mathrm{pyk}}{=}$ "$*$ proof of $*$ reads $*$"]

[302] $[\mathrm{Line}\, \mathsf{l} : \mathsf{a} \gg \mathsf{i}; \mathsf{p} \stackrel{\mathrm{pyk}}{=}$ "line $*$ because $*$ indeed $*$ end line $*$"]

[303] $[\mathrm{Last\ line}\, \mathsf{a} \gg \mathsf{i}\, \Box \stackrel{\mathrm{pyk}}{=}$ "because $*$ indeed $*$ qed"]

[304] $[\mathrm{Line}\, \mathsf{l} : \mathrm{Premise} \gg \mathsf{i}; \mathsf{p} \stackrel{\mathrm{pyk}}{=}$ "line $*$ premise $*$ end line $*$"]

[305] $[\mathrm{Line}\, \mathsf{l} : \text{Side-condition} \gg \mathsf{i}; \mathsf{p} \stackrel{\mathrm{pyk}}{=}$ "line $*$ side condition $*$ end line $*$"]

[306] $[\mathrm{Arbitrary} \gg \mathsf{i}; \mathsf{p} \stackrel{\mathrm{pyk}}{=}$ "arbitrary $*$ end line $*$"]

[307] $[\mathrm{Local} \gg \mathsf{u} = \mathsf{v}; \mathsf{p} \stackrel{\mathrm{pyk}}{=}$ "locally define $*$ as $*$ end line $*$"]

Some of the constructs look different when used in proofs. As an example, the "Arbitrary" construct above gets a line number when used in a proof (or, more precisely, when typeset using the tex aspect instead of the tex name aspect).

Line numbers in proofs are generated automatically. As an example, one may write

> line ell a because tactic reflexivity indeed meta a math equal meta a end line ...

in a pyk source text to get a numbered proof line saying [Reflexivity$_1 \gg \mathcal{A} = \mathcal{A}$]. The line number is assigned automatically, and afterwards references to "ell a" will refer to that line number.

A Constructs

A.1 Pyk definitions

This appendix contains pyk definitions which the author has not yet turned into footnotes.

For convenience, we introduce a variable named [$*$] which we use in pyk, TEX, and priority definitions where parameter names are ignored.

[$* \overset{\text{pyk}}{=}$ "x"]
[$* \text{ ' } * \overset{\text{pyk}}{=}$ "$*$ apply $*$"]
[$* \text{ ` } * \overset{\text{pyk}}{=}$ "$*$ tagged apply $*$"]
[$\lambda * .* \overset{\text{pyk}}{=}$ "lambda $*$ dot $*$"]
[$\Lambda * \overset{\text{pyk}}{=}$ "tagging $*$"]
[$\mathsf{T} \overset{\text{pyk}}{=}$ "true"]
[$\text{if}(*,*,*) \overset{\text{pyk}}{=}$ "if $*$ then $*$ else $*$ end if"]
[val $\overset{\text{pyk}}{=}$ "value"]
[claim $\overset{\text{pyk}}{=}$ "claim"]
[$[* \overset{*}{\Rightarrow} *] \overset{\text{pyk}}{=}$ "introduce $*$ of $*$ as $*$ end introduce"]
[$\bot \overset{\text{pyk}}{=}$ "bottom"]
[$\text{f}(*) \overset{\text{pyk}}{=}$ "function f of $*$ end function"]
[$(*)^{\text{I}} \overset{\text{pyk}}{=}$ "identity $*$ end identity"]
[$\mathsf{F} \overset{\text{pyk}}{=}$ "false"]
[$\underline{0} \overset{\text{pyk}}{=}$ "untagged zero"]
[$\underline{1} \overset{\text{pyk}}{=}$ "untagged one"]
[$\underline{2} \overset{\text{pyk}}{=}$ "untagged two"]
[$\underline{3} \overset{\text{pyk}}{=}$ "untagged three"]
[$\underline{4} \overset{\text{pyk}}{=}$ "untagged four"]

[$\underline{5} \overset{\text{pyk}}{=}$ “untagged five”]
[$\underline{6} \overset{\text{pyk}}{=}$ “untagged six”]
[$\underline{7} \overset{\text{pyk}}{=}$ “untagged seven”]
[$\underline{8} \overset{\text{pyk}}{=}$ “untagged eight”]
[$\underline{9} \overset{\text{pyk}}{=}$ “untagged nine”]
[$0 \overset{\text{pyk}}{=}$ “zero”]
[$1 \overset{\text{pyk}}{=}$ “one”]
[$2 \overset{\text{pyk}}{=}$ “two”]
[$3 \overset{\text{pyk}}{=}$ “three”]
[$4 \overset{\text{pyk}}{=}$ “four”]
[$5 \overset{\text{pyk}}{=}$ “five”]
[$6 \overset{\text{pyk}}{=}$ “six”]
[$7 \overset{\text{pyk}}{=}$ “seven”]
[$8 \overset{\text{pyk}}{=}$ “eight”]
[$9 \overset{\text{pyk}}{=}$ “nine”]
[$\mathsf{a} \overset{\text{pyk}}{=}$ “var a”]
[$\mathsf{b} \overset{\text{pyk}}{=}$ “var b”]
[$\mathsf{c} \overset{\text{pyk}}{=}$ “var c”]
[$\mathsf{d} \overset{\text{pyk}}{=}$ “var d”]
[$\mathsf{e} \overset{\text{pyk}}{=}$ “var e”]
[$\mathsf{f} \overset{\text{pyk}}{=}$ “var f”]
[$\mathsf{g} \overset{\text{pyk}}{=}$ “var g”]
[$\mathsf{h} \overset{\text{pyk}}{=}$ “var h”]
[$\mathsf{i} \overset{\text{pyk}}{=}$ “var i”]
[$\mathsf{j} \overset{\text{pyk}}{=}$ “var j”]
[$\mathsf{k} \overset{\text{pyk}}{=}$ “var k”]
[$\mathsf{l} \overset{\text{pyk}}{=}$ “var l”]
[$\mathsf{m} \overset{\text{pyk}}{=}$ “var m”]
[$\mathsf{n} \overset{\text{pyk}}{=}$ “var n”]
[$\mathsf{o} \overset{\text{pyk}}{=}$ “var o”]
[$\mathsf{p} \overset{\text{pyk}}{=}$ “var p”]
[$\mathsf{q} \overset{\text{pyk}}{=}$ “var q”]
[$\mathsf{r} \overset{\text{pyk}}{=}$ “var r”]
[$\mathsf{s} \overset{\text{pyk}}{=}$ “var s”]
[$\mathsf{t} \overset{\text{pyk}}{=}$ “var t”]

$[\mathsf{u} \overset{\text{pyk}}{=} \text{“var u”}]$

$[\mathsf{v} \overset{\text{pyk}}{=} \text{“var v”}]$

$[\mathsf{w} \overset{\text{pyk}}{=} \text{“var w”}]$

$[(*)^{\mathrm{M}} \overset{\text{pyk}}{=} \text{“tagged parenthesis } * \text{ end tagged”}]$

$[\mathrm{If}(*,*,*) \overset{\text{pyk}}{=} \text{“tagged if } * \text{ then } * \text{ else } * \text{ end if”}]$

$[\mathrm{array}\{*\} * \text{end array} \overset{\text{pyk}}{=} \text{“array } * \text{ is } * \text{ end array”}]$

$[\mathsf{l} \overset{\text{pyk}}{=} \text{“left”}]$

$[\mathsf{c} \overset{\text{pyk}}{=} \text{“center”}]$

$[\mathsf{r} \overset{\text{pyk}}{=} \text{“right”}]$

$[\mathrm{empty} \overset{\text{pyk}}{=} \text{“empty”}]$

$[\langle * \mid * := * \rangle \overset{\text{pyk}}{=} \text{“substitute } * \text{ set } * \text{ to } * \text{ end substitute”}]$

$[\mathcal{M}(*) \overset{\text{pyk}}{=} \text{“map tag } * \text{ end tag”}]$

$[\mathbf{apply}(*,*) \overset{\text{pyk}}{=} \text{“apply } * \text{ to } * \text{ end apply”}]$

$[\mathbf{apply}_1(*,*) \overset{\text{pyk}}{=} \text{“apply one } * \text{ to } * \text{ end apply”}]$

$[*^{\mathrm{H}} \overset{\text{pyk}}{=} \text{“} * \text{ raw head”}]$

$[*^{\mathrm{T}} \overset{\text{pyk}}{=} \text{“} * \text{ raw tail”}]$

$[*^{\mathrm{c}} \overset{\text{pyk}}{=} \text{“} * \text{ is cardinal”}]$

$[*^{\mathrm{d}} \overset{\text{pyk}}{=} \text{“} * \text{ is data”}]$

$[*^{\mathrm{U}} \overset{\text{pyk}}{=} \text{“} * \text{ cardinal untag”}]$

$[*^{\mathrm{h}} \overset{\text{pyk}}{=} \text{“} * \text{ head”}]$

$[*^{\mathrm{t}} \overset{\text{pyk}}{=} \text{“} * \text{ tail”}]$

$[*^{\mathrm{s}} \overset{\text{pyk}}{=} \text{“} * \text{ is singular”}]$

$[*^{\mathrm{B}} \overset{\text{pyk}}{=} \text{“} * \text{ boolean retract”}]$

$[*^{\mathrm{C}} \overset{\text{pyk}}{=} \text{“} * \text{ cardinal retract”}]$

$[*^{\mathrm{M}} \overset{\text{pyk}}{=} \text{“} * \text{ tagged retract”}]$

$[\tilde{\mathcal{U}}(*) \overset{\text{pyk}}{=} \text{“raw map untag } * \text{ end untag”}]$

$[\mathcal{U}(*) \overset{\text{pyk}}{=} \text{“map untag } * \text{ end untag”}]$

$[\mathcal{U}^{\mathrm{M}}(*) \overset{\text{pyk}}{=} \text{“normalizing untag } * \text{ end untag”}]$

$[* \mathbin{\therefore} * \overset{\text{pyk}}{=} \text{“} * \text{ raw pair } *\text{”}]$

$[* \mathbin{\underline{\therefore}} * \overset{\text{pyk}}{=} \text{“} * \text{ eager pair } *\text{”}]$

$[* \mathbin{::} * \overset{\text{pyk}}{=} \text{“} * \text{ tagged pair } *\text{”}]$

$[* \mathbin{\underline{+2*}} * \overset{\text{pyk}}{=} \text{“} * \text{ untagged double } *\text{”}]$

$[* :: * \overset{\text{pyk}}{=} \text{“} * \text{ pair } *\text{”}]$

$[* +2* * \overset{\text{pyk}}{=} \text{“} * \text{ double } *\text{”}]$

$[* = * \overset{\text{pyk}}{=} \text{“} * \text{ math equal } *\text{”}]$

$[* \overset{D}{\approx} * \overset{pyk}{=} \text{“}* \text{ data equal } *\text{”}]$

$[* \overset{C}{\approx} * \overset{pyk}{=} \text{“}* \text{ cardinal equal } *\text{”}]$

$[* \overset{P}{\approx} * \overset{pyk}{=} \text{“}* \text{ peano equal } *\text{”}]$

$[* \approx * \overset{pyk}{=} \text{“}* \text{ tagged equal } *\text{”}]$

$[* \overset{+}{\rightarrow} * \overset{pyk}{=} \text{“}* \text{ reduce to } *\text{”}]$

$[\neg * \overset{pyk}{=} \text{“not } *\text{”}]$

$[* \wedge * \overset{pyk}{=} \text{“}* \text{ and } *\text{”}]$

$[* \vee * \overset{pyk}{=} \text{“}* \text{ or } *\text{”}]$

$[* \parallel * \overset{pyk}{=} \text{“}* \text{ parallel } *\text{”}]$

$[* \overset{B}{\approx} * \overset{pyk}{=} \text{“}* \text{ boolean equal } *\text{”}]$

$[* : * \overset{pyk}{=} \text{“}* \text{ guard } *\text{”}]$

$[*!* \overset{pyk}{=} \text{“}* \text{ tagged guard } *\text{”}]$

$[* \left\{ \begin{array}{l} * \\ * \end{array} \right. \overset{pyk}{=} \text{“}* \text{ select } * \text{ else } * \text{ end select”}]$

$[*\&* \overset{pyk}{=} \text{“}* \text{ tab } *\text{”}]$

$[*\backslash\backslash* \overset{pyk}{=} \text{“}* \text{ row}$

$*\text{”}]$

$[\text{macro} \overset{pyk}{=} \text{“macro”}]$

$[\text{"value"} \overset{pyk}{=} \text{“value aspect”}]$

$[\text{"macro"} \overset{pyk}{=} \text{“macro aspect”}]$

$[\mathcal{E}(*,*,*) \overset{pyk}{=} \text{“eval } * \text{ stack } * \text{ cache } * \text{ end eval”}]$

$[\mathcal{E}_2(*,*,*,*,*) \overset{pyk}{=} \text{“eval two } * \text{ ref } * \text{ id } * \text{ stack } * \text{ cache } * \text{ end eval”}]$

$[\mathcal{E}_3(*,*,*,*) \overset{pyk}{=} \text{“eval three } * \text{ function } * \text{ stack } * \text{ cache } * \text{ end eval”}]$

$[\mathcal{E}_4(*,*,*,*) \overset{pyk}{=} \text{“eval four } * \text{ arguments } * \text{ stack } * \text{ cache } * \text{ end eval”}]$

$[\mathbf{lookup}(*,*,*) \overset{pyk}{=} \text{“lookup } * \text{ stack } * \text{ default } * \text{ end lookup”}]$

$[\mathbf{abstract}(*,*,*,*) \overset{pyk}{=} \text{“abstract } * \text{ term } * \text{ stack } * \text{ cache } * \text{ end abstract”}]$

$[*^{a} \overset{pyk}{=} \text{“}* \text{ is atomic”}]$

$[\lceil * \rceil \overset{pyk}{=} \text{“quote } * \text{ end quote”}]$

$[*0 \overset{pyk}{=} \text{“}* \text{ bit nil”}]$

$[*1 \overset{pyk}{=} \text{“}* \text{ bit one”}]$

$[0\text{b} \overset{pyk}{=} \text{“binary”}]$

$[* \overset{t}{=} * \overset{pyk}{=} \text{“}* \text{ term equal } *\text{”}]$

$[* \overset{t^*}{=} * \overset{pyk}{=} \text{“}* \text{ term list equal } *\text{”}]$

$[*[\,*\,] \overset{pyk}{=} \text{“}* \text{ assoc } * \text{ end assoc”}]$

$[\mathcal{M}(*,*,*) \overset{pyk}{=} \text{“expand } * \text{ state } * \text{ cache } * \text{ end expand”}]$

$[\mathcal{M}_2(*,*,*,*) \overset{\text{pyk}}{=} \text{“expand two } * \text{ definition } * \text{ state } * \text{ cache } * \text{ end expand”}]$

$[\mathcal{M}^*(*,*,*) \overset{\text{pyk}}{=} \text{“expand list } * \text{ state } * \text{ cache } * \text{ end expand”}]$

$[\mathrm{s}_0 \overset{\text{pyk}}{=} \text{“macro state”}]$

$[\mathbf{zip}(*,*) \overset{\text{pyk}}{=} \text{“zip } * \text{ with } * \text{ end zip”}]$

$[\mathbf{assoc}_1(*,*,*) \overset{\text{pyk}}{=} \text{“assoc one } * \text{ address } * \text{ index } * \text{ end assoc”}]$

$[*_{\text{-}}\{*\} \overset{\text{pyk}}{=} \text{“} * \text{ sub } * \text{ end sub”}]$

$[(*)^{\mathbf{p}} \overset{\text{pyk}}{=} \text{“protect } * \text{ end protect”}]$

$[\mathrm{self} \overset{\text{pyk}}{=} \text{“self”}]$

$([[* \ddot{=} *] \overset{\text{pyk}}{\rightarrow} \text{macrodefine} * \text{as} * \text{enddefine}])^{\mathbf{p}}$

$[[* \doteq *] \overset{\text{pyk}}{=} \text{“value define } * \text{ as } * \text{ end define”}]$

$[[* \acute{=} *] \overset{\text{pyk}}{=} \text{“intro define } * \text{ as } * \text{ end define”}]$

$[[* \overset{\text{pyk}}{=} *] \overset{\text{pyk}}{=} \text{“pyk define } * \text{ as } * \text{ end define”}]$

$[[* \overset{\text{tex}}{=} *] \overset{\text{pyk}}{=} \text{“tex define } * \text{ as } * \text{ end define”}]$

$[[* \overset{\text{name}}{=} *] \overset{\text{pyk}}{=} \text{“tex name define } * \text{ as } * \text{ end define”}]$

$[(*) \overset{\text{pyk}}{=} \text{“parenthesis } * \text{ end parenthesis”}]$

$[\tilde{\mathcal{M}}_1 \overset{\text{pyk}}{=} \text{“macro define one”}]$

$[\tilde{\mathcal{M}}_2(*) \overset{\text{pyk}}{=} \text{“macro define two } * \text{ end define”}]$

$[\tilde{\mathcal{M}}_3(*) \overset{\text{pyk}}{=} \text{“macro define three } * \text{ end define”}]$

$[\tilde{\mathcal{M}}_4(*,*,*,*) \overset{\text{pyk}}{=} \text{“macro define four } * \text{ state } * \text{ cache } * \text{ definition } * \text{ end define”}]$

$[\tilde{\mathcal{M}}(*,*,*) \overset{\text{pyk}}{=} \text{“state expand } * \text{ state } * \text{ cache } * \text{ end expand”}]$

$[\tilde{\mathcal{Q}}(*,*,*) \overset{\text{pyk}}{=} \text{“quote expand } * \text{ term } * \text{ stack } * \text{ end expand”}]$

$[\tilde{\mathcal{Q}}_2(*,*,*) \overset{\text{pyk}}{=} \text{“quote expand two } * \text{ term } * \text{ stack } * \text{ end expand”}]$

$[\tilde{\mathcal{Q}}_3(*,*,*,*) \overset{\text{pyk}}{=} \text{“quote expand three } * \text{ term } * \text{ stack } * \text{ value } * \text{ end expand”}]$

$[\tilde{\mathcal{Q}}^*(*,*,*) \overset{\text{pyk}}{=} \text{“quote expand star } * \text{ term } * \text{ stack } * \text{ end expand”}]$

A.2 TEX definitions

$[[\mathsf{x} \bowtie \mathsf{y}] \overset{\text{tex}}{=}$ “
```
[#1/tex name/tex.
\bowtie#2.
]”]
```

$[[\mathsf{y} \overset{\mathsf{x}}{\rightarrow} \mathsf{z}] \overset{\text{tex}}{=}$ “
```
[#2/tex name/tex.
\stackrel{#1.
}{\rightarrow}#3.
]”]
```

[pyk $\overset{\text{tex}}{=}$ “
\mathrm{pyk}”]

[x $\overset{\text{tex}}{=}$ “
\mathsf{x}”]

[y $\overset{\text{tex}}{=}$ “
\mathsf{y}”]

[z $\overset{\text{tex}}{=}$ “
\mathsf{z}”]

[tex $\overset{\text{tex}}{=}$ “
\mathrm{tex}”]

[name $\overset{\text{tex}}{=}$ “
\mathrm{name}”]

[x $\overset{\text{tex}}{=}$ “$#1.$”]

[x $\overset{\text{name}}{=}$ “
\ \$#1.\$\linebreak[0]\ ”]

[bracket x end bracket $\overset{\text{tex}}{=}$ “$[#1.]$”]

[bracket x end bracket $\overset{\text{name}}{=}$ “
\mbox{bracket $#1.$ end bracket}”]

[big bracket x end bracket $\overset{\text{tex}}{=}$ “$\left[#1.\right]$”]

[big bracket x end bracket $\overset{\text{name}}{=}$ “
\mbox{big bracket $#1.$ end bracket}”]

[**flush left** [x] $\overset{\text{tex}}{=}$ “
\begin {flushleft}#1.
\end {flushleft}”]

[**flush left** [x] $\overset{\text{name}}{=}$ “
\mathbf{flush\ left\ }[#1.
]”]

[x^0 $\overset{\text{tex}}{=}$ “#1.
{}^{0}”]

[x^1 $\overset{\text{tex}}{=}$ “#1.
{}^{1}”]

[x^2 $\overset{\text{tex}}{=}$ “#1.
{}^{2}”]

$[\mathsf{x}^3 \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad \{\}\hat{}\{3\}\text{”}]$

$[\mathsf{x}^4 \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad \{\}\hat{}\{4\}\text{”}]$

$[\mathsf{x}^5 \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad \{\}\hat{}\{5\}\text{”}]$

$[\mathsf{x}^6 \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad \{\}\hat{}\{6\}\text{”}]$

$[\mathsf{x}^7 \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad \{\}\hat{}\{7\}\text{”}]$

$[\mathsf{x}^8 \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad \{\}\hat{}\{8\}\text{”}]$

$[\mathsf{x}^9 \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad \{\}\hat{}\{9\}\text{”}]$

$[\mathsf{x} + \mathsf{y} \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad + \#2.\text{”}]$

$[\mathsf{x} +_0 \mathsf{y} \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad \backslash\text{mathop}\{+\_0\}\ \#2.\text{”}]$

$[\mathsf{x} +_1 \mathsf{y} \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad \backslash\text{mathop}\{+\_1\}\ \#2.\text{”}]$

$[\mathsf{x} < \mathsf{y} \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad < \#2.\text{”}]$

$[\mathsf{x} <' \mathsf{y} \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad <\text{'}\ \#2.\text{”}]$

$[\mathsf{x} \leq' \mathsf{y} \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad \backslash\text{le'}\ \#2.\text{”}]$

$[\mathsf{x} - \mathsf{y} \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad \text{- } \#2.\text{”}]$

$[\mathsf{x} -_0 \mathsf{y} \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad \backslash\text{mathop}\{\text{-}\_0\}\ \#2.\text{”}]$

$[\mathsf{x} -_1 \mathsf{y} \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad \backslash\text{mathop}\{\text{-}\_1\}\ \#2.\text{”}]$

$[x \cdot y \overset{\text{tex}}{=} \text{“\#1.}$
$\text{\cdot \#2.”}]$

$[x \cdot_0 y \overset{\text{tex}}{=} \text{“\#1.}$
$\text{\cdot\_0 \#2.”}]$

$[\text{bit}(x, y) \overset{\text{tex}}{=} \text{“}$
$\text{bit(\#1.}$
$\text{, \#2.}$
$\text{)”}]$

$[\text{bit}_1(x, y) \overset{\text{tex}}{=} \text{“}$
$\text{bit\_1(\#1.}$
$\text{, \#2.}$
$\text{)”}]$

$[x^{\text{r}} \overset{\text{tex}}{=} \text{“\#1.}$
$\text{\{\}\^{}\{r\}”}]$

$[x^{\text{i}} \overset{\text{tex}}{=} \text{“\#1.}$
$\text{\{\}\^{}\{i\}”}]$

$[x^{\text{d}} \overset{\text{tex}}{=} \text{“\#1.}$
$\text{\{\}\^{}\{d\}”}]$

$[x^{\text{R}} \overset{\text{tex}}{=} \text{“\#1.}$
$\text{\{\}\^{}\{R\}”}]$

$[x \overset{\text{r}}{=} y \overset{\text{tex}}{=} \text{“\#1.}$
$\text{\stackrel\{r\}\{=\} \#2.”}]$

$[\text{identifier}(x) \overset{\text{tex}}{=} \text{“}$
$\text{identifier(\#1.}$
$\text{)”}]$

$[\text{identifier}_1(x, y) \overset{\text{tex}}{=} \text{“}$
$\text{identifier\_\{1\}(\#1.}$
$\text{, \#2.}$
$\text{)”}]$

$[a[i \to v] \overset{\text{tex}}{=} \text{“\#1.}$
$\text{[\#2.}$
$\text{\{\rightarrow\} \#3.}$
$\text{]”}]$

$[\text{array-plus}(x, y) \overset{\text{tex}}{=} \text{“}$
$\text{array\mbox\{-\}\linebreak[0]plus(\#1.}$
$\text{, \#2.}$
$\text{)”}]$

[array-remove(i, a, l) $\stackrel{\text{tex}}{=}$ “
array\mbox{-}\linebreak[0]remove(#1.
, #2.
, #3.
)”]

[array-put(i, v, a, l) $\stackrel{\text{tex}}{=}$ “
array\mbox{-}\linebreak[0]put(#1.
, #2.
, #3.
, #4.
)”]

[array-add(i, v, i′, v′, l) $\stackrel{\text{tex}}{=}$ “
array\mbox{-}\linebreak[0]add(#1.
, #2.
, #3.
, #4.
, #5.
)”]

[a[i⇒v] $\stackrel{\text{tex}}{=}$ “#1.
[#2.
{\Rightarrow} #3.
]”]

[x′ $\stackrel{\text{tex}}{=}$ “#1.”’]

