A Logiweb base page

Klaus Grue

GRD-2009-11-13.UTC:08:44:26.210720

Contents

1 Introduction **3**
1.1 Electronic appendices . . . 3
1.2 Proclamations . . . 4
1.3 Aspect declarations . . . 5
1.4 Definition of pre- and post-associativity . . . 9
1.5 The Logiweb system . . . 9

2 Computation **10**
2.1 Reduction system . . . 10
2.2 Normal forms . . . 10
2.3 Maps . . . 10
2.4 Full reduction system . . . 11
2.5 Elementary definitions . . . 12
2.6 Booleans . . . 14
2.7 Naturals . . . 14
2.8 Bottom . . . 16

3 Tagged values **16**
3.1 Tags . . . 16
3.2 Tag querying . . . 17
3.3 Guards . . . 17
3.4 Normalization . . . 17
3.5 Booleans . . . 18
3.6 Integers . . . 18
3.7 Integer addition . . . 19
3.8 Integer subtraction . . . 19
3.9 Integer multiplication . . . 19
3.10 Integer comparison . . . 19
3.11 Integer construction . . . 20
3.12 Logical operations on integers . . . 21
3.13 Vector operations . . . 22
3.14 Division . . . 26
3.15 Pairs . . . 27

3.16 Exceptions . . . 28
3.17 Maps . . . 29
3.18 Objects . . . 29

4 Evaluation **29**
4.1 Let construct . . . 29
4.2 Simple accessors . . . 29
4.3 Representation of terms . . . 30
4.4 Tree equality . . . 31
4.5 Arrays . . . 32
4.6 Evaluator . . . 33
4.7 Debugging aids . . . 33
4.8 Verification . . . 34
4.9 Idiosyncrasies . . . 35

5 Compilation **37**
5.1 Fixed point combinator . . . 37
5.2 Compiler . . . 37
5.3 Code constructors . . . 38
5.4 Compilation of individual constructs . . . 39
5.5 Claim evaluation . . . 40

6 Macro expansion **41**
6.1 Macro expansion engine . . . 41
6.2 Backquoting . . . 41
6.3 Elementary macros . . . 42
6.4 Definition macros . . . 43
6.5 Parentheses . . . 45
6.6 Numerals . . . 45
6.7 Formating of the expansion . . . 46
6.8 Visibility constructs . . . 47
6.9 Tuples . . . 47
6.10 Eager definitions . . . 48
6.11 Late definitions . . . 48
6.12 Let constructs . . . 48
6.13 Backquoting . . . 50
6.14 Rendering . . . 51

7 The Logiweb machine **52**
7.1 Messages . . . 52
7.2 Machine invocation . . . 53
7.3 Input-eval-output-loop . . . 53
7.4 Handlers and processes . . . 54
7.5 Character constants . . . 54
7.6 Hello World . . . 55
7.7 Echo . . . 55

7.8 Eecho . . . 55
7.9 Execution constructs . . . 56

8 The lgcio interface **56**
8.1 The lgcio request . . . 57
8.2 Constants used by the lgcio interface . . . 58
8.3 Data conversion . . . 59
8.4 Individual lgcio requests . . . 60
8.5 The lgcio interface definition . . . 62
8.6 Test of the lgcio interface . . . 81

9 Compiler support functions **84**
9.1 General functions . . . 84
9.2 Conversion to mixed endian hexadecimal . . . 84
9.3 Rack conversion . . . 84
9.4 Rack cleaning and restoring . . . 86

10 Ripemd160 **88**
10.1 Introduction . . . 88
10.2 Elementary 32 bit functions . . . 89
10.3 Bit combination functions . . . 89
10.4 Constants . . . 89
10.5 Macros used in rounds . . . 90
10.6 Initial quintuple . . . 91
10.7 Compress function . . . 91
10.8 Conversion of buffer to array . . . 95
10.9 Conversion of buffer to Ripemd-160 code . . . 95

1 Introduction

Logiweb [2] is an open source system available under GNU GPL for distribution of mathematical definitions, lemmas, and proofs.

The present document is part of a *Logiweb page*. A Logiweb page is a cluster of documents which has been submitted to the Logiweb system. Such a cluster typically consists of a main document like the present one plus a number of electronic appendices.

The present Logiweb page (i.e. the document cluster to which the present document belongs) is a Logiweb *base* page. A Logiweb base page is a Logiweb page which references no other Logiweb pages. Logiweb base pages are self-contained pages which define elementary concepts.

1.1 Electronic appendices

The present Logiweb page comprises one html and two PDF files, located the following places:

- ../../../logiweb/01AB1F51C8C17606A5C0331B5689B4858C796547B9A0A4AEF0BCB2BB0806/page/page.pdf
- ../../../logiweb/01AB1F51C8C17606A5C0331B5689B4858C796547B9A0A4AEF0BCB2BB0806/page/appendix.pdf
- ../../../logiweb/01AB1F51C8C17606A5C0331B5689B4858C796547B9A0A4AEF0BCB2BB0806/page/page.pdf [3]
- ../../../logiweb/01AB1F51C8C17606A5C0331B5689B4858C796547B9A0A4AEF0BCB2BB0806/page/page.html

The first link points to the present paper. The second link points to an electronic appendix with definitions that would bore most readers (such as definitions of how each construct should be rendered). The second link is included in the BibTeX bibliography [3] for easy reference. The third link points to a table of contents.

1.2 Proclamations

Logiweb has a number of predefined semantic concepts. Each predefined concept has a name where a name is a string of characters.

Furthermore, Logiweb has a *proclamation* mechanism which allows to connect syntactic constructs with semantic concepts. Some proclamations are given in the following.

[$\lambda x.y \bowtie$ 'lambda'] connects the syntactic construct $\lambda x.y$ with the semantic concept whose name reads "lambda". This proclamation tells Logiweb that $\lambda x.y$ denotes *lambda abstraction* in the present context. The proclamation has effect on the present Logiweb page and all Logiweb pages which reference the present page.

[x ' $y \bowtie$ 'apply'] proclaims x ' y to denote *functional application*.

[$\mathsf{T} \bowtie$ 'true'] proclaims T to denote *truth*.

[**If** x **then** y **else** $z \bowtie$ 'if'] proclaims **If** x **then** y **else** z to denote an *if-then-else construct*. This and the previous 3 constructs are explained in more detail in Section 2.

[$\lceil x \rceil \bowtie$ 'quote'] proclaims $\lceil x \rceil$ to denote a data structure which represents the term x. This is explained in more detail in Section 4.3.

[$[x \bowtie y] \bowtie$ 'proclaim'] proclaims $[x \bowtie y]$ to denote *proclamation*. Each construct in Logiweb is identified by a reference and an index, both of which are integers. The reference identifies the Logiweb page which introduces the construct. If a page is a 'base' page in the sense that it references no other pages, then the construct with index 1 of that page is born as a proclamation construct. Such a proclamation construct can

declare other constructs to be proclamation constructs. But it also has to secure itself by declaring itself as a proclamation construct as it otherwise looses its proclamation ability. The present page happens to be a base page and the construct above happens to be construct number 1 of the present page.

$[[y \xrightarrow{x} z] \bowtie$ 'define'] says that $[y \xrightarrow{x} z]$ *defines* the x *aspect* of y to be z. As an example of use, $[x! \xrightarrow{\text{val}} \textbf{if } x = 0 \textbf{ then } 1 \textbf{ else } x \cdot (x-1)!]$ defines the *value aspect* of $x!$ such that $[3! = 6]^{\cdot}$.

$[[y \xrightarrow{x!} z] \bowtie$ 'define'] says that $[y \xrightarrow{x!} z]$ means exactly the same as $[y \xrightarrow{x} z]$. The difference is in how the right hand side of the definition is rendered. $[y \xrightarrow{x} z]$ renders z using the *tex use aspect* of z whereas $[y \xrightarrow{x!} z]$ uses the *tex show aspect*. For more on the tex use and tex show aspects see Section 1.3 and see the newline construct at the end of Section 2.5.

[[def x of y as z] $\bowtie$ 'define'] defines [def x of y as z] as an alternative for $[y \xrightarrow{x!} z]$. [def x of y as z] is used by the lgc compiler for translating the '""N' and '""C' directives into name and charge definitions for all constructs on a page.

$[[y \xrightarrow{x}\!\!\rightarrow z] \bowtie$ 'introduce'] says that $[y \xrightarrow{x}\!\!\rightarrow z]$ means exactly the same as $[y \xrightarrow{x!} z]$, except that $[y \xrightarrow{x}\!\!\rightarrow z]$ may affect CPU run time dramatically in a few, vital cases c.f. Section 2.5.

[hide(x) $\bowtie$ 'hide'] proclaims that x in hide(x) should be hidden from *harvesting*. Harvesting is the process in which proclamations, definitions, and other Logiweb *revelations* are collected from a page. Hiding may be useful e.g. when talking about revelation constructs without wanting them to take effect. For more on hiding see Section 6.8.

Logiweb has predefined concepts with the names used above (apply, lambda, true, if, quote, proclaim, define, introduce, hide, pre, and post). Logiweb has no other predefined concepts than those. Attempts to proclaim a construct to denote some unknown concept are ignored. The *Logiweb compiler* (a tool for constructing Logiweb pages) issues a warning in case of unrecognized proclamations.

1.3 Aspect declarations

On Logiweb, constructs that assign meaning to constructs always assign meaning to a particular *aspect* of the construct. Some aspects are defined by the following *aspect declarations*.

[msg $\overset{\text{`message'}!}{\rightarrow}$ 'message'] defines msg to represent the *message aspect*. The message aspect of a construct indicates which aspect the construct represents. For an example of use, see the next entry.

Section 1.2 proclaimed two definitions constructs, one with and one without an exclamation mark. The one with the exclamation mark renders its arguments using the tex show aspect whereas the one without renders its aspect and right hand side using the tex use aspect. The tex show aspect of a string comprises the string with quotes around whereas the tex use aspect of a string is the string without quotes. The exclamation mark version of the definition construct is used here to make it easy distinguish strings from non-strings.

[val $\overset{\text{msg!}}{\rightarrow}$ 'value'] defines val to represent the *value aspect*. As an example, the definition [$x! \overset{\text{val!}}{\rightarrow}$ **if** $x = 0$ **then** 1 **else** $x \cdot (x-1)!$] from Section 1.2 used val to represent the value aspect. One may replace msg and val by the strings they denote as in [val $\overset{\text{`message'}!}{\rightarrow}$ 'value'] and [$x! \overset{\text{`value'}!}{\rightarrow}$ **if** $x = 0$ **then** 1 **else** $x \cdot (x-1)!$].

[name $\overset{\text{msg!}}{\rightarrow}$ 'name'] defines name to represent the *logiweb name aspect*. The name aspect of a construct is a plain text representation for the construct. Such a plain text representation is useful when editing Logiweb pages using an ordinary text editor. As an example, the name aspects of $\lambda x.y$, u, and v are `\".`, `vu`, and `vv`, respectively. For that reason, one may type `\ u . vv` in a text editor, run the text through a Logiweb compiler, and get a Logiweb page containing $\lambda u.v$. For examples of name definitions see Section "Name definitions" in [3].

[lgcname $\overset{\text{msg!}}{\rightarrow}$ 'name'] defines name the same way as name. name is used by the lgc compiler for translating the '""N' escape sequence to name definitions for all constructs on a page.

[lgccharge $\overset{\text{msg!}}{\rightarrow}$ 'charge'] defines charge to represent the *logiweb charge aspect*. charge is used by the lgc compiler for translating the '""C' escape sequence to charge definitions for all constructs on a page. The charge aspect of a construct is a dotted list of integers represented as a string. Charges define the precedence of operators such that constructs with high charges have low priority and vice versa. As an example, $x + y$ has greater charge than $x \cdot y$ so $a \cdot b + c \cdot d$ means $(a \cdot b) + (c \cdot d)$.

[use $\overset{\text{msg!}}{\rightarrow}$ 'use'] defines use to represent the *tex use aspect*. The tex use aspect of a construct indicates how the aspect should be rendered in TEX when using the construct. For examples of tex use definitions see Section "TEX definitions" in [3].

[show $\overset{\text{msg!}}{\rightarrow}$ 'show'] defines show to represent the *tex show aspect*. The tex show aspect of a construct indicates how the aspect should be rendered in TEX when talking about the construct. Most constructs look the same when using them and when talking about them. For that reason, the tex show aspect defaults to the tex use aspect. As an example, $\lambda x.y$ has no tex show aspect so $\lambda x.y$ looks the same when using lambda abstraction and when talking about lambda abstraction. Some constructs look different when using them and when talking about them, c.f. Section 6.8. For examples of tex show definitions see Section "TEX definitions" in [3]. For more on the use of tex use and tex show aspects, see the newline construct at the end of Section 2.5.

[prio $\overset{\text{msg!}}{\rightarrow}$ 'priority'] defines prio to represent the *priority aspect*. Before Logiweb Version 0.2.0, the priority aspect of a Logiweb page indicated the priority and associativity of all constructs on the page itself plus all constructs on directly referenced Logiweb pages. From Logiweb Version 0.2.0, the charge aspect is used in place of the priority aspect. However, a Logiweb Version 0.1.x compiler is required for compiling the Logiweb Version 0.2.0 compiler, and 0.1.x compilers need the priority aspect. For that reason, the priority aspect is kept until further.

The statement above that the priority aspect of a Logiweb page indicates the priorities is not completely true. Aspects attach to constructs, not to pages. However, each Logiweb page has a *page construct* which represents the page (c.f. Section 4.3). The page construct of the present page reads "base" and the priority table in [3] is indeed a definition of the priority aspect of this "base" construct.

[macro $\overset{\text{msg!}}{\rightarrow}$ 'macro'] defines macro to represent the *macro aspect*. The macro aspect defines how a Logiweb page is macro expanded, c.f. Section 6.

[render $\overset{\text{msg!}}{\rightarrow}$ 'render'] defines render to represent the *render aspect*. Logiweb has a predefined way of rendering Logiweb pages, but rendering can be modified using the render aspect.

[claim $\overset{\text{msg!}}{\rightarrow}$ 'claim'] defines claim to represent the *claim aspect*. The claim aspect defines how a Logiweb page is verified, c.f. Section 4.8.

[unpack $\overset{\text{msg!}}{\rightarrow}$ 'unpack'] defines unpack to represent the *unpack aspect*. When stored on disk or transmitted over a network, Logiweb pages are stored as *Logiweb vectors*, i.e. as sequences of bytes. Logiweb has a predefined way of unpacking Logiweb vectors into Logiweb pages, but unpacking can be modified using the unpack aspect.

[execute $\overset{\text{msg!}}{\rightarrow}$ 'execute'] defines execute to represent the *execute aspect*. If the execute aspect of a string s is defined then the Logiweb compiler generates an executable named s whose behavior is defined by the right hand side of the execute definition.

[exampleaspect0 $\overset{\text{msg!}}{\rightarrow}$ 'example aspect/kg'] defines exampleaspect0 to represent an aspect named 'example aspect/kg'. The exampleaspect0 aspect is a *user aspect* in the sense that Logiweb does not know about it in advance. To avoid name conflicts, one should give any user aspect a personal touch. The exampleaspect0 aspect above is represented by a string which includes the initials of the present author. In general, user aspects should contain at least one character which is not a small latin letter (a to z) and not a space. The example aspect above is ok since it contains a slash. Aspects which contain only small latin letters and spaces are reserved for future extensions of Logiweb. The only exception is the 'destructure' aspect defined later which consists of only small latin letters for historical reasons.

[exampleaspect1 $\overset{\text{msg!}}{\rightarrow}$ exampleaspect2] defines exampleaspect1 to represent a user aspect represented by the construct exampleaspect2. exampleaspec is a construct introduced on the present page. Like any other Logiweb construct, it is identified by a *reference* and an *index*. The reference is a world-wide unique natural number which identifies the present page. The index is a natural number which distinguishes the exampleaspect2 construct from the other constructs introduced on the present page. The definition of exampleaspect1 above defines exampleaspect1 to denote an aspect which is represented by the reference and identifier of exampleaspect2. Contrary to the definition of the exampleaspect0 aspects above, there is no risk of name conflicts. On the other hand, the exampleaspect2 ceases to exist if the present page is ever deleted from Logiweb.

[exampleaspect2 $\overset{\text{msg!}}{\rightarrow}$ exampleaspect2] defines exampleaspect2 to represent the user aspect represented by exampleaspect2 itself. Hence, exampleaspect and exampleaspect2 denote the same aspect.

Logiweb assigns a particular semantics to the value, name, use, show, priority, macro, claim, message, unpack, and execute aspects. Logiweb does not assign semantics to any other aspects, but it allows users to declare arbitrary user aspects like the example aspects above. It is up to the authors of Logiweb pages to assign semantics to user aspects. As an example, the present page declares a 'destructure' user aspect in Section 6.12 and defines a destructuring 'let' macro such that 'let' uses the 'destructure' aspect. Apart from the destructure aspect, any user defined aspect should contain at least one character which is neither a small latin letter (a to z) nor a space.

1.4 Definition of pre- and post-associativity

[**Preassociative** $x; y \overset{\text{prio!}}{\rightarrow}$ 'pre'] defines **Preassociative** $x; y$ to denote *preassociativity.* A preassociative construct is leftassociative in text that runs left to right, rightassociative in text that runs right to left, topassociative in text that runs from top to bottom, counter-clockwise-associative in text written in clock-wise spirals, and so on. **Preassociative** $x; y$ states that all constructs listed in the x position are preassociative and that all of them have greater priority than those listed in the y position. For a table of associativites and priorities see the chapter named "Priority table" in [3].

[**Postassociative** $x; y \overset{\text{prio!}}{\rightarrow}$ 'post'] defines **Postassociative** $x; y$ to denote *postassociativity.*

1.5 The Logiweb system

From a standardization point of view, the Logiweb system comprises two standards, one which defines the syntax and semantics of Logiweb pages and one which defines a protocol for exchanging information about Logiweb pages.