[rack $\stackrel{\text{tex}}{=}$ “
rack”]

["vector" $\stackrel{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {vector}\mbox {\tt \char34}”]

["bibliography" $\stackrel{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {bibliography}\mbox {\tt \char34}”]

["dictionary" $\stackrel{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {dictionary}\mbox {\tt \char34}”]

["body" $\stackrel{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {body}\mbox {\tt \char34}”]

["codex" $\stackrel{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {codex}\mbox {\tt \char34}”]

["expansion" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {expansion}\mbox {\tt \char34}”]

["code" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {code}\mbox {\tt \char34}”]

["cache" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {cache}\mbox {\tt \char34}”]

["diagnose" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {diagnose}\mbox {\tt \char34}”]

["value" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {value}\mbox {\tt \char34}”]

["pyk" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {pyk}\mbox {\tt \char34}”]

["tex" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {tex}\mbox {\tt \char34}”]

["texname" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {texname}\mbox {\tt \char34}”]

["message" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {message}\mbox {\tt \char34}”]

["macro" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {macro}\mbox {\tt \char34}”]

["definition" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {definition}\mbox {\tt \char34}”]

["unpack" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {unpack}\mbox {\tt \char34}”]

["claim" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {claim}\mbox {\tt \char34}”]

["priority" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {priority}\mbox {\tt \char34}”]

[$\mathbf{aspect}(\mathsf{a},\mathsf{c})$ $\overset{\text{tex}}{=}$ “
\mathbf{aspect}(#1.
, #2.
)”]

[**aspect**$(*,*,*)$ $\overset{\text{tex}}{=}$ “
\mathbf{aspect}(#1.
, #2.
, #3.
)”]

["lambda" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {lambda}\mbox {\tt \char34}”]

["apply" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {apply}\mbox {\tt \char34}”]

["true" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {true}\mbox {\tt \char34}”]

["if" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {if}\mbox {\tt \char34}”]

["quote" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {quote}\mbox {\tt \char34}”]

["proclaim" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {proclaim}\mbox {\tt \char34}”]

["define" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {define}\mbox {\tt \char34}”]

["introduce" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {introduce}\mbox {\tt \char34}”]

["hide" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {hide}\mbox {\tt \char34}”]

["pre" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {pre}\mbox {\tt \char34}”]

["post" $\overset{\text{tex}}{=}$ “
\mbox {\tt \char34}\mathrm {post}\mbox {\tt \char34}”]

[(x) $\overset{\text{tex}}{=}$ “
\left(#1.
\right) ”]

[Λx.y $\overset{\text{tex}}{=}$ “
\Lambda #1.
. #2.”]

[**let** x $\doteqdot$ y in z $\overset{\text{tex}}{=}$ “
\mathbf{let\ } #1.
\mathrel{\ddot{=}} #2.
\mathrel{\ in\ } #3.”]

[**Priority table**[x] $\overset{\text{tex}}{=}$ “
\mathbf{Priority\ table} #1.
\mathbf{End\ table}”]

[**Priority table**[x] $\overset{\text{name}}{=}$ “
\mathbf{Priority\ table} [#1.
]”]

[$\langle * \rangle$ $\overset{\text{tex}}{=}$ “
\langle #1.
\rangle ”]

[x, y $\overset{\text{tex}}{=}$ “#1.
, \linebreak [0] #2.”]

[$\mathbf{tuple}_1(*)$ $\overset{\text{tex}}{=}$ “
\mathbf {tuple}_1(#1.
) ”]

[$\mathbf{tuple}_2(*)$ $\overset{\text{tex}}{=}$ “
\mathbf {tuple}_2(#1.
) ”]

[ragged right $\overset{\text{tex}}{=}$ “
\raggedright”]

[ragged right $\overset{\text{name}}{=}$ “
ragged\ right”]

[ragged right expansion $\overset{\text{tex}}{=}$ “”]

[ragged right expansion $\overset{\text{name}}{=}$ “
ragged\ right\ expansion\ ”]

[newline x $\overset{\text{tex}}{=}$ “
\newline #1.”]

[newline x $\overset{\text{name}}{=}$ “
newline\ #1.”]

[macro newline x $\overset{\text{tex}}{=}$ “
\newline #1.”]

```
[macro newline x ≝^name “
        macro\ newline\ #1.”]

[(x)^v ≝^tex “#1/tex name.”]

[(x)^v ≝^name “
        ( #1.
        )^{\bf v}”]

[if x then y else z ≝^tex “
        {\bf if} \ #1.
        \ {\bf then} \ #2.
        \ {\bf else} \ #3.”]

[let x = y in z ≝^tex “
        \mathbf{let\ } #1.
        = #2.
        \mathbf{\ in\ } #3.”]

[let_1(f, y) ≝^tex “
        let_1( #1.
        , #2.
        )”]

[let_2(f, y) ≝^tex “
        let_2( #1.
        , #2.
        )”]

[x ⩓ y ≝^tex “#1.
        \mathrel{\ddot{\wedge}} #2.”]

[x ⩔ y ≝^tex “#1.
        \mathrel{\ddot{\vee}} #2.”]

[x ⤇ y ≝^tex “#1.
        \mathrel{\ddot{\Rightarrow}} #2.”]

[display(x) ≝^tex “

        \addvspace{\abovedisplayskip}

        \setlength{\leftskip}{\mathindent}\noindent #1.
        \everypar{\setlength{\parindent}{\docparindent}}
        \setlength{\parindent}{0mm}

        \setlength{\leftskip}{0mm}
```

\addvspace{\belowdisplayskip}

”]

[display(x) $\overset{\text{name}}{=}$ “
display(#1.
)”]

[statement(x) $\overset{\text{tex}}{=}$ “

\addvspace{\abovedisplayskip}

\setlength{\leftskip}{0mm}\noindent #1.
\everypar{\setlength{\parindent}{\docparindent}}
\setlength{\parindent}{0mm}

\setlength{\leftskip}{0mm}
\addvspace{\belowdisplayskip}

”]

[statement(x) $\overset{\text{name}}{=}$ “
statement(#1.
)”]

[intro(x, i, p, t) $\overset{\text{tex}}{=}$ “\index{#2.: #3. @#2.: $[#1/tex name/tex.]$ #3.}%
\index{pyk: #3. $[#1/tex name/tex.]$}%
\tex{
$[#1/tex name/tex.
\stackrel {\mathrm {tex}}{=} #4/tex name.
]$}$[#1/tex name/tex.%
]$\footnote{$[#1/tex name/tex.
\stackrel {\mathrm {pyk}}{=} #3/tex name.
]$}”]

[intro(x, i, p, t) $\overset{\text{name}}{=}$ “
intro(#1.
, #2.
, #3.
, #4.
)”]

[intro(x, p, t) $\overset{\text{tex}}{=}$ “\index{\alpha #2. @\back \makebox[20mm][l]{$[#1/tex
name/tex.]$}#2.}%
\index{pyk: #2. $[#1/tex name/tex.]$}%
\tex{
$[#1/tex name/tex.

\stackrel {\mathrm {tex}}{=} #3/tex name.
]$}$[#1/tex name/tex.%
]$\footnote{$[#1/tex name/tex.
\stackrel {\mathrm {pyk}}{=} #2/tex name.
]$}"]

[intro(x, p, t) $\stackrel{\text{name}}{=}$ "
intro(#1.
, #2.
, #3.
)"]

[x/intro(y, p, t) $\stackrel{\text{tex}}{=}$ "#1.%
\footnote{$[#2/tex name/tex.
\stackrel {\mathrm {pyk}}{=} #3/tex name.
]$}\index{\alpha #3. @\back \makebox[20mm][l]{$[#2/tex
name/tex.]$}#3.}%
\index{pyk: #3. $[#2/tex name/tex.]$}%
\tex{
$[#2/tex name/tex.
\stackrel {\mathrm {tex}}{=} #4/tex name.
]$}"]

[x/intro(y, p, t) $\stackrel{\text{name}}{=}$ "#1.
/intro(#2.
, #3.
, #4.
)"]

[x/indexintro(y, i, p, t) $\stackrel{\text{tex}}{=}$ "#1.%
\footnote{$[#2/tex name/tex.
\stackrel {\mathrm {pyk}}{=} #4/tex name.
]$}\index{#3.: #4. @#3.: $[#2/tex name/tex.]$ #4.}%
\index{pyk: #4. $[#2/tex name/tex.]$}%
\tex{
$[#2/tex name/tex.
\stackrel {\mathrm {tex}}{=} #5/tex name.
]$}"]

[x/indexintro(y, i, p, t) $\stackrel{\text{name}}{=}$ "#1.
/indexintro(#2.
, #3.
, #4.
, #5.
)"]

[x/nameintro(y, p, t, n) $\stackrel{\text{tex}}{=}$ "#1.%
\footnote{$[#2/tex name/tex.

\stackrel {\mathrm {pyk}}{=} #3/tex name.
]$}\index{\alpha #3. @\back \makebox[20mm][l]{$[#2/tex name/tex.]$}#3.}%
\index{pyk: #3. $[#2/tex name/tex.]$}%
\tex{
$[#2/tex name/tex.
\stackrel {\mathrm {tex}}{=} #4/tex name.
]$}
\tex{
$[#2/tex name/tex.
\stackrel {\mathrm {name}}{=} #5/tex name.
]$}"]

[x/nameintro$(\mathsf{y},\mathsf{p},\mathsf{t},\mathsf{n})$ $\overset{\text{name}}{=}$ "#1.
/nameintro(#2.
,#3.
,#4.
,#5.
)"]

[x/bothintro$(\mathsf{y},\mathsf{i},\mathsf{p},\mathsf{t},\mathsf{n})$ $\overset{\text{tex}}{=}$ "#1.%
\footnote{$[#2/tex name/tex.
\stackrel {\mathrm {pyk}}{=} #4/tex name.
]$}\index{#3.: #4. @#3.: $[#2/tex name/tex.]$ #4.}%
\index{pyk: #4. $[#2/tex name/tex.]$}%
\tex{
$[#2/tex name/tex.
\stackrel {\mathrm {tex}}{=} #5/tex name.
]$}
\tex{
$[#2/tex name/tex.
\stackrel {\mathrm {name}}{=} #6/tex name.
]$}"]

[x/bothintro$(\mathsf{y},\mathsf{i},\mathsf{p},\mathsf{t},\mathsf{n})$ $\overset{\text{name}}{=}$ "#1.
/bothintro(#2.
,#3.
,#4.
,#5.
,#6.
)"]

$[[* \overset{\text{claim}}{=} *] \overset{\text{tex}}{=}$ "
[#1/tex name/tex.
\stackrel {claim}{=}#2.
]"]

$[x \tilde{\wedge} y \overset{\text{tex}}{=} \text{“}\#1.$
$\backslash\text{mathrel}\{\backslash\text{tilde}\{\backslash\text{wedge}\}\}\ \#2.\text{”}]$

$[x \wedge_c y \overset{\text{tex}}{=} \text{“}\#1.$
$\backslash\text{wedge\_c}\ \#2.\text{”}]$

$[\mathbf{check}(*,*) \overset{\text{tex}}{=} \text{“}$
$\backslash\text{mathbf}\{\text{check}\}(\ \#1.$
$,\ \#2.$
$)\text{”}]$

$[\mathbf{check}_2(*,*,*) \overset{\text{tex}}{=} \text{“}$
$\backslash\text{mathbf}\{\text{check}\}\_2(\ \#1.$
$,\ \#2.$
$,\ \#3.$
$)\text{”}]$

$[\mathbf{check}_3(*,*,*) \overset{\text{tex}}{=} \text{“}$
$\backslash\text{mathbf}\{\text{check}\}\_3(\ \#1.$
$,\ \#2.$
$,\ \#3.$
$)\text{”}]$

$[\mathbf{check}^*(*,*) \overset{\text{tex}}{=} \text{“}$
$\backslash\text{mathbf}\{\text{check}\}\hat{}*(\ \#1.$
$,\ \#2.$
$)\text{”}]$

$[\mathbf{check}^*_2(*,*,*) \overset{\text{tex}}{=} \text{“}$
$\backslash\text{mathbf}\{\text{check}\}\hat{}*\_2(\ \#1.$
$,\ \#2.$
$,\ \#3.$
$)\text{”}]$

$[[*]^{\cdot} \overset{\text{tex}}{=} \text{“}$
$\backslash\text{relax}\ [\ \#1.$
$\backslash\text{relax}\ ]\hat{}\{\backslash\text{cdot}\}\ \text{”}]$

$[[x]^{-} \overset{\text{tex}}{=} \text{“}$
$\backslash\text{relax}\ [\ \#1.$
$\backslash\text{relax}\ ]\hat{}\{\text{-}\}\ \text{”}]$

$[[*]^{\circ} \overset{\text{tex}}{=} \text{“}$
$\backslash\text{relax}\ [\ \#1.$
$\backslash\text{relax}\ ]\hat{}\{\backslash\text{circ}\}\ \text{”}]$

$[x \text{ spy } y \overset{\text{tex}}{=} \text{“}\#1.$
$\backslash\text{mathrel}\{\text{spy}\}\#2.\text{”}]$

$[[\mathsf{x}]^{\cdot} \overset{\text{tex}}{=} \text{“}$
$\quad [\#1.$
$\quad]\hat{}\{\backslash\text{cdot}\}\text{”}]$

$[[\mathsf{x}]^{-} \overset{\text{tex}}{=} \text{“}$
$\quad [\#1.$
$\quad]\hat{}\text{-”}]$

$[\text{msg} \overset{\text{tex}}{=} \text{“}$
$\quad \text{msg”}]$

$[[\mathsf{x} \overset{\text{msg}}{=} \mathsf{y}] \overset{\text{tex}}{=} \text{“}$
$\quad [\#1/\text{tex name}/\text{tex}.$
$\quad \backslash\text{stackrel } \{\text{msg}\}\{=\} \#2.$
$\quad]\text{”}]$

$[\langle\text{stmt}\rangle \overset{\text{tex}}{=} \text{“}$
$\quad \{<\}\text{stmt}\{>\}\text{”}]$

$[\text{stmt} \overset{\text{tex}}{=} \text{“}$
$\quad \text{stmt”}]$

$[[\mathsf{x} \overset{\text{stmt}}{=} \mathsf{y}] \overset{\text{tex}}{=} \text{“}$
$\quad [\#1/\text{tex name}/\text{tex}.$
$\quad \backslash\text{stackrel } \{\text{stmt}\}\{=\}\#2.$
$\quad]\text{”}]$

$[\text{HeadNil}' \overset{\text{tex}}{=} \text{“}$
$\quad \text{HeadNil}'\text{”}]$

$[\text{HeadPair}' \overset{\text{tex}}{=} \text{“}$
$\quad \text{HeadPair}'\text{”}]$

$[\text{Transitivity}' \overset{\text{tex}}{=} \text{“}$
$\quad \text{Transitivity}'\text{”}]$

$[\text{Contra}' \overset{\text{tex}}{=} \text{“}$
$\quad \text{Contra}'\text{”}]$

$[\mathrm{T}'_{\mathrm{E}} \overset{\text{tex}}{=} \text{“}$
$\quad \text{T'\_\{E\}”}]$

$[\mathrm{L}_1 \overset{\text{tex}}{=} \text{“}$
$\quad \text{L\_\{1\}”}]$

$[\mathsf{x} \vdash \mathsf{y} \overset{\text{tex}}{=} \text{“}\#1.$
$\quad \backslash\text{vdash } \#2.\text{”}]$

[$\mathsf{x} \Vdash \mathsf{y} \overset{\text{tex}}{=}$ "#1.
\mathrel {\makebox [0mm][l]{$\vdash $}\, {\vdash }} #2."]

[$\forall \mathsf{x}\colon \mathsf{y} \overset{\text{tex}}{=}$ "
\forall #1.
\colon #2."]

[$\perp\!\!\!\perp \overset{\text{tex}}{=}$ "
{\makebox [0mm][l]{$\bot $}\, {\bot }}"]

[$\mathsf{x} \oplus \mathsf{y} \overset{\text{tex}}{=}$ "#1.
\mathrel {\oplus} #2."]

[$\underline{\mathsf{x}} \overset{\text{tex}}{=}$ "\underline{#1.}"]

[$\mathcal{A} \overset{\text{tex}}{=}$ "{\cal A}"]

[$\mathcal{B} \overset{\text{tex}}{=}$ "{\cal B}"]

[$\mathcal{C} \overset{\text{tex}}{=}$ "{\cal C}"]

[$\mathcal{D} \overset{\text{tex}}{=}$ "{\cal D}"]

[$\mathcal{E} \overset{\text{tex}}{=}$ "{\cal E}"]

[$\mathcal{F} \overset{\text{tex}}{=}$ "{\cal F}"]

[$\mathcal{G} \overset{\text{tex}}{=}$ "{\cal G}"]

[$\mathcal{H} \overset{\text{tex}}{=}$ "{\cal H}"]

[$\mathcal{I} \overset{\text{tex}}{=}$ "{\cal I}"]

[$\mathcal{J} \overset{\text{tex}}{=}$ "{\cal J}"]

[$\mathcal{K} \overset{\text{tex}}{=}$ "{\cal K}"]

[$\mathcal{L} \overset{\text{tex}}{=}$ "{\cal L}"]

[$\mathcal{M} \overset{\text{tex}}{=}$ "{\cal M}"]

[$\mathcal{N} \overset{\text{tex}}{=}$ "{\cal N}"]

[$\mathcal{O} \overset{\text{tex}}{=}$ "{\cal O}"]

[$\mathcal{P} \overset{\text{tex}}{=}$ "{\cal P}"]

[$\mathcal{Q} \overset{\text{tex}}{=}$ "{\cal Q}"]

$[\mathcal{R} \overset{\text{tex}}{=} \text{“\{\textbackslash cal R\}”}]$

$[\mathcal{S} \overset{\text{tex}}{=} \text{“\{\textbackslash cal S\}”}]$

$[\mathcal{T} \overset{\text{tex}}{=} \text{“\{\textbackslash cal T\}”}]$

$[\mathcal{U} \overset{\text{tex}}{=} \text{“\{\textbackslash cal U\}”}]$

$[\mathcal{V} \overset{\text{tex}}{=} \text{“\{\textbackslash cal V\}”}]$

$[\mathcal{W} \overset{\text{tex}}{=} \text{“\{\textbackslash cal W\}”}]$

$[\mathcal{X} \overset{\text{tex}}{=} \text{“\{\textbackslash cal X\}”}]$

$[\mathcal{Y} \overset{\text{tex}}{=} \text{“\{\textbackslash cal Y\}”}]$

$[\mathcal{Z} \overset{\text{tex}}{=} \text{“\{\textbackslash cal Z\}”}]$

$[\mathsf{t}^{\mathcal{V}} \overset{\text{tex}}{=} \text{“\#1.}$
$\text{\{\} \^{} \{\textbackslash cal V\}”}]$

$[\mathsf{t}^{\mathcal{C}} \overset{\text{tex}}{=} \text{“\#1.}$
$\text{\{\} \^{} \{\textbackslash cal C\}”}]$

$[\mathsf{t}^{\mathcal{C}^*} \overset{\text{tex}}{=} \text{“\#1.}$
$\text{\{\} \^{} \{\{\textbackslash cal C\} \^{} \{ \textbackslash ast \}\}”}]$

$[\mathsf{x} \text{ free in } \mathsf{y} \overset{\text{tex}}{=} \text{“\#1.}$
$\text{\textbackslash mathrel \{free\textbackslash\ in\} \#2.”}]$

$[\mathsf{x} \text{ free in}^* \mathsf{y} \overset{\text{tex}}{=} \text{“\#1.}$
$\text{\textbackslash mathrel \{free\textbackslash\ in\}\^{}\{\textbackslash ast\} \#2.”}]$

$[\mathsf{a} \text{ free for } \mathsf{x} \text{ in } \mathsf{b} \overset{\text{tex}}{=} \text{“\#1.}$
$\text{\textbackslash mathrel \{free\textbackslash\ for\} \#2.}$
$\text{\textbackslash mathrel \{in\} \#3.”}]$

$[\mathsf{a} \text{ free for}^* \mathsf{x} \text{ in } \mathsf{b} \overset{\text{tex}}{=} \text{“\#1.}$
$\text{\textbackslash mathrel \{free\textbackslash\ for\}\^{}\{\textbackslash ast\} \#2.}$
$\text{\textbackslash mathrel \{in\} \#3.”}]$

$[\langle \mathsf{a} \,|\mathsf{x} := \mathsf{b}\rangle \overset{\text{tex}}{=} \text{“}$
$\text{\textbackslash langle \#1.}$
$\text{\textbackslash,\{\textbackslash protect\textbackslash vert\}\#2.}$
$\text{\{:=\}\textbackslash,\#3.}$
$\text{\textbackslash rangle ”}]$

$[\langle {}^{*}\mathsf{a}\,|\mathsf{x}{:=}\,\mathsf{b}\rangle \overset{\text{tex}}{=} \text{“}$
\langle ^ { \ast } #1.
\,{\protect\vert}#2.
{:=}\,#3.
\rangle ”]

$[\mathsf{x} \in_{\mathrm{t}} \mathsf{y} \overset{\text{tex}}{=} \text{“}$#1.
\in_t #2.”]