From an implementation point of view, the current implementation of the Logiweb system comprises two programs called the *Logiweb compiler* and the *Logiweb server*, respectively, plus a number of auxilliary files and programs.

From a user point of view, Logiweb comprises a number of Logiweb pages which the user can browse using an ordinary web browser or search using an ordinary web search engine.

Users who merely want to read existing Logiweb pages can do with an ordinary web browser. Users will need the Logiweb compiler in the following cases:

1. when authoring and submitting new Logiweb pages.

2. when mirroring Logiweb pages, i.e. when a user wants to make a particular Logiweb page available on the users own web site.

3. when executing computable functions on Logiweb pages (c.f. the "execute" aspect in Section 1.3).

In general, users will not be aware of Logiweb servers. All Logiweb servers in the world cooperate on indexing all Logiweb pages in the world, which is needed in connection with the internal referencing system of Logiweb. When searching for Logiweb pages, users should use an ordinary web search engine.

Even though Logiweb has several components and even though one can view Logiweb from several points of view, the only piece of Logiweb software users will typically meet is the Logiweb compiler. Typically, when the present paper states that Logiweb is able to perform various tasks, those tasks are actually undertaken by the Logiweb compiler.

Note that the Logiweb compiler is just one, possible implementation of Logiweb. A wysiwyg Logiweb browser/editor was once implemented but is not currently maintained, but one may foresee other tools than the Logiweb compiler for viewing, authoring, mirroring, and executing Logiweb pages.

2 Computation

Logiweb (i.e. the Logiweb compiler, c.f. Section 1.5) is able to *reduce* computable terms to *normal form*. It does so using the *Logiweb computing engine*.

The Logiweb engine is just a computing engine; it has no facilities for communicating with the outside world. The Logiweb machine (c.f. Section 7) contains the Logiweb engine and also contains facilities for input/output.

2.1 Reduction system

The engine reduces terms according to the following reduction system:

$$
\begin{array}{lcl}
(\lambda x.y)\,'\,z & \xrightarrow{+} & \langle y\,|x{:=}\,z\rangle \\
\mathsf{T}\,'\,z & \xrightarrow{+} & \mathsf{T} \\
\textbf{If } \mathsf{T} \textbf{ then } u \textbf{ else } v & \xrightarrow{+} & u \\
\textbf{If } \lambda x.y \textbf{ then } u \textbf{ else } v & \xrightarrow{+} & v
\end{array}
$$

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

2.2 Normal forms

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 $\lambda x.y$ 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 leftmost reduction. The Logiweb engine stops again when the term reaches root normal form, if ever.

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

2.3 Maps

The reduction system above is pure lambda calculus extended with an uhr-element T and an if-then-else construct which can test for equality to T. It is out of the scope of the present page to explain why that extension is vital.

It is possible to reason about lambda calculus extended with T and **If** x **then** y **else** z using *map theory* [1]. Using the terminology of [1] we shall refer to values of terms built from $\lambda x.y$, x ' y, T, and **If** x **then** y **else** z as *maps*.

Map theory defines a notion of equality named $x \equiv y$. We shall use $x = y$ to denote that x and y are equal modulo identifications whereas we use $x \equiv y$ to denote genuine equality.

In other words, we use a sign with two horizontal bars, $x = y$ to denote that x and y are very much equal and one with 3 horizontal bars, $x \equiv y$ to denote that x and y are even more equal.

The use of $x = y$ to denote an equivalence relation (equality modulo identifications) and $x \equiv y$ to denote equality is apparently opposite to normal mathematical use. In practice, the use of $x = y$ to denote equality modulo identification is very much in line with mathematical litterature. As an example, if one identifies the bidual of a Hilbert space with the Hilbert space itself, then it is common practice to use $x = y$ between members of the bidual and members of the Hilbert space.

Anyone who can come up with a better name than $x \equiv y$ for true, identification disrespecting equality is welcome to contact the author.

Note that $x \equiv y$ has much lower priority than $x = y$ according to the priority table in [3]. As an example, $\lambda x.x = x \equiv y$ means $(\lambda x.(x = x)) \equiv y$.

The reduction rules of Section 2.1 correspond to the following axioms of map theory:

$$
\begin{array}{lcl}
(\lambda x.y)\text{ '}\, z & \equiv & \langle y \,|x{:=}\, z\rangle \\
\mathsf{T}\text{ '}\, z & \equiv & \mathsf{T} \\
\textbf{If } \mathsf{T} \textbf{ then } u \textbf{ else } v & \equiv & u \\
\textbf{If } \lambda x.y \textbf{ then } u \textbf{ else } v & \equiv & v
\end{array}
$$

2.4 Full reduction system

The reduction system in Section 2.1 gives a good description of the Logiweb engine from a theoretical point of view, but a more complete description is given in the following:

- The engine reduces constructs proclaimed to denote lambda abstraction, functional application, truth, and if-then-else as indicated in Section 2.1. Section 1.2 proclaimed $\lambda x.y$, x ' y, T, and **If** x **then** y **else** z to denote these concepts, respectively.

- The engine reduces constructs proclaimed to denote quoting to the value described in Section 4.3. Section 1.2 proclaimed $\lceil x \rceil$ to denote quoting. $\lceil x \rceil$ reduces to a term built up from $\lambda x.y$, x ' y, T, and **If** x **then** y **else** z. The value of $\lceil x \rceil$ represents the term x.

- The engine reduces constructs that have a value definition according to that definition, c.f. Section 2.5.

- The engine reduces page constructs to terms which (in principle) are built up from $\lambda x.y$, x ' y, T, and **If** x **then** y **else** z. The value of a page construct is called the *cache* of the given page. The cache of a page is a huge data structure which represents a lot of information about the given page and all its transitively referenced pages. Among other, the cache contains the page before and after macro expansion, a structure which contains all definitions harvested from the page, and a structure which contains compiled versions of all value definitions.

- The engine reduces all other constructs (non-page constructs that are neither proclaimed nor defined) to T. This rather arbitrary convention is made to make the behaviour of the engine well-defined in all cases.

- The engine would be rediculously slow if it wasn't for the optimizations explained in Section 2.5.

2.5 Elementary definitions

Consider the following definitions:

[x LazyPair $y \overset{\text{val}}{\to} \lambda z.$**If** z **then** x **else** y]. The engine reduces x LazyPair y to the function normal form $\lambda z.$**If** z **then** x **else** y. As we shall see, it is possible to extract x and y from the return value, so the return value is a *pairing* construct.

[$\mathsf{F} \overset{\text{val}}{\to\to} \mathsf{T}$ LazyPair T]. The engine reduces F to the function normal form $\lambda x.$**If** x **then** T **else** T. We shall use F to represent *falsehood* as opposed to T which we use to represent truth.

[x Head $\overset{\text{val}}{\to} x$ ' T]. We have

$$\begin{aligned}
&(u \text{ LazyPair } v) \text{ Head} \equiv \\
&(\lambda x.\textbf{If } x \textbf{ then } u \textbf{ else } v) \text{ Head} \equiv \\
&(\lambda x.\textbf{If } x \textbf{ then } u \textbf{ else } v) \text{ '} \mathsf{T} \equiv \\
&\textbf{If } \mathsf{T} \textbf{ then } u \textbf{ else } v \equiv \\
&u
\end{aligned}$$

Hence, $(u$ LazyPair $v)$ Head $\equiv u$ whenever u and v differ from $\perp$.

[x Tail $\overset{\text{val}}{\to} x$ ' F]. Like above, we have that $(u$ LazyPair $v)$ Tail $\equiv v$ whenever u and v differ from $\perp$. This supports the claim above that x LazyPair y is a pairing construct.

[x Guard $y \overset{\text{val}}{\to}$ **If** x **then** y **else** y]. The engine reduces x Guard y by reducing x, discarding the value, reducing y, and returning the value of y. Hence x Guard y has the following properties:

- T Guard $y \equiv y$

- $\lambda u.v$ Guard $y \equiv y$
- $\perp$ Guard $y \equiv \perp$.

$\perp$ above is defined in Section 2.8. $\perp$ is a bottom term, i.e. a term which the engine reduces forever without reaching a root normal form (c.f. Section 2.2).

$[x$ Pair $y \overset{\text{val}}{\rightarrow} x$ Guard y Guard x LazyPair $y]$. The engine reduces x Pair y by reducing x, discarding the value, reducing y, discarding the value, and then returning the function normal form $\lambda z.$**If** z **then** x **else** y. x Pair y is *eager* or *strict* in the sense that

- $\perp$ Pair $y \equiv \perp$
- x Pair $\perp \equiv \perp$

protect$([(x) \overset{\text{val}}{\longrightarrow} x])$. This definition says that a parenthesis does not affect the value of its contents. Section 6.5 macro defines parentheses such that they disappear during macro expansion which is another way of saying that a parenthesis does not affect the value of its contents. The purpose of the value definition above is to ensure that $(x) \equiv x$ even in situations where (x) is not macro expanded. The definition above is itself protected against macro expansion with the protect(x) construct which is defined in Section 6.3. Without that protection, the left hand side of the definition above would be macro expanded from (x) to x so that the definition would define x to be equal to itself. Due to the minimal semantics convention, that would define x to be equal to $\perp$.

protect$([$newline $x \overset{\text{val}}{\longrightarrow} x])$. This construct leaves its argument unaffected just like parentheses do; and it is macro defined in Section 6.5 just like parentheses are. But the newline construct is rendered differently. When talking about the construct, it is rendered as above. But when using it as in the formula $2 +$
3 it affects line breaking. In the formula, the newline construct appears right after the plus sign. This effect is acheived by definitions in [3]. In [3], the 'tex show' definition of the newline construct defines how the construct should be rendered when it occurs in the left hand side of a definition or some other context where the construct is talked about. Furthermore, the 'tex use' definition of the newline construct in [3] defines how the newline construct should be rendered when it is used rather than talked about. Finally, in [3], the 'tex use' definition of the definition construct specifies that the left hand side of a definition should be rendered using the 'tex show' definition rather than the 'tex use' definition.

The definition of F above is an example of an "introduction". Formally, an introduction has exactly the same meaning as a definition. However, each

implementation of Logiweb is supposed so contain a finite list of functions which have been hand-coded for efficiency. The nulary function F is one such function.

For each hand-coded function, the implementation must contain a definition for an equivalent function expressed solely using the reduction system in 2.1 plus quoting. When the implementation sees an introduction, it runs through the list of equivalent functions to see if the right hand side matches one of them. If a match is found then the introduced construct is translated into the associated hand-coded function. If no match is found then the introduction is treated as an ordinary definition. As long as implementors are careful to ensure equivalence between hand-coded functions and equivalent functions, this ensures portability of code between different implementations of Logiweb.

Matching of right hand sides against equivalent functions is quite strict: One may change names of variables and names of functions, but apart from that, an exact match is required. It is decidable (and fast to decide) whether two functions match.

2.6 Booleans

$$[x \left\langle \begin{matrix} y \\ z \end{matrix} \right. \bowtie \text{‘if’}]$$

$$[\text{Not } x \xrightarrow{\text{val}} \textbf{If } x \textbf{ then } \mathsf{F} \textbf{ else } \mathsf{T}]$$

$$[x \text{ And } y \xrightarrow{\text{val}} x \left\langle \begin{matrix} \textbf{If } y \textbf{ then } \mathsf{T} \textbf{ else } \mathsf{F} \\ \textbf{If } y \textbf{ then } \mathsf{F} \textbf{ else } \mathsf{F} \end{matrix} \right.]$$

$$[x \text{ Or } y \xrightarrow{\text{val}} x \left\langle \begin{matrix} \textbf{If } y \textbf{ then } \mathsf{T} \textbf{ else } \mathsf{T} \\ \textbf{If } y \textbf{ then } \mathsf{T} \textbf{ else } \mathsf{F} \end{matrix} \right.]$$

$$[x \text{ Iff } y \xrightarrow{\text{val}} x \left\langle \begin{matrix} \textbf{If } y \textbf{ then } \mathsf{T} \textbf{ else } \mathsf{F} \\ \textbf{If } y \textbf{ then } \mathsf{F} \textbf{ else } \mathsf{T} \end{matrix} \right.]$$

$$[x \text{ TheBool} \xrightarrow{\text{val}} \textbf{If } x \textbf{ then } \mathsf{T} \textbf{ else } \mathsf{F}]$$

$$\begin{aligned}[x \text{ Equal } y \xrightarrow{\text{val}} & \textbf{If } x \textbf{ then} \\ & \textbf{If } y \textbf{ then } \mathsf{T} \textbf{ else } \mathsf{F} \textbf{ else} \\ & \textbf{If } y \textbf{ then } \mathsf{F} \textbf{ else } x \text{ Head Equal } y \text{ Head And } x \text{ Tail Equal } y \text{ Tail}]\end{aligned}$$

2.7 Naturals

We shall refer to the *natural numbers* 0, 1, 2, and so on as *naturals*. We shall say that a list b_0 Pair b_1 Pair $\ldots$ Pair b_n Pair T is an *untagged representation* of the natural

$$\sum_{\mathrm{i}=0}^{\mathrm{n}} 2^{\mathrm{i}} \mathrm{c}_{\mathrm{i}}$$

where c_i is zero or one when b_i is true or false, respectively. When a list is used as an untagged representation of a natural, we shall refer to the list as an *untagged natural*. The following functions apply to untagged naturals.

[Zero $\stackrel{\text{val}}{\rightarrow}$ T]

[x NatPair y $\stackrel{\text{val}}{\rightarrow}$ **If** x And y **then** Zero **else** x TheBool Pair y]

[x TheNat $\stackrel{\text{val}}{\rightarrow}$ **If** x **then** Zero **else** x Head NatPair x Tail TheNat]

[One $\stackrel{\text{val}}{\rightarrow}$ F Pair Zero]

[Two $\stackrel{\text{val}}{\rightarrow}$ T Pair One]

[Three $\stackrel{\text{val}}{\rightarrow}$ F Pair One]

[Four $\stackrel{\text{val}}{\rightarrow}$ T Pair Two]

[Five $\stackrel{\text{val}}{\rightarrow}$ F Pair Two]

[Six $\stackrel{\text{val}}{\rightarrow}$ T Pair Three]

[Seven $\stackrel{\text{val}}{\rightarrow}$ F Pair Three]

[Eight $\stackrel{\text{val}}{\rightarrow}$ T Pair Four]

[Nine $\stackrel{\text{val}}{\rightarrow}$ F Pair Four]

[Ten $\stackrel{\text{val}}{\rightarrow}$ T Pair Five]

[Xor (x , y , c) $\stackrel{\text{val}}{\rightarrow}$ x Head Iff y Head Iff c]

[Carry (x , y , c) $\stackrel{\text{val}}{\rightarrow}$ **If** x Head **then** y Head Or c **else** y Head And c]

[Plus (x , y , c) $\stackrel{\text{val}}{\rightarrow}$ **If** x And y And c **then** Zero **else** Xor (x , y , c) NatPair Plus (x Tail , y Tail , Carry (x , y , c))]

[x Plus y $\stackrel{\text{val}}{\rightarrow}$ Plus (x , y , T)]

[Borrow (x , y , b) $\stackrel{\text{val}}{\rightarrow}$ **If** x Head **then** y Head And b **else** y Head Or b]

[Compare (x , y , b) $\stackrel{\text{val}}{\rightarrow}$ **If** x And y **then** b **else** Compare (x Tail , y Tail , Borrow (x , y , b))]

[x LT y $\stackrel{\text{val}}{\rightarrow}$ Compare (y , x , F)]

[Minus (x , y , b) $\stackrel{\text{val}}{\rightarrow}$ **If** x And y **then** Zero **else** Xor (x , y , b) NatPair Minus (x Tail , y Tail , Borrow (x , y , b))]

$[x \text{ Minus } y \overset{\text{val}}{\to} \text{Minus } (\ x \ , \ y \ , \ \mathsf{T} \ )]$

$[x \text{ Times } y \overset{\text{val}}{\to} \textbf{If } x \textbf{ then } \text{Zero } \textbf{else } (\textbf{If } x \text{ Head } \textbf{then } \text{Zero } \textbf{else } y) \text{ Plus } (\mathsf{T} \text{ Pair } x \text{ Tail Times } y)]$

2.8 Bottom

$[\bot \overset{\text{val}}{\to\to} (\lambda x.x \text{ ' } x) \text{ ' } (\lambda x.x \text{ ' } x)]$

Optimized bottom is a peculiar optimized function. Evaluation of the right hand side above takes forever. So the semantics of $\bot$ is that it does not return a value. Optimized bottom is faithful to that: it never returns a value. What it does do is that it prints and error message and kills its caller. So the user can sit comfortably one step further away and watch the caller being killed.

One could suggest some "bessermachen" and suggest some sort of catcher which allows to test whether or not some computation results in a call to optimized bottom. That, however, will give grave portability problems and grave problems for reasoning about programs. What you need if you think of this "bessermachen" are the exceptions described in Section 3.

It should be noted that not only users can sit comfortably one step away and see a caller being killed by $\bot$. Also computers can do that. In the Logiweb machine described in Section 7, a handler could use its interface to the outside world to look into the machine itself and inspect the fate of a subordinate process.

3 Tagged values

3.1 Tags

Tagged values have form t LazyPair v where t is a *tag* which identifies the type of the value and v represents the value itself.

Tags have form r Pair i where r and i are untagged naturals. We shall refer to r and i as the *reference* and *index* of the tag, respectively.

The reference of a tag is supposed to be either zero or to be the reference of a Logiweb page. The former case is used for the *predefined* types listed later. Individual users who want to introduce their own types should use tags of form r Pair i where r is the reference of a Logiweb page they have published.

The predefined types are: *Booleans*, *integers*, *pairs*, *exceptions*, and *maps*. Maps are just arbitrary values with a tag on.