$[\mathsf{x} \subseteq_{\mathrm{T}} \mathsf{y} \overset{\text{tex}}{=} \text{“}$#1.
\subseteq_T #2.”]

$[\mathsf{x} \overset{\mathrm{T}}{=} \mathsf{y} \overset{\text{tex}}{=} \text{“}$#1.
\stackrel{T}{=} #2.”]

$[\emptyset \overset{\text{tex}}{=} \text{“}$
\emptyset ”]

$[\mathsf{x} \cup \{\mathsf{y}\} \overset{\text{tex}}{=} \text{“}$#1.
\cup \{ #2.
\}”]

$[\mathsf{x} \backslash \{\mathsf{y}\} \overset{\text{tex}}{=} \text{“}$#1.
\backslash \{ #2.
\}”]

$[\mathsf{x} \cup \mathsf{y} \overset{\text{tex}}{=} \text{“}$#1.
\cup #2.”]

$[\mathsf{x} \overset{\mathrm{s}}{=} \mathsf{y} \overset{\text{tex}}{=} \text{“}$#1.
\stackrel{s}{=} #2.”]

$[\mathsf{x}^{\mathrm{I}} \overset{\text{tex}}{=} \text{“}$#1.
{} ^ { I }”]

$[\mathsf{x}^{\triangleright} \overset{\text{tex}}{=} \text{“}$#1.
{} ^ { \rhd }”]

$[\mathsf{x} @ \mathsf{y} \overset{\text{tex}}{=} \text{“}$#1.
\mathop {\char64} #2.”]

$[\mathsf{x}^{\mathrm{V}} \overset{\text{tex}}{=} \text{“}$#1.
{} ^ { V } ”]

$[\mathcal{T}(\mathsf{x}) \overset{\text{tex}}{=} \text{“}$
{\cal T}(#1.
)”]

$[\mathsf{x}^{+} \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad \{\}\ \hat{}\ \{\ +\ \}\ \text{”}]$

$[\mathsf{x}^{-} \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad \{\}\ \hat{}\ \{\ \text{-}\ \}\ \text{”}]$

$[\mathsf{x}\ \text{i.e.}\ \mathsf{y} \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad \backslash\text{mathrel}\ \{\text{i.e.}\}\ \#2.\text{”}]$

$[\mathsf{x}^{*} \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad \{\}\ \hat{}\ \{\ \backslash\text{ast}\ \}\ \text{”}]$

$[\mathsf{x};\mathsf{y} \overset{\text{tex}}{=} \text{“\#1.}$
$\qquad\qquad ;\ \#2.\text{”}]$

$[\text{Remainder} \overset{\text{tex}}{=} \text{“}$
$\qquad\qquad \text{Remainder”}]$

$[\mathcal{S}^{\mathrm{I}}(\mathsf{x},\mathsf{y}) \overset{\text{tex}}{=} \text{“}$
$\qquad\qquad \{\backslash\text{cal S}\}\hat{}\{\text{I}\}(\#1.$
$\qquad\qquad ,\ \#2.$
$\qquad\qquad)\text{”}]$

$[\mathcal{S}^{\triangleright}(\mathsf{x},\mathsf{y}) \overset{\text{tex}}{=} \text{“}$
$\qquad\qquad \{\backslash\text{cal S}\}\hat{}\{\backslash\text{rhd}\}(\#1.$
$\qquad\qquad ,\ \#2.$
$\qquad\qquad)\text{”}]$

$[\mathcal{S}_1^{\triangleright}(\mathsf{x},\mathsf{y},\mathsf{z}) \overset{\text{tex}}{=} \text{“}$
$\qquad\qquad \{\backslash\text{cal S}\}\_\{1\}\hat{}\{\backslash\text{rhd}\}(\#1.$
$\qquad\qquad ,\ \#2.$
$\qquad\qquad ,\ \#3.$
$\qquad\qquad)\text{”}]$

$[\mathcal{S}^{\mathrm{E}}(\mathsf{x},\mathsf{y}) \overset{\text{tex}}{=} \text{“}$
$\qquad\qquad \{\backslash\text{cal S}\}\hat{}\{\text{E}\}(\#1.$
$\qquad\qquad ,\ \#2.$
$\qquad\qquad)\text{”}]$

$[\mathcal{S}_1^{\mathrm{E}}(\mathsf{x},\mathsf{y},\mathsf{z}) \overset{\text{tex}}{=} \text{“}$
$\qquad\qquad \{\backslash\text{cal S}\}\_\{1\}\hat{}\{\text{E}\}(\#1.$
$\qquad\qquad ,\ \#2.$
$\qquad\qquad ,\ \#3.$
$\qquad\qquad)\text{”}]$

$[\mathcal{S}^{+}(\mathsf{x},\mathsf{y}) \overset{\text{tex}}{=} \text{“}$
$\qquad\qquad \{\backslash\text{cal S}\}\hat{}\{+\}(\#1.$

, #2.
)”]

$[\mathcal{S}_1^+(\mathsf{x},\mathsf{y},\mathsf{z}) \overset{\text{tex}}{=} “$
{\cal S}_{1}^{+}(#1.
, #2.
, #3.
)”]

$[\mathcal{S}^-(\mathsf{x},\mathsf{y}) \overset{\text{tex}}{=} “$
{\cal S}^{-}(#1.
, #2.
)”]

$[\mathcal{S}_1^-(\mathsf{x},\mathsf{y},\mathsf{z}) \overset{\text{tex}}{=} “$
{\cal S}_{1}^{-}(#1.
, #2.
, #3.
)”]

$[\mathcal{S}^*(\mathsf{x},\mathsf{y}) \overset{\text{tex}}{=} “$
{\cal S}^{\ast}(#1.
, #2.
)”]

$[\mathcal{S}_1^*(\mathsf{x},\mathsf{y},\mathsf{z}) \overset{\text{tex}}{=} “$
{\cal S}_{1}^{\ast}(#1.
, #2.
, #3.
)”]

$[\mathcal{S}_2^*(\mathsf{c},\mathsf{t},\mathsf{q},\mathsf{d}) \overset{\text{tex}}{=} “$
{\cal S}_{2}^{\ast}(#1.
, #2.
, #3.
, #4.
)”]

$[\mathcal{S}^@(\mathsf{x},\mathsf{y}) \overset{\text{tex}}{=} “$
{\cal S}^{\char64}(#1.
, #2.
)”]

$[\mathcal{S}_1^@(\mathsf{c},\mathsf{t},\mathsf{q}) \overset{\text{tex}}{=} “$
{\cal S}_{1}^{\char64}(#1.
, #2.
, #3.
)”]

$[\mathcal{S}^{\vdash}(\mathsf{x},\mathsf{y}) \overset{\text{tex}}{=} \text{“}$
{\cal S}^{\vdash}(#1.
, #2.
)”]

$[\mathcal{S}_1^{\vdash}(\mathsf{x},\mathsf{y},\mathsf{z},\mathsf{u}) \overset{\text{tex}}{=} \text{“}$
{\cal S}_{1}^{\vdash}(#1.
, #2.
, #3.
, #4.
)”]

$[\mathcal{S}^{\dashv\vdash}(\mathsf{x},\mathsf{y}) \overset{\text{tex}}{=} \text{“}$
{\cal S}^{\makebox [0mm][l]{\scriptsize $\vdash $}\,{\vdash }}(#1.
, #2.
)”]

$[\mathcal{S}_1^{\dashv\vdash}(\mathsf{x},\mathsf{y},\mathsf{z},\mathsf{u}) \overset{\text{tex}}{=} \text{“}$
{\cal S}_{1}^{\makebox [0mm][l]{\scriptsize $\vdash $}\,{\vdash }}(#1.
, #2.
, #3.
, #4.
)”]

$[\mathcal{S}^{\text{i.e.}}(\mathsf{x},\mathsf{y}) \overset{\text{tex}}{=} \text{“}$
{\cal S}^{i.e.}(#1.
, #2.
)”]

$[\mathcal{S}_1^{\text{i.e.}}(\mathsf{x},\mathsf{y},\mathsf{z},\mathsf{u}) \overset{\text{tex}}{=} \text{“}$
{\cal S}_{1}^{i.e.}(#1.
, #2.
, #3.
, #4.
)”]

$[\mathcal{S}_2^{\text{i.e.}}(\mathsf{c},\mathsf{t},\mathsf{a},\mathsf{q},\mathsf{d}) \overset{\text{tex}}{=} \text{“}$
{\cal S}_{2}^{i.e.}(#1.
, #2.
, #3.
, #4.
, #5.
)”]

$[\mathcal{S}^{\forall}(\mathsf{x},\mathsf{y}) \stackrel{\text{tex}}{=} \text{“}$
{\cal S}^{\forall}(#1.
, #2.
)”]

$[\mathcal{S}_1^{\forall}(\mathsf{c},\mathsf{t},\mathsf{v},\mathsf{q}) \stackrel{\text{tex}}{=} \text{“}$
{\cal S}_{1}^{\forall}(#1.
, #2.
, #3.
, #4.
)”]

$[\mathcal{S}^{;}(\mathsf{x},\mathsf{y}) \stackrel{\text{tex}}{=} \text{“}$
{\cal S}^{;}(#1.
, #2.
)”]

$[\mathcal{S}_1^{;}(\mathsf{x},\mathsf{y},\mathsf{z}) \stackrel{\text{tex}}{=} \text{“}$
{\cal S}_{1}^{;}(#1.
, #2.
, #3.
)”]

$[\mathcal{S}_2^{;}(\mathsf{c},\mathsf{t},\mathsf{p},\mathsf{q}) \stackrel{\text{tex}}{=} \text{“}$
{\cal S}_{2}^{;}(#1.
, #2.
, #3.
, #4.
)”]

$[\mathsf{x}\text{-color}(\mathsf{y}) \stackrel{\text{tex}}{=} \text{“}$#1.
\mbox {-color}(#2.
)”]

$[\mathsf{x}\text{-color}^*(\mathsf{y}) \stackrel{\text{tex}}{=} \text{“}$#1.
\mbox {-color}^{\ast}(#2.
)”]

$[\text{error}(\mathsf{m},\mathsf{t}) \stackrel{\text{tex}}{=} \text{“}$
error(#1/tex name.
,#2.
)”]

$[\text{error}_2(\mathsf{m},\mathsf{t}) \stackrel{\text{tex}}{=} \text{“}$
error_{2}(#1/tex name.
,#2.
)”]

$[\mathsf{x}^{\mathrm{E}} \overset{\text{tex}}{=} \text{“}$#1.
{} ^ { E }”]

$[\mathcal{S}(\mathsf{x},\mathsf{y}) \overset{\text{tex}}{=} \text{“}$
{\cal S}(#1.
, #2.
)”]

$[\mathsf{p}$ proves $\mathsf{t} \overset{\text{tex}}{=} \text{“}$#1.
\ proves\ #2.”]

$[\mathrm{proof}(\mathsf{p},\mathsf{t},\mathsf{c}) \overset{\text{tex}}{=} \text{“}$
proof(#1.
, #2.
, #3.
)”]

$[\mathrm{proof}_2(\mathsf{q},\mathsf{t}) \overset{\text{tex}}{=} \text{“}$
proof_{2}(#1.
, #2.
)”]

$[\mathsf{x} \in_{\mathrm{c}} \mathsf{y} \overset{\text{tex}}{=} \text{“}$#1.
\in_c #2.”]

$[\mathrm{claims}(\mathsf{t},\mathsf{c},\mathsf{r}) \overset{\text{tex}}{=} \text{“}$
claims(#1.
, #2.
, #3.
)”]

$[\mathrm{claims}_2(\mathsf{t},\mathsf{c},\mathsf{r}) \overset{\text{tex}}{=} \text{“}$
claims_2(#1.
, #2.
, #3.
)”]

[<proof> $\overset{\text{tex}}{=} \text{“}$
{<}proof{>}”]

[proof $\overset{\text{tex}}{=} \text{“}$
proof”]

[[**Lemma** x: y] $\overset{\text{tex}}{=} \text{“}$
[\mathbf{Lemma\ } #1.
\colon #2.
]”]

$[[\textbf{Proof of } \mathsf{x}\colon \mathsf{y}] \overset{\text{tex}}{=} \text{“}$
[\mathbf{Proof\ of\ }#1/tex name/tex.
\colon #2.
]”]

$[[\mathsf{x}\ \textbf{lemma}\ \mathsf{y}\colon \mathsf{z}] \overset{\text{tex}}{=} \text{“}$
[#1.
\mathbf{\ lemma\ } #2.
\colon #3.
]”]

$[\text{HeadNil}'' \overset{\text{tex}}{=} \text{“}$
HeadNil’”]

$[[\mathsf{x}\ \textbf{antilemma}\ \mathsf{y}\colon \mathsf{z}] \overset{\text{tex}}{=} \text{“}$
[#1.
\mathbf{\ antilemma\ } #2.
\colon #3.
]”]

$[\text{Contra}'' \overset{\text{tex}}{=} \text{“}$
Contra’”]

$[\mathcal{V}_1(\mathsf{c}) \overset{\text{tex}}{=} \text{“}$
{\cal V}_1(#1.
)”]

$[\mathcal{V}_2(\mathsf{c}, \mathsf{p}) \overset{\text{tex}}{=} \text{“}$
{\cal V}_2(#1.
, #2.
)”]

$[\mathcal{V}_3(\mathsf{c}, \mathsf{r}, \mathsf{p}, \mathsf{d}) \overset{\text{tex}}{=} \text{“}$
{\cal V}_3(#1.
, #2.
, #3.
, #4.
)”]

$[\mathcal{V}_4(\mathsf{c}, \mathsf{p}) \overset{\text{tex}}{=} \text{“}$
{\cal V}_4(#1.
, #2.
)”]

$[\mathcal{V}_7(\mathsf{c}, \mathsf{r}, \mathsf{i}, \mathsf{q}) \overset{\text{tex}}{=} \text{“}$
{\cal V}_7(#1.
, #2.

, #3.
, #4.
)"]

[$\mathcal{V}_6(\mathsf{c},\mathsf{r},\mathsf{p},\mathsf{q})$ $\stackrel{\text{tex}}{=}$ "
{\cal V}_6(#1.
, #2.
, #3.
, #4.
)"]

[$\mathcal{V}_5(\mathsf{c},\mathsf{r},\mathsf{a},\mathsf{q})$ $\stackrel{\text{tex}}{=}$ "
{\cal V}_5(#1.
, #2.
, #3.
, #4.
)"]

[Rule tactic $\stackrel{\text{tex}}{=}$ "
Rule\ tactic"]

[rule(c,p) $\stackrel{\text{tex}}{=}$ "
rule(#1.
, #2.
)"]

[rule$_1(\mathsf{s},\mathsf{t})$ $\stackrel{\text{tex}}{=}$ "
rule_1(#1.
, #2.
)"]

[Cut(a,b) $\stackrel{\text{tex}}{=}$ "
Cut(#1.
, #2.
)"]

[Head$_\oplus(\mathsf{s})$ $\stackrel{\text{tex}}{=}$ "
Head_{\oplus} (#1.
)"]

[Tail$_\oplus(\mathsf{s})$ $\stackrel{\text{tex}}{=}$ "
Tail_{\oplus} (#1.
)"]

[[x **rule** y: z] $\stackrel{\text{tex}}{=}$ "
[#1.
\mathbf{\ rule\ } #2.

\colon #3.
]”]

[[x **antirule** y: z] $\overset{\text{tex}}{=}$ “
[#1.
\mathbf{\ antirule\ } #2.
\colon #3.
]”]

[HeadPair″ $\overset{\text{tex}}{=}$ “
HeadPair’’”]

[Transitivity″ $\overset{\text{tex}}{=}$ “
Transitivity’’”]

[HeadNil $\overset{\text{tex}}{=}$ “
HeadNil”]

[HeadPair $\overset{\text{tex}}{=}$ “
HeadPair”]

[Transitivity $\overset{\text{tex}}{=}$ “
Transitivity”]

[Contra $\overset{\text{tex}}{=}$ “
Contra”]

[$\mathrm{T_E}$ $\overset{\text{tex}}{=}$ “
T_E”]

[[**Theory** n] $\overset{\text{tex}}{=}$ “
[\mathbf{Theory\ } #1.
]”]

[$\mathrm{theory_2}(\mathsf{t}, \mathsf{c})$ $\overset{\text{tex}}{=}$ “
theory_2(#1.
, #2.
)”]

[$\mathrm{theory_3}(\mathsf{c}, \mathsf{n})$ $\overset{\text{tex}}{=}$ “
theory_3(#1.
, #2.
)”]

[$\mathrm{theory_4}(\mathsf{c}, \mathsf{n}, \mathsf{s})$ $\overset{\text{tex}}{=}$ “
theory_4(#1.
, #2.
, #3.
)”]

$[\mathrm{Plus}(\mathsf{a},\mathsf{b}) \stackrel{\mathrm{tex}}{=} \text{“}$
Plus(#1.
, #2.
)”]

$[\mathrm{Reflexivity} \stackrel{\mathrm{tex}}{=} \text{“}$
Reflexivity”]

$[\mathrm{Commutativity} \stackrel{\mathrm{tex}}{=} \text{“}$
Commutativity”]

$[\mathrm{parm}(\mathsf{t},\mathsf{s},\mathsf{n}) \stackrel{\mathrm{tex}}{=} \text{“}$
parm(#1.
,#2.
,#3.
)”]

$[\mathrm{parm}^*(\mathsf{t},\mathsf{s},\mathsf{n}) \stackrel{\mathrm{tex}}{=} \text{“}$
parm^*(#1.
,#2.
,#3.
)”]

$[\mathrm{inst}(\mathsf{t},\mathsf{s}) \stackrel{\mathrm{tex}}{=} \text{“}$
inst(#1.
,#2.
)”]

$[\mathrm{inst}^*(\mathsf{t},\mathsf{s}) \stackrel{\mathrm{tex}}{=} \text{“}$
inst^*(#1.
,#2.
)”]

$[\mathrm{occur}(\mathsf{t},\mathsf{u},\mathsf{s}) \stackrel{\mathrm{tex}}{=} \text{“}$
occur(#1.
,#2.
,#3.
)”]

$[\mathrm{occur}^*(\mathsf{t},\mathsf{u},\mathsf{s}) \stackrel{\mathrm{tex}}{=} \text{“}$
occur^*(#1.
,#2.
,#3.
)”]

$[\mathrm{unify}(\mathsf{t}=\mathsf{u},\mathsf{s}) \stackrel{\mathrm{tex}}{=} \text{“}$
unify(#1.

=#2.
,#3.
)"]

[unify*(t = u, s) $\stackrel{\text{tex}}{=}$ "
unify^*(#1.
=#2.
,#3.
)"]

[unify$_2$(t = u, s) $\stackrel{\text{tex}}{=}$ "
unify_2(#1.
=#2.
,#3.
)"]

[Reflexivity$_1$ $\stackrel{\text{tex}}{=}$ "
Reflexivity_1"]

[Commutativity$_1$ $\stackrel{\text{tex}}{=}$ "
Commutativity_1"]

[<tactic> $\stackrel{\text{tex}}{=}$ "
{<}tactic{>}"]

[tactic $\stackrel{\text{tex}}{=}$ "
tactic"]

[[x $\stackrel{\text{tactic}}{=}$ y] $\stackrel{\text{tex}}{=}$ "
[#1/tex name/tex.
\stackrel {tactic}{=}#2.
]"]

[$\mathcal{P}$(t, s, c) $\stackrel{\text{tex}}{=}$ "
{\cal P}(#1.
, #2.
, #3.
)"]

[$\mathcal{P}^*$(t, s, c) $\stackrel{\text{tex}}{=}$ "
{\cal P}^*(#1.
, #2.
, #3.
)"]

[p_0 $\stackrel{\text{tex}}{=}$ "
p_0"]

[$\mathsf{x} \rhd \mathsf{y} \stackrel{\text{tex}}{=}$ "#1.
\rhd #2."]