The tags of the predefined types are:

$[\text{BoolTag} \overset{\text{val}}{\to} \text{Zero Pair Zero}]$

$[\text{IntTag} \overset{\text{val}}{\to} \text{Zero Pair One}]$

[PairTag $\overset{\text{val}}{\to}$ Zero Pair Two]

[ExTag $\overset{\text{val}}{\to}$ Zero Pair Three]

[MapTag $\overset{\text{val}}{\to}$ Zero Pair Four]

3.2 Tag querying

[x Tag $\overset{\text{val}}{\to}$ x Head Head TheNat Pair x Head Tail TheNat]

[x BoolP $\overset{\text{val}}{\to}$ x Tag Equal BoolTag]

[x IntP $\overset{\text{val}}{\to}$ x Tag Equal IntTag]

[x PairP $\overset{\text{val}}{\to}$ x Tag Equal PairTag]

[x ExP $\overset{\text{val}}{\to}$ x Tag Equal ExTag]

[x MapP $\overset{\text{val}}{\to}$ x Tag Equal MapTag]

[x ObjectP $\overset{\text{val}}{\to}$ Not (x BoolP Or x IntP Or x PairP Or x ExP Or x MapP)]

3.3 Guards

[$x{\in}\mathbf{V}\colon y \overset{\text{val}}{\to\to}$ **If** x^{N} ExP **then** x **else** y]

[x .then. $y \overset{\text{val}}{\to\to}$ **If** x^{N} ExP **then** x **else** y]

[$x{\in}\mathbf{B}\colon y \overset{\text{val}}{\to\to} x{\in}\mathbf{V}\colon y{\in}\mathbf{V}\colon$ **If** x^{N} BoolP **then** y **else** $\bullet$]

[$x{\in}\mathbf{Z}\colon y \overset{\text{val}}{\to\to} x{\in}\mathbf{V}\colon y{\in}\mathbf{V}\colon$ **If** x^{N} IntP **then** y **else** $\bullet$]

[$x{\in}\mathbf{P}\colon y \overset{\text{val}}{\to\to} x{\in}\mathbf{V}\colon y{\in}\mathbf{V}\colon$ **If** x^{N} PairP **then** y **else** $\bullet$]

[$x{\in}\mathbf{M}\colon y \overset{\text{val}}{\to\to} x{\in}\mathbf{V}\colon y{\in}\mathbf{V}\colon$ **If** x^{N} MapP **then** y **else** $\bullet$]

[$x{\in}\mathbf{O}\colon y \overset{\text{val}}{\to\to} x{\in}\mathbf{V}\colon y{\in}\mathbf{V}\colon$ **If** x^{N} ObjectP **then** y **else** $\bullet$]

3.4 Normalization

[$x^{\text{N}} \overset{\text{val}}{\to\to}$

If x BoolP **then** x TheBool **else**
If x IntP **then** int (x Tail) **else**
If x PairP **then** x Tail Head : : x Tail Tail **else**
If x ExP **then** x Tail$^{\bullet}$ **else**
If x MapP **then** map (x Tail) **else**
x Tag Pair x Tail^{N}]

[norm $x \overset{\text{val}}{\to\to} x^{\text{N}}$]

3.5 Booleans

$[x \in \mathbf{B} \xrightarrow{\text{val}} x{\in}\mathbf{V}\colon x \text{ BoolP}]$

$[\textbf{if } x \textbf{ then } y \textbf{ else } z \xrightarrow{\text{val}} x{\in}\mathbf{V}\colon \textbf{If } \text{norm } x \textbf{ then } y \textbf{ else } z]$

$[\textbf{not } x \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon \textbf{if } x \textbf{ then } \mathsf{F} \textbf{ else } \mathsf{T}]$

$[\text{notnot } x \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon \textbf{if } x \textbf{ then } \mathsf{T} \textbf{ else } \mathsf{F}]$

$[x \textbf{ and } y \xrightarrow{\text{val}} \textbf{If } x^{\text{N}} \textbf{ then } y^{\text{N}} \textbf{ else } x^{\text{N}}]$

$[x \textbf{ or } y \xrightarrow{\text{val}} \textbf{if } x^{\text{N}} \textbf{ then } \mathsf{T} \textbf{ else } y^{\text{N}}]$

$[x = y \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon y{\in}\mathbf{V}\colon \text{equal1 } (\ x^{\text{N}}\ ,\ y^{\text{N}}\ )]$

$$\begin{aligned}[\text{equal1 } (\ x\ ,\ y\ ) \xrightarrow{\text{val}}\ & \\ & \textbf{if } x \in \mathbf{B} \textbf{ then } x \text{ Equal } y \textbf{ else} \\ & \textbf{if } x \in \mathbf{Z} \textbf{ then } x \text{ Equal } y \textbf{ else} \\ & \textbf{if } x \in \mathbf{P} \textbf{ then } y \in \mathbf{P} \textbf{ and } x^{\text{h}} = y^{\text{h}} \textbf{ and } x^{\text{t}} = y^{\text{t}} \textbf{ else} \\ & \textbf{if } x \in \mathbf{M} \textbf{ then } y \in \mathbf{M} \textbf{ else} \\ & x \text{ Tag Equal } y \text{ Tag } \textbf{and } x \text{ Tail} = y \text{ Tail}]\end{aligned}$$

$[x \neq y \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon y{\in}\mathbf{V}\colon \textbf{not } x = y]$

$[x \in \mathbf{T} \xrightarrow{\text{val}} x{\in}\mathbf{V}\colon x \in \mathbf{B} \textbf{ and } \text{notnot } x]$

$[x \in \mathbf{F} \xrightarrow{\text{val}} x{\in}\mathbf{V}\colon x \in \mathbf{B} \textbf{ and not } x]$

3.6 Integers

$[x \in \mathbf{Z} \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x \text{ IntP}]$

$[\text{int } (\ x\ ) \xrightarrow{\text{val}} \text{TheInt } (\ x \text{ Head TheBool}\ ,\ x \text{ Tail TheNat}\ )]$

$[\text{PlusTag } x \xrightarrow{\text{val}} \text{TheInt } (\ \mathsf{T}\ ,\ x\ )]$

$[\text{MinusTag } x \xrightarrow{\text{val}} \text{TheInt } (\ \mathsf{F}\ ,\ x\ )]$

$[x \text{ Sign} \xrightarrow{\text{val}} x \text{ Tail Head}]$

$[x \text{ Mag} \xrightarrow{\text{val}} x \text{ Tail Tail}]$

$[+x \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x{\in}\mathbf{Z}\colon x]$

$[{}^{+}x \xrightarrow{\text{val}} +x]$

3.7 Integer addition

$[\text{TheInt}\ (\ s\ ,\ v\ ) \overset{\text{val}}{\to} \text{IntTag Pair}\ (s\ \text{Or}\ v)\ \text{Pair}\ v]$

$[x + y \overset{\text{val}}{\to\to} \text{norm}\ x{\in}\mathbf{V}{:}\, y{\in}\mathbf{V}{:}\, \text{plus1}\ (\ x^{\text{N}}\ ,\ y^{\text{N}}\ )]$

$[\text{plus1}\ (\ x\ ,\ y\ ) \overset{\text{val}}{\to} x{\in}\mathbf{Z}{:}\, y{\in}\mathbf{Z}{:}\, \text{plus2}\ (\ x\ \text{Sign}\ ,\ y\ \text{Sign}\ ,\ x\ \text{Mag}\ ,\ y\ \text{Mag}\ )]$

$$[\text{plus2}\ (\ s\ ,\ t\ ,\ x\ ,\ y\ ) \overset{\text{val}}{\to} s \left\langle \begin{array}{l} t \left\langle \begin{array}{l} \text{PlusTag}\ x\ \text{Plus}\ y \\ x\ \text{LT}\ y \left\langle \begin{array}{l} \text{MinusTag}\ y\ \text{Minus}\ x \\ \text{PlusTag}\ x\ \text{Minus}\ y \end{array} \right. \end{array} \right. \\ t \left\langle \begin{array}{l} x\ \text{LT}\ y \left\langle \begin{array}{l} \text{PlusTag}\ y\ \text{Minus}\ x \\ \text{MinusTag}\ x\ \text{Minus}\ y \end{array} \right. \\ \text{MinusTag}\ x\ \text{Plus}\ y \end{array} \right. \end{array} \right. \quad]$$

3.8 Integer subtraction

$[-x \overset{\text{val}}{\to\to} \text{norm}\ x{\in}\mathbf{V}{:}\, \text{minus1}\ (\ x^{\text{N}}\ )]$

$[{}^{-}x \overset{\text{val}}{\to\to} -x]$

$[\text{minus1}\ (\ x\ ) \overset{\text{val}}{\to} x{\in}\mathbf{Z}{:}\, \text{minus2}\ (\ x\ \text{Sign}\ ,\ x\ \text{Mag}\ )]$

$[\text{minus2}\ (\ s\ ,\ x\ ) \overset{\text{val}}{\to} \text{TheInt}\ (\ \text{Not}\ s\ ,\ x\ )]$

$[x - y \overset{\text{val}}{\to\to} \text{norm}\ x{\in}\mathbf{V}{:}\, y{\in}\mathbf{V}{:}\, x + -y]$

3.9 Integer multiplication

$[x \cdot y \overset{\text{val}}{\to\to} \text{norm}\ x{\in}\mathbf{V}{:}\, y{\in}\mathbf{V}{:}\, \text{times1}\ (\ x^{\text{N}}\ ,\ y^{\text{N}}\ )]$

$[\text{times1}\ (\ x\ ,\ y\ ) \overset{\text{val}}{\to} x{\in}\mathbf{Z}{:}\, y{\in}\mathbf{Z}{:}\, \text{times2}\ (\ x\ \text{Sign}\ ,\ y\ \text{Sign}\ ,\ x\ \text{Mag}\ ,\ y\ \text{Mag}\ )]$

$[\text{times2}\ (\ s\ ,\ t\ ,\ x\ ,\ y\ ) \overset{\text{val}}{\to} \text{TheInt}\ (\ s\ \text{Iff}\ t\ ,\ x\ \text{Times}\ y\ )]$

3.10 Integer comparison

$[x < y \overset{\text{val}}{\to\to} \text{norm}\ x{\in}\mathbf{V}{:}\, y{\in}\mathbf{V}{:}\, x{\in}\mathbf{Z}{:}\, y{\in}\mathbf{Z}{:}\, \text{lt1}\ (\ x^{\text{N}}\ ,\ y^{\text{N}}\ )]$

$[\text{lt1}\ (\ x\ ,\ y\ ) \overset{\text{val}}{\to} \text{lt2}\ (\ x\ \text{Sign}\ ,\ y\ \text{Sign}\ ,\ x\ \text{Mag}\ ,\ y\ \text{Mag}\ )]$

$$[\text{lt2}\ (\ s\ ,\ t\ ,\ x\ ,\ y\ ) \overset{\text{val}}{\to} s \left\langle \begin{array}{l} t \left\langle \begin{array}{l} x\ \text{LT}\ y \\ \mathsf{F} \end{array} \right. \\ t \left\langle \begin{array}{l} \mathsf{T} \\ y\ \text{LT}\ x \end{array} \right. \end{array} \right. \quad]$$

$[x > y \overset{\text{val}}{\to\to} \text{norm}\ x{\in}\mathbf{V}{:}\, y{\in}\mathbf{V}{:}\, x{\in}\mathbf{Z}{:}\, y{\in}\mathbf{Z}{:}\, y < x]$

$[x \leq y \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon y{\in}\mathbf{V}\colon x{\in}\mathbf{Z}\colon y{\in}\mathbf{Z}\colon \textbf{not } x > y]$

$[x \geq y \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon y{\in}\mathbf{V}\colon x{\in}\mathbf{Z}\colon y{\in}\mathbf{Z}\colon \textbf{not } x < y]$

3.11 Integer construction

An integer like 123 can be expressed as an invisible zero followed by three suffix operators $x1$, $x2$, and $x3$. The suffix operators multiply x by ten and add one, two, and three, respectively.

$[\text{Base} \xrightarrow{\text{val}} \text{norm PlusTag Ten}]$

$[\{\} \xrightarrow{\text{val}} \text{norm PlusTag Zero}]$

Invisible zero.

$[x0 \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x \cdot \text{Base}]$

$[x1 \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x \cdot \text{Base} + \text{PlusTag One}]$

$[x2 \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x \cdot \text{Base} + \text{PlusTag Two}]$

$[x3 \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x \cdot \text{Base} + \text{PlusTag Three}]$

$[x4 \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x \cdot \text{Base} + \text{PlusTag Four}]$

$[x5 \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x \cdot \text{Base} + \text{PlusTag Five}]$

$[x6 \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x \cdot \text{Base} + \text{PlusTag Six}]$

$[x7 \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x \cdot \text{Base} + \text{PlusTag Seven}]$

$[x8 \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x \cdot \text{Base} + \text{PlusTag Eight}]$

$[x9 \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x \cdot \text{Base} + \text{PlusTag Nine}]$

$[0 \xrightarrow{\text{val}} \text{norm } 0]$

$[1 \xrightarrow{\text{val}} \text{norm } 1]$

$[2 \xrightarrow{\text{val}} \text{norm } 2]$

$[3 \xrightarrow{\text{val}} \text{norm } 3]$

$[4 \xrightarrow{\text{val}} \text{norm } 4]$

$[5 \xrightarrow{\text{val}} \text{norm } 5]$

$[6 \xrightarrow{\text{val}} \text{norm } 6]$

[7 $\stackrel{\text{val}}{\rightarrow}$ norm 7]

[8 $\stackrel{\text{val}}{\rightarrow}$ norm 8]

[9 $\stackrel{\text{val}}{\rightarrow}$ norm 9]

3.12 Logical operations on integers

[evenp (x) $\stackrel{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: $x{\in}\mathbf{Z}$: x Mag Head TheBool]

[oddp (x) $\stackrel{\text{val}}{\rightarrow}$ norm $x{\in}\mathbf{V}$: $x{\in}\mathbf{Z}$: **not** evenp (x)]

[half (x) $\stackrel{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: $x{\in}\mathbf{Z}$: norm **If** x Sign **then** PlusTag x Mag Tail **else** MinusTag (x Plus One) Mag Tail]

[small (x) $\stackrel{\text{val}}{\rightarrow}$ norm $x{\in}\mathbf{V}$: norm $x{\in}\mathbf{Z}$: $^{-}1 \leq x$ **and** $x \leq 0$]

[double (b , i) $\stackrel{\text{val}}{\rightarrow}$ norm $b{\in}\mathbf{V}$: $i{\in}\mathbf{V}$: $b{\in}\mathbf{B}$: $i{\in}\mathbf{Z}$: **if** b **then** $1 + 2 \cdot i$ **else** $2 \cdot i$]

[lognot (x) $\stackrel{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: $x{\in}\mathbf{Z}$: $^{-}1 - x$]

[logior (x , y) $\stackrel{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: $y{\in}\mathbf{V}$: $x{\in}\mathbf{Z}$: $y{\in}\mathbf{Z}$:
if $x = {}^{-}1$ **then** $^{-}1$ **else if** $y = {}^{-}1$ **then** $^{-}1$ **else**
if $x = 0$ **then** y **else if** $y = 0$ **then** x **else** double (oddp (x) **or** oddp (y) , logior (half (x) , half (y)))]

[logxor (x , y) $\stackrel{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: $y{\in}\mathbf{V}$: $x{\in}\mathbf{Z}$: $y{\in}\mathbf{Z}$:
if $x = {}^{-}1$ **then** lognot (y) **else if** $y = {}^{-}1$ **then** lognot (x) **else**
if $x = 0$ **then** y **else if** $y = 0$ **then** x **else** double (oddp (x) $\neq$ oddp (y) , logxor (half (x) , half (y)))]

[logand (x , y) $\stackrel{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: $y{\in}\mathbf{V}$: $x{\in}\mathbf{Z}$: $y{\in}\mathbf{Z}$:
if $x = {}^{-}1$ **then** y **else if** $y = {}^{-}1$ **then** x **else**
if $x = 0$ **then** 0 **else if** $y = 0$ **then** 0 **else** double (oddp (x) **and** oddp (y) , logand (half (x) , half (y)))]

[logeqv (x , y) $\stackrel{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: $y{\in}\mathbf{V}$: $x{\in}\mathbf{Z}$: $y{\in}\mathbf{Z}$: lognot (logxor (x , y))]

[lognand (x , y) $\stackrel{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: $y{\in}\mathbf{V}$: $x{\in}\mathbf{Z}$: $y{\in}\mathbf{Z}$: lognot (logand (x , y))]

[lognor (x , y) $\stackrel{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: $y{\in}\mathbf{V}$: $x{\in}\mathbf{Z}$: $y{\in}\mathbf{Z}$: lognot (logior (x , y))]

[logandc1 (x , y) $\stackrel{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: $y{\in}\mathbf{V}$: $x{\in}\mathbf{Z}$: $y{\in}\mathbf{Z}$: logand (lognot (x) , y)]

[logandc2 (x , y) $\stackrel{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: $y{\in}\mathbf{V}$: $x{\in}\mathbf{Z}$: $y{\in}\mathbf{Z}$: logand (x , lognot (y))]

[logorc1 (x , y) $\stackrel{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: $y{\in}\mathbf{V}$: $x{\in}\mathbf{Z}$: $y{\in}\mathbf{Z}$: logior (lognot (x) , y)]

[logorc2 (x , y) $\overset{\text{val}}{\longrightarrow}$ norm $x{\in}\mathbf{V}$: $y{\in}\mathbf{V}$: $x{\in}\mathbf{Z}$: $y{\in}\mathbf{Z}$: logior (x , lognot (y))]

[logtest (x , y) $\overset{\text{val}}{\longrightarrow}$ norm $x{\in}\mathbf{V}$: $y{\in}\mathbf{V}$: $x{\in}\mathbf{Z}$: $y{\in}\mathbf{Z}$: logand (x , y) $\neq 0$]

[ash (i , c) $\overset{\text{val}}{\longrightarrow}$ norm $i{\in}\mathbf{V}$: $c{\in}\mathbf{V}$: $i{\in}\mathbf{Z}$: $c{\in}\mathbf{Z}$: **if** $c \geq 0$ **then** ash+ (i , c) **else** ash- (i , $-c$)]

[ash+ (i , c) $\overset{\text{val}}{\rightarrow}$ norm $i{\in}\mathbf{V}$: $c{\in}\mathbf{V}$: $i{\in}\mathbf{Z}$: $c{\in}\mathbf{Z}$: **if** $c = 0$ **then** i **else** ash+ ($2 \cdot i$, $c - 1$)]

[ash- (i , c) $\overset{\text{val}}{\rightarrow}$ norm $i{\in}\mathbf{V}$: $c{\in}\mathbf{V}$: $i{\in}\mathbf{Z}$: $c{\in}\mathbf{Z}$: **if** $c = 0$ **then** i **else** ash- (half (i) , $c - 1$)]

[logbitp (x , i) $\overset{\text{val}}{\longrightarrow}$ norm $x{\in}\mathbf{V}$: $i{\in}\mathbf{V}$: $x{\in}\mathbf{Z}$: $i{\in}\mathbf{Z}$: **if** $x < 0$ **then** $\bullet$ **else** oddp (ash (i , $-x$))]

[logcount (x) $\overset{\text{val}}{\longrightarrow}$ norm $x{\in}\mathbf{V}$: $x{\in}\mathbf{Z}$: **if** $x < 0$ **then** logcount1 (lognot (x)) **else** logcount1 (x)]

[logcount1 (x) $\overset{\text{val}}{\rightarrow}$ norm $x{\in}\mathbf{V}$: $x{\in}\mathbf{Z}$: **if** $x = 0$ **then** 0 **else** logcount1 (half (x)) + **if** evenp (x) **then** 0 **else** 1]

[integer-length (x) $\overset{\text{val}}{\longrightarrow}$ norm $x{\in}\mathbf{V}$: $x{\in}\mathbf{Z}$: **if** small (x) **then** 0 **else** 1 + integer-length (half (x))]

3.13 Vector operations

A byte is an integer in the range 0..255. We shall represent sequences of bytes as *vectors* as follows: Given a sequence of bytes, add the number 1 at the end of the sequence, then interpret the sequence little endian base 256. In that way, sequences of bytes are represented by cardinals (positive integers). As an example, the sequence $\langle 65, 66 \rangle$ is represented by the number $65 + 256 \cdot 66 + 256 \cdot 256 \cdot 1$.

Strings like 'AB' are treated as vectors. The value of 'AB' is $65 + 256 \cdot 66 + 256 \cdot 256 \cdot 1$ because A and B have unicode 65 and 66, respectively. Strings are encoded using UTF-8 so strings are always sequences of bytes even when characters in the strings have unicodes above 255.

If a number like $65{+}256{\cdot}66{+}256{\cdot}256{\cdot}2$ is interpreted as a vector, it represents the vector consisting of the bytes 65 and 66. Hence, when translating integers to vectors, the stop byte can have any, non-zero value.

We define the following functions on vectors:

vector (x). Normalize the vector (change the stop byte to a one-byte).

vector-suffix (x , b). Return the subvector beginning at byte number b (where the first byte is byte number zero): vector-suffix ('ABCD' , 2) = 'CD'.

vector-prefix (x , e). Return the subvector ending before byte number e: vector-prefix (‘ABCD’ , 2) = ‘AB’.

vector-subseq (x , b , e). Return the subvector beginning at b and ending before e: vector-subseq (‘ABCD’ , 1 , 3) = ‘BC’.

vector-length (x). Return the length of the vector: vector-length (‘ABCD’) = 4.

vector-index (x , i). Return the i’th byte: vector-index (‘ABCD’ , 1) = 66.

[vector-head (x) $\stackrel{\bullet}{=}$ vector-index (x , 0)].

[vector-tail (x) $\stackrel{\bullet}{=}$ vector-suffix (x , 1)].

We shall represent byte sequences in four ways:

byte* A list of bytes.

vector An integer which encodes the list little endian base 256 with a stop byte at the end.

bt A *byte tree*, i.e. a structure built up from bytes and conses. To convert it to a sequence, scan the tree root to leaf, left to right, and collect all bytes found. Ignore anything which is not a byte.

t A *byte vector*, i.e. a structure built up from vectors and conses. To convert it to a sequence, scan the tree root to leaf, left to right, and append all vectors (integers) found. Ignore anything which is not a vector.

The following functions convert between the representations. Examples:

$$\begin{array}{lcl}
\text{vector2byte}*\ (\ \text{‘ABC’}\ ) & = & \langle 65, 66, 67\rangle \\
\text{bt2byte}*\ (\ (65::66)::-1::256::\mathsf{T}::67::\mathsf{F}::68\ ) & = & \langle 65, 66, 67, 68\rangle \\
\text{bt2vector}\ (\ (65::66)::-1::256::\mathsf{T}::67::\mathsf{F}::68\ ) & = & \text{‘ABCD’} \\
\text{vt2byte}*\ (\ (\text{‘A’}::\text{‘BC’})::-1::\text{‘’}::\mathsf{F}::\text{‘D’}\ ) & = & \langle 65, 66, 67, 68\rangle \\
\text{vt2vector}\ (\ (\text{‘A’}::\text{‘BC’})::-1::\text{‘’}::\mathsf{F}::\text{‘D’}\ ) & = & \text{‘ABCD’}
\end{array}$$

In the implementations, care is taken to make the functions tail recursive. Furthermore, whenever tail recursion had to favor either recursion in the head or in the tail of pairs, recursion in the tail has been favored. For that reason, the functions can handle very long lists without stack overflow. This is done at the cost of having to reverse lists occasionally.
The definitions read:

[vector-mask $\stackrel{\text{val}}{\rightarrow}$ norm 255]

[octet-base $\stackrel{\text{val}}{\rightarrow}$ norm 256]

[vector-empty (x) $\stackrel{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: $x \leq$ vector-mask]

[vector-head1 (x) $\overset{\text{val}}{\rightarrow}$ norm $x{\in}\mathbf{V}$: logand (vector-mask , x)]

[vector-tail1 (x) $\overset{\text{val}}{\rightarrow}$ norm $x{\in}\mathbf{V}$: ash (x , $^{-}8$)]

[vector-cons (x , y) $\overset{\text{val}}{\rightarrow}$ norm $x{\in}\mathbf{V}$: $y{\in}\mathbf{V}$: logior (x , ash (y , 8))]

[vector (x) $\overset{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: **if** vector-empty (x) **then** **else** vector-cons (vector-head1 (x) , vector (vector-tail1 (x)))]

[vector-norm (x) $\overset{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: **if** $x < 0$ **then** T **else if** integer-length (x) mod $8 = 1$ **then** x **else** T]

[vector-suffix (x , b) $\overset{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: $b{\in}\mathbf{V}$: **if** $b \leq 0$ **then** vector (x) **else** vector-suffix (vector-tail1 (x) , $b - 1$)]

[vector-prefix (x , e) $\overset{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: $e{\in}\mathbf{V}$: $e{\in}\mathbf{Z}$: **if** vector-empty (x) **or** $e \leq 0$ **then** **else** vector-cons (vector-head1 (x) , vector-prefix (vector-tail1 (x) , $e - 1$))]

[vector-subseq (x , b , e) $\overset{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: $b{\in}\mathbf{V}$: $e{\in}\mathbf{V}$: **if** $b \leq 0$ **then** vector-prefix (x , e) **else** vector-subseq (vector-tail1 (x) , $b - 1$, $e - 1$)]

[vector-length (x) $\overset{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: **if** vector-empty (x) **then** 0 **else** 1 + vector-length (vector-tail1 (x))]

[vector-index (x , i) $\overset{\text{val}}{\rightarrow\rightarrow}$ **if** $i < 0$ **or** vector-length (x) $\leq i$ **then** $\bullet$ **else** vector-head1 (vector-suffix (x , i))]

[vector2byte$*$ (x) $\overset{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: vector2byte$*$1 (x , T)]

[vector2byte$*$1 (x , r) $\overset{\text{val}}{\rightarrow}$ norm $x{\in}\mathbf{V}$: $r{\in}\mathbf{V}$:
if vector-empty (x) **then** reverse (r) **else**
vector2byte$*$1 (vector-tail1 (x) , vector-head1 (x) : : r)]

[vector2vector$*$ (x) $\overset{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: vector2vector$*$1 (x , T)]

[vector2vector$*$1 (x , r) $\overset{\text{val}}{\rightarrow}$ norm $x{\in}\mathbf{V}$: $r{\in}\mathbf{V}$:
if vector-empty (x) **then** reverse (r) **else**
vector2vector$*$1 (vector-tail1 (x) , vector-head1 (x)+octet-base : : r)]

[bt2byte$*$ (x) $\overset{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: reverse (bt2byte$*$1 (x , T))]

[bt2byte$*$1 (x , r) $\overset{\text{val}}{\rightarrow}$ norm $x{\in}\mathbf{V}$: $r{\in}\mathbf{V}$:
if $x \in \mathbf{P}$ **then** bt2byte$*$1 (x^{t} , bt2byte$*$1 (x^{h} , r)) **else**
if $x \in \mathbf{Z}$ **and** $0 \leq x$ **and** $x \leq$ vector-mask **then** x : : r **else** r]

[bt2vector∗ (x) $\xrightarrow{\text{val}}$ norm $x{\in}\mathbf{V}$: reverse (bt2vector∗1 (x , $\top$))]

[bt2vector∗1 (x , r) $\xrightarrow{\text{val}}$ norm $x{\in}\mathbf{V}$: $r{\in}\mathbf{V}$:
if $x \in \mathbf{P}$ **then** bt2vector∗1 (x^{t} , bt2vector∗1 (x^{h} , r)) **else**
if $x \in \mathbf{Z}$ **and** $0 \leq x$ **and** $x \leq$ vector-mask **then** x+octet-base : : r **else**
r]

[bt2vector (x) $\xrightarrow{\text{val}}$ norm $x{\in}\mathbf{V}$: revbyte∗2vector (bt2byte∗1 (x , $\top$) , ‘’)]

[revbyte∗2vector (x , r) $\xrightarrow{\text{val}}$ norm $x{\in}\mathbf{V}$: $r{\in}\mathbf{V}$:
if $x \in \mathbf{A}$ **then** r **else**
revbyte∗2vector (x^{t} , vector-cons (x^{h} , r))]

Reverse the list x of bytes, revappend them to the vector r, and return them as a vector.

[vector-revappend (x , y) $\xrightarrow{\text{val}}$ norm $x{\in}\mathbf{V}$: $y{\in}\mathbf{V}$:
if vector-empty (x) **then** y **else**
vector-revappend (vector-tail1 (x) , vector-head1 (x) : : y)]

Convert the vector x to a sequence of bytes and revappend it to the sequence y of bytes.

[vt2byte∗ (x) $\xrightarrow{\text{val}}$ norm $x{\in}\mathbf{V}$: reverse (vt2byte∗1 (x , $\top$))]

[vt2byte∗1 (x , r) $\xrightarrow{\text{val}}$ norm $x{\in}\mathbf{V}$: $r{\in}\mathbf{V}$:
if $x \in \mathbf{P}$ **then** vt2byte∗1 (x^{t} , vt2byte∗1 (x^{h} , r)) **else**
if $x \in \mathbf{Z}$ **then** vector-revappend (x , r) **else** r]

[vector-revappend1 (x , y) $\xrightarrow{\text{val}}$ norm $x{\in}\mathbf{V}$: $y{\in}\mathbf{V}$:
if vector-empty (x) **then** y **else**
vector-revappend1 (vector-tail1 (x) , vector-head1 (x)+octet-base : : y
)]

Convert the vector x to a sequence of bytes and revappend it to the sequence y of bytes.

[vt2vector∗ (x) $\xrightarrow{\text{val}}$ norm $x{\in}\mathbf{V}$: reverse (vt2vector∗1 (x , $\top$))]

[vt2vector∗1 (x , r) $\xrightarrow{\text{val}}$ norm $x{\in}\mathbf{V}$: $r{\in}\mathbf{V}$:
if $x \in \mathbf{P}$ **then** vt2vector∗1 (x^{t} , vt2vector∗1 (x^{h} , r)) **else**
if $x \in \mathbf{Z}$ **then** vector-revappend1 (x , r) **else** r]

[vt2vector (x) $\xrightarrow{\text{val}}$ norm $x{\in}\mathbf{V}$: revbyte∗2vector (vt2byte∗1 (x , $\top$) , ‘’)]

3.14 Division

The division routines in the following return quotient/remainder pairs of form $q :: r$ where q is the quotient and r is the remainder. When dividing the dividend x by the divisor y, the resulting quotient/remainder pair satisfies $x = q \cdot y + r$. The differences between the various kinds of divisions lie in the direction in which the quotient is rounded and, hence, the possible interval for the remainder.

There is general agreement about how negative dividends x should be handled. But there is no similar agreement about the handling of negative divisors y. For that reason, we simply proclaim division by negative numbers to be as undefined as division by zero.

We now define four division routines, floor (x , y), ceiling (x , y), truncate (x , y), and round (x , y), one for each kind of rounding defined by the IEEE floating point standard. These division routines raise an exception in case of division by zero or division by a negative number. The names of the division routines are inspired by Common Lisp, c.f. the Common Lisp Hyperspec, which is available on the World Wide Web.

We consider floor (x , y), which does rounding towards minus infinity, to be the most useful integer division routine, and introduce x div y and x mod y to denote the quotient and remainder, respectively, returned by floor (x , y).

To define division, we first define restricted forms of division floor1 (x , y), ceiling1 (x , y), and round1 (x , y) which can handle positive arguments. We also introduce a construct reverse quotient (p) for changing the sign of both quotient and remainder in a quotient/remainder pair. The restricted division routines may loop indefinitely when the input values are out of range.

[floor1 (x , y) $\stackrel{\text{val}}{\rightarrow}$
if $x < y$ **then** $0 :: x$ **else**
let floor1 (x , $2 \cdot y$) **be** z **in**
if $z^{\text{t}} < y$ **then** $2 \cdot z^{\text{h}} :: z^{\text{t}}$ **else** $2 \cdot z^{\text{h}} + 1 :: z^{\text{t}} - y$]

Divide the natural x by the positive integer y with rounding towards minus infinity.

[ceiling1 (x , y) $\stackrel{\text{val}}{\rightarrow}$
let floor1 (x , y) **be** z **in**
if $z^{\text{t}} = 0$ **then** z **else** $z^{\text{h}} + 1 :: z^{\text{t}} - y$]

Divide the positive integer x by the positive integer y with rounding towards plus infinity and return af pair $q :: r$ of quotient d and remainder r.

[round1 (x , y) $\stackrel{\text{val}}{\rightarrow}$
let floor1 (x , y) **be** z **in**
if $z^{\text{t}} \cdot 2 < y$ **then** z **else**
if $z^{\text{t}} \cdot 2 > y$ **then** $z^{\text{h}} + 1 :: z^{\text{t}} - y$ **else**
if evenp (z^{h}) **then** z **else**
$z^{\text{h}} + 1 :: z^{\text{t}} - y$]

Divide the natural x by the positive integer y with rounding towards nearest/even, c.f. the IEEE floating point standard or the Common Lisp Hyperspec.

[reverse quotient (p) $\overset{\text{val}}{\rightarrow} -p^{\text{h}} :: -p^{\text{t}}$]

Negate both quotient and remainder of a quotient/remainder pair.

[truncate (x , y) $\overset{\text{val}}{\rightarrow\rightarrow}$
if $y \leq 0$ **then** $\bullet$ **else**
if $0 \leq x$ **then** floor1 (x , y) **else**
reverse quotient (floor1 ($-x$, y))]

Divide the integer x by the positive integer y with rouding towards zero.

[floor (x , y) $\overset{\text{val}}{\rightarrow\rightarrow}$
if $y \leq 0$ **then** $\bullet$ **else**
if $0 \leq x$ **then** floor1 (x , y) **else**
reverse quotient (ceiling1 ($-x$, y))]

Divide the integer x by the positive integer y with rouding towards minus infinity.

[ceiling (x , y) $\overset{\text{val}}{\rightarrow\rightarrow}$
if $y \leq 0$ **then** $\bullet$ **else**
if $0 < x$ **then** ceiling1 (x , y) **else**
reverse quotient (floor1 ($-x$, y))]

Divide the integer x by the positive integer y with rouding towards plus infinity.

[round (x , y) $\overset{\text{val}}{\rightarrow\rightarrow}$
if $y \leq 0$ **then** $\bullet$ **else**
if $0 < x$ **then** round1 (x , y) **else**
reverse quotient (round1 ($-x$, y))]

Divide the integer x by the positive integer y with rouding towards nearest/even, c.f. the IEEE floating point standard or the Common Lisp Hyperspec.

[x div $y \overset{\text{val}}{\rightarrow\rightarrow}$ floor (x , y)^{h}]

[x mod $y \overset{\text{val}}{\rightarrow\rightarrow}$ floor (x , y)^{t}]

3.15 Pairs

[$x \in \mathbf{P} \overset{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: x PairP]

[$a \in \mathbf{A} \overset{\text{val}}{\rightarrow}$ norm $a{\in}\mathbf{V}$: norm $a{\in}\mathbf{V}$: **not** $a \in \mathbf{P}$]

$[x :: y \xrightarrow{\text{val}} x{\in}\mathbf{V}: y{\in}\mathbf{V}: \text{PairTag Pair } x^{\text{N}} \text{ Pair } y^{\text{N}}]$

$[x^{\text{h}} \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}: \textbf{if } x \in \mathbf{A} \textbf{ then } x^{\text{N}} \textbf{ else } x^{\text{N}} \text{ Tail Head}]$

$[x^{\text{t}} \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}: \textbf{if } x \in \mathbf{A} \textbf{ then } x^{\text{N}} \textbf{ else } x^{\text{N}} \text{ Tail Tail}]$

$[\text{append } (\ x \ , \ y \ ) \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}: y{\in}\mathbf{V}:$
$\textbf{if } x \in \mathbf{A} \textbf{ then } y \textbf{ else } x^{\text{h}} :: \text{append } (\ x^{\text{t}} \ , \ y \ )]$

$[\text{reverse } (\ x \ ) \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}: \text{revappend } (\ x \ , \ \mathsf{T} \ )]$

$[\text{revappend } (\ x \ , \ y \ ) \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}: y{\in}\mathbf{V}:$
$\textbf{if } x \in \mathbf{A} \textbf{ then } y \textbf{ else } \text{revappend } (\ x^{\text{t}} \ , \ x^{\text{h}} :: y \ )]$

$[\text{nth } (\ n \ , \ x \ ) \xrightarrow{\text{val}} \text{norm } n{\in}\mathbf{V}: x{\in}\mathbf{V}:$
$\textbf{if } n \leq 0 \textbf{ then } x^{\text{h}} \textbf{ else } \text{nth } (\ n - 1 \ , \ x^{\text{t}} \ )]$

$[\text{length } (\ x \ ) \stackrel{\bullet}{=} \text{length1 } (\ x \ , \ 0 \ )]$.