[$\mathsf{x} \Rrightarrow \mathsf{y} \stackrel{\text{tex}}{=}$ "#1.
\mathrel {\makebox [0mm][l]{$\rhd $}\, {\rhd }} #2."]

[$\mathsf{x} \gg \mathsf{y} \stackrel{\text{tex}}{=}$ "#1.
\gg #2."]

[$\text{conclude}_1(\mathsf{t}, \mathsf{c}) \stackrel{\text{tex}}{=}$ "
conclude_1 (#1.
, #2.
)"]

[$\text{conclude}_2(\mathsf{a}, \mathsf{t}, \mathsf{c}) \stackrel{\text{tex}}{=}$ "
conclude_2 (#1.
, #2.
, #3.
)"]

[$\text{conclude}_3(\mathsf{a}, \mathsf{t}, \mathsf{l}, \mathsf{s}) \stackrel{\text{tex}}{=}$ "
conclude_3 (#1.
, #2.
, #3.
, #4.
)"]

[$\text{conclude}_4(\mathsf{a}, \mathsf{l}) \stackrel{\text{tex}}{=}$ "
conclude_4 (#1.
, #2.
)"]

[t **proof of** $\mathsf{s} : \mathsf{p} \stackrel{\text{tex}}{=}$ "
\if\relax\csname lgwprooflinep\endcsname
\def\lgwprooflinep{x}
\newcount\lgwproofline
\fi
\begingroup
\def\insideproof{x}
\lgwproofline=0 #1.
\mathbf {\ proof\ of\ } #2.
\colon #3.
\gdef\lgwella{\relax}
\gdef\lgwellb{\relax}
\gdef\lgwellc{\relax}
\gdef\lgwelld{\relax}
\gdef\lgwelle{\relax}

```
\gdef\lgwellf{\relax}
\gdef\lgwellg{\relax}
\gdef\lgwellh{\relax}
\gdef\lgwelli{\relax}
\gdef\lgwellj{\relax}
\gdef\lgwellk{\relax}
\gdef\lgwelll{\relax}
\gdef\lgwellm{\relax}
\gdef\lgwelln{\relax}
\gdef\lgwello{\relax}
\gdef\lgwellp{\relax}
\gdef\lgwellq{\relax}
\gdef\lgwellr{\relax}
\gdef\lgwells{\relax}
\gdef\lgwellt{\relax}
\gdef\lgwellu{\relax}
\gdef\lgwellv{\relax}
\gdef\lgwellw{\relax}
\gdef\lgwellx{\relax}
\gdef\lgwelly{\relax}
\gdef\lgwellz{\relax}
\gdef\lgwellbiga{\relax}
\gdef\lgwellbigb{\relax}
\gdef\lgwellbigc{\relax}
\gdef\lgwellbigd{\relax}
\gdef\lgwellbige{\relax}
\gdef\lgwellbigf{\relax}
\gdef\lgwellbigg{\relax}
\gdef\lgwellbigh{\relax}
\gdef\lgwellbigi{\relax}
\gdef\lgwellbigj{\relax}
\gdef\lgwellbigk{\relax}
\gdef\lgwellbigl{\relax}
\gdef\lgwellbigm{\relax}
\gdef\lgwellbign{\relax}
\gdef\lgwellbigo{\relax}
\gdef\lgwellbigp{\relax}
\gdef\lgwellbigq{\relax}
\gdef\lgwellbigr{\relax}
\gdef\lgwellbigs{\relax}
\gdef\lgwellbigt{\relax}
\gdef\lgwellbigu{\relax}
\gdef\lgwellbigv{\relax}
\gdef\lgwellbigw{\relax}
\gdef\lgwellbigx{\relax}
\gdef\lgwellbigy{\relax}
```

```
\gdef\lgwellbigz{\relax}
\endgroup "]

[t proof of s : p ^name= "#1.
\mathbf{\ proof\ of\ } #2.
: #3."]

[Line l : a ≫ i; p ^tex= "
\newline \makebox [0.1\textwidth]{}%
\parbox [b]{0.4\textwidth }{\raggedright
\setlength {\parindent }{-0.1\textwidth }%
\makebox [0.1\textwidth ][l]{$#1.
$:}$#2.
{}\gg {}$}\quad
\parbox [t]{0.4\textwidth }{$#3.
$\hfill \makebox [0mm][l]{\quad ;}}#4."]

[Line l : a ≫ i; p ^name= "
Line \, #1.
: #2.
\gg #3.
; #4."]

[Last line a ≫ i □ ^tex= "
\newline \makebox [0.1\textwidth]{}%
\parbox [b]{0.4\textwidth }{\raggedright
\setlength {\parindent }{-0.1\textwidth }%
\makebox [0.1\textwidth ][l]{$
\if \relax \csname lgwprooflinep\endcsname L_? \else
\global \advance \lgwproofline by 1
L\ifnum \lgwproofline <10 0\fi \number \lgwproofline
\fi
$:}$#1.
{}\gg {}$}\quad
\parbox [t]{0.4\textwidth }{$#2.
$\hfill \makebox [0mm][l]{\quad \makebox[0mm]{$\Box$}}}"]

[Last line a ≫ i □ ^name= "
Last\ line \, #1.
\gg #2.
\, \Box"]

[Line l : Premise ≫ i; p ^tex= "
\newline \makebox [0.1\textwidth ][l]{$#1.
$:}\makebox [0.4\textwidth ][l]{$Premise{}\gg{}$}\quad
\parbox [t]{0.4\textwidth }{$#2.
$\hfill \makebox [0mm][l]{\quad ;}}#3."]
```

```
[Line l : Premise ≫ i; p ^name= “
        Line \, #1.
        : Premise \gg #2.
        ; #3.”]

[Line l : Side-condition ≫ i; p ^tex= “
        \newline \makebox [0.1\textwidth ][l]{$#1.
        $:}\makebox [0.4\textwidth ][l]{%
        $\mbox{Side-condition}{}\gg{}$}\quad
        \parbox [t]{0.4\textwidth }{$#2.
        $\hfill \makebox [0mm][l]{\quad ; }}#3.”]

[Line l : Side-condition ≫ i; p ^name= “
        Line \, #1.
        : \mbox{Side-condition} \gg #2.
        ; #3.”]

[Arbitrary ≫ i; p ^tex= “
        \newline \makebox [0.1\textwidth ][l]{$
        \if \relax \csname lgwprooflinep\endcsname L_? \else
        \global \advance \lgwproofline by 1
        L\ifnum \lgwproofline <10 0\fi \number \lgwproofline
        \fi
        $:}\makebox [0.4\textwidth ][l]{$Arbitrary{}\gg{}$}\quad
        \parbox [t]{0.4\textwidth }{$#1.
        $\hfill \makebox [0mm][l]{\quad ; }}#2.”]

[Arbitrary ≫ i; p ^name= “
        Arbitrary \gg #1.
        ; #2.”]

[Local ≫ a = i; p ^tex= “
        \newline\makebox[0.1\textwidth][l]{$
        \if \relax \csname lgwprooflinep\endcsname L_? \else
        \global \advance \lgwproofline by 1
        L\ifnum \lgwproofline <10 0\fi \number \lgwproofline
        \fi
        $:}%
        \makebox[0.4\textwidth][l]{$Local{}\gg{}$}%
        \quad%
        \parbox[t]{0.4\textwidth}{$#1.
        = #2.
        $\hfill\makebox[0mm][l]{\quad ; }}#3.”]

[Local ≫ a = i; p ^name= “
        Local \gg #1.
```

= #2.
; #3."]

A.3 Further TEX definitions

This section contains tex definitions in an old style which the author has not yet turned into the style used in Appendix A.2

[$*$ $\overset{\text{tex}}{=}$ "
\ast "]

[$*$ ' $*$ $\overset{\text{tex}}{=}$ "#1.
\mathbin {\mbox {'}}#2."]

[$*$ ‘ $*$ $\overset{\text{tex}}{=}$ "#1.
\mathbin {\mbox {‘}}#2."]

[$\lambda * .*$ $\overset{\text{tex}}{=}$ "
\lambda #1.
.#2."]

[$\Lambda *$ $\overset{\text{tex}}{=}$ "
\Lambda #1."]

[T $\overset{\text{tex}}{=}$ "
\mathsf {T}"]

[$\mathrm{if}(*, *, *)$ $\overset{\text{tex}}{=}$ "
\mathrm {if}(#1.
, \linebreak [0]#2.
, \linebreak [0]#3.
)"]

[val $\overset{\text{tex}}{=}$ "
\mathrm {val}"]

[claim $\overset{\text{tex}}{=}$ "
\mathrm {claim}"]

[[$* \overset{*}{\Rightarrow} *$] $\overset{\text{tex}}{=}$ "
[#2/tex name/tex.
\stackrel {#1.
}{\Rightarrow }#3.
]"]

[$\bot$ $\overset{\text{tex}}{=}$ "
\bot "]

[$\mathrm{f}(*)$ $\overset{\text{tex}}{=}$ "
f(#1.
)"]

[$(*)^{\mathrm{I}}$ $\overset{\text{tex}}{=}$ "
(#1.
){}^{I} "]

[$\mathsf{F} \overset{\text{tex}}{=}$ “
\mathsf {F}”]
[$\underline{0} \overset{\text{tex}}{=}$ “
\underline {0}”]
[$\underline{1} \overset{\text{tex}}{=}$ “
\underline {1}”]
[$\underline{2} \overset{\text{tex}}{=}$ “
\underline {2}”]
[$\underline{3} \overset{\text{tex}}{=}$ “
\underline {3}”]
[$\underline{4} \overset{\text{tex}}{=}$ “
\underline {4}”]
[$\underline{5} \overset{\text{tex}}{=}$ “
\underline {5}”]
[$\underline{6} \overset{\text{tex}}{=}$ “
\underline {6}”]
[$\underline{7} \overset{\text{tex}}{=}$ “
\underline {7}”]
[$\underline{8} \overset{\text{tex}}{=}$ “
\underline {8}”]
[$\underline{9} \overset{\text{tex}}{=}$ “
\underline {9}”]
[$0 \overset{\text{tex}}{=}$ “
0”]
[$1 \overset{\text{tex}}{=}$ “
1”]
[$2 \overset{\text{tex}}{=}$ “
2”]
[$3 \overset{\text{tex}}{=}$ “
3”]
[$4 \overset{\text{tex}}{=}$ “
4”]
[$5 \overset{\text{tex}}{=}$ “
5”]
[$6 \overset{\text{tex}}{=}$ “
6”]
[$7 \overset{\text{tex}}{=}$ “
7”]
[$8 \overset{\text{tex}}{=}$ “
8”]

$[9 \overset{\text{tex}}{=} \text{“}$
9”]
$[\mathsf{a} \overset{\text{tex}}{=} \text{“}$
\mathsf {a}”]
$[\mathsf{b} \overset{\text{tex}}{=} \text{“}$
\mathsf {b}”]
$[\mathsf{c} \overset{\text{tex}}{=} \text{“}$
\mathsf {c}”]
$[\mathsf{d} \overset{\text{tex}}{=} \text{“}$
\mathsf {d}”]
$[\mathsf{e} \overset{\text{tex}}{=} \text{“}$
\mathsf {e}”]
$[\mathsf{f} \overset{\text{tex}}{=} \text{“}$
\mathsf {f}”]
$[\mathsf{g} \overset{\text{tex}}{=} \text{“}$
\mathsf {g}”]
$[\mathsf{h} \overset{\text{tex}}{=} \text{“}$
\mathsf {h}”]
$[\mathsf{i} \overset{\text{tex}}{=} \text{“}$
\mathsf {i}”]
$[\mathsf{j} \overset{\text{tex}}{=} \text{“}$
\mathsf {j}”]
$[\mathsf{k} \overset{\text{tex}}{=} \text{“}$
\mathsf {k}”]
$[\mathsf{l} \overset{\text{tex}}{=} \text{“}$
\mathsf {l}”]
$[\mathsf{m} \overset{\text{tex}}{=} \text{“}$
\mathsf {m}”]
$[\mathsf{n} \overset{\text{tex}}{=} \text{“}$
\mathsf {n}”]
$[\mathsf{o} \overset{\text{tex}}{=} \text{“}$
\mathsf {o}”]
$[\mathsf{p} \overset{\text{tex}}{=} \text{“}$
\mathsf {p}”]
$[\mathsf{q} \overset{\text{tex}}{=} \text{“}$
\mathsf {q}”]
$[\mathsf{r} \overset{\text{tex}}{=} \text{“}$
\mathsf {r}”]
$[\mathsf{s} \overset{\text{tex}}{=} \text{“}$
\mathsf {s}”]

$[\mathsf{t} \overset{\text{tex}}{=} \text{“}$
\mathsf {t}”]
$[\mathsf{u} \overset{\text{tex}}{=} \text{“}$
\mathsf {u}”]
$[\mathsf{v} \overset{\text{tex}}{=} \text{“}$
\mathsf {v}”]
$[\mathsf{w} \overset{\text{tex}}{=} \text{“}$
\mathsf {w}”]
$[(*)^{\mathrm{M}} \overset{\text{tex}}{=} \text{“}$
(#1.
)ˆM”]
$[\mathrm{If}(*,*,*) \overset{\text{tex}}{=} \text{“}$
\mathrm {If}(#1.
, \linebreak [0]#2.
, \linebreak [0]#3.
)”]
$[\mathrm{array}\{*\}\,*\,\mathrm{end\ array} \overset{\text{tex}}{=} \text{“}$
\begin {array}{#1.
}#2.
\end {array}”]
$[\mathrm{array}\{*\}\,*\,\mathrm{end\ array} \overset{\text{name}}{=} \text{“}$\mathrm {array}\{#1.
\}#2.
\mathrm {end\ array}”]
$[\mathrm{l} \overset{\text{tex}}{=} \text{“}$
l”]
$[\mathrm{c} \overset{\text{tex}}{=} \text{“}$
c”]
$[\mathrm{r} \overset{\text{tex}}{=} \text{“}$
r”]
$[\mathrm{empty} \overset{\text{tex}}{=} \text{“}$
”]
$[\mathrm{empty} \overset{\text{name}}{=} \text{“}$
\mathrm {empty}”]
$[\langle * \mid * := * \rangle \overset{\text{tex}}{=} \text{“}$
\langle #1.
\, {|}#2.
{:=}\, #3.
\rangle ”]
$[\mathcal{M}(*) \overset{\text{tex}}{=} \text{“}$
{\cal M}(#1.
)”]
$[\mathbf{apply}(*,*) \overset{\text{tex}}{=} \text{“}$
\mathbf {apply}(#1.

,#2.
)”]

[$\mathbf{apply}_1(*,*) \overset{\text{tex}}{=}$ “
\mathbf {apply}_1(#1.
,#2.
)”]

[$*^{\mathrm{H}} \overset{\text{tex}}{=}$ “#1.
{}^H”]

[$*^{\mathrm{T}} \overset{\text{tex}}{=}$ “#1.
{}^T”]

[$*^{\mathrm{c}} \overset{\text{tex}}{=}$ “#1.
{}^c”]

[$*^{\mathrm{d}} \overset{\text{tex}}{=}$ “#1.
{}^d”]

[$*^{\mathrm{U}} \overset{\text{tex}}{=}$ “#1.
{}^U”]

[$*^{\mathrm{h}} \overset{\text{tex}}{=}$ “#1.
{}^h”]

[$*^{\mathrm{t}} \overset{\text{tex}}{=}$ “#1.
{}^t”]

[$*^{\mathrm{s}} \overset{\text{tex}}{=}$ “#1.
{}^s”]

[$*^{\mathrm{B}} \overset{\text{tex}}{=}$ “#1.
{}^B”]

[$*^{\mathrm{C}} \overset{\text{tex}}{=}$ “#1.
{}^C”]

[$*^{\mathrm{M}} \overset{\text{tex}}{=}$ “#1.
{}^M”]

[$\tilde{\mathcal{U}}(*) \overset{\text{tex}}{=}$ “
\tilde {{\cal U}}(#1.
)”]

[$\mathcal{U}(*) \overset{\text{tex}}{=}$ “
{\cal U}(#1.
)”]

[$\mathcal{U}^{\mathrm{M}}(*) \overset{\text{tex}}{=}$ “
{\cal U}^M(#1.
)”]

[$* \therefore * \overset{\text{tex}}{=}$ “#1.
\mathrel { \dot { . \, . } }#2.”]

[$* \underline{\therefore} * \overset{\text{tex}}{=}$ “#1.
\mathrel { \underline { \dot { . \, . } } }#2.”]

$[* \underline{::} * \overset{\text{tex}}{=} \text{“}\#1.$
$\mathrel{ \underline{ : \, : } }\#2.\text{”}]$
$[* \underline{+2*} * \overset{\text{tex}}{=} \text{“}\#1.$
$\mathrel{ \underline{ \{+\} \, 2 \, \ast \, } }\#2.\text{”}]$
$[\mathsf{x} :: \mathsf{y} \overset{\text{tex}}{=} \text{“}\#1.$
$\mathrel{ \{ : \, : \} }\#2.\text{”}]$
$[* +2* * \overset{\text{tex}}{=} \text{“}\#1.$
$\mathrel{ \{+\} \, 2 \, \ast \, }\#2.\text{”}]$
$[* \overset{\text{B}}{\approx} * \overset{\text{tex}}{=} \text{“}\#1.$
$\stackrel{\text{B}}{\{}\approx \}\#2.\text{”}]$
$[* \overset{\text{D}}{\approx} * \overset{\text{tex}}{=} \text{“}\#1.$
$\stackrel{\text{D}}{\{}\approx \}\#2.\text{”}]$
$[* \overset{\text{C}}{\approx} * \overset{\text{tex}}{=} \text{“}\#1.$
$\stackrel{\text{C}}{\{}\approx \}\#2.\text{”}]$
$[* \overset{\text{P}}{\approx} * \overset{\text{tex}}{=} \text{“}\#1.$
$\stackrel{\text{P}}{\{}\approx \}\#2.\text{”}]$
$[* \approx * \overset{\text{tex}}{=} \text{“}\#1.$
$\approx \#2.\text{”}]$
$[* = * \overset{\text{tex}}{=} \text{“}\#1.$
$=\#2.\text{”}]$
$[* \overset{+}{\to} * \overset{\text{tex}}{=} \text{“}\#1.$
$\stackrel{+}{\{}\rightarrow \}\#2.\text{”}]$
$[\neg * \overset{\text{tex}}{=} \text{“}$
$\{\neg \}\#1.\text{”}]$
$[* \wedge * \overset{\text{tex}}{=} \text{“}\#1.$
$\wedge \#2.\text{”}]$
$[* \vee * \overset{\text{tex}}{=} \text{“}\#1.$
$\vee \#2.\text{”}]$
$[* \parallel * \overset{\text{tex}}{=} \text{“}\#1.$
$\parallel \#2.\text{”}]$
$[* : * \overset{\text{tex}}{=} \text{“}\#1.$
$:\#2.\text{”}]$
$[*!* \overset{\text{tex}}{=} \text{“}\#1.$
$!\#2.\text{”}]$
$[* \left\{ \begin{array}{l} * \\ * \end{array} \right. \overset{\text{tex}}{=} \text{“}\#1.$
$\left\{\{\protect \begin{array}\{l\}\#2.$
$\backslash\backslash\#3.$
$\protect \end \{array\}\right.\text{”}]$
$[*\&* \overset{\text{tex}}{=} \text{“}\#1.$
$\&\#2.\text{”}]$

$[*\&* \overset{\text{name}}{=} \text{“}\#1.$
$\backslash\& \#2.\text{”}]$
$[*\backslash\backslash* \overset{\text{tex}}{=} \text{“}\#1.$
$\backslash\backslash \#2.\text{”}]$
$[*\backslash\backslash* \overset{\text{name}}{=} \text{“}\#1.$
$\backslash\text{backslash } \backslash\text{backslash } \#2.\text{”}]$
$[\text{macro} \overset{\text{tex}}{=} \text{“}$
$\backslash\text{mathrm } \{\text{macro}\}\text{”}]$
$[\text{"value"} \overset{\text{tex}}{=} \text{“}$
$\backslash\text{mbox } \{\backslash\text{tt } \backslash\text{char34}\}\backslash\text{mathrm } \{\text{value}\}\backslash\text{mbox } \{\backslash\text{tt } \backslash\text{char34}\}\text{”}]$
$[\text{"macro"} \overset{\text{tex}}{=} \text{“}$
$\backslash\text{mbox } \{\backslash\text{tt } \backslash\text{char34}\}\backslash\text{mathrm } \{\text{macro}\}\backslash\text{mbox } \{\backslash\text{tt } \backslash\text{char34}\}\text{”}]$
$[\mathcal{E}(*,*,*) \overset{\text{tex}}{=} \text{“}$
$\{\backslash\text{cal E}\}(\#1.$
$,\#2.$
$,\#3.$
$)\text{”}]$
$[\mathcal{E}_2(*,*,*,*,*) \overset{\text{tex}}{=} \text{“}$
$\{\backslash\text{cal E}\}\_2(\#1.$
$,\#2.$
$,\#3.$
$,\#4.$
$,\#5.$
$)\text{”}]$
$[\mathcal{E}_3(*,*,*,*) \overset{\text{tex}}{=} \text{“}$
$\{\backslash\text{cal E}\}\_3(\#1.$
$,\#2.$
$,\#3.$
$,\#4.$
$)\text{”}]$
$[\mathcal{E}_4(*,*,*,*) \overset{\text{tex}}{=} \text{“}$
$\{\backslash\text{cal E}\}\_4(\#1.$
$,\#2.$
$,\#3.$
$,\#4.$
$)\text{”}]$
$[\mathbf{lookup}(*,*,*) \overset{\text{tex}}{=} \text{“}$
$\backslash\text{mathbf } \{\text{lookup}\}(\#1.$
$,\#2.$
$,\#3.$
$)\text{”}]$
$[\mathbf{abstract}(*,*,*,*) \overset{\text{tex}}{=} \text{“}$
$\backslash\text{mathbf } \{\text{abstract}\}(\#1.$

, #2.
, #3.
, #4.
)”]
[$*^{a} \overset{\text{tex}}{=}$ “#1.
{}ˆa”]
[$\lceil * \rceil \overset{\text{tex}}{=}$ “
\lceil #1.
\rceil ”]
[$*0 \overset{\text{tex}}{=}$ “#1.
0”]
[$*1 \overset{\text{tex}}{=}$ “#1.
1”]
[0b $\overset{\text{tex}}{=}$ “
0 \mathrm {b}”]
[$* \overset{\text{t}}{=} * \overset{\text{tex}}{=}$ “#1.
\stackrel {t}{=}#2.”]
[$* \overset{\text{t}^*}{=} * \overset{\text{tex}}{=}$ “#1.
\stackrel {tˆ*}{=}#2.”]
[$*[\, *\,] \overset{\text{tex}}{=}$ “#1.
{[}#2.
{]}”]
[$\mathcal{M}(*, *, *) \overset{\text{tex}}{=}$ “
{\cal M}(#1.
, #2.
, #3.
)”]
[$\mathcal{M}_2(*, *, *, *) \overset{\text{tex}}{=}$ “
{\cal M}_2(#1.
, #2.
, #3.
, #4.
)”]
[$s_0 \overset{\text{tex}}{=}$ “
s_0”]
[$\mathcal{M}^*(*, *, *) \overset{\text{tex}}{=}$ “
{\cal M}ˆ*(#1.
, #2.
, #3.
)”]
[$\mathbf{zip}(*, *) \overset{\text{tex}}{=}$ “
\mathbf {zip}(#1.