Compute length of list.

$[\text{length1 } (\ x \ , \ r \ ) \stackrel{\bullet}{=} \textbf{if } x \in \mathbf{A} \textbf{ then } r \textbf{ else } \text{length1 } (\ x^{\text{t}} \ , \ r + 1 \ )]$.

Compute the sum of the integer r and the length of the list x.

$[\text{list-prefix } (\ l \ , \ n \ ) \stackrel{\bullet}{=}$
$\textbf{if } n = 0 \textbf{ or } l \in \mathbf{A} \textbf{ then } \mathsf{T} \textbf{ else } l^{\text{h}} :: \text{list-prefix } (\ l^{\text{t}} \ , \ n - 1 \ )]$

Return the first n elements of the list l.

$[\text{list-suffix } (\ l \ , \ n \ ) \stackrel{\bullet}{=}$
$\textbf{if } n = 0 \textbf{ then } l \textbf{ else } \text{list-suffix } (\ l^{\text{t}} \ , \ n - 1 \ )]$

Remove the first n elements from the list l.

3.16 Exceptions

$[x^{\bullet} \xrightarrow{\text{val}} x{\in}\mathbf{V}: \text{ExTag Pair } x^{\text{N}}]$

$[\bullet \xrightarrow{\text{val}} \text{norm } \mathsf{T}^{\bullet}]$

$[\text{catch } (\ x \ ) \xrightarrow{\text{val}} \text{Catch } (\ x^{\text{N}} \ )]$

$[x^{\circ} \xrightarrow{\text{val}} \text{Catch } (\ x^{\text{N}} \ )]$

$[\text{Catch } (\ x \ ) \xrightarrow{\text{val}} \textbf{If } x \text{ ExP } \textbf{then } \mathsf{T} :: x \text{ Tail } \textbf{else } \mathsf{F} :: x]$

3.17 Maps

$[x \in \mathbf{M} \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x \text{ MapP}]$

$[\text{map } (\ x\ ) \xrightarrow{\text{val}} \text{MapTag LazyPair } x]$

$[x^{\text{M}^\circ} \xrightarrow{\text{val}} \text{map } (\ x^{\text{N}}\ )]$

$[x^{\text{M}} \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x^{\text{M}^\circ}]$

$[x^{\text{U}} \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x{\in}\mathbf{M}\colon x \text{ Tail}^{\text{N}}]$

$[x \text{ ''} y \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon y{\in}\mathbf{V}\colon x{\in}\mathbf{M}\colon y{\in}\mathbf{M}\colon \text{map } (\ x \text{ Tail ' } y \text{ Tail }\ )]$

$[x^{\text{R}} \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x{\in}\mathbf{M}\colon x \text{ Tail TheBool}]$

3.18 Objects

$[x \in \mathbf{O} \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x \text{ ObjectP}]$

$[\text{object } (\ x\ ) \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon \text{Object } (\ x^{\text{hh}}\ ,\ x^{\text{ht}}\ ,\ x^{\text{t}}\ )]$

$[\text{Object } (\ x\ ,\ y\ ,\ z\ ) \xrightarrow{\text{val}} x{\in}\mathbf{Z}\colon y{\in}\mathbf{Z}\colon z{\in}\mathbf{V}\colon \textbf{if } x < 0 \textbf{ or } y < 0 \textbf{ or } x = 0 \textbf{ and } y \leq 4 \textbf{ then } \bullet \textbf{ else } (x \text{ Mag Pair } y \text{ Mag}) \text{ Pair } z]$

$[\text{destruct } (\ x\ ) \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x{\in}\mathbf{O}\colon ((\text{PlusTag } (x \text{ Head Head})::\text{PlusTag } (x \text{ Head Tail}))::x \text{ Tail})^{\text{N}}]$

4 Evaluation

4.1 Let construct

$[\textbf{let } x \textbf{ be } y \xrightarrow{\text{val}} x{\in}\mathbf{V}\colon y \text{ ' } x]$
$[x \textbf{ in } y \bowtie \text{`lambda'}]$

4.2 Simple accessors

$[x^0 \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x^{\text{h}}]$

$[x^1 \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x^{\text{t}0}]$

$[x^2 \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x^{\text{t}1}]$

$[x^3 \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x^{\text{t}2}]$

$[x^4 \xrightarrow{\text{val}} \text{norm } x{\in}\mathbf{V}\colon x^{\text{t}3}]$

$[x^5 \stackrel{\text{val}}{\to} \text{norm } x{\in}\mathbf{V}\colon x^{\mathrm{t}4}]$

$[x^6 \stackrel{\text{val}}{\to} \text{norm } x{\in}\mathbf{V}\colon x^{\mathrm{t}5}]$

$[x^7 \stackrel{\text{val}}{\to} \text{norm } x{\in}\mathbf{V}\colon x^{\mathrm{t}6}]$

$[x^8 \stackrel{\text{val}}{\to} \text{norm } x{\in}\mathbf{V}\colon x^{\mathrm{t}7}]$

$[x^9 \stackrel{\text{val}}{\to} \text{norm } x{\in}\mathbf{V}\colon x^{\mathrm{t}8}]$

4.3 Representation of terms

Section 1.2 proclaimed $\lceil x \rceil$ to denote *quoting*. The value of $\lceil x \rceil$ is a structure which represents the term x. As an example, the value of $\lceil u+2 \rceil$ represents the plus construct applied to the u construct and the 2 construct.

As mentioned in Section 1.3, every Logiweb construct is represented by a *reference* and an *index*. The reference is a world-wide unique natural number which identifies the *home page* of the construct, i.e. the page on which the construct is introduced. The index is a natural number which distinguishes the construct from the other constructs introduced on that home page.

The value of $\lceil u + 2 \rceil$ has form

$$(r :: i :: d) :: x :: y :: \mathsf{T}$$

where r and i are the reference and index, respectively, of the plus construct. x and y represent the terms u and 2 respectively. d is .

The purpose of debugging information is to allow users to locate the source of errors. When a user receives an error message saying e.g. that a particular proof is in error, then the error message is likely to contain a term from the macro expanded version of the page. On the present page, the debugging information of a term indicates the location a term had before macro expansion. Users have control over the debugging information through the unpacking and macro expansion machinery.

On the present page, the debugging information of a term has form

$$(p_n :: \cdots :: p_1 :: r' :: \mathsf{T}) :: \mathsf{T}$$

where r' is the reference of the page on which the term occurs and $p_n, \ldots, p_1$ indicate that before macro expansion, the term was subtree number p_n of subtree number p_{n-1} and so on of the root of the page.

Strings are represented in a particular way in that they have form

$$(r :: i :: d) :: \mathsf{T}$$

where r is zero and i is a natural number which represents the string. i is constructed thus: write the string as a sequence of bytes using Unicode and UTF-8 encoding (a *byte* is a natural number between 0 and 255, inclusive). Then add a byte with value 1 at the end. Finally, convert the sequence of bytes to a natural number using little endian radix 256 representation.

On many occasions, there is a need to assign aspects to a page. As an example, Section 4.8 assigns a 'claim' aspect to the present page. As another example, Section 6 assigns a 'macro' aspect to the present page. Formally, definitions can only assign aspects to constructs and not to pages. That problem is solved by the convention that every page has a *page construct* which represents the page. The page construct of a page has index zero and aspects of the page construct should be interpretted as aspects of the page. As examples, the name and show aspects of a page symbol define the name of the page expressed in plain text (UTF-8) and TEX, respectively.

The following constructs allows to access the reference, index, and debugging information of a term x

$[x^{\mathrm{r}} \stackrel{\mathrm{val}}{\to} \text{norm } x{\in}\mathbf{V}\colon x^{\mathrm{hh}}]$

$[x^{\mathrm{i}} \stackrel{\mathrm{val}}{\to} \text{norm } x{\in}\mathbf{V}\colon x^{\mathrm{hth}}]$

$[x^{\mathrm{d}} \stackrel{\mathrm{val}}{\to} \text{norm } x{\in}\mathbf{V}\colon x^{\mathrm{htt}}]$

4.4 Tree equality

$[x \stackrel{\mathrm{t}}{=} y \stackrel{\mathrm{val}}{\to} \text{norm } x{\in}\mathbf{V}\colon y{\in}\mathbf{V}\colon x^{\mathrm{r}} = y^{\mathrm{r}}$ **and** $x^{\mathrm{i}} = y^{\mathrm{i}}$ **and** $x^{\mathrm{t}} \stackrel{\mathrm{t}^*}{=} y^{\mathrm{t}}]$ True if the terms x and y are equal modulo differences in debugging information. As an example, $\lceil z \rceil = \lceil z \rceil$ is false because the two instances of z occur different places on the page whereas $\lceil z \rceil \stackrel{\mathrm{t}}{=} \lceil z \rceil$ because the $\stackrel{\mathrm{t}}{=}$ operation disregards debugging information.

$[x \stackrel{\mathrm{t}^*}{=} y \stackrel{\mathrm{val}}{\to} \text{norm } x{\in}\mathbf{V}\colon y{\in}\mathbf{V}\colon$ **if** $x \in \mathbf{A}$ **then** $y \in \mathbf{A}$ **else if** $y \in \mathbf{A}$ **then** F **else** $x^{\mathrm{h}} \stackrel{\mathrm{t}}{=} y^{\mathrm{h}}$ **and** $x^{\mathrm{t}} \stackrel{\mathrm{t}^*}{=} y^{\mathrm{t}}]$ Coordinatewise application of $u \stackrel{\mathrm{t}}{=} v$ to the lists x and y of terms.

$[x \stackrel{\mathrm{r}}{=} y \stackrel{\mathrm{val}}{\to} \text{norm } x{\in}\mathbf{V}\colon y{\in}\mathbf{V}\colon x^{\mathrm{r}} = y^{\mathrm{r}}$ **and** $x^{\mathrm{i}} = y^{\mathrm{i}}]$ True if the roots of terms x and y are equal modulo differences in debugging information.

[lookup (x , s , d) $\stackrel{\mathrm{val}}{\to}$ norm $x{\in}\mathbf{V}\colon s{\in}\mathbf{V}\colon d{\in}\mathbf{V}\colon$ **if** $s \in \mathbf{A}$ **then** d **else if** $x \stackrel{\mathrm{t}}{=} s^{\mathrm{hh}}$ **then** s^{ht} **else** lookup (x , s^{t} , d)] Look up the term x in the stack s. A *stack* is an association list which associates terms to values. When the term x is not found in the stack s, the default d is returned.

[zip (x , y) $\stackrel{\mathrm{val}}{\to}$ norm $x{\in}\mathbf{V}\colon y{\in}\mathbf{V}\colon$
if $x \in \mathbf{A}$ **or** $y \in \mathbf{A}$ **then** $\top$ **else**
$(x^{\mathrm{h}} :: y^{\mathrm{h}}) :: \text{zip } (x^{\mathrm{t}} , y^{\mathrm{t}})]$ Zip the list x of terms and the list y of values into a stack (may of course be used for arbitrary lists).

4.5 Arrays

$[a[i] \stackrel{\text{val}}{\to} \text{norm } a{\in}\mathbf{V}{:}\, i{\in}\mathbf{V}{:}\, i{\in}\mathbf{Z}{:}\, \text{array1 } (\ a\ ,\ i\ ,\ 0\ )]$

$[\text{array1 } (\ a\ ,\ i\ ,\ x\ ) \stackrel{\text{val}}{\to} \text{norm } a{\in}\mathbf{V}{:}\, i{\in}\mathbf{V}{:}\, x{\in}\mathbf{V}{:}\, i{\in}\mathbf{Z}{:}\, x{\in}\mathbf{Z}{:}$
$\quad \textbf{if } a \in \mathbf{A} \textbf{ then } \top \textbf{ else}$
$\quad \textbf{if } a^{\text{h}} \in \mathbf{Z} \textbf{ then if } i = a^{\text{h}} \textbf{ then } a^{\text{t}} \textbf{ else } \top \textbf{ else}$
$\quad \text{array1 } (\ \textbf{if } \text{logbitp } (\ x\ ,\ i\ ) \textbf{ then } a^{\text{t}} \textbf{ else } a^{\text{h}}\ ,\ i\ ,\ x+1\ )]$

$[a[i{\to}v] \stackrel{\text{val}}{\to} \text{norm } a{\in}\mathbf{V}{:}\, i{\in}\mathbf{V}{:}\, v{\in}\mathbf{V}{:}\, i{\in}\mathbf{Z}{:}\, \text{array2 } (\ a\ ,\ i\ ,\ v\ ,\ 0\ )]$

$[\text{array2 } (\ a\ ,\ i\ ,\ v\ ,\ x\ ) \stackrel{\text{val}}{\to} \text{norm } a{\in}\mathbf{V}{:}\, i{\in}\mathbf{V}{:}\, v{\in}\mathbf{V}{:}\, x{\in}\mathbf{V}{:}\, i{\in}\mathbf{Z}{:}\, x{\in}\mathbf{Z}{:}$
$\quad \textbf{if } a \in \mathbf{A} \textbf{ then } \text{array3 } (\ i\ ,\ v\ ) \textbf{ else}$
$\quad \textbf{if } a^{\text{h}} \in \mathbf{Z} \textbf{ then}$
$\quad \textbf{if } i = a^{\text{h}} \textbf{ then}$
$\quad \text{array3 } (\ i\ ,\ v\ ) \textbf{ else}$
$\quad \textbf{if } v \textbf{ then } a \textbf{ else}$
$\quad \text{array5 } (\ a^{\text{h}}\ ,\ a^{\text{t}}\ ,\ i\ ,\ v\ ,\ x\ ) \textbf{ else}$
$\quad \textbf{if } \text{logbitp } (\ x\ ,\ i\ ) \textbf{ then}$
$\quad \text{array4 } (\ a^{\text{h}}\ ,\ \text{array2 } (\ a^{\text{t}}\ ,\ i\ ,\ v\ ,\ x+1\ )\ ) \textbf{ else}$
$\quad \text{array4 } (\ \text{array2 } (\ a^{\text{h}}\ ,\ i\ ,\ v\ ,\ x+1\ )\ ,\ a^{\text{t}}\ )]$

$[\text{array3 } (\ i\ ,\ v\ ) \stackrel{\text{val}}{\to} \text{norm } i{\in}\mathbf{V}{:}\, v{\in}\mathbf{V}{:}\, i{\in}\mathbf{Z}{:}\, \textbf{if } v = \top \textbf{ then } \top \textbf{ else } i :: v]$

$[\text{array4 } (\ a\ ,\ b\ ) \stackrel{\text{val}}{\to} \text{norm } a{\in}\mathbf{V}{:}\, b{\in}\mathbf{V}{:}$
$\quad \textbf{if } a \in \mathbf{A} \textbf{ then}$
$\quad \textbf{if } b \in \mathbf{A} \textbf{ then } \top \textbf{ else if } b^{\text{h}} \in \mathbf{Z} \textbf{ then } b \textbf{ else } a :: b \textbf{ else}$
$\quad \textbf{if } b \in \mathbf{A} \textbf{ and } a^{\text{h}} \in \mathbf{Z} \textbf{ then } a \textbf{ else } a :: b]$

$[\text{array5 } (\ i\ ,\ v\ ,\ j\ ,\ w\ ,\ x\ ) \stackrel{\text{val}}{\to} \text{norm } i{\in}\mathbf{V}{:}\, v{\in}\mathbf{V}{:}\, j{\in}\mathbf{V}{:}\, w{\in}\mathbf{V}{:}\, x{\in}\mathbf{V}{:}\, i{\in}\mathbf{Z}{:}\, j{\in}\mathbf{Z}{:}\, x{\in}\mathbf{Z}{:}$
$\quad \textbf{if } \text{logbitp } (\ x\ ,\ i\ ) \textbf{ then}$
$\quad \textbf{if } \text{logbitp } (\ x\ ,\ j\ ) \textbf{ then}$
$\quad \top :: \text{array5 } (\ i\ ,\ v\ ,\ j\ ,\ w\ ,\ x+1\ ) \textbf{ else}$
$\quad (j :: w) :: (i :: v) \textbf{ else}$
$\quad \textbf{if } \text{logbitp } (\ x\ ,\ j\ ) \textbf{ then}$
$\quad (i :: v) :: (j :: w) \textbf{ else}$
$\quad \text{array5 } (\ i\ ,\ v\ ,\ j\ ,\ w\ ,\ x+1\ ) :: \top]$

$[a[i{\Rightarrow}v] \stackrel{\text{val}}{\to} \text{norm } a{\in}\mathbf{V}{:}\, i{\in}\mathbf{V}{:}\, v{\in}\mathbf{V}{:}\, \textbf{if } i \in \mathbf{A} \textbf{ then } v \textbf{ else } a[i^{\text{h}}{\to}a[i^{\text{h}}][i^{\text{t}}{\Rightarrow}v]]]$

$[\text{push } (\ a\ ,\ i\ ,\ v\ ) \stackrel{\text{val}}{\to} \text{norm } a{\in}\mathbf{V}{:}\, i{\in}\mathbf{V}{:}\, v{\in}\mathbf{V}{:}\, a[i{\to}v :: a[i]]]$

$[\text{pop } (\ a\ ,\ i\ ) \stackrel{\text{val}}{\to} \text{norm } a{\in}\mathbf{V}{:}\, i{\in}\mathbf{V}{:}\, a[i{\to}a[i]^{\text{t}}]]$

$[\text{get}{*}\ (\ a\ ,\ i\ ) \stackrel{\text{val}}{\to} \text{norm } a{\in}\mathbf{V}{:}\, i{\in}\mathbf{V}{:}\, \textbf{if } i \in \mathbf{A} \textbf{ then } a \textbf{ else } \text{get}{*}\ (\ a[i^{\text{h}}]\ ,\ i^{\text{t}}\ )]$

$[\text{push}{*}\ (\ a\ ,\ i\ ,\ v\ ) \stackrel{\text{val}}{\to} \text{norm } a{\in}\mathbf{V}{:}\, i{\in}\mathbf{V}{:}\, v{\in}\mathbf{V}{:}\, a[i{\Rightarrow}v :: \text{get}{*}\ (\ a\ ,\ i\ )]]$

[pop∗ (a , i) $\overset{\text{val}}{\rightarrow}$ norm $a{\in}\mathbf{V}$: $i{\in}\mathbf{V}$: $a[i{\Rightarrow}\text{get}*\ (\ a\ ,\ i\ )^{\text{t}}]$]

[array-domain (x) $\overset{\bullet\bullet}{=}$
if $x \in \mathbf{A}$ **then** $\top$ **else**
if $x^{\text{h}} \in \mathbf{Z}$ **then** $\langle x^{\text{h}} \rangle$ **else**
sort-merge (array-domain (x^{h}) , array-domain (x^{t}))]

Return the domain of the array x sorted in ascending order.

[sort-merge (x , y) $\overset{\bullet\bullet}{=}$
if $x \in \mathbf{A}$ **then** y **else** sort-merge1 (y , x)]

Merge the ascending lists x and y into a new, ascending list.

[sort-merge1 (x , y) $\overset{\bullet\bullet}{=}$
if $x \in \mathbf{A}$ **then** y **else**
if $x^{\text{h}} < y^{\text{h}}$ **then**
x^{h} : : sort-merge1 (x^{t} , y) **else**
y^{h} : : sort-merge1 (y^{t} , x)]

Same as sort-merge (x , y) but assumes y to be a pair.

4.6 Evaluator

[eval (t , s , c) $\overset{\text{val}}{\rightarrow}$ norm $t{\in}\mathbf{V}$: $s{\in}\mathbf{V}$: $c{\in}\mathbf{V}$:
let t^{r} **be** r **in**
let t^{i} **be** i **in**
if $r = 0$ **then** i^{M} **else**
if $i = 0$ **then if** $r = c[0]$ **then** c^{M} **else** $c[c[0]][\text{cluster}][r]^{\text{M}}$ **else**
let $c[r][\text{code}][i]$ **be** f **in**
if $f \in \mathbf{M}$ **then** eval1 (f , t^{t} , s , c) **else**
if $f = \top$ **then** lookup (t , s , map ($\top$)) **else**
if $f = 1$ **then** $t^{1\text{M}}$ **else**
map ($\lambda t.\lambda s.\lambda c.\lambda x.$eval (t^2 , (t^1 : : map (x)) : : s , c) Tail) " t^{M} "
s^{M} " c^{M}]

[eval1 (f , t , s , c) $\overset{\text{val}}{\rightarrow}$ norm $f{\in}\mathbf{V}$: $t{\in}\mathbf{V}$: $s{\in}\mathbf{V}$: $c{\in}\mathbf{V}$: $f{\in}\mathbf{M}$:
if $t \in \mathbf{A}$ **then** f **else**
eval1 (f " eval (t^{h} , s , c) , t^{t} , s , c)]

4.7 Debugging aids

[spy (x) $\overset{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: spy]

[trace (x) $\overset{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: trace]

[print (x) $\overset{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: print]

[timer (x) $\overset{\text{val}}{\rightarrow\rightarrow}$ norm $x{\in}\mathbf{V}$: timer]

spy (x) is faithful to the definition above in that it discards x and returns spy. As a side effect, however, it sets a variable named *spy* to the value of x. It is possible to inspect that variable in several ways: One may ask the Logiweb compiler to print it out using the spy flag. Or one may use the quit flag to enter a read-eval-print loop in which one may issue commands like (spy), (spycd ...), (spyls ...), (spy/), (spy..), and (spypwd) to navigate and inspect the value of the *spy* variable. The *spy* variable is set by all test constructs so that if a test case makes the computer loop indefinitely then one has a chance of finding out which one by inspection of the *spy* variable.

trace (x) is like spy (x) but prints out the value of x to standard output. To get output from a trace (x), the construct has to be executed, which may happen during testing, macro expansion, custom unpacking, or custom rendering. The easiest way to get output from trace is to include it in a test construct.

print (x) is like trace (x) but just dumps the vector tree x to standard output as a sequence of bytes. Note the print (x) does not even append a newline at the end (println (x) defined later does that). Printing a linefeed (code 10) is supposed to generate the newline sequence of the host operating system.

timer (x) returns the string "timer" and is thus easy to recognize from the other ones. The timer construct is supposed to be used as the x in the x .then. y construct which discards x and returns y. Whenever timer (x) is evaluated, the timer associated with x starts counting. If no timer is associated with x, then one is created. The timer counts until a call of timer (y) which causes the timer associated with x to stop and the one associated with y to start counting. Just before the Logiweb compiler exits, it prints the values of all timers except the one associated with T. Hence, timer (T) effectively stops counting. Note that the active timer keeps counting until the timer values are reported.

[x **prog1** $y \overset{\bullet\bullet}{=} x$]

[measure (x , y) $\ddot{=}$ trace (x) .then. timer (x) .then. y **prog1** timer (T)]

4.8 Verification

[base $\overset{\text{claim}}{\rightarrow}$ test1]

[$x \wedge_{\mathrm{c}} y \overset{\text{val}}{\rightarrow} \lambda c.x$ ' c **and** y ' c]

[test1 $\overset{\text{val}}{\rightarrow} \lambda c$.test2 (c)]

[test2 (c) $\overset{\text{val}}{\rightarrow}$ norm $c \in \mathbf{V}$:
let test3 ($c[c[0]]$[expansion] , c)° **be** p **in**
if $p^{\mathrm{t}} \neq \mathsf{T}$ **then** p^{t} **else**
if p^{h} **then** ⌈'In testsuite: unprocessed exception'⌉ **else**
spy (T) .then. T]

[diagnose x end diagnose $\overset{\text{val}}{\to}$ norm $x{\in}\mathbf{V}$:
$(\lceil x\rceil^{\mathrm{r}} :: \lceil x\rceil^{\mathrm{i}} :: x^{\mathrm{d}}) :: x :: \mathsf{T}$]

[test3 (t , c) $\overset{\text{val}}{\to}$ norm $t{\in}\mathbf{V}$: $c{\in}\mathbf{V}$:
let t^{r} **be** r **in**
let t^{i} **be** i **in**
if $r = 0$ **or** $i = 0$ **then** T **else**
let $c[r]$[codex]$[r][i][0]$[claim] **be** d **in**
if $d \in \mathbf{P}$ **then** (eval (d^3 , T , c) " $(t :: c :: \mathsf{T})^{\mathrm{M}})^{\mathrm{U}}$ **else**
if $c[r]$[codex]$[r][i][0]$[definition] $\neq \mathsf{T}$ **then** T **else** test3∗ (t^{t} , c)]

[test3∗ (t , c) $\overset{\text{val}}{\to}$ norm $t{\in}\mathbf{V}$: $c{\in}\mathbf{V}$: **if** $t \in \mathbf{A}$ **then** T **else** test3 (t^{h} , c) **and** test3∗ (t^{t} , c)]

[make math x end math $\overset{\text{val}}{\to}$ norm $x{\in}\mathbf{V}$: $(\lceil\ \$x\$\ \rceil^{\mathrm{r}} :: \lceil\ \$x\$\ \rceil^{\mathrm{i}} :: x^{\mathrm{d}}) :: x :: \mathsf{T}$]

$[[u]^{\cdot} \overset{\text{claim}}{\to} \lambda x.\text{ttst1 (} x \text{)}]$

[ttst1 (x) $\overset{\text{val}}{\to}$ norm $x{\in}\mathbf{V}$: **let** x^0 **be** t **in let** x^1 **be** c **in** spy (t) .then. **if** eval (t^1 , T , c)$^{\mathrm{U}\circ} = \mathsf{F} :: \mathsf{T}$ **then** T **else** make math t end math]

$[[u]^{-} \overset{\text{claim}}{\to} \lambda x.\text{ftst1 (} x \text{)}]$

[ftst1 (x) $\overset{\text{val}}{\to}$ norm $x{\in}\mathbf{V}$: **let** x^0 **be** t **in let** x^1 **be** c **in** spy (t) .then. **if** eval (t^1 , T , c)$^{\mathrm{U}\circ} = \mathsf{F} :: \mathsf{F}$ **then** T **else** make math t end math]

$[[u = v]^{=} \overset{\text{claim}}{\to} \lambda x.\text{etst1 (} x \text{)}]$

[etst1 (x) $\overset{\text{val}}{\to}$ norm $x{\in}\mathbf{V}$: **let** x^0 **be** t **in let** x^1 **be** c **in** spy (t) .then. **if** eval (t^1 , T , c)$^{\mathrm{U}\circ}$ = eval (t^2 , T , c)$^{\mathrm{U}\circ}$ **then** T **else** make math t end math]

$[\mathsf{T}]^{\cdot}$

$[\mathsf{F}]^{-}$

$[2 = 2]^{=}$

$[2^{\bullet} = 2^{\bullet}]^{=}$

4.9 Idiosyncrasies

Logiweb reacts in a predictable way to definitions, even if they look dubious:

[testfunc1 (x) $\doteq x + 3$]

This is an ordinary definition. Thus we have: [testfunc1 (2) = 5]$^{=}$.

$[\text{testfunc2 (} x \text{ , } x \text{)} \doteq x]$

When parameters are repeated, the first one is used: $[\text{testfunc2 (2 , 3)} = 2]^{=}$. In the codex, the definition is recorded as it is: $[\text{self}[\lceil\text{testfunc2 (2 , 3)}\rceil^{\text{r}}][\text{codex}][\lceil\text{testfunc2 (2 , 3)}\rceil^{\text{r}}][\lceil\text{testfunc2 (2 , 3)}\rceil^{\text{i}}][0][\text{value}] \overset{\text{t}}{=} \lceil[\text{testfunc2 (} x \text{ , } x \text{)} \doteq x]\rceil]^{\cdot}$.

$[\text{testfunc3} \doteq 2]$ and $[\text{testfunc3} \doteq 3]$

When a definition is stated more than once, then the rightmost one counts: $[\text{testfunc3} = 3]^{=}$. The codex always contains the definition that counts.

$[\text{testfunc4} \doteq \lceil[\text{testfunc4} \doteq 2]\rceil]$

When a page is harvested, all functions that occur inside other functions are disregarded: $[\text{testfunc4} \overset{\text{t}}{=} \text{hide}(\lceil[\text{testfunc4} \doteq 2]\rceil)]^{\cdot}$. Without the use of hide(x) above, the definition of testfunc4 inside the test case would override the first definition of testfunc4.

$[\text{testfunc5 (} x \text{)} \doteq y]$

Unbound variables equal T: $[\text{testfunc5 (2)} = \mathsf{T}]^{=}$. y is a variable because its root has no value definition: $[\text{self}[\lceil y\rceil^{\text{r}}][\text{code}][\lceil y\rceil^{\text{i}}] = \mathsf{T}]^{=}$

$[\text{testfunc6 (2)} \doteq 2]$

One may foolishly use non-variables as parameters, but it is impossible to get access to their values since their value definitions take precedence over their dynamic bindings: $[\text{testfunc6 (3)} = 2]^{=}$

$[\text{testfunc7 (} x \text{ \_ \{ 4 \})} \doteq x_4 + 3]$

Compound variables work: $[\text{testfunc7 (2)} = 5]^{=}$. The variable x_4 consists of a binary subscript operator applied to x and 4. x_4 is a variable because the subscript operator has no value definition. When deciding whether or not a term is a variable, only the root of the term matters. The fact that 4 does have a value definition has no influence here.

$[\text{testfunc8 (} x \text{ \_ \{ } 2+2 \text{ \} , } x \text{ \_ \{ 4 \})} \doteq x_{2+2} + x_4]$

x_{2+2} and x_4 are distinct variables: $[\text{testfunc8 (2 , 3)} = 5]^{=}$. x_{2+2} and x_4 are distinct variables because they are distinct trees: $[\lceil x_{2+2}\rceil \overset{\text{t}}{=} \lceil x_4\rceil = \mathsf{F}]^{=}$. The fact that $2+2=4$ has no influence here.

$[\text{base} \overset{\text{val}}{\rightarrow} 2+3]$

One cannot change the value of the page symbol, but the definition is still recorded in the codex: $[\text{base}[\text{base}[0]][\text{`codex'}][\lceil\text{base}\rceil^{\text{r}}][\lceil\text{base}\rceil^{\text{i}}][0][\text{`value'}] \overset{\text{t}}{=} \lceil[\text{base} \overset{\text{val}}{\rightarrow} 2+3]\rceil]^{\cdot}$

Furthermore, and more important, the definition is compiled and stored in the code: $[\text{base}[\text{base}[0]][\text{`code'}][0]\ \text{Tail} = 5]^{=}$. This could

be useful for recording values which are costly to compute and thus should only be computed once.

5 Compilation

5.1 Fixed point combinator

$[\mathsf{Y} \overset{\text{val}}{\to\to} \lambda f.(\lambda x.f\ '\ (x\ '\ x))\ '\ (\lambda x.f\ '\ (x\ '\ x))]$

The compiler defined in the following uses the fixed point operator Y. On some systems, the fixed point operator may be optimized using circular structures. Doing so may give a huge speedup in cases where the compiler itself runs non-optimized.

5.2 Compiler

$[\text{compile}\ (\ c\ ) \overset{\text{val}}{\to\to} \text{norm}\ c{\in}\mathbf{V}:$
$(\text{map}\ (\ \lambda c.\mathsf{Y}\ '\ \lambda C.\text{compile1}\ (\ c[0]\ ,\ c\ ,\ \text{map}\ (\ C\ )\ )\ )\ ''\ c^{\text{M}})^{\text{U}}]$

Compile the cache c. $c[0]$ is the reference r of the cache. C is the cache being returned from compile (c). That result is made available to compile1 by the Y combinator. The compile1 function has to approach C with caution: if compile1 untags C, then the fixed point operator will return $\perp$.

The compile function sets $c[r]$[code] to a compiled version of $c[r]$ [codex] and sets $c[r]$[diagnose] to the result of applying $c[r]$[claim] to c. The diagnose $c[r]$[diagnose] and each compiled function $c[r]$ [code][i] is map tagged (using map (x)) such that the diagnose and the compiled functions are computed lazily.

The compile1 function performs a few sanity checks and throws an exception if it finds something wrong.

$[\text{compile1}\ (\ r\ ,\ c\ ,\ C\ ) \overset{\text{val}}{\to} \text{norm}\ r{\in}\mathbf{V}: c{\in}\mathbf{V}: C{\in}\mathbf{V}:$
if not $r \in \mathbf{Z}$ **then** $\bullet$ **else**
let $c[r::\text{code}::\mathsf{T}{\Rightarrow}\text{compile2}\ (\ r\ ,\ c\ ,\ C\ )]$ **be** c **in**
let $c[r::\text{diagnose}::\mathsf{T}{\Rightarrow}\text{compile-claim}\ (\ r\ ,\ c\ ,\ C\ )]$ **be** c **in** $c]$

Compile cache c which has reference r. C is the compiled cache as explained above.

$[\text{compile2}\ (\ r\ ,\ c\ ,\ C\ ) \overset{\text{val}}{\to} \text{norm}\ r{\in}\mathbf{V}: c{\in}\mathbf{V}: C{\in}\mathbf{V}:$
$\text{compile3}\ (\ C\ ,\ c[r][\text{codex}][r]\ ,\ \mathsf{T}\ )]$

Compile the codex part of the cache c. C is the compiled cache (see above).

$[\text{compile3}\ (\ C\ ,\ x\ ,\ X\ ) \overset{\text{val}}{\to} \text{norm}\ C{\in}\mathbf{V}: x{\in}\mathbf{V}: X{\in}\mathbf{V}:$
if x **then** X **else**

if $x^{\mathrm{h}} \in \mathbf{Z}$ **then** $X[x^{\mathrm{h}} \rightarrow$compile4 (x^{t} , C)] **else**
compile3 (C , x^{h} , compile3 (C , x^{t} , X))]

Compile the codex part x of the cache c. C is the compiled cache (see above) and the result is accumulated in X.

[compile4 (x , C) $\stackrel{\mathrm{val}}{\rightarrow}$ norm $x \in \mathbf{V}$: $C \in \mathbf{V}$:
let $x[0]$[value] **be** d **in**
if d **then** $\top$ **else**
if not $d \in \mathbf{P}$ **or not** $d^{\mathrm{h}} \in \mathbf{P}$ **then** $\bullet$ **else**
if $d^{\mathrm{r}} \neq 0$ **then**
map ($\lambda d.\lambda C.$compile-code (d , C) Tail ' $\top$) " d^{M} " C **else**
if not $d^{\mathrm{ht}} \in \mathbf{P}$ **then** $\bullet$ **else**
let d^{hth} **be** i **in**
if $i =$ lambda **then** 0 **else**
if $i =$ quote **then** 1 **else**
if $i =$ true **then** $\top^{\mathrm{M}}$ **else**
if $i =$ apply **then** map ($\lambda x.\lambda y.x$ ' y) **else**
if $i =$ if **then** map ($\lambda x.\lambda y.\lambda z.$**If** x **then** y **else** z) **else** $\bullet$]

Compile the codex branch x by looking up the value definition d in it (if any) and compile d.