,#2.
)”]
$[\mathbf{assoc}_1(*,*,*) \overset{\text{tex}}{=} $“
\mathbf {assoc}_1(#1.
,#2.
,#3.
)”]
$[*_\{*\} \overset{\text{tex}}{=} $“#1.
_{#2.
}”]
$[*_\{*\} \overset{\text{name}}{=} $“#1.
_\{#2.
\}”]
$[(*)^{\mathbf{p}} \overset{\text{tex}}{=} $“(#1.
)^{\mathbf {p}}”]
$([[* \ddot{=} *] \overset{\text{tex}}{\to} $“
[#1/tex name/tex.
\mathrel {\ddot {=}}#2.
]”])^{\mathbf{p}}$
$[[* \dot{=} *] \overset{\text{tex}}{=} $“
[#1/tex name/tex.
\mathrel {\dot {=}}#2.
]”]
$[[* \acute{=} *] \overset{\text{tex}}{=} $“
[#1/tex name/tex.
\mathrel {\acute {=}}#2.
]”]
$[[* \overset{\text{pyk}}{=} *] \overset{\text{tex}}{=} $“
[#1/tex name/tex.
\stackrel {\mathrm {pyk}}{=} #2/tex name.
]”]
$[[* \overset{\text{tex}}{=} *] \overset{\text{tex}}{=} $“
[#1/tex name/tex.
\stackrel {\mathrm {tex}}{=} #2/tex name.
]”]
$[[* \overset{\text{name}}{=} *] \overset{\text{tex}}{=} $“
[#1/tex name/tex.
\stackrel {\mathrm {name}}{=} #2/tex name.
]”]
$[\tilde{\mathcal{M}}_1 \overset{\text{tex}}{=} $“
\tilde {{\cal M}}_1”]
$[\tilde{\mathcal{M}}_2(*) \overset{\text{tex}}{=} $“
\tilde {{\cal M}}_2(#1.

)"]
[$\tilde{\mathcal{M}}_3(*) \overset{\text{tex}}{=}$ "
\tilde {{\cal M}}_3(#1.
)"]
[$\tilde{\mathcal{M}}_4(*,*,*,*) \overset{\text{tex}}{=}$ "
\tilde {{\cal M}}_4(#1.
,#2.
,#3.
,#4.
)"]
[$\tilde{\mathcal{M}}(*,*,*) \overset{\text{tex}}{=}$ "
\tilde {{\cal M}}(#1.
,#2.
,#3.
)"]
[$\tilde{\mathcal{Q}}(*,*,*) \overset{\text{tex}}{=}$ "
\tilde {{\cal Q}}(#1.
,#2.
,#3.
)"]
[$\tilde{\mathcal{Q}}_2(*,*,*) \overset{\text{tex}}{=}$ "
\tilde {{\cal Q}}_2(#1.
,#2.
,#3.
)"]
[$\tilde{\mathcal{Q}}_3(*,*,*,*) \overset{\text{tex}}{=}$ "
\tilde {{\cal Q}}_3(#1.
,#2.
,#3.
,#4.
)"]
[$\tilde{\mathcal{Q}}^*(*,*,*) \overset{\text{tex}}{=}$ "
\tilde {{\cal Q}}^*(#1.
,#2.
,#3.
)"]
[$(*) \overset{\text{tex}}{=}$ "
(#1.
)"]

A.4 Line numbers

The following definitions are experimental definitions of line numbers named "ell a", "ell b", etc. which expand into [L_{01}], [L_{02}], etc. in such a way that

proof lines are numbered in succession. The “ell dummy” operator expands into different line numbers each time it is used and can be used for lines that are never referenced.

[$L_a \overset{pyk}{=}$ “ell a”] [$L_b \overset{pyk}{=}$ “ell b”] [$L_c \overset{pyk}{=}$ “ell c”] [$L_d \overset{pyk}{=}$ “ell d”] [$L_e \overset{pyk}{=}$ “ell e”]
[$L_f \overset{pyk}{=}$ “ell f”] [$L_g \overset{pyk}{=}$ “ell g”] [$L_h \overset{pyk}{=}$ “ell h”] [$L_i \overset{pyk}{=}$ “ell i”] [$L_j \overset{pyk}{=}$ “ell j”]
[$L_k \overset{pyk}{=}$ “ell k”] [$L_l \overset{pyk}{=}$ “ell l”] [$L_m \overset{pyk}{=}$ “ell m”] [$L_n \overset{pyk}{=}$ “ell n”] [$L_o \overset{pyk}{=}$ “ell o”]
[$L_p \overset{pyk}{=}$ “ell p”] [$L_q \overset{pyk}{=}$ “ell q”] [$L_r \overset{pyk}{=}$ “ell r”] [$L_s \overset{pyk}{=}$ “ell s”] [$L_t \overset{pyk}{=}$ “ell t”]
[$L_u \overset{pyk}{=}$ “ell u”] [$L_v \overset{pyk}{=}$ “ell v”] [$L_w \overset{pyk}{=}$ “ell w”] [$L_x \overset{pyk}{=}$ “ell x”] [$L_y \overset{pyk}{=}$ “ell y”]
[$L_z \overset{pyk}{=}$ “ell z”] [$L_A \overset{pyk}{=}$ “ell big a”] [$L_B \overset{pyk}{=}$ “ell big b”] [$L_C \overset{pyk}{=}$ “ell big c”]
[$L_D \overset{pyk}{=}$ “ell big d”] [$L_E \overset{pyk}{=}$ “ell big e”] [$L_F \overset{pyk}{=}$ “ell big f”] [$L_G \overset{pyk}{=}$ “ell big g”]
[$L_H \overset{pyk}{=}$ “ell big h”] [$L_I \overset{pyk}{=}$ “ell big i”] [$L_J \overset{pyk}{=}$ “ell big j”] [$L_K \overset{pyk}{=}$ “ell big k”]
[$L_L \overset{pyk}{=}$ “ell big l”] [$L_M \overset{pyk}{=}$ “ell big m”] [$L_N \overset{pyk}{=}$ “ell big n”] [$L_O \overset{pyk}{=}$ “ell big o”]
[$L_P \overset{pyk}{=}$ “ell big p”] [$L_Q \overset{pyk}{=}$ “ell big q”] [$L_R \overset{pyk}{=}$ “ell big r”] [$L_S \overset{pyk}{=}$ “ell big s”]
[$L_T \overset{pyk}{=}$ “ell big t”] [$L_U \overset{pyk}{=}$ “ell big u”] [$L_V \overset{pyk}{=}$ “ell big v”] [$L_W \overset{pyk}{=}$ “ell big w”]
[$L_X \overset{pyk}{=}$ “ell big x”] [$L_Y \overset{pyk}{=}$ “ell big y”] [$L_Z \overset{pyk}{=}$ “ell big z”] [$L_? \overset{pyk}{=}$ “ell dummy”]
[$L_a \overset{name}{=}$ “L_a”] [$L_b \overset{name}{=}$ “L_b”] [$L_c \overset{name}{=}$ “L_c”] [$L_d \overset{name}{=}$ “L_d”] [$L_e \overset{name}{=}$ “L_e”]
[$L_f \overset{name}{=}$ “L_f”] [$L_g \overset{name}{=}$ “L_g”] [$L_h \overset{name}{=}$ “L_h”] [$L_i \overset{name}{=}$ “L_i”] [$L_j \overset{name}{=}$ “L_j”]
[$L_k \overset{name}{=}$ “L_k”] [$L_l \overset{name}{=}$ “L_l”] [$L_m \overset{name}{=}$ “L_m”] [$L_n \overset{name}{=}$ “L_n”] [$L_o \overset{name}{=}$ “L_o”]
[$L_p \overset{name}{=}$ “L_p”] [$L_q \overset{name}{=}$ “L_q”] [$L_r \overset{name}{=}$ “L_r”] [$L_s \overset{name}{=}$ “L_s”] [$L_t \overset{name}{=}$ “L_t”]
[$L_u \overset{name}{=}$ “L_u”] [$L_v \overset{name}{=}$ “L_v”] [$L_w \overset{name}{=}$ “L_w”] [$L_x \overset{name}{=}$ “L_x”]
[$L_y \overset{name}{=}$ “L_y”] [$L_z \overset{name}{=}$ “L_z”] [$L_A \overset{name}{=}$ “L_A”] [$L_B \overset{name}{=}$ “L_B”]
[$L_C \overset{name}{=}$ “L_C”] [$L_D \overset{name}{=}$ “L_D”] [$L_E \overset{name}{=}$ “L_E”] [$L_F \overset{name}{=}$ “L_F”]
[$L_G \overset{name}{=}$ “L_G”] [$L_H \overset{name}{=}$ “L_H”] [$L_I \overset{name}{=}$ “L_I”] [$L_J \overset{name}{=}$ “L_J”]
[$L_K \overset{name}{=}$ “L_K”] [$L_L \overset{name}{=}$ “L_L”] [$L_M \overset{name}{=}$ “L_M”] [$L_N \overset{name}{=}$ “L_N”]
[$L_O \overset{name}{=}$ “L_O”] [$L_P \overset{name}{=}$ “L_P”] [$L_Q \overset{name}{=}$ “L_Q”] [$L_R \overset{name}{=}$ “L_R”]
[$L_S \overset{name}{=}$ “L_S”] [$L_T \overset{name}{=}$ “L_T”] [$L_U \overset{name}{=}$ “L_U”] [$L_V \overset{name}{=}$ “L_V”]
[$L_W \overset{name}{=}$ “L_W”] [$L_X \overset{name}{=}$ “L_X”] [$L_Y \overset{name}{=}$ “L_Y”] [$L_Z \overset{name}{=}$ “L_Z”]
[$L_? \overset{name}{=}$ “L_?”]

[$L_a \overset{tex}{=}$ “
\if \relax \csname lgwprooflinep\endcsname L_a \else
\if \relax \csname lgwella\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwella {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwella \fi ”]
[$L_b \overset{tex}{=}$ “
\if \relax \csname lgwprooflinep\endcsname L_b \else
\if \relax \csname lgwellb\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellb {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }

```
\fi \lgwellb \fi "]
    [L_c ^tex= "
\if \relax \csname lgwprooflinep\endcsname L_c \else
\if \relax \csname lgwellc\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellc {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellc \fi "]
    [L_d ^tex= "
\if \relax \csname lgwprooflinep\endcsname L_d \else
\if \relax \csname lgwelld\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwelld {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwelld \fi "]
    [L_e ^tex= "
\if \relax \csname lgwprooflinep\endcsname L_e \else
\if \relax \csname lgwelle\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwelle {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwelle \fi "]
    [L_f ^tex= "
\if \relax \csname lgwprooflinep\endcsname L_f \else
\if \relax \csname lgwellf\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellf {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellf \fi "]
    [L_g ^tex= "
\if \relax \csname lgwprooflinep\endcsname L_g \else
\if \relax \csname lgwellg\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellg {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellg \fi "]
    [L_h ^tex= "
\if \relax \csname lgwprooflinep\endcsname L_h \else
\if \relax \csname lgwellh\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellh {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellh \fi "]
    [L_i ^tex= "
\if \relax \csname lgwprooflinep\endcsname L_i \else
\if \relax \csname lgwelli\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwelli {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwelli \fi "]
```

```
    [L_j ≝ “
\if \relax \csname lgwprooflinep\endcsname L_j \else
\if \relax \csname lgwellj\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellj {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellj \fi ”]
    [L_k ≝ “
\if \relax \csname lgwprooflinep\endcsname L_k \else
\if \relax \csname lgwellk\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellk {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellk \fi ”]
    [L_l ≝ “
\if \relax \csname lgwprooflinep\endcsname L_l \else
\if \relax \csname lgwelll\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwelll {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwelll \fi ”]
    [L_m ≝ “
\if \relax \csname lgwprooflinep\endcsname L_m \else
\if \relax \csname lgwellm\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellm {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellm \fi ”]
    [L_n ≝ “
\if \relax \csname lgwprooflinep\endcsname L_n \else
\if \relax \csname lgwelln\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwelln {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwelln \fi ”]
    [L_o ≝ “
\if \relax \csname lgwprooflinep\endcsname L_o \else
\if \relax \csname lgwello\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwello {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwello \fi ”]
    [L_p ≝ “
\if \relax \csname lgwprooflinep\endcsname L_p \else
\if \relax \csname lgwellp\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellp {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellp \fi ”]
    [L_q ≝ “
\if \relax \csname lgwprooflinep\endcsname L_q \else
```

```
\if \relax \csname lgwellq\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellq {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellq \fi "]
    [L_r ^{tex} = "
\if \relax \csname lgwprooflinep\endcsname L_r \else
\if \relax \csname lgwellr\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellr {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellr \fi "]
    [L_s ^{tex} = "
\if \relax \csname lgwprooflinep\endcsname L_s \else
\if \relax \csname lgwells\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwells {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwells \fi "]
    [L_t ^{tex} = "
\if \relax \csname lgwprooflinep\endcsname L_t \else
\if \relax \csname lgwellt\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellt {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellt \fi "]
    [L_u ^{tex} = "
\if \relax \csname lgwprooflinep\endcsname L_u \else
\if \relax \csname lgwellu\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellu {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellu \fi "]
    [L_v ^{tex} = "
\if \relax \csname lgwprooflinep\endcsname L_v \else
\if \relax \csname lgwellv\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellv {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellv \fi "]
    [L_w ^{tex} = "
\if \relax \csname lgwprooflinep\endcsname L_w \else
\if \relax \csname lgwellw\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellw {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellw \fi "]
    [L_x ^{tex} = "
\if \relax \csname lgwprooflinep\endcsname L_x \else
\if \relax \csname lgwellx\endcsname
\global \advance \lgwproofline by 1
```

```
\xdef \lgwellx {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellx \fi "]
    [L_y ^{tex} = "
\if \relax \csname lgwprooflinep\endcsname L_y \else
\if \relax \csname lgwelly\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwelly {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwelly \fi "]
    [L_z ^{tex} = "
\if \relax \csname lgwprooflinep\endcsname L_z \else
\if \relax \csname lgwellz\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellz {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellz \fi "]
    [L_A ^{tex} = "
\if \relax \csname lgwprooflinep\endcsname L_A \else
\if \relax \csname lgwellbiga\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbiga {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbiga \fi "]
    [L_B ^{tex} = "
\if \relax \csname lgwprooflinep\endcsname L_B \else
\if \relax \csname lgwellbigb\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigb {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigb \fi "]
    [L_C ^{tex} = "
\if \relax \csname lgwprooflinep\endcsname L_C \else
\if \relax \csname lgwellbigc\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigc {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigc \fi "]
    [L_D ^{tex} = "
\if \relax \csname lgwprooflinep\endcsname L_D \else
\if \relax \csname lgwellbigd\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigd {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigd \fi "]
    [L_E ^{tex} = "
\if \relax \csname lgwprooflinep\endcsname L_E \else
\if \relax \csname lgwellbige\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbige {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbige \fi "]
```

[$L_F \overset{\text{tex}}{=}$ “
\if \relax \csname lgwprooflinep\endcsname L_F \else
\if \relax \csname lgwellbigf\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigf {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigf \fi ”]
[$L_G \overset{\text{tex}}{=}$ “
\if \relax \csname lgwprooflinep\endcsname L_G \else
\if \relax \csname lgwellbigg\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigg {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigg \fi ”]
[$L_H \overset{\text{tex}}{=}$ “
\if \relax \csname lgwprooflinep\endcsname L_H \else
\if \relax \csname lgwellbigh\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigh {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigh \fi ”]
[$L_I \overset{\text{tex}}{=}$ “
\if \relax \csname lgwprooflinep\endcsname L_I \else
\if \relax \csname lgwellbigi\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigi {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigi \fi ”]
[$L_J \overset{\text{tex}}{=}$ “
\if \relax \csname lgwprooflinep\endcsname L_J \else
\if \relax \csname lgwellbigj\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigj {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigj \fi ”]
[$L_K \overset{\text{tex}}{=}$ “
\if \relax \csname lgwprooflinep\endcsname L_K \else
\if \relax \csname lgwellbigk\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigk {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigk \fi ”]
[$L_L \overset{\text{tex}}{=}$ “
\if \relax \csname lgwprooflinep\endcsname L_L \else
\if \relax \csname lgwellbigl\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigl {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigl \fi ”]
[$L_M \overset{\text{tex}}{=}$ “
\if \relax \csname lgwprooflinep\endcsname L_M \else

```
\if \relax \csname lgwellbigm\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigm {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigm \fi ”]
    [L_N ^{tex} = “
\if \relax \csname lgwprooflinep\endcsname L_N \else
\if \relax \csname lgwellbign\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbign {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbign \fi ”]
    [L_O ^{tex} = “
\if \relax \csname lgwprooflinep\endcsname L_O \else
\if \relax \csname lgwellbigo\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigo {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigo \fi ”]
    [L_P ^{tex} = “
\if \relax \csname lgwprooflinep\endcsname L_P \else
\if \relax \csname lgwellbigp\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigp {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigp \fi ”]
    [L_Q ^{tex} = “
\if \relax \csname lgwprooflinep\endcsname L_Q \else
\if \relax \csname lgwellbigq\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigq {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigq \fi ”]
    [L_R ^{tex} = “
\if \relax \csname lgwprooflinep\endcsname L_R \else
\if \relax \csname lgwellbigr\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigr {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigr \fi ”]
    [L_S ^{tex} = “
\if \relax \csname lgwprooflinep\endcsname L_S \else
\if \relax \csname lgwellbigs\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigs {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigs \fi ”]
    [L_T ^{tex} = “
\if \relax \csname lgwprooflinep\endcsname L_T \else
\if \relax \csname lgwellbigt\endcsname
\global \advance \lgwproofline by 1
```

```
\xdef \lgwellbigt {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigt \fi "]
    [L_U ^{tex} =
\if \relax \csname lgwprooflinep\endcsname L_U \else
\if \relax \csname lgwellbigu\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigu {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigu \fi "]
    [L_V ^{tex} =
\if \relax \csname lgwprooflinep\endcsname L_V \else
\if \relax \csname lgwellbigv\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigv {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigv \fi "]
    [L_W ^{tex} =
\if \relax \csname lgwprooflinep\endcsname L_W \else
\if \relax \csname lgwellbigw\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigw {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigw \fi "]
    [L_X ^{tex} =
\if \relax \csname lgwprooflinep\endcsname L_X \else
\if \relax \csname lgwellbigx\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigx {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigx \fi "]
    [L_Y ^{tex} =
\if \relax \csname lgwprooflinep\endcsname L_Y \else
\if \relax \csname lgwellbigy\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigy {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigy \fi "]
    [L_Z ^{tex} =
\if \relax \csname lgwprooflinep\endcsname L_Z \else
\if \relax \csname lgwellbigz\endcsname
\global \advance \lgwproofline by 1
\xdef \lgwellbigz {L\ifnum \lgwproofline <10 0\fi \number \lgwproofline }
\fi \lgwellbigz \fi "]
    [L_? ^{tex} =
\if \relax \csname lgwprooflinep\endcsname L_? \else
\global \advance \lgwproofline by 1
L\ifnum \lgwproofline <10 0\fi \number \lgwproofline
\fi "]
```