5.3 Code constructors

[make-constant (v) $\stackrel{\mathrm{val}}{\rightarrow}$ norm $v \in \mathbf{V}$: map ($\lambda v.\lambda s.v$) " v^{M}]

Convert the value v into an uncurried function which returns v. s is the "stack", i.e. the list of values of all bound variables.

[deBruijn (v , a , b) $\stackrel{\mathrm{val}}{\rightarrow}$ norm $v \in \mathbf{V}$: $a \in \mathbf{V}$: $b \in \mathbf{V}$:
if a **then** deBruijn1 (v , b) **else**
if not $a \in \mathbf{P}$ **then** $\bullet$ **else**
if $v \stackrel{\mathrm{t}}{=} a^{\mathrm{h}}$ **then** 0 **else** $1 +$ deBruijn (v , a^{t} , b)]

[deBruijn1 (v , b) $\stackrel{\mathrm{val}}{\rightarrow}$ norm $v \in \mathbf{V}$: $b \in \mathbf{V}$:
if not $b \in \mathbf{P}$ **then** $\bullet$ **else**
if $v \stackrel{\mathrm{t}}{=} b^{\mathrm{h}}$ **then** deBruijn2 (b^{t}) **else** deBruijn1 (v , b^{t})]

[deBruijn2 (b) $\stackrel{\mathrm{val}}{\rightarrow}$ norm $b \in \mathbf{V}$:
if b **then** 0 **else if** $b \in \mathbf{P}$ **then** $1 +$ deBruijn2 (b^{t}) **else** $\bullet$]

Find the position of the variable v in the variable lists a and b. Throw exception if the variable is not found. The list a contains variables bound by lambdas in deBruijn index order. The list b contains arguments in order of appearance which is opposite to the deBruijn index order.

[lazy-nth (i , s) $\stackrel{\mathrm{val}}{\rightarrow}$ **If** $i = 0$ **then** s Head **else** lazy-nth ($i - 1$, s Tail)]

[make-variable (v , a , b) $\stackrel{\text{val}}{\rightarrow}$ norm $v{\in}\mathbf{V}$: $a{\in}\mathbf{V}$: $b{\in}\mathbf{V}$:
let deBruijn (v , a , b)$^{\circ\text{t}}$ **be** i **in**
if i **then** make-constant ($\top$) **else**
map ($\lambda i.\lambda s.$lazy-nth (i , s)) " i^{M}]

Convert the variable v into an uncurried function which returns v. a is the argument list, i.e. the list of names of all bound variables. make-variable looks up the deBruijn index of v in a and b and returns an accessor for that index.

[make-lambda (f) $\stackrel{\text{val}}{\rightarrow}$ norm $f{\in}\mathbf{V}$:
map ($\lambda f.\lambda s.\lambda x.f$ ' (x LazyPair s)) " f]

Add lambda abstraction to the uncurried function f. s is the stack and x is the lambda variable to be pushed on the stack.

[make-lambdas (a , f) $\stackrel{\text{val}}{\rightarrow}$ norm $a{\in}\mathbf{V}$: $f{\in}\mathbf{V}$:
if not $a \in \mathbf{P}$ **then** f **else**
make-lambda (make-lambdas (a^{t} , f))]

Add one lambda per argument in the argument list a.

5.4 Compilation of individual constructs

[compile-code (d , c) $\stackrel{\text{val}}{\rightarrow}$ norm $d{\in}\mathbf{V}$: $c{\in}\mathbf{V}$:
if not $d^{\text{ttt}} \in \mathbf{P}$ **or not** d^{tttt} **or not** $d^2 \in \mathbf{P}$ **then** $\bullet$ **else**
compile-code1 ($d^{2\text{t}}$, d^3 , c)]

[compile-code1 (b , t , c) $\stackrel{\text{val}}{\rightarrow}$ norm $b{\in}\mathbf{V}$: $t{\in}\mathbf{V}$: $c{\in}\mathbf{V}$:
make-lambdas (b , compile-code2 ($\top$, b , t , c))]

Compile t and add one lambda per argument.

[compile-code2 (a , b , t , c) $\stackrel{\text{val}}{\rightarrow}$ norm $a{\in}\mathbf{V}$: $b{\in}\mathbf{V}$: $t{\in}\mathbf{V}$: $c{\in}\mathbf{V}$:
if not $t \in \mathbf{P}$ **or not** $t^{\text{h}} \in \mathbf{P}$ **or not** $t^{\text{ht}} \in \mathbf{P}$ **then** $\bullet$ **else**
let t^{r} **be** r **in**
let t^{i} **be** i **in**
if not $r \in \mathbf{Z}$ **or not** $i \in \mathbf{Z}$ **then** $\bullet$ **else**
if $r = 0$ **then** make-constant (i) **else**
if $i = 0$ **then if** $r = c[0]$ **then** make-constant (c) **else**
make-constant ($c[c[0]]$[cluster]$[r]$) **else**
let $c[r]$[code]$[i]$ **be** f **in**
if $f \in \mathbf{M}$ **then**
compile-code2$*$ (map ($\lambda f.\lambda s.f$) " f , a , b , t^{t} , c) **else**
if $f = \top$ **then** make-variable (t , a , b) **else**
if $f = 1$ **then** make-constant (t^1) **else**
make-lambda (compile-code2 ($t^1 :: a$, b , t^2 , c))]

Compile the right hand side t of the definition being compiled in the environment defined by the association list s. First set r and i to

the reference and index, respectively, of the root of t. If $r = 0$ then t is a string and i is the value to be returned. If i is zero then t is a page symbol whose value is either c or can be looked up in the cache branch. Otherwise look up the definition f of the root of t.

If f is a map then apply it to compiled versions of the subterms of t. If f is T then t is a variable to be looked up in the environment s. If f is 1 then t is a quote construct whose sole argument is returned. Otherwise, t is a lambda construct.

[compile-code2∗ (f , a , b , t , c) $\stackrel{\text{val}}{\to}$ norm $f{\in}\mathbf{V}$: $a{\in}\mathbf{V}$: $b{\in}\mathbf{V}$: $t{\in}\mathbf{V}$: $c{\in}\mathbf{V}$:
if t **then** f **else**
compile-code2∗ (map ($\lambda f.\lambda x.\lambda s.f$ ' s ' (x ' s)) " f "
compile-code2 (a , b , t^{h} , c) , a , b , t^{t} , c)]

Apply the function f to compiled versions of the elements of the list t of terms.

5.5 Claim evaluation

[prune (t , C) $\stackrel{\text{val}}{\to}$ norm $t{\in}\mathbf{V}$: $C{\in}\mathbf{V}$: **if** t **then** T **else** prune1 (t , C)]

[prune1 (t , C) $\stackrel{\text{val}}{\to}$ norm $t{\in}\mathbf{V}$: $C{\in}\mathbf{V}$:
let $(C[0]::0)::\mathsf{T}$ **be** b **in**
if not $t \in \mathbf{P}$ **or not** $t^{\mathrm{h}} \in \mathbf{P}$ **or not** $t^{\mathrm{ht}} \in \mathbf{P}$ **then** b **else**
let t^{r} **be** r **in**
let t^{i} **be** i **in**
if not $r \in \mathbf{Z}$ **or not** $i \in \mathbf{Z}$ **then** b **else**
if $r = 0$ **then if** t^{t} **then** t **else** b **else**
if $i = 0$ **then if** t^{t} **then** t **else** b **else**
let $C[r]$[dictionary][i] **be** a **in**
if not $a \in \mathbf{Z}$ **or** $a < 0$ **then** b **else**
let prune∗ (a , t^{t} , C)$^{\circ}$ **be** v **in**
if v^{h} **then** b **else** $t^{\mathrm{h}}::v^{\mathrm{t}}$]

[prune∗ (a , t , C) $\stackrel{\text{val}}{\to}$ norm $a{\in}\mathbf{V}$: $t{\in}\mathbf{V}$: $C{\in}\mathbf{V}$:
if $a = 0$ **then if** t **then** T **else** $\bullet$ **else**
if not $t \in \mathbf{P}$ **then** $\bullet$ **else**
prune1 (t^{h} , C)::prune∗ ($a - 1$, t^{t} , C)]

[eval-claim $\stackrel{\text{val}}{\to}$ norm map ($\lambda d.\lambda C$.prune ((eval (d^3 , T , C) " C^{M})^{U} , C))]

[compile-claim (r , c , C) $\stackrel{\text{val}}{\to}$ norm $r{\in}\mathbf{V}$: $c{\in}\mathbf{V}$: $C{\in}\mathbf{V}$:
let $c[r]$[codex][r][0][0][claim] **be** d **in**
if $d \in \mathbf{P}$ **then** eval-claim " d^{M} " C **else**
let $c[r]$[bibliography]^1 **be** r **in if** r **then** T^{M} **else**
let $c[r]$[codex][r][0][0][claim] **be** d **in**
if $d \in \mathbf{P}$ **then** eval-claim " d^{M} " C **else** T^{M}]

6 Macro expansion

6.1 Macro expansion engine

[base $\stackrel{\text{macro}}{\to}$ macro1]

[macro1 $\stackrel{\text{val}}{\to} \lambda c.$macro2 (c)]

[macro2 (c) $\stackrel{\text{val}}{\to}$ norm $c{\in}\mathbf{V}$:
macro3 ($c[c[0]]$[body] , macrostate0 , c)]

[macro3 (t , s , c) $\stackrel{\text{val}}{\to}$ norm $t{\in}\mathbf{V}$: $s{\in}\mathbf{V}$: $c{\in}\mathbf{V}$:
let t^{r} **be** r **in**
let t^{i} **be** i **in**
if $r = 0$ **then** t **else**
if $i = 0$ **then** t **else**
let $c[r]$[codex]$[r][i][0]$[macro] **be** d **in**
if $d = \top$ **then** t^{h} :: macro3∗ (t^{t} , s , c) **else**
(eval (d^3 , $\top$, c) " $(t :: s :: c :: \top)^{\text{M}})^{\text{U}}$]

[macro3∗ (t , s , c) $\stackrel{\text{val}}{\to}$ norm $t{\in}\mathbf{V}$: $s{\in}\mathbf{V}$: $c{\in}\mathbf{V}$:
if $t \in \mathbf{A}$ **then** $\top$ **else** macro3 (t^{h} , s , c) :: macro3∗ (t^{t} , s , c)]

[macro4 (x) $\stackrel{\text{val}}{\to}$ norm $x{\in}\mathbf{V}$: macro3 (x^0 , x^1 , x^2)]

[macrostate0 $\stackrel{\text{val}}{\to}$ norm map ($\lambda x.$macro4 (x)) :: $\top$]

[stateexpand (t , s , c) $\stackrel{\text{val}}{\to}$ norm $t{\in}\mathbf{V}$: $s{\in}\mathbf{V}$: $c{\in}\mathbf{V}$: $(s^{\text{h}}$ " $(t :: s :: c :: \top)^{\text{M}})^{\text{U}}$]

[stateexpand∗ (t , s , c) $\stackrel{\text{val}}{\to}$ norm $t{\in}\mathbf{V}$: $s{\in}\mathbf{V}$: $c{\in}\mathbf{V}$:
if $t \in \mathbf{A}$ **then** $\top$ **else**
stateexpand (t^{h} , s , c) :: stateexpand∗ (t^{t} , s , c)]

6.2 Backquoting

[substitute (r , t , s) $\stackrel{\text{val}}{\to}$ norm $r{\in}\mathbf{V}$: $t{\in}\mathbf{V}$: $s{\in}\mathbf{V}$:
let lookup (t , s , $\top$) **be** d **in**
if $d = \top$ **then** $(t^{\text{r}} :: t^{\text{i}} :: r^{\text{d}})$ :: substitute∗ (r , t^{t} , s) **else** d]
Replace subterms of t according to the stack s. Set the debugging information of nodes taken from t to the debugging information of the root of r

[substitute∗ (r , t , s) $\stackrel{\text{val}}{\to}$ norm $r{\in}\mathbf{V}$: $t{\in}\mathbf{V}$: $s{\in}\mathbf{V}$:
if $t \in \mathbf{A}$ **then** $\top$ **else**
substitute (r , t^{h} , s) :: substitute∗ (r , t^{t} , s)] Coordinatwise application of substitute to the elements of the list t.

$[\text{expand}\ (\ d\ ,\ x\ ) \overset{\text{val}}{\rightarrow} \text{norm}\ d{\in}\mathbf{V}{:}\, x{\in}\mathbf{V}{:}$
$\quad$ **let** x^0 **be** t **in let** zip $(\ d^{1\mathrm{t}}\ ,\ t^{\mathrm{t}}\ )$ **be** s **in**
$\quad$ stateexpand (substitute (t , d^2 , s) , x^1 , x^2)]

x is supposed to have form $t :: s :: c :: \mathsf{T}$. Expand the term t accoding the the macro definition d, then macro expand the result as specified by x. The macro definition is supposed to have form $[x \ddot{=} y]$ where the root of x equals the root of t.

6.3 Elementary macros

$\text{protect}([\text{protect}(u) \overset{\text{macro}}{\rightarrow} \lambda x.x^{01}])$

The protect(x) construct protects x against macro expansion. During macro expansion, the protection construct itself disappears. Note that the protection construct is used to protect the definition itself. Otherwise the left hand side of the definition would be macro expanded!

$\text{protect}([[u \overset{\circ}{=} v] \overset{\text{macro}}{\rightarrow} \lambda x.\text{Macrodefine}\ (\ x\ )])$

The $[u \overset{\circ}{=} v]$ construct macro expands into $\text{protect}([u \overset{\text{macro}}{\rightarrow} v])$. Thus it allows to state a macro definition which is itself protected against macro definition.

$[\text{Macrodefine}\ (\ x\ ) \overset{\text{val}}{\rightarrow} \text{norm}\ x{\in}\mathbf{V}{:}$
$\quad$ **if** $x = \mathsf{T}$ **then** $\lceil \mathsf{T} \rceil$ **else**
$\quad$ **let** x^0 **be** t **in**
$\quad$ **let** t^1 **be** u **in let** t^2 **be** v **in**
$\quad$ **let** $(\lceil u \rceil :: u) :: (\lceil v \rceil :: v) :: \mathsf{T}$ **be** s **in**
$\quad$ substitute (t , $\lceil \text{protect}([u \overset{\text{macro}}{\rightarrow} v]) \rceil$, s)]

This function performs the actual work of the $[u \overset{\circ}{=} v]$ macro. If x equals T then we are probably at a stage of codification where the argument has not yet get a sensible value. In this case just return a valid term. Otherwise, the argument x is supposed to have form $t :: s :: c :: \mathsf{T}$ where t is the term to be expanded, s is the macro state, and c is the cache of the page on which that term occurs. Macrodefine (x) does not need the state and cache but extracts the term t from x. The term has form $[u \overset{\circ}{=} v]$ where u and v are the actual left and right hand sides of the macro definition. Macrodefine (x) extracts these two terms end forms a stack s which maps the variables u and v to their respective values. Then Macrodefine (x) calls substitute (t , $\lceil \cdots \rceil$, s) to construct $\text{protect}([u \overset{\text{macro}}{\rightarrow} v])$. The debugging information of the latter is set to the debugging information of the original term t so that one can locate where the macro expanded term came from.

$[[u \ddot{=} v] \overset{\circ}{=} \lambda x.\text{macrodefine}\ (\ x\ )]$

This construct allows to state simple macro definitions in which a left hand side is macro expanded into a right hand side by simple substitution.

[macrodefine (x) $\stackrel{\text{val}}{\rightarrow}$ norm $x{\in}\mathbf{V}$:
let x^0 **be** t **in**
let t^1 **be** u **in**
let $(\lceil t \rceil :: t) :: (\lceil u \rceil :: u) :: \top$ **be** s **in**
substitute (t , $\lceil [u \stackrel{\text{macro}}{\rightarrow} \lambda x.\text{expand} (\lceil t \rceil , x)] \rceil$, s)]

This construct is similar to Macrodefine (x).

[self $\stackrel{\circ}{=}$ λx.makeself (x)] The self construct expands into the page symbol of the page on which the self construct occurs.

[makeself (x) $\stackrel{\text{val}}{\rightarrow}$ norm $x{\in}\mathbf{V}$:
let x^0 **be** t **in let** x^2 **be** c **in**
let t^{d} **be** d **in let** $c[0]$ **be** r **in**
$((r :: 0 :: d) :: \top)$]

The makeself (x) extracts the term t of form self as well as the cache c of the page on which self occurs. Then makeself (x) extracts the debugging information d from t and extracts the reference r of the page on which self occurs. Finally, makeself (x) constructs a tree with reference r, index 0, debugging information d, and no subtrees.

[root protect(u) $\stackrel{\circ}{=}$ λx.rootprotect (x)]

The root protect(u) construct protects the root of the term u against macro expansion but does allow the subtrees of u to be expanded.

[rootprotect (x) $\stackrel{\text{val}}{\rightarrow}$ norm $x{\in}\mathbf{V}$:
let x^0 **be** t **in let** t^1 **be** u **in**
let u^{r} **be** r **in let** u^{i} **be** i **in let** t^{d} **be** d **in**
$(r :: i :: d) ::$ stateexpand$*$ (u^{t} , x^1 , x^2)]

6.4 Definition macros

$[[x \stackrel{\text{render}}{=} y] \stackrel{..}{=} [\text{root protect}(x) \stackrel{\text{render!}}{\rightarrow} y]]$

Define the render aspect of a construct. The construct itself is protected against macro expansion.

$[[x \stackrel{\text{use}}{=} y] \stackrel{..}{=} [\text{root protect}(x) \stackrel{\text{use!}}{\rightarrow} y]]$

Define the tex use aspect of a construct. The construct itself is protected against macro expansion.

$[[x \stackrel{\text{show}}{=} y] \stackrel{..}{=} [\text{root protect}(x) \stackrel{\text{show!}}{\rightarrow} y]]$

$[[x \stackrel{.}{=} y] \stackrel{..}{=} [\text{root protect}(x) \stackrel{\text{val}}{\rightarrow} y]]$

$[[x \overset{\text{msg}}{=} y] \ddot{=} [\text{root protect}(x) \overset{\text{msg}}{\rightarrow} y]]$

$[[x \overset{\text{exec}}{=} y] \ddot{=} [\text{root protect}(x) \overset{\text{execute}}{\rightarrow} y]]$

$[\textbf{Priority table}\ x \ddot{=} [\text{self} \overset{\text{prio}}{\rightarrow} \text{protect}(x)]]$

Define the priority table of the page.