A.5 Characters

A.5.1 Pyk aspects of characters

[
x $\overset{\text{pyk}}{=}$ “
unicode newline ∗”]
[x $\overset{\text{pyk}}{=}$ “unicode space ∗”]
[!x $\overset{\text{pyk}}{=}$ “unicode exclamation mark ∗”]
[”x $\overset{\text{pyk}}{=}$ “unicode quotation mark ∗”]
[#x $\overset{\text{pyk}}{=}$ “unicode number sign ∗”]
[$x $\overset{\text{pyk}}{=}$ “unicode dollar sign ∗”]
[%x $\overset{\text{pyk}}{=}$ “unicode percent ∗”]
[&x $\overset{\text{pyk}}{=}$ “unicode ampersand ∗”]
[’x $\overset{\text{pyk}}{=}$ “unicode apostrophe ∗”]
[(x $\overset{\text{pyk}}{=}$ “unicode left parenthesis ∗”]
[)x $\overset{\text{pyk}}{=}$ “unicode right parenthesis ∗”]
[∗x $\overset{\text{pyk}}{=}$ “unicode asterisk ∗”]
[+x $\overset{\text{pyk}}{=}$ “unicode plus sign ∗”]
[,x $\overset{\text{pyk}}{=}$ “unicode comma ∗”]
[-x $\overset{\text{pyk}}{=}$ “unicode hyphen ∗”]
[.x $\overset{\text{pyk}}{=}$ “unicode period ∗”]
[/x $\overset{\text{pyk}}{=}$ “unicode slash ∗”]
[0x $\overset{\text{pyk}}{=}$ “unicode zero ∗”]
[1x $\overset{\text{pyk}}{=}$ “unicode one ∗”]
[2x $\overset{\text{pyk}}{=}$ “unicode two ∗”]
[3x $\overset{\text{pyk}}{=}$ “unicode three ∗”]
[4x $\overset{\text{pyk}}{=}$ “unicode four ∗”]
[5x $\overset{\text{pyk}}{=}$ “unicode five ∗”]
[6x $\overset{\text{pyk}}{=}$ “unicode six ∗”]
[7x $\overset{\text{pyk}}{=}$ “unicode seven ∗”]
[8x $\overset{\text{pyk}}{=}$ “unicode eight ∗”]
[9x $\overset{\text{pyk}}{=}$ “unicode nine ∗”]
[:x $\overset{\text{pyk}}{=}$ “unicode colon ∗”]
[;x $\overset{\text{pyk}}{=}$ “unicode semicolon ∗”]
[<x $\overset{\text{pyk}}{=}$ “unicode less than ∗”]
[=x $\overset{\text{pyk}}{=}$ “unicode equal sign ∗”]

[>x $\overset{\text{pyk}}{=}$ “unicode greater than ∗”]
[?x $\overset{\text{pyk}}{=}$ “unicode question mark ∗”]
[@x $\overset{\text{pyk}}{=}$ “unicode commercial at ∗”]
[Ax $\overset{\text{pyk}}{=}$ “unicode capital a ∗”]
[Bx $\overset{\text{pyk}}{=}$ “unicode capital b ∗”]
[Cx $\overset{\text{pyk}}{=}$ “unicode capital c ∗”]
[Dx $\overset{\text{pyk}}{=}$ “unicode capital d ∗”]
[Ex $\overset{\text{pyk}}{=}$ “unicode capital e ∗”]
[Fx $\overset{\text{pyk}}{=}$ “unicode capital f ∗”]
[Gx $\overset{\text{pyk}}{=}$ “unicode capital g ∗”]
[Hx $\overset{\text{pyk}}{=}$ “unicode capital h ∗”]
[Ix $\overset{\text{pyk}}{=}$ “unicode capital i ∗”]
[Jx $\overset{\text{pyk}}{=}$ “unicode capital j ∗”]
[Kx $\overset{\text{pyk}}{=}$ “unicode capital k ∗”]
[Lx $\overset{\text{pyk}}{=}$ “unicode capital l ∗”]
[Mx $\overset{\text{pyk}}{=}$ “unicode capital m ∗”]
[Nx $\overset{\text{pyk}}{=}$ “unicode capital n ∗”]
[Ox $\overset{\text{pyk}}{=}$ “unicode capital o ∗”]
[Px $\overset{\text{pyk}}{=}$ “unicode capital p ∗”]
[Qx $\overset{\text{pyk}}{=}$ “unicode capital q ∗”]
[Rx $\overset{\text{pyk}}{=}$ “unicode capital r ∗”]
[Sx $\overset{\text{pyk}}{=}$ “unicode capital s ∗”]
[Tx $\overset{\text{pyk}}{=}$ “unicode capital t ∗”]
[Ux $\overset{\text{pyk}}{=}$ “unicode capital u ∗”]
[Vx $\overset{\text{pyk}}{=}$ “unicode capital v ∗”]
[Wx $\overset{\text{pyk}}{=}$ “unicode capital w ∗”]
[Xx $\overset{\text{pyk}}{=}$ “unicode capital x ∗”]
[Yx $\overset{\text{pyk}}{=}$ “unicode capital y ∗”]
[Zx $\overset{\text{pyk}}{=}$ “unicode capital z ∗”]
[[x $\overset{\text{pyk}}{=}$ “unicode left bracket ∗”]
[\x $\overset{\text{pyk}}{=}$ “unicode backslash ∗”]
[]x $\overset{\text{pyk}}{=}$ “unicode right bracket ∗”]
[^x $\overset{\text{pyk}}{=}$ “unicode circumflex ∗”]
[_x $\overset{\text{pyk}}{=}$ “unicode underscore ∗”]
[ax $\overset{\text{pyk}}{=}$ “unicode small a ∗”]

[bx $\overset{\text{pyk}}{=}$ “unicode small b ∗”]
[cx $\overset{\text{pyk}}{=}$ “unicode small c ∗”]
[dx $\overset{\text{pyk}}{=}$ “unicode small d ∗”]
[ex $\overset{\text{pyk}}{=}$ “unicode small e ∗”]
[fx $\overset{\text{pyk}}{=}$ “unicode small f ∗”]
[gx $\overset{\text{pyk}}{=}$ “unicode small g ∗”]
[hx $\overset{\text{pyk}}{=}$ “unicode small h ∗”]
[ix $\overset{\text{pyk}}{=}$ “unicode small i ∗”]
[jx $\overset{\text{pyk}}{=}$ “unicode small j ∗”]
[kx $\overset{\text{pyk}}{=}$ “unicode small k ∗”]
[lx $\overset{\text{pyk}}{=}$ “unicode small l ∗”]
[mx $\overset{\text{pyk}}{=}$ “unicode small m ∗”]
[nx $\overset{\text{pyk}}{=}$ “unicode small n ∗”]
[ox $\overset{\text{pyk}}{=}$ “unicode small o ∗”]
[px $\overset{\text{pyk}}{=}$ “unicode small p ∗”]
[qx $\overset{\text{pyk}}{=}$ “unicode small q ∗”]
[rx $\overset{\text{pyk}}{=}$ “unicode small r ∗”]
[sx $\overset{\text{pyk}}{=}$ “unicode small s ∗”]
[tx $\overset{\text{pyk}}{=}$ “unicode small t ∗”]
[ux $\overset{\text{pyk}}{=}$ “unicode small u ∗”]
[vx $\overset{\text{pyk}}{=}$ “unicode small v ∗”]
[wx $\overset{\text{pyk}}{=}$ “unicode small w ∗”]
[xx $\overset{\text{pyk}}{=}$ “unicode small x ∗”]
[yx $\overset{\text{pyk}}{=}$ “unicode small y ∗”]
[zx $\overset{\text{pyk}}{=}$ “unicode small z ∗”]
[‘x $\overset{\text{pyk}}{=}$ “unicode grave accent ∗”]
[{x $\overset{\text{pyk}}{=}$ “unicode left brace ∗”]
[|x $\overset{\text{pyk}}{=}$ “unicode vertical line ∗”]
[}x $\overset{\text{pyk}}{=}$ “unicode right brace ∗”]
[˜x $\overset{\text{pyk}}{=}$ “unicode tilde ∗”]

A.5.2 Tex aspects of characters

[
x $\overset{\text{tex}}{=}$ “
#1.”]
[x $\overset{\text{tex}}{=}$ “ #1.”]

[!x $\overset{\text{tex}}{=}$ “!#1.”]
[”x $\overset{\text{tex}}{=}$ “”#1.”]
[#x $\overset{\text{tex}}{=}$ “#.#1.”]
[$x $\overset{\text{tex}}{=}$ “$#1.”]
[%x $\overset{\text{tex}}{=}$ “%#1.”]
[&x $\overset{\text{tex}}{=}$ “&#1.”]
[’x $\overset{\text{tex}}{=}$ “’#1.”]
[(x $\overset{\text{tex}}{=}$ “(#1.”]
[)x $\overset{\text{tex}}{=}$ “)#1.”]
[∗x $\overset{\text{tex}}{=}$ “∗#1.”]
[+x $\overset{\text{tex}}{=}$ “+#1.”]
[,x $\overset{\text{tex}}{=}$ “,#1.”]
[-x $\overset{\text{tex}}{=}$ “-#1.”]
[.x $\overset{\text{tex}}{=}$ “.#1.”]
[/x $\overset{\text{tex}}{=}$ “/#1.”]
[0x $\overset{\text{tex}}{=}$ “0#1.”]
[1x $\overset{\text{tex}}{=}$ “1#1.”]
[2x $\overset{\text{tex}}{=}$ “2#1.”]
[3x $\overset{\text{tex}}{=}$ “3#1.”]
[4x $\overset{\text{tex}}{=}$ “4#1.”]
[5x $\overset{\text{tex}}{=}$ “5#1.”]
[6x $\overset{\text{tex}}{=}$ “6#1.”]
[7x $\overset{\text{tex}}{=}$ “7#1.”]
[8x $\overset{\text{tex}}{=}$ “8#1.”]
[9x $\overset{\text{tex}}{=}$ “9#1.”]
[:x $\overset{\text{tex}}{=}$ “:#1.”]
[;x $\overset{\text{tex}}{=}$ “;#1.”]
[<x $\overset{\text{tex}}{=}$ “<#1.”]
[=x $\overset{\text{tex}}{=}$ “=#1.”]
[>x $\overset{\text{tex}}{=}$ “>#1.”]
[?x $\overset{\text{tex}}{=}$ “?#1.”]
[@x $\overset{\text{tex}}{=}$ “@#1.”]
[Ax $\overset{\text{tex}}{=}$ “A#1.”]
[Bx $\overset{\text{tex}}{=}$ “B#1.”]
[Cx $\overset{\text{tex}}{=}$ “C#1.”]
[Dx $\overset{\text{tex}}{=}$ “D#1.”]
[Ex $\overset{\text{tex}}{=}$ “E#1.”]
[Fx $\overset{\text{tex}}{=}$ “F#1.”]

[Gx $\overset{\text{tex}}{=}$ “G#1.”]
[Hx $\overset{\text{tex}}{=}$ “H#1.”]
[Ix $\overset{\text{tex}}{=}$ “I#1.”]
[Jx $\overset{\text{tex}}{=}$ “J#1.”]
[Kx $\overset{\text{tex}}{=}$ “K#1.”]
[Lx $\overset{\text{tex}}{=}$ “L#1.”]
[Mx $\overset{\text{tex}}{=}$ “M#1.”]
[Nx $\overset{\text{tex}}{=}$ “N#1.”]
[Ox $\overset{\text{tex}}{=}$ “O#1.”]
[Px $\overset{\text{tex}}{=}$ “P#1.”]
[Qx $\overset{\text{tex}}{=}$ “Q#1.”]
[Rx $\overset{\text{tex}}{=}$ “R#1.”]
[Sx $\overset{\text{tex}}{=}$ “S#1.”]
[Tx $\overset{\text{tex}}{=}$ “T#1.”]
[Ux $\overset{\text{tex}}{=}$ “U#1.”]
[Vx $\overset{\text{tex}}{=}$ “V#1.”]
[Wx $\overset{\text{tex}}{=}$ “W#1.”]
[Xx $\overset{\text{tex}}{=}$ “X#1.”]
[Yx $\overset{\text{tex}}{=}$ “Y#1.”]
[Zx $\overset{\text{tex}}{=}$ “Z#1.”]
[[x $\overset{\text{tex}}{=}$ “[#1.”]
[\x $\overset{\text{tex}}{=}$ “\#1.”]
[]x $\overset{\text{tex}}{=}$ “]#1.”]
[^x $\overset{\text{tex}}{=}$ “^#1.”]
[_x $\overset{\text{tex}}{=}$ “_#1.”]
[ax $\overset{\text{tex}}{=}$ “a#1.”]
[bx $\overset{\text{tex}}{=}$ “b#1.”]
[cx $\overset{\text{tex}}{=}$ “c#1.”]
[dx $\overset{\text{tex}}{=}$ “d#1.”]
[ex $\overset{\text{tex}}{=}$ “e#1.”]
[fx $\overset{\text{tex}}{=}$ “f#1.”]
[gx $\overset{\text{tex}}{=}$ “g#1.”]
[hx $\overset{\text{tex}}{=}$ “h#1.”]
[ix $\overset{\text{tex}}{=}$ “i#1.”]
[jx $\overset{\text{tex}}{=}$ “j#1.”]
[kx $\overset{\text{tex}}{=}$ “k#1.”]
[lx $\overset{\text{tex}}{=}$ “l#1.”]
[mx $\overset{\text{tex}}{=}$ “m#1.”]

[nx $\overset{\text{tex}}{=}$ “n#1.”]
[ox $\overset{\text{tex}}{=}$ “o#1.”]
[px $\overset{\text{tex}}{=}$ “p#1.”]
[qx $\overset{\text{tex}}{=}$ “q#1.”]
[rx $\overset{\text{tex}}{=}$ “r#1.”]
[sx $\overset{\text{tex}}{=}$ “s#1.”]
[tx $\overset{\text{tex}}{=}$ “t#1.”]
[ux $\overset{\text{tex}}{=}$ “u#1.”]
[vx $\overset{\text{tex}}{=}$ “v#1.”]
[wx $\overset{\text{tex}}{=}$ “w#1.”]
[xx $\overset{\text{tex}}{=}$ “x#1.”]
[yx $\overset{\text{tex}}{=}$ “y#1.”]
[zx $\overset{\text{tex}}{=}$ “z#1.”]
[‘x $\overset{\text{tex}}{=}$ “‘#1.”]
[{x $\overset{\text{tex}}{=}$ “{#1.”]
[|x $\overset{\text{tex}}{=}$ “|#1.”]
[}x $\overset{\text{tex}}{=}$ “}#1.”]
[˜x $\overset{\text{tex}}{=}$ “˜#1.”]

A.5.3 Tex name aspects of characters

[
x $\overset{\text{name}}{=}$ “
\newline #1.”]
[x $\overset{\text{name}}{=}$ “
\linebreak [0]\ \hskip0em plus2.0em{}#1.”]
[#x $\overset{\text{name}}{=}$ “
\#.#1.”]
[$x $\overset{\text{name}}{=}$ “
\$#1.”]
[%x $\overset{\text{name}}{=}$ “
\%#1.”]
[&x $\overset{\text{name}}{=}$ “
\&#1.”]
[’x $\overset{\text{name}}{=}$ “
\mbox {’}#1.”]
[∗x $\overset{\text{name}}{=}$ “
{∗}#1.”]
[+x $\overset{\text{name}}{=}$ “
{+}#1.”]

$[\mathsf{-x} \overset{\text{name}}{=} \text{“}\backslash\text{mbox\{-\}\#1.”}]$

$[\mathsf{:x} \overset{\text{name}}{=} \text{“}\{:\}\#1.”]$

$[\mathsf{<x} \overset{\text{name}}{=} \text{“}\{<\}\#1.”]$

$[\mathsf{=x} \overset{\text{name}}{=} \text{“}\{=\}\#1.”]$

$[\mathsf{>x} \overset{\text{name}}{=} \text{“}\{>\}\#1.”]$

$[\mathsf{\backslash x} \overset{\text{name}}{=} \text{“}\backslash\text{mbox }\{\$\backslash\text{backslash }\$\}\#1.”]$

$[\mathsf{\hat{}x} \overset{\text{name}}{=} \text{“}\{\backslash\text{char94}\}\#1.”]$

$[\mathsf{\_x} \overset{\text{name}}{=} \text{“}\backslash\_\#1.”]$

$[\mathsf{`x} \overset{\text{name}}{=} \text{“}\backslash\text{mbox }\{\text{`}\}\#1.”]$

$[\mathsf{\{x} \overset{\text{name}}{=} \text{“}\backslash\{\#1.”]$

$[\mathsf{\}x} \overset{\text{name}}{=} \text{“}\backslash\}\#1.”]$

$[\mathsf{\tilde{}x} \overset{\text{name}}{=} \text{“}\backslash\text{char126 }\#1.”]$

B Test

$[\mathsf{T} + 0 \approx \mathsf{T}]^{\cdot}$

$[0 + \mathsf{T} \approx \mathsf{T}]^{\cdot}$

$[\mathsf{T} + \mathsf{T} \approx \mathsf{T}]^{\cdot}$

$[0 + 0 \approx 0]^{\cdot}$

$[0 + 1 \approx 1]^{\cdot}$

$[0 + 2 \approx 2]^{\cdot}$

$[0 + 3 \approx 3]^{\cdot}$

$[1 + 0 \approx 1]^{\cdot}$

$[1 + 1 \approx 2]^{\cdot}$

$[1 + 2 \approx 3]^{\cdot}$

$[1 + 3 \approx 4]^{\cdot}$

$[2 + 0 \approx 2]^{\cdot}$

$[2 + 1 \approx 3]^{\cdot}$

$[2 + 2 \approx 4]^{\cdot}$

$[2 + 3 \approx 5]^{\cdot}$

$[3 + 0 \approx 3]^{\cdot}$

$[3 + 1 \approx 4]^{\cdot}$

$[3 + 2 \approx 5]^{\cdot}$

$[3 + 3 \approx 6]^{\cdot}$

$[\mathsf{T} < 0 \approx \mathsf{F}]^{\cdot}$

$[0 < \mathsf{T} \approx \mathsf{F}]^{\cdot}$

$[\mathsf{T} < \mathsf{T} \approx \mathsf{F}]^{\cdot}$

$[0 < 0 \approx \mathsf{F}]^{\cdot}$

$[0 < 1 \approx \mathsf{T}]^{\cdot}$

$[0 < 2 \approx \mathsf{T}]^{\cdot}$

$[0 < 3 \approx \mathsf{T}]^{\cdot}$

$[1 < 0 \approx \mathsf{F}]^{\cdot}$

$[1 < 1 \approx \mathsf{F}]^{\cdot}$

$[1 < 2 \approx \mathsf{T}]^{\cdot}$

$[1 < 3 \approx \mathsf{T}]^{\cdot}$

$[2 < 0 \approx \mathsf{F}]^{\cdot}$

$[2 < 1 \approx \mathsf{F}]^{\cdot}$

$[2 < 2 \approx \mathsf{F}]^{\cdot}$

$[2 < 3 \approx \mathsf{T}]^{\cdot}$

$[3 < 0 \approx \mathsf{F}]^{\cdot}$

$[3 < 1 \approx \mathsf{F}]^{\cdot}$

$[3 < 2 \approx \mathsf{F}]^{\cdot}$

$[3 < 3 \approx \mathsf{F}]^{\cdot}$

$[\mathsf{T} - 0 \approx \mathsf{T}]^{\cdot}$

$[0 - \mathsf{T} \approx \mathsf{T}]^{\cdot}$

$[\mathsf{T} - \mathsf{T} \approx \mathsf{T}]^{\cdot}$

$[0 - 0 \approx 0]^{\cdot}$

$[0 - 1 \approx 0]^{\cdot}$

$[0 - 2 \approx 0]^{\cdot}$

$[0 - 3 \approx 0]^{\cdot}$

$[1 - 0 \approx 1]^{\cdot}$

$[1 - 1 \approx 0]^{\cdot}$

$[1 - 2 \approx 0]^{\cdot}$

$[1 - 3 \approx 0]^{\cdot}$

$[2 - 0 \approx 2]^{\cdot}$

$[2 - 1 \approx 1]^{\cdot}$

$[2 - 2 \approx 0]^{\cdot}$

$[2 - 3 \approx 0]^{\cdot}$

$[3 - 0 \approx 3]^{\cdot}$

$[3 - 1 \approx 2]^{\cdot}$

$[3 - 2 \approx 1]^{\cdot}$

$[3 - 3 \approx 0]^{\cdot}$

$[\mathsf{T} \cdot 0 \approx \mathsf{T}]^{\cdot}$

$[0 \cdot \mathsf{T} \approx \mathsf{T}]^{\cdot}$

$[\mathsf{T} \cdot \mathsf{T} \approx \mathsf{T}]^{\cdot}$

$[0 \cdot 0 \approx 0]^{\cdot}$

$[0 \cdot 1 \approx 0]^{\cdot}$

$[0 \cdot 2 \approx 0]^{\cdot}$

$[0 \cdot 3 \approx 0]^{\cdot}$

$[1 \cdot 0 \approx 0]^{\cdot}$

$[1 \cdot 1 \approx 1]^{\cdot}$

$[1 \cdot 2 \approx 2]^{\cdot}$

$[1 \cdot 3 \approx 3]^{\cdot}$

$[2 \cdot 0 \approx 0]^{\cdot}$

$[2 \cdot 1 \approx 2]^{\cdot}$

$[2 \cdot 2 \approx 4]^{\cdot}$

$[2 \cdot 3 \approx 6]^{\cdot}$

$[3 \cdot 0 \approx 0]^{\cdot}$

$[3 \cdot 1 \approx 3]^{\cdot}$

$[3 \cdot 2 \approx 6]^{\cdot}$

$[3 \cdot 3 \approx 9]^{\cdot}$

$[\mathrm{bit}(\mathsf{T}, 0) \approx \mathsf{T}]^{\cdot}$

$[\mathrm{bit}(0, \mathsf{T}) \approx \mathsf{T}]^{\cdot}$

$[\mathrm{bit}(\mathsf{T}, \mathsf{T}) \approx \mathsf{T}]^{\cdot}$

$[\mathrm{bit}(0, 0) \approx \mathsf{T}]^{\cdot}$

$[\mathrm{bit}(0, 1) \approx \mathsf{F}]^{\cdot}$

$[\mathrm{bit}(0, 2) \approx \mathsf{T}]^{\cdot}$

$[\mathrm{bit}(0, 3) \approx \mathsf{F}]^{\cdot}$

$[\mathrm{bit}(1, 0) \approx \mathsf{T}]^{\cdot}$

$[\mathrm{bit}(1, 1) \approx \mathsf{T}]^{\cdot}$

$[\mathrm{bit}(1, 2) \approx \mathsf{F}]^{\cdot}$

$[\mathrm{bit}(1, 3) \approx \mathsf{F}]^{\cdot}$

$[\mathrm{bit}(2, 0) \approx \mathsf{T}]^{\cdot}$

$[\mathrm{bit}(2, 1) \approx \mathsf{T}]^{\cdot}$

$[\mathrm{bit}(2, 2) \approx \mathsf{T}]^{\cdot}$

$[\mathrm{bit}(2, 3) \approx \mathsf{T}]^{\cdot}$

$[\mathrm{bit}(3, 0) \approx \mathsf{T}]^{\cdot}$

$[\mathrm{bit}(3, 1) \approx \mathsf{T}]^{\cdot}$

$[\mathrm{bit}(3, 2) \approx \mathsf{T}]^{\cdot}$

$[\mathrm{bit}(3, 3) \approx \mathsf{T}]^{\cdot}$

$[\mathrm{identifier}(\ulcorner \text{“code”} \urcorner) \approx 0\mathrm{b}11001010110010001101111011000 11]^{\cdot}$

$[\mathsf{T}[0{\to}1] \approx 0 :: 1]^{\cdot}$

$[\mathsf{T}[0{\to}1][1{\to}2] \approx (0 :: 1) :: (1 :: 2)]^{\cdot}$

$[\mathsf{T}[0{\to}1][2{\to}4] \approx ((0 :: 1) :: (2 :: 4)) :: \mathsf{T}]^{\cdot}$

$[\mathsf{T}[1{\to}2][3{\to}6] \approx \mathsf{T} :: ((1 :: 2) :: (3 :: 6))]^{\cdot}$

$[\mathsf{T}[0{\to}1][2{\to}4][0{\to}\mathsf{T}] \approx \mathsf{T}[2{\to}4]]^{\cdot}$

$[\mathsf{T}[0{\to}1][2{\to}4][2{\to}\mathsf{T}] \approx \mathsf{T}[0{\to}1]]^{\cdot}$

$[\mathsf{T}[1{\to}2][3{\to}6][1{\to}\mathsf{T}] \approx \mathsf{T}[3{\to}6]]^{\cdot}$

$[\mathsf{T}[1{\to}2][3{\to}6][3{\to}\mathsf{T}] \approx \mathsf{T}[1{\to}2]]^{\cdot}$

$[\mathsf{T}[1{\to}7][2{\to}8][4{\to}9][1] \approx 7]^{\cdot}$

$[\mathsf{T}[1{\to}7][2{\to}8][4{\to}9][2] \approx 8]^{\cdot}$

$[\mathsf{T}[1{\to}7][2{\to}8][4{\to}9][4] \approx 9]^{\cdot}$

$[\mathsf{T}[\langle 1,2,3\rangle{\Rightarrow}1][\langle 2\rangle{\Rightarrow}2][\langle 1,1\rangle{\Rightarrow}3][\langle 1,2,4\rangle{\Rightarrow}4][1][2][3] \approx 1]^{\cdot}$

$[\langle 2,3,2\rangle \approx \mathbf{let}\ \mathsf{x} = 2\ \mathbf{in}\ \langle \mathsf{x},3,\mathsf{x}\rangle]^{\cdot}$

$[0 \approx \mathbf{let}\ \mathsf{x} = \mathsf{F}\ \mathbf{in}\ \mathsf{T} \therefore \mathsf{F}]^{\cdot}$

$[0 \approx \mathbf{let}\ \mathsf{x} = \mathsf{F}\ \mathbf{in}\ \mathsf{T} \therefore \mathsf{x}]^{\cdot}$

$[\mathsf{T} :: 0 \approx (\underline{0} \therefore \underline{0} \therefore \mathsf{T}) \therefore \lambda \mathsf{y}.\mathsf{y}]^{\cdot}$

$[\mathsf{T} :: \mathsf{T} \approx \mathbf{let}\ \mathsf{x} = \lambda \mathsf{y}.\mathsf{y}\ \mathbf{in}\ (\underline{0} \therefore \underline{0} \therefore \mathsf{T}) \therefore \mathsf{x}]^{\cdot}$

$[(\ulcorner 2 \urcorner \vdash \ulcorner 3 \urcorner) \stackrel{\mathrm{t}}{=} \ulcorner 2 \vdash 3 \urcorner]^{\cdot}$

$[(\ulcorner 2 \urcorner \Vdash \ulcorner 3 \urcorner) \stackrel{\mathrm{t}}{=} \ulcorner 2 \Vdash 3 \urcorner]^{\cdot}$

$[(\forall \ulcorner 2 \urcorner : \ulcorner 3 \urcorner) \stackrel{\mathrm{t}}{=} \ulcorner \forall 2 : 3 \urcorner]^{\cdot}$

$[\perp\!\!\!\perp \stackrel{\mathrm{t}}{=} \ulcorner \perp\!\!\!\perp \urcorner]^{\cdot}$

$[(\ulcorner 2 \urcorner \oplus \ulcorner 3 \urcorner) \stackrel{\mathrm{t}}{=} \ulcorner 2 \oplus 3 \urcorner]^{\cdot}$

$[(\ulcorner 2 \urcorner^{\mathrm{I}}) \stackrel{\mathrm{t}}{=} \ulcorner 2^{\mathrm{I}} \urcorner]^{\cdot}$

$[(\ulcorner 2 \urcorner^{\triangleright}) \stackrel{\mathrm{t}}{=} \ulcorner 2^{\triangleright} \urcorner]^{\cdot}$

$[(\ulcorner 2 \urcorner^{\mathrm{V}}) \stackrel{\mathrm{t}}{=} \ulcorner 2^{\mathrm{V}} \urcorner]^{\cdot}$

$[(\ulcorner 2 \urcorner^{+}) \stackrel{\mathrm{t}}{=} \ulcorner 2^{+} \urcorner]^{\cdot}$

$[(\ulcorner 2 \urcorner^{-}) \overset{\mathrm{t}}{=} \ulcorner 2^{-} \urcorner]^{\cdot}$

$[(\ulcorner 2 \urcorner^{*}) \overset{\mathrm{t}}{=} \ulcorner 2^{*} \urcorner]^{\cdot}$

$[(\ulcorner 2 \urcorner \mathbin{@} \ulcorner 3 \urcorner) \overset{\mathrm{t}}{=} \ulcorner 2 \mathbin{@} 3 \urcorner]^{\cdot}$

$[(\ulcorner 2 \urcorner \text{ i.e. } \ulcorner 3 \urcorner) \overset{\mathrm{t}}{=} \ulcorner 2 \text{ i.e. } 3 \urcorner]^{\cdot}$

$[(\ulcorner 2 \urcorner ; \ulcorner 3 \urcorner) \overset{\mathrm{t}}{=} \ulcorner 2 ; 3 \urcorner]^{\cdot}$

$[\ulcorner \mathcal{P} \urcorner^{\mathcal{V}}]^{\cdot}$

$[\ulcorner \mathsf{p} \urcorner^{\mathcal{V}}]^{-}$

$[\ulcorner \mathsf{x} :: \mathsf{y} :: \mathsf{z} \urcorner^{\mathcal{C}}]^{\cdot}$

$[\ulcorner \mathsf{x} :: \mathcal{Y} :: \mathsf{z} \urcorner^{\mathcal{C}}]^{-}$

$[\ulcorner \mathcal{Y} \urcorner \text{ free in } \ulcorner \mathcal{X} :: \mathcal{Y} :: \mathcal{Z} \urcorner]^{\cdot}$

$[\ulcorner \mathcal{Y} \urcorner \text{ free in } \ulcorner \mathcal{X} :: \mathsf{y} :: \mathcal{Z} \urcorner]^{-}$

$[\ulcorner \mathsf{x} :: \mathcal{Y} :: \mathsf{z} \urcorner \text{ free for } \ulcorner \mathcal{X} \urcorner \text{ in } \ulcorner \forall \mathcal{X} \colon \mathsf{b} :: \mathcal{X} :: \mathsf{c} \urcorner]^{\cdot}$

$[\ulcorner \mathsf{x} :: \mathcal{Y} :: \mathsf{z} \urcorner \text{ free for } \ulcorner \mathcal{X} \urcorner \text{ in } \ulcorner \forall \mathcal{Y} \colon \mathsf{b} :: \mathcal{X} :: \mathsf{c} \urcorner]^{-}$

$[\ulcorner \mathsf{x} :: \mathcal{Y} :: \mathsf{z} \urcorner \text{ free for } \ulcorner \mathcal{Y} \urcorner \text{ in } \ulcorner \forall \mathcal{X} \colon \mathsf{b} :: \mathcal{X} :: \mathsf{c} \urcorner]^{\cdot}$

$[\ulcorner \mathsf{x} :: \mathcal{Y} :: \mathsf{z} \urcorner \text{ free for } \ulcorner \mathcal{Y} \urcorner \text{ in } \ulcorner \forall \mathcal{Y} \colon \mathsf{b} :: \mathcal{X} :: \mathsf{c} \urcorner]^{\cdot}$

$[\langle \ulcorner \mathcal{A} :: (\forall \mathcal{A} \colon \mathcal{A}) :: \mathcal{B} :: \mathcal{A} \urcorner \,|\, \ulcorner \mathcal{A} \urcorner := \ulcorner 2 \urcorner \rangle \overset{\mathrm{t}}{=} \ulcorner 2 :: (\forall \mathcal{A} \colon \mathcal{A}) :: \mathcal{B} :: 2 \urcorner]^{\cdot}$

$[\ulcorner \mathsf{y} \urcorner \in_{\mathrm{t}} \langle \ulcorner \mathsf{x} \urcorner, \ulcorner \mathsf{y} \urcorner, \ulcorner \mathsf{z} \urcorner \rangle]^{\cdot}$

$[\ulcorner \mathsf{u} \urcorner \in_{\mathrm{t}} \langle \ulcorner \mathsf{x} \urcorner, \ulcorner \mathsf{y} \urcorner, \ulcorner \mathsf{z} \urcorner \rangle]^{-}$

$[\langle \ulcorner \mathsf{z} \urcorner, \ulcorner \mathsf{x} \urcorner \rangle \subseteq_{\mathrm{T}} \langle \ulcorner \mathsf{x} \urcorner, \ulcorner \mathsf{y} \urcorner, \ulcorner \mathsf{z} \urcorner \rangle]^{\cdot}$

$[\langle \ulcorner \mathsf{z} \urcorner, \ulcorner \mathsf{u} \urcorner \rangle \subseteq_{\mathrm{T}} \langle \ulcorner \mathsf{x} \urcorner, \ulcorner \mathsf{y} \urcorner, \ulcorner \mathsf{z} \urcorner \rangle]^{-}$

$[\langle \ulcorner \mathsf{z} \urcorner, \ulcorner \mathsf{y} \urcorner, \ulcorner \mathsf{x} \urcorner \rangle \overset{\mathrm{T}}{=} \langle \ulcorner \mathsf{x} \urcorner, \ulcorner \mathsf{y} \urcorner, \ulcorner \mathsf{z} \urcorner \rangle]^{\cdot}$

$[\langle \ulcorner \mathsf{z} \urcorner, \ulcorner \mathsf{x} \urcorner \rangle \overset{\mathrm{T}}{=} \langle \ulcorner \mathsf{x} \urcorner, \ulcorner \mathsf{y} \urcorner, \ulcorner \mathsf{z} \urcorner \rangle]^{-}$

$[\langle \ulcorner \mathsf{z} \urcorner, \ulcorner \mathsf{y} \urcorner, \ulcorner \mathsf{x} \urcorner \rangle \overset{\mathrm{T}}{=} \langle \ulcorner \mathsf{x} \urcorner, \ulcorner \mathsf{z} \urcorner \rangle]^{-}$

$[\emptyset \cup \{\ulcorner \mathsf{x} \urcorner\} \cup \{\ulcorner \mathsf{y} \urcorner\} \cup \{\ulcorner \mathsf{z} \urcorner\} \cup \{\ulcorner \mathsf{y} \urcorner\} \overset{\mathrm{t}^{*}}{=} \langle \ulcorner \mathsf{z} \urcorner, \ulcorner \mathsf{y} \urcorner, \ulcorner \mathsf{x} \urcorner \rangle]^{\cdot}$

$[\langle \ulcorner \mathsf{x} \urcorner, \ulcorner \mathsf{y} \urcorner, \ulcorner \mathsf{z} \urcorner \rangle \backslash \{\ulcorner \mathsf{y} \urcorner\} \overset{\mathrm{t}^{*}}{=} \langle \ulcorner \mathsf{x} \urcorner, \ulcorner \mathsf{z} \urcorner \rangle]^{\cdot}$

$[\langle \ulcorner \mathsf{x} \urcorner, \ulcorner \mathsf{y} \urcorner, \ulcorner \mathsf{z} \urcorner \rangle \backslash \{\ulcorner \mathsf{u} \urcorner\} \overset{\mathrm{t}^{*}}{=} \langle \ulcorner \mathsf{x} \urcorner, \ulcorner \mathsf{y} \urcorner, \ulcorner \mathsf{z} \urcorner \rangle]^{\cdot}$

$[\langle \lceil \mathsf{a} \rceil, \lceil \mathsf{y} \rceil, \lceil \mathsf{b} \rceil \rangle \cup \langle \lceil \mathsf{x} \rceil, \lceil \mathsf{y} \rceil, \lceil \mathsf{z} \rceil \rangle \stackrel{\mathrm{t}^*}{=} \langle \lceil \mathsf{a} \rceil, \lceil \mathsf{b} \rceil, \lceil \mathsf{x} \rceil, \lceil \mathsf{y} \rceil, \lceil \mathsf{z} \rceil \rangle]^{\cdot}$

$[\langle \langle \lceil 1 \rceil, \lceil 2 \rceil \rangle, \langle \lceil 3 \rceil, \lceil 4 \rceil \rangle, \lceil 5 \rceil \rangle \stackrel{\mathrm{s}}{=} \langle \langle \lceil 2 \rceil, \lceil 1 \rceil \rangle, \langle \lceil 4 \rceil, \lceil 3 \rceil \rangle, \lceil 5 \rceil \rangle]^{\cdot}$

$[\langle \langle \lceil 6 \rceil, \lceil 2 \rceil \rangle, \langle \lceil 3 \rceil, \lceil 4 \rceil \rangle, \lceil 5 \rceil \rangle \stackrel{\mathrm{s}}{=} \langle \langle \lceil 2 \rceil, \lceil 1 \rceil \rangle, \langle \lceil 4 \rceil, \lceil 3 \rceil \rangle, \lceil 5 \rceil \rangle]^{-}$

$[\langle \langle \lceil 1 \rceil, \lceil 6 \rceil \rangle, \langle \lceil 3 \rceil, \lceil 4 \rceil \rangle, \lceil 5 \rceil \rangle \stackrel{\mathrm{s}}{=} \langle \langle \lceil 2 \rceil, \lceil 1 \rceil \rangle, \langle \lceil 4 \rceil, \lceil 3 \rceil \rangle, \lceil 5 \rceil \rangle]^{-}$

$[\langle \langle \lceil 1 \rceil, \lceil 2 \rceil \rangle, \langle \lceil 6 \rceil, \lceil 4 \rceil \rangle, \lceil 5 \rceil \rangle \stackrel{\mathrm{s}}{=} \langle \langle \lceil 2 \rceil, \lceil 1 \rceil \rangle, \langle \lceil 4 \rceil, \lceil 3 \rceil \rangle, \lceil 5 \rceil \rangle]^{-}$

$[\langle \langle \lceil 1 \rceil, \lceil 2 \rceil \rangle, \langle \lceil 3 \rceil, \lceil 6 \rceil \rangle, \lceil 5 \rceil \rangle \stackrel{\mathrm{s}}{=} \langle \langle \lceil 2 \rceil, \lceil 1 \rceil \rangle, \langle \lceil 4 \rceil, \lceil 3 \rceil \rangle, \lceil 5 \rceil \rangle]^{-}$

$[\langle \langle \lceil 1 \rceil, \lceil 2 \rceil \rangle, \langle \lceil 3 \rceil, \lceil 4 \rceil \rangle, \lceil 6 \rceil \rangle \stackrel{\mathrm{s}}{=} \langle \langle \lceil 2 \rceil, \lceil 1 \rceil \rangle, \langle \lceil 4 \rceil, \lceil 3 \rceil \rangle, \lceil 5 \rceil \rangle]^{-}$

$[\mathcal{U}(\mathcal{E}(\lceil \mathsf{F} +2* \, 2 \approx 5 \rceil, \mathsf{T}, \mathrm{self}))]^{\cdot}$

$[\mathcal{U}^{\mathrm{M}}(\mathcal{E}(\lceil \mathsf{F} +2* \, 2 \approx 5 \rceil, \mathsf{T}, \mathrm{self}))]^{\cdot}$

$[\mathcal{U}^{\mathrm{M}}(\mathcal{E}(\lceil \mathcal{T}(\mathsf{F} +2* \, 2 \approx 5) \rceil, \mathsf{T}, \mathrm{self}) \text{ ‘ } \mathrm{self})]^{\cdot}$

$[\mathcal{S}(\mathrm{self}, \lceil \mathcal{A}^{\mathrm{I}\triangleright} \rceil) \stackrel{\mathrm{s}}{=} \langle \langle \lceil \mathcal{A} \rceil \rangle, \emptyset, \lceil \mathcal{A} \rceil \rangle]^{\cdot}$

$[\lceil 2 \rceil\text{-color}(\lceil 3 \rceil \vdash \lceil 4 \rceil) \stackrel{\mathrm{t}}{=} \lceil 3 \vdash 4 \rceil]^{\cdot}$

$[\mathcal{S}(\mathrm{self}, \lceil (\mathcal{A} \oplus \mathcal{B})^{\mathrm{I}} \rceil) \stackrel{\mathrm{s}}{=} \langle \emptyset, \emptyset, \lceil (\mathcal{A} \oplus \mathcal{B}) \vdash (\mathcal{A} \oplus \mathcal{B}) \rceil \rangle]^{\cdot}$

$[\mathcal{S}(\mathrm{self}, \lceil \mathcal{A} \vdash \mathcal{B}^{\mathrm{I}} \rceil) \stackrel{\mathrm{s}}{=} \langle \emptyset, \emptyset, \lceil \mathcal{A} \vdash \mathcal{B} \vdash \mathcal{B} \rceil \rangle]^{\cdot}$

$[\lceil \mathrm{checker} \rceil \in_{\mathrm{c}} \lceil \mathrm{checker} \wedge_{\mathrm{c}} \mathrm{verifier} \rceil]^{\cdot}$

$[\lceil \mathrm{verifier} \rceil \in_{\mathrm{c}} \lceil \mathrm{checker} \wedge_{\mathrm{c}} \mathrm{verifier} \rceil]^{\cdot}$

$[\lceil 7 \rceil \in_{\mathrm{c}} \lceil \mathrm{checker} \wedge_{\mathrm{c}} \mathrm{verifier} \rceil]^{-}$

$[\mathrm{claims}(\lceil \mathrm{verifier} \rceil, \mathrm{self}, \mathrm{self}[0])]^{\cdot}$

$[\mathrm{claims}(\lceil 7 \rceil, \mathrm{self}, \mathrm{self}[0])]^{-}$

$[\mathrm{inst}(\mathrm{parm}(\lceil \forall \mathsf{x}\colon \forall \mathsf{y}\colon \mathsf{x} + \mathsf{y} \rceil, \mathsf{T}, 4), \mathsf{T}[4 \rightarrow \lceil \mathsf{u} \rceil][8 \rightarrow \lceil \mathsf{v} \rceil]) \stackrel{\mathrm{t}}{=} \lceil \forall \mathsf{u}\colon \forall \mathsf{v}\colon \mathsf{u} + \mathsf{v} \rceil]^{\cdot}$

math test occur eight in parameter term quote not all var x indeed var x end quote substitution true end occur end test end math

math false test occur nine in parameter term quote not all var x indeed var x end quote substitution true end occur end test end math

$[\mathrm{unify}(\lceil 2 \rceil = \lceil 2 \rceil, \mathsf{T})]^{\cdot}$

$[\mathrm{unify}(\lceil 2 \rceil = \lceil 3 \rceil, \mathsf{T}) \approx 0]^{\cdot}$

$[\mathrm{unify}(\lceil 2 + 3 \rceil = \lceil 2 + 3 \rceil, \mathsf{T})]^{\cdot}$

$[\mathrm{unify}(\lceil 2 + 3 \rceil = \lceil 2 + 4 \rceil, \mathsf{T}) \approx 0]^{\cdot}$

$[\text{unify}(\lceil 2 \rceil = 3, \mathsf{T})[3] \stackrel{\text{t}}{=} \lceil 2 \rceil]^{\cdot}$

$[\text{unify}(3 = \lceil 2 \rceil, \mathsf{T})[3] \stackrel{\text{t}}{=} \lceil 2 \rceil]^{\cdot}$

$[\text{unify}(\langle \lceil \mathsf{x} + \mathsf{y} \rceil^{\mathrm{R}}, 1, \lceil 3 \rceil \rangle = \lceil 2 + 3 \rceil, \mathsf{T})[1] \stackrel{\text{t}}{=} \lceil 2 \rceil]^{\cdot}$

$[\text{unify}(\langle \lceil \mathsf{x} + \mathsf{y} \rceil^{\mathrm{R}}, 1, \lceil 3 \rceil \rangle = \langle \lceil \mathsf{x} + \mathsf{y} \rceil^{\mathrm{R}}, \lceil 2 \rceil, 2 \rangle, \mathsf{T})[1] \stackrel{\text{t}}{=} \lceil 2 \rceil]^{\cdot}$

$[\text{unify}(\text{parm}(\lceil \mathcal{A} + 3 \rceil, \langle \lceil \mathcal{A} \rceil :: 1 \rangle, 1) = \text{parm}(\lceil 2 + \mathcal{B} \rceil, \langle \lceil \mathcal{B} \rceil :: 2 \rangle, 1), \mathsf{T})[1] \stackrel{\text{t}}{=} \lceil 2 \rceil]^{\cdot}$

$[\text{unify}(\text{parm}(\lceil \mathcal{A} + 3 \rceil, \langle \lceil \mathcal{A} \rceil :: 1 \rangle, 1) = \text{parm}(\lceil 2 + \mathcal{B} \rceil, \langle \lceil \mathcal{B} \rceil :: 2 \rangle, 1), \mathsf{T})[2] \stackrel{\text{t}}{=} \lceil 3 \rceil]^{\cdot}$

$[\text{inst}(\text{parm}(\lceil \mathcal{A} + \mathcal{B} \rceil, \langle \lceil \mathcal{A} \rceil :: 1, \lceil \mathcal{B} \rceil :: 2 \rangle, 1), \text{unify}(\text{parm}(\lceil \mathcal{A} + 3 \rceil, \langle \lceil \mathcal{A} \rceil :: 1 \rangle, 1) = \text{parm}(\lceil 2 + \mathcal{B} \rceil, \langle \lceil \mathcal{B} \rceil :: 2 \rangle, 1), \mathsf{T})) \stackrel{\text{t}}{=} \lceil 2 + 3 \rceil]^{\cdot}$

C Priority table

([ijcar base $\stackrel{\text{prio}}{\rightarrow}$
Preassociative
[ijcar base], [bracket $*$ end bracket], [big bracket $*$ end bracket], [\$ $*$ \$],
[**flush left** $[*]$], [x], [y], [z], $[[* \bowtie *]]$, $[[* \stackrel{*}{\rightarrow} *]]$, [pyk], [tex], [name], [prio], $[*]$, $[\mathsf{T}]$,
$[\text{if}(*, *, *)]$, $[[* \stackrel{*}{\Rightarrow} *]]$, [val], [claim], $[\bot]$, $[\text{f}(*)]$, $[(*)^{\text{I}}]$, $[\mathsf{F}]$, $[\underline{0}]$, $[\underline{1}]$, $[\underline{2}]$, $[\underline{3}]$, $[\underline{4}]$, $[\underline{5}]$, $[\underline{6}]$,
$[\underline{7}]$, $[\underline{8}]$, $[\underline{9}]$, [0], [1], [2], [3], [4], [5], [6], [7], [8], [9], [a], [b], [c], [d], [e], [f], [g], [h], [i], [j],
[k], [l], [m], [n], [o], [p], [q], [r], [s], [t], [u], [v], [w], $[(*)^{\text{M}}]$, $[\text{If}(*, *, *)]$,
[array\{$*$\} $*$ end array], [l], [c], [r], [empty], $[\langle * \mid * := * \rangle]$, $[\mathcal{M}(*)]$, $[\tilde{\mathcal{U}}(*)]$, $[\mathcal{U}(*)]$,
$[\mathcal{U}^{\text{M}}(*)]$, [**apply**$(*, *)$], [**apply**$_1(*, *)$], [identifier$(*)$], [identifier$_1(*, *)$], [array-plus$(*, *)$], [array-remove$(*, *, *)$], [array-put$(*, *, *, *)$], [array-add$(*, *, *, *, *)$],
[bit$(*, *)$], [bit$_1(*, *)$], [rack], ["vector"], ["bibliography"], ["dictionary"],
["body"], ["codex"], ["expansion"], ["code"], ["cache"], ["diagnose"], ["pyk"],
["tex"], ["texname"], ["value"], ["message"], ["macro"], ["definition"],
["unpack"], ["claim"], ["priority"], ["lambda"], ["apply"], ["true"], ["if"],
["quote"], ["proclaim"], ["define"], ["introduce"], ["hide"], ["pre"], ["post"],
$[\mathcal{E}(*, *, *)]$, $[\mathcal{E}_2(*, *, *, *, *)]$, $[\mathcal{E}_3(*, *, *, *)]$, $[\mathcal{E}_4(*, *, *, *)]$, [**lookup**$(*, *, *)$],
[**abstract**$(*, *, *, *)$], $[\lceil * \rceil]$, $[\mathcal{M}(*, *, *)]$, $[\mathcal{M}_2(*, *, *, *)]$, $[\mathcal{M}^*(*, *, *)]$, [macro],
[s_0], [**zip**$(*, *)$], [**assoc**$_1(*, *, *)$], $[(*)^{\text{P}}]$, [self], $[[* \ddot{=} *]]$, $[[* \dot{=} *]]$, $[[* \acute{=} *]]$,
$[[* \stackrel{\text{pyk}}{=} *]]$, $[[* \stackrel{\text{tex}}{=} *]]$, $[[* \stackrel{\text{name}}{=} *]]$, [**Priority table**$[*]$], $[\tilde{\mathcal{M}}_1]$, $[\tilde{\mathcal{M}}_2(*)]$, $[\tilde{\mathcal{M}}_3(*)]$,
$[\tilde{\mathcal{M}}_4(*, *, *, *)]$, $[\tilde{\mathcal{M}}(*, *, *)]$, $[\tilde{\mathcal{Q}}(*, *, *)]$, $[\tilde{\mathcal{Q}}_2(*, *, *)]$, $[\tilde{\mathcal{Q}}_3(*, *, *, *)]$, $[\tilde{\mathcal{Q}}^*(*, *, *)]$,
$[(*)]$, $[(*)]$, [display$(*)$], [statement$(*)$], $[[*]^{\cdot}]$, $[[*]^{-}]$, [**aspect**$(*, *)$],
[**aspect**$(*, *, *)$], $[\langle * \rangle]$, [**tuple**$_1(*)$], [**tuple**$_2(*)$], [let$_2(*, *)$], [let$_1(*, *)$],
$[[* \stackrel{\text{claim}}{=} *]]$, [checker], [**check**$(*, *)$], [**check**$_2(*, *, *)$], [**check**$_3(*, *, *)$],
[**check**$^*(*, *)$], [**check**$^*_2(*, *, *)$], $[[*]^{\cdot}]$, $[[*]^{-}]$, $[[*]^{\circ}]$, [msg], $[[* \stackrel{\text{msg}}{=} *]]$, [<stmt>],
[stmt], $[[* \stackrel{\text{stmt}}{=} *]]$, [HeadNil$'$], [HeadPair$'$], [Transitivity$'$], $[\perp\!\!\!\perp]$, [Contra$'$], [T'_{E}],
[L_1], $[\underline{*}]$, $[\mathcal{A}]$, $[\mathcal{B}]$, $[\mathcal{C}]$, $[\mathcal{D}]$, $[\mathcal{E}]$, $[\mathcal{F}]$, $[\mathcal{G}]$, $[\mathcal{H}]$, $[\mathcal{I}]$, $[\mathcal{J}]$, $[\mathcal{K}]$, $[\mathcal{L}]$, $[\mathcal{M}]$, $[\mathcal{N}]$, $[\mathcal{O}]$, $[\mathcal{P}]$, $[\mathcal{Q}]$,
$[\mathcal{R}]$, $[\mathcal{S}]$, $[\mathcal{T}]$, $[\mathcal{U}]$, $[\mathcal{V}]$, $[\mathcal{W}]$, $[\mathcal{X}]$, $[\mathcal{Y}]$, $[\mathcal{Z}]$, $[\langle * \mid * := * \rangle]$, $[\langle {}^{*} * \mid * := * \rangle]$, $[\emptyset]$, [Remainder],

$[(*)^{\mathbf{v}}]$, [intro(∗, ∗, ∗, ∗)], [intro(∗, ∗, ∗)], [error(∗, ∗)], [error$_2$(∗, ∗)], [proof(∗, ∗, ∗)], [proof$_2$(∗, ∗)], $[\mathcal{S}(*,*)]$, $[\mathcal{S}^{\mathrm{I}}(*,*)]$, $[\mathcal{S}^{\triangleright}(*,*)]$, $[\mathcal{S}_1^{\triangleright}(*,*,*)]$, $[\mathcal{S}^{\mathrm{E}}(*,*)]$, $[\mathcal{S}_1^{\mathrm{E}}(*,*,*)]$, $[\mathcal{S}^{+}(*,*)]$, $[\mathcal{S}_1^{+}(*,*,*)]$, $[\mathcal{S}^{-}(*,*)]$, $[\mathcal{S}_1^{-}(*,*,*)]$, $[\mathcal{S}^{*}(*,*)]$, $[\mathcal{S}_1^{*}(*,*,*)]$, $[\mathcal{S}_2^{*}(*,*,*,*)]$, $[\mathcal{S}^{@}(*,*)]$, $[\mathcal{S}_1^{@}(*,*,*)]$, $[\mathcal{S}^{\vdash}(*,*)]$, $[\mathcal{S}_1^{\vdash}(*,*,*,*)]$, $[\mathcal{S}^{\Vdash}(*,*)]$, $[\mathcal{S}_1^{\Vdash}(*,*,*,*)]$, $[\mathcal{S}^{\mathrm{i.e.}}(*,*)]$, $[\mathcal{S}_1^{\mathrm{i.e.}}(*,*,*,*)]$, $[\mathcal{S}_2^{\mathrm{i.e.}}(*,*,*,*,*)]$, $[\mathcal{S}^{\forall}(*,*)]$, $[\mathcal{S}_1^{\forall}(*,*,*,*)]$, $[\mathcal{S}^{\mathrm{;}}(*,*)]$, $[\mathcal{S}_1^{\mathrm{;}}(*,*,*)]$, $[\mathcal{S}_2^{\mathrm{;}}(*,*,*,*)]$, $[\mathcal{T}(*)]$, [claims(∗, ∗, ∗)], [claims$_2$(∗, ∗, ∗)], [<proof>], [proof], [[**Lemma** ∗: ∗]], [[**Proof of** ∗: ∗]], [[∗ **lemma** ∗: ∗]], [[∗ **antilemma** ∗: ∗]], [[∗ **rule** ∗: ∗]], [[∗ **antirule** ∗: ∗]], [verifier], $[\mathcal{V}_1(*)]$, $[\mathcal{V}_2(*,*)]$, $[\mathcal{V}_3(*,*,*,*)]$, $[\mathcal{V}_4(*,*)]$, $[\mathcal{V}_5(*,*,*,*)]$, $[\mathcal{V}_6(*,*,*,*)]$, $[\mathcal{V}_7(*,*,*,*)]$, [Cut(∗, ∗)], [Head$_{\oplus}$(∗)], [Tail$_{\oplus}$(∗)], [rule$_1$(∗, ∗)], [rule(∗, ∗)], [Rule tactic], [Plus(∗, ∗)], [[**Theory** ∗]], [theory$_2$(∗, ∗)], [theory$_3$(∗, ∗)], [theory$_4$(∗, ∗, ∗)], [HeadNil″], [HeadPair″], [Transitivity″], [Contra″], [HeadNil], [HeadPair], [Transitivity], [Contra], [T$_{\mathrm{E}}$], [ragged right], [ragged right expansion], [parm(∗, ∗, ∗)], [parm*(∗, ∗, ∗)], [inst(∗, ∗)], [inst*(∗, ∗)], [occur(∗, ∗, ∗)], [occur*(∗, ∗, ∗)], [unify(∗ = ∗, ∗)], [unify*(∗ = ∗, ∗)], [unify$_2$(∗ = ∗, ∗)], [L$_{\mathrm{a}}$], [L$_{\mathrm{b}}$], [L$_{\mathrm{c}}$], [L$_{\mathrm{d}}$], [L$_{\mathrm{e}}$], [L$_{\mathrm{f}}$], [L$_{\mathrm{g}}$], [L$_{\mathrm{h}}$], [L$_{\mathrm{i}}$], [L$_{\mathrm{j}}$], [L$_{\mathrm{k}}$], [L$_{\mathrm{l}}$], [L$_{\mathrm{m}}$], [L$_{\mathrm{n}}$], [L$_{\mathrm{o}}$], [L$_{\mathrm{p}}$], [L$_{\mathrm{q}}$], [L$_{\mathrm{r}}$], [L$_{\mathrm{s}}$], [L$_{\mathrm{t}}$], [L$_{\mathrm{u}}$], [L$_{\mathrm{v}}$], [L$_{\mathrm{w}}$], [L$_{\mathrm{x}}$], [L$_{\mathrm{y}}$], [L$_{\mathrm{z}}$], [L$_{\mathrm{A}}$], [L$_{\mathrm{B}}$], [L$_{\mathrm{C}}$], [L$_{\mathrm{D}}$], [L$_{\mathrm{E}}$], [L$_{\mathrm{F}}$], [L$_{\mathrm{G}}$], [L$_{\mathrm{H}}$], [L$_{\mathrm{I}}$], [L$_{\mathrm{J}}$], [L$_{\mathrm{K}}$], [L$_{\mathrm{L}}$], [L$_{\mathrm{M}}$], [L$_{\mathrm{N}}$], [L$_{\mathrm{O}}$], [L$_{\mathrm{P}}$], [L$_{\mathrm{Q}}$], [L$_{\mathrm{R}}$], [L$_{\mathrm{S}}$], [L$_{\mathrm{T}}$], [L$_{\mathrm{U}}$], [L$_{\mathrm{V}}$], [L$_{\mathrm{W}}$], [L$_{\mathrm{X}}$], [L$_{\mathrm{Y}}$], [L$_{\mathrm{Z}}$], [L$_{?}$], [Reflexivity], [Reflexivity$_1$], [Commutativity], [Commutativity$_1$], [<tactic>], [tactic], [[∗ $\overset{\text{tactic}}{=}$ ∗]], $[\mathcal{P}(*,*,*)]$, $[\mathcal{P}^*(*,*,*)]$, [p$_0$], [conclude$_1$(∗, ∗)], [conclude$_2$(∗, ∗, ∗)], [conclude$_3$(∗, ∗, ∗, ∗)], [conclude$_4$(∗, ∗)];

Preassociative

[∗_{∗}], [∗/indexintro(∗, ∗, ∗, ∗)], [∗/intro(∗, ∗, ∗)], [∗/bothintro(∗, ∗, ∗, ∗, ∗)], [∗/nameintro(∗, ∗, ∗, ∗)], [∗′], [∗[∗]], [∗[∗⟶∗]], [∗[∗⇒∗]], [∗0], [∗1], [0b], [∗-color(∗)], [∗-color*(∗)], [∗$^{\mathrm{H}}$], [∗$^{\mathrm{T}}$], [∗$^{\mathrm{U}}$], [∗$^{\mathrm{h}}$], [∗$^{\mathrm{t}}$], [∗$^{\mathrm{s}}$], [∗$^{\mathrm{c}}$], [∗$^{\mathrm{d}}$], [∗$^{\mathrm{a}}$], [∗$^{\mathrm{C}}$], [∗$^{\mathrm{M}}$], [∗$^{\mathrm{B}}$], [∗$^{\mathrm{r}}$], [∗$^{\mathrm{i}}$], [∗$^{\mathrm{d}}$], [∗$^{\mathrm{R}}$], [∗^{0}], [∗^{1}], [∗^{2}], [∗^{3}], [∗^{4}], [∗^{5}], [∗^{6}], [∗^{7}], [∗^{8}], [∗^{9}], [∗$^{\mathrm{E}}$], [∗$^{\mathcal{V}}$], [∗$^{\mathcal{C}}$], [∗$^{\mathcal{C}^*}$];

Preassociative

[“ ∗ ”], [], [(∗)$^{\mathbf{t}}$], [string(∗) + ∗], [string(∗) ++ ∗], [∗], [∗], [!∗], [”∗], [#∗], [$∗], [%∗], [&∗], [’∗], [(∗], [)∗], [∗∗], [+∗], [, ∗], [-∗], [.∗], [/∗], [0∗], [1∗], [2∗], [3∗], [4∗], [5∗], [6∗], [7∗], [8∗], [9∗], [:∗], [; ∗], [<∗], [=∗], [>∗], [?∗], [@∗], [A∗], [B∗], [C∗], [D∗], [E∗], [F∗], [G∗], [H∗], [I∗], [J∗], [K∗], [L∗], [M∗], [N∗], [O∗], [P∗], [Q∗], [R∗], [S∗], [T∗], [U∗], [V∗], [W∗], [X∗], [Y∗], [Z∗], [[∗], [\∗], []∗], [ˆ∗], [_∗], [‘∗], [a∗], [b∗], [c∗], [d∗], [e∗], [f∗], [g∗], [h∗], [i∗], [j∗], [k∗], [l∗], [m∗], [n∗], [o∗], [p∗], [q∗], [r∗], [s∗], [t∗], [u∗], [v∗], [w∗], [x∗], [y∗], [z∗], [{∗], [|∗], [}∗], [˜∗], [**Preassociative** ∗; ∗], [**Postassociative** ∗; ∗], [[∗], ∗], [priority ∗ end], [newline ∗], [macro newline ∗];

Preassociative

[∗ ’ ∗], [∗ ‘ ∗];

Preassociative

[∗ · ∗], [∗ ·$_0$ ∗];

Preassociative

[∗ + ∗], [∗ +$_0$ ∗], [∗ +$_1$ ∗], [∗ − ∗], [∗ −$_0$ ∗], [∗ −$_1$ ∗];

Preassociative

[∗ ∪ {∗}], [∗ ∪ ∗], [∗\{∗}];

Postassociative
$[* \mathrel{\therefore} *], [* \mathrel{\underline{\therefore}} *], [* \mathrel{\underline{::}} *], [* \mathrel{\underline{+2*}} *], [* :: *], [* +2* *];$
Postassociative
$[*, *];$
Preassociative
$[* \overset{\mathrm{B}}{\approx} *], [* \overset{\mathrm{D}}{\approx} *], [* \overset{\mathrm{C}}{\approx} *], [* \overset{\mathrm{P}}{\approx} *], [* \approx *], [* = *], [* \overset{+}{\to} *], [* \overset{\mathrm{t}}{=} *], [* \overset{\mathrm{t}^*}{=} *], [* \overset{\mathrm{r}}{=} *],$
$[* \in_{\mathrm{t}} *], [* \subseteq_{\mathrm{T}} *], [* \overset{\mathrm{T}}{=} *], [* \overset{\mathrm{s}}{=} *], [* \text{ free in } *], [* \text{ free in}^* *], [* \text{ free for } * \text{ in } *],$
$[* \text{ free for}^* * \text{ in } *], [* \in_{\mathrm{c}} *], [* < *], [* <' *], [* \leq' *];$
Preassociative
$[\neg *];$
Preassociative
$[* \wedge *], [* \ddot{\wedge} *], [* \tilde{\wedge} *], [* \wedge_{\mathrm{c}} *];$
Preassociative
$[* \vee *], [* \parallel *], [* \ddot{\vee} *];$
Postassociative
$[* \ddot{\Rightarrow} *];$
Postassociative
$[* : *], [* \text{ spy } *], [*!*];$
Preassociative
$[* \begin{cases} * \\ * \end{cases}];$
Preassociative
$[\lambda * .*], [\Lambda * .*], [\Lambda *], [\textbf{if } * \textbf{ then } * \textbf{ else } *], [\textbf{let } * = * \textbf{ in } *], [\textbf{let } * \ddot{=} * \text{ in } *];$
Preassociative
$[*^{\mathrm{I}}], [*^{\triangleright}], [*^{\mathrm{V}}], [*^{+}], [*^{-}], [*^*];$
Preassociative
$[* @ *], [* \triangleright *], [* \blacktriangleright *], [* \gg *];$
Postassociative
$[* \vdash *], [* \Vdash *], [* \text{ i.e. } *];$
Preassociative
$[\forall *: *];$
Postassociative
$[* \oplus *];$
Postassociative
$[*; *];$
Preassociative
$[* \text{ proves } *];$
Preassociative
$[* \textbf{ proof of } * : *], [\text{Line } * : * \gg *; *], [\text{Last line } * \gg * \Box],$
$[\text{Line } * : \text{Premise} \gg *; *], [\text{Line } * : \text{Side-condition} \gg *; *], [\text{Arbitrary} \gg *; *],$
$[\text{Local} \gg * = *; *];$
Postassociative
$[* \text{ then } *], [*[*]*];$
Preassociative
$[*\&*];$

Preassociative
$[*\backslash\backslash *];])^{\mathbf{P}}$

D Index

[1] C. Berline and K. Grue. A κ-denotational semantics for Map Theory in ZFC+SI. *Theoretical Computer Science*, 179(1–2):137–202, jun 1997.

[2] A. Church. *The Calculi of Lambda-Conversion*. Princeton University Press, 1941.

[3] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 12(XXXVIII):173–198, 1931.

[4] K. Grue. Map theory. *Theoretical Computer Science*, 102(1):1–133, jul 1992.

[5] K. Grue. *Mathematics and Computation (lecture notes)*, volume 1–3. DIKU, 7. edition, 2001.
http://www.diku.dk/~grue/papers/mac0102/.

[6] K. Grue. Logiweb. In Fairouz Kamareddine, editor, *Mathematical Knowledge Management Symposium 2003*, volume 93 of *Electronic Notes in Theoretical Computer Science*, pages 70–101. Elsevier, 2004.

[7] D. Knuth. *The TeXbook*. Addison Wesley, 1983.

[8] C. P. Wadsworth M. J. Gordon, A. J. Milner. *Edinburgh LCF, A mechanised logic of computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

[9] J. McCarthy. Recursive functions of symbolic expressions and their computation by machine. *Communications of the ACM*, pages 184–195, 1960.

[10] E. Mendelson. *Introduction to Mathematical Logic*. Wadsworth and Brooks, 3. edition, 1987.

[11] Guy L. Steele. *Common Lisp—The Language*. Digital Press, second edition, 1990.