$[[\textbf{Verifier}{:}x] \ddot{=} [\text{self} \overset{\text{claim}}{\rightarrow} x]]$

Define the claim of the page.

$[[\textbf{Unpacker}{:}x] \ddot{=} [\text{self} \overset{\text{unpack}}{\rightarrow} x]]$

Define the unpacker of the page.

$[[\textbf{Renderer}{:}x] \ddot{=} [\text{self} \overset{\text{render}}{\rightarrow} x]]$

Define the renderer of the page.

$[[\textbf{Expander}{:}x] \ddot{=} [\text{self} \overset{\text{macro}}{\rightarrow} x]]$

Define the macro expansion engine of the page.

$\text{protect}([[x \overset{\text{use}}{=} y] \bowtie \text{‘hide’}])$

Ensure that the tex use definition construct can be used on the present page. If a tex use definition construct is used to define the tex use aspect of a revelation construct (e.g. a definition or proclamation construct) and if the tex use definition construct is not macro expanded (e.g. early in the codification process where the macro expander is not yet operational) then the hiding specified above ensures that the revelation construct in the left hand side of the tex use definition has no effect.

$\text{protect}([[x \overset{\text{show}}{=} y] \bowtie \text{‘hide’}])$

Ensure that the tex show definition construct can be used on the present page.

$\text{protect}([\textbf{Priority table}\ x \bowtie \text{‘hide’}])$

Ensure that the priority table construct can be used on the present page.

$\text{protect}([[x \ddot{=} y] \bowtie \text{‘hide’}])$

Ensure that macro definitions can expand into definitions without causing trouble at early stages of codification.

6.5 Parentheses

[$(x) \ddot{=} x$]

Ordinary parentheses.

[newline $x \ddot{=} x$]

Put newline before x. The construct has charge 32.

[tight newline $x \ddot{=} x$]

Put newline before x. The construct has charge 2.

[include (x) $\overset{\bullet\bullet}{=} x$]

This has value x but renders x as an elipsis.

[x tight endline $\ddot{=} x$]

Put newline after x. The construct has charge 4.

[x endline $\ddot{=} x$]

Put newline after x. The construct has charge 50.

6.6 Numerals

As mentioned in Section 3.11, an integer like 123 can be expressed as an invisible zero followed by three suffix operators $x1$, $x2$, and $x3$. The source of 123 reads `%% %1 %2 %3` where `%%` is the invisible zero and `%1`, `%2`, and `%3` are the three suffix operators. This way of entering numbers is rather inconvenient, however.

It is more convenient to enter a number like 123 as `123` where `3` is a nulary operator and `1` and `2` are prefix operators. Macros which support that are given in the following. The macros defined in the following convert `123` into `%% %1 %2 %3`.

Seen in isolation, it would be much easier to treat `123` as a nulary operator `1` followed by suffix operators `2` and `3`. Doing so, however, is a waste of good names. As an example, it is convenient to have constructs named things like `myname` and `myname2` in Logiweb source files. If we introduced a suffix `2`, however, `myname2` could be interpreted either as the `myname` construct followed by a suffix `2` or as the `myname2` construct. This is why numerals are constructed by prefix operators and why the burden of reversing the digit lists is left to the macro expander.

[$0x \overset{\circ}{=} \lambda x.$numeral (x)]

[$1x \overset{\circ}{=} \lambda x.$numeral (x)]

[$2x \overset{\circ}{=} \lambda x.$numeral (x)]

[$3x \overset{\circ}{=} \lambda x.$numeral (x)]

$[4x \overset{\circ}{=} \lambda x.\text{numeral}\ (\ x\ )]$

$[5x \overset{\circ}{=} \lambda x.\text{numeral}\ (\ x\ )]$

$[6x \overset{\circ}{=} \lambda x.\text{numeral}\ (\ x\ )]$

$[7x \overset{\circ}{=} \lambda x.\text{numeral}\ (\ x\ )]$

$[8x \overset{\circ}{=} \lambda x.\text{numeral}\ (\ x\ )]$

$[9x \overset{\circ}{=} \lambda x.\text{numeral}\ (\ x\ )]$

$[\text{numeral}\ (\ x\ ) \overset{\text{val}}{\to} \text{norm}\ x{\in}\mathbf{V}{:}\ \text{num1}\ (\ x^0\ ,\ x^0\ ,\ \text{substitute}\ (\ x^0\ ,\ \lceil\rceil\ ,\ \mathsf{T}\ )\ )]$

$\text{protect}([\text{num1}\ (\ r\ ,\ t\ ,\ u\ ) \overset{\text{val}}{\to} \text{norm}\ r{\in}\mathbf{V}{:}\ t{\in}\mathbf{V}{:}\ u{\in}\mathbf{V}{:}$
$\quad \textbf{if}\ t^{\text{t}}\ \textbf{then}\ \text{num2}\ (\ r\ ,\ t\ ,\ u\ )\ \textbf{else}$
$\quad \textbf{let}\ (\lceil x\rceil :: u) :: \mathsf{T}\ \textbf{be}\ s\ \textbf{in}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 0x\rceil\ \textbf{then}\ \text{num1}\ (\ r\ ,\ t^1\ ,\ \text{substitute}\ (\ r\ ,\ \lceil x0\rceil\ ,\ s\ )\ )\ \textbf{else}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 1x\rceil\ \textbf{then}\ \text{num1}\ (\ r\ ,\ t^1\ ,\ \text{substitute}\ (\ r\ ,\ \lceil x1\rceil\ ,\ s\ )\ )\ \textbf{else}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 2x\rceil\ \textbf{then}\ \text{num1}\ (\ r\ ,\ t^1\ ,\ \text{substitute}\ (\ r\ ,\ \lceil x2\rceil\ ,\ s\ )\ )\ \textbf{else}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 3x\rceil\ \textbf{then}\ \text{num1}\ (\ r\ ,\ t^1\ ,\ \text{substitute}\ (\ r\ ,\ \lceil x3\rceil\ ,\ s\ )\ )\ \textbf{else}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 4x\rceil\ \textbf{then}\ \text{num1}\ (\ r\ ,\ t^1\ ,\ \text{substitute}\ (\ r\ ,\ \lceil x4\rceil\ ,\ s\ )\ )\ \textbf{else}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 5x\rceil\ \textbf{then}\ \text{num1}\ (\ r\ ,\ t^1\ ,\ \text{substitute}\ (\ r\ ,\ \lceil x5\rceil\ ,\ s\ )\ )\ \textbf{else}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 6x\rceil\ \textbf{then}\ \text{num1}\ (\ r\ ,\ t^1\ ,\ \text{substitute}\ (\ r\ ,\ \lceil x6\rceil\ ,\ s\ )\ )\ \textbf{else}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 7x\rceil\ \textbf{then}\ \text{num1}\ (\ r\ ,\ t^1\ ,\ \text{substitute}\ (\ r\ ,\ \lceil x7\rceil\ ,\ s\ )\ )\ \textbf{else}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 8x\rceil\ \textbf{then}\ \text{num1}\ (\ r\ ,\ t^1\ ,\ \text{substitute}\ (\ r\ ,\ \lceil x8\rceil\ ,\ s\ )\ )\ \textbf{else}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 9x\rceil\ \textbf{then}\ \text{num1}\ (\ r\ ,\ t^1\ ,\ \text{substitute}\ (\ r\ ,\ \lceil x9\rceil\ ,\ s\ )\ )\ \textbf{else}$
$\quad \text{substitute}\ (\ r\ ,\ \lceil\bullet\rceil\ ,\ \mathsf{T}\ )])$

$[\text{num2}\ (\ r\ ,\ t\ ,\ u\ ) \overset{\text{val}}{\to} \text{norm}\ r{\in}\mathbf{V}{:}\ t{\in}\mathbf{V}{:}\ u{\in}\mathbf{V}{:}$
$\quad \textbf{let}\ (\lceil x\rceil :: u) :: \mathsf{T}\ \textbf{be}\ s\ \textbf{in}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 0\rceil\ \textbf{then}\ \text{substitute}\ (\ r\ ,\ \lceil x0\rceil\ ,\ s\ )\ \textbf{else}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 1\rceil\ \textbf{then}\ \text{substitute}\ (\ r\ ,\ \lceil x1\rceil\ ,\ s\ )\ \textbf{else}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 2\rceil\ \textbf{then}\ \text{substitute}\ (\ r\ ,\ \lceil x2\rceil\ ,\ s\ )\ \textbf{else}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 3\rceil\ \textbf{then}\ \text{substitute}\ (\ r\ ,\ \lceil x3\rceil\ ,\ s\ )\ \textbf{else}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 4\rceil\ \textbf{then}\ \text{substitute}\ (\ r\ ,\ \lceil x4\rceil\ ,\ s\ )\ \textbf{else}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 5\rceil\ \textbf{then}\ \text{substitute}\ (\ r\ ,\ \lceil x5\rceil\ ,\ s\ )\ \textbf{else}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 6\rceil\ \textbf{then}\ \text{substitute}\ (\ r\ ,\ \lceil x6\rceil\ ,\ s\ )\ \textbf{else}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 7\rceil\ \textbf{then}\ \text{substitute}\ (\ r\ ,\ \lceil x7\rceil\ ,\ s\ )\ \textbf{else}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 8\rceil\ \textbf{then}\ \text{substitute}\ (\ r\ ,\ \lceil x8\rceil\ ,\ s\ )\ \textbf{else}$
$\quad \textbf{if}\ t \overset{\text{r}}{=} \lceil 9\rceil\ \textbf{then}\ \text{substitute}\ (\ r\ ,\ \lceil x9\rceil\ ,\ s\ )\ \textbf{else}$
$\quad \text{substitute}\ (\ r\ ,\ \lceil\bullet\rceil\ ,\ \mathsf{T}\ )]$

6.7 Formating of the expansion

[make macro expanded version ragged right $\ddot{=}$]

6.8 Visibility constructs

The following constructs affect *visibility*:

show(x) renders the constructs in x using the tex show aspect.

!x: Save as show(x) but shorter.

macroshow(x) is similar to show(x) but also macro expands like parentheses so that it disappears in the expansion of the page.

hide(x) hides x from harvesting, c.f. Section 1.2.

hideshow(x) renders the constructs in x using the tex show aspect, protects x from macro expansion, and and hides x from harvesting.

All constructs above are themselves invisible when rendered using the tex use aspect. They are visible above because they are all enclosed in a show(x) construct. The properties of the constructs are declared in the following:

$[\text{show}(x) \overset{\text{val}}{\rightarrow\rightarrow} x]$

$[!x \overset{\text{val}}{\rightarrow\rightarrow} x]$

$[\text{macroshow}(x) \ddot{=} x]$

$[\text{hide}(x) \overset{\text{val}}{\rightarrow\rightarrow} x]$

$[\text{hide}(x) \bowtie \text{`hide'}]$

$[\text{hideshow}(x) \overset{\text{val}}{\rightarrow\rightarrow} x]$

$[\text{hideshow}(x) \ddot{=} \text{protect}(x)]$

$[\text{hideshow}(x) \bowtie \text{`hide'}]$

6.9 Tuples

$[\langle\,\rangle \doteq \text{norm } \mathsf{T}]$

$[\langle u\rangle \overset{\circ}{=} \lambda x.\text{tuple1 } (\ x\ )]$

$[\text{tuple1 } (\ x\ ) \overset{\text{val}}{\rightarrow} \text{norm } x{\in}\mathbf{V}\colon \text{stateexpand } (\ \text{tuple2 } (\ x^0\ ,\ x^{01}\ )\ ,\ x^1\ ,\ x^2\ )]$

$[\text{tuple2 } (\ t\ ,\ u\ ) \overset{\text{val}}{\rightarrow} \text{norm } t{\in}\mathbf{V}\colon u{\in}\mathbf{V}\colon$
$\quad$ **if not** $u \overset{\text{r}}{=} \lceil x, y\rceil$ **then**
$\quad$ substitute $(\ t\ ,\ \lceil u::\langle\,\rangle\rceil\ ,\ (\lceil u\rceil :: u) :: \mathsf{T}\ )$ **else**
$\quad$ substitute $(\ t\ ,\ \lceil x::y\rceil\ ,\ (\lceil x\rceil :: u^1) ::(\lceil y\rceil :: \text{tuple2 } (\ t\ ,\ u^2\ )) :: \mathsf{T}\ )]$

6.10 Eager definitions

$[[u \overset{\bullet}{=} v] \overset{\circ}{=} \lambda x.\text{eager1 } (\ x \ )]$

Construct of eager definitions.

$[\text{eager1 } (\ x \ ) \overset{\text{val}}{\to} \text{norm } x{\in}\mathbf{V}:$
$\mathbf{let}\ x^0\ \mathbf{be}\ t\ \mathbf{in}$
$\mathbf{let}\ t^1\ \mathbf{be}\ u\ \mathbf{in}\ \mathbf{let}\ t^2\ \mathbf{be}\ v\ \mathbf{in}$
$\mathbf{let}\ (\lceil u \rceil :: u) :: (\lceil w \rceil :: \text{eager2 } (\ t \ , \ u^{\text{t}} \ , \ v \ )) :: \mathsf{T}\ \mathbf{be}\ s\ \mathbf{in}$
$\text{stateexpand } (\text{ substitute } (\ t \ , \ \lceil [u \overset{\text{val}}{\to} \text{norm } w] \rceil \ , \ s \ ) \ , \ x^1 \ , \ x^2 \ )]$

$[\text{eager2 } (\ t \ , \ p \ , \ v \ ) \overset{\text{val}}{\to} \text{norm } t{\in}\mathbf{V}: p{\in}\mathbf{V}: v{\in}\mathbf{V}:$
$\mathbf{if}\ p \in \mathbf{A}\ \mathbf{then}\ v\ \mathbf{else}$
$\mathbf{let}\ (\lceil x \rceil :: p^{\text{h}}) :: (\lceil v \rceil :: \text{eager2 } (\ t \ , \ p^{\text{t}} \ , \ v \ )) :: \mathsf{T}\ \mathbf{be}\ s\ \mathbf{in}$
$\text{substitute } (\ t \ , \ \lceil x{\in}\mathbf{V}: v \rceil \ , \ s \ )]$

$[[\text{message } x : y] \ddot{=} [x \overset{\text{msg}}{=} y][x \overset{\bullet}{=} y]]$

6.11 Late definitions

The following constructs allow to state "late definitions" which only work after macros are in place. Useful only on the present page to avoid spurious warnings about mistakenly unfit functions or unrecognized optimized functions.

$[[u \overset{\bullet\bullet}{=} v] \ddot{=} [u \overset{\bullet}{=} v]]$

[late unhide x end unhide $\bowtie$ 'hide']

[late unhide x end unhide $\ddot{=}$ x]

$[[u \overset{\text{Late}}{\to\to} v] \ddot{=} \text{late unhide } [u \overset{\text{val}}{\to\to} v] \text{ end unhide}]$

6.12 Let constructs

We now define two let constructs. The first one allows to make a local macro definition. The second one allows to bind a pattern to a value. When binding a pattern to a value, the value is destructured according to the pattern and each variable in the pattern is bound to a component of the value. Destructuring is user definable. Below, we merely define how a pair is destructured into head and tail.

$[\mathbf{let}\ u \ddot{=} v\ \mathbf{in}\ w \overset{\circ}{=} \lambda x.\text{macrolet1 } (\ x \ )]$

The construct above locally defines u to macro expand to v inside w.

$[\text{macrolet1}\ (\ x\ ) \overset{\text{val}}{\to} \text{norm}\ x{\in}\mathbf{V}$:
$\quad$ **let** x^0 **be** t **in let** x^1 **be** s **in let** x^2 **be** c **in**
$\quad$ **let** t^1 **be** u **in let** t^3 **be** w **in**
$\quad$ **let** u^{r} **be** r **in let** u^{i} **be** i **in**
$\quad$ **let** $c[r::\text{codex}::r::i::0::\text{macro}::\mathsf{T}{\Rightarrow}\text{macrodefine}\ (\ x\ )]$ **be** c **in**
$\quad$ stateexpand $(\ w\ ,\ s\ ,\ c\ )]$

The function above implements local macro definitions.

$[\text{dest} \overset{\text{msg}}{\to} \text{destructure}]$

The destructure aspect allows to define how a value is destructured by a particular pattern. We only define destructuring of pairs in the following.

$[u::v \overset{\text{dest}}{\to} (\textbf{if}\ *\in\mathbf{A}\ \textbf{then}\ *\ \textbf{else}\ *^{\mathrm{h}})::(\textbf{if}\ *\in\mathbf{A}\ \textbf{then}\ *\ \textbf{else}\ *^{\mathrm{t}})::\mathsf{T}]$

The definition above describes how a pair is destructured into head and tail. If a non-pair x is destructured as a pair then both the head and the tail part is bound to x. In the definition, $*$ denotes the value to be destructured.

$[[x \overset{\text{dest}}{=} y] \ddot{=} [\text{root protect}(x) \overset{\text{dest}}{\to} y]]$

The macro above allows to define destructuring properties of a construct. We do not use the macro on the present page, however, to avoid problems with late definitions, c.f. Section 6.10.

$[\textbf{let}\ u = v\ \textbf{in}\ w \overset{\circ}{=} \lambda x.\text{let1}\ (\ x\ )]$

$[\text{let1}\ (\ x\ ) \overset{\text{val}}{\to} \text{norm}\ x{\in}\mathbf{V}$:
$\quad$ **let** x^0 **be** t **in let** x^1 **be** s **in let** x^2 **be** c **in**
$\quad$ **let** t^1 **be** u **in let** t^2 **be** v **in let** t^3 **be** w **in**
$\quad$ **let** stateexpand $(\ u\ ,\ s\ ,\ c\ )$ **be** u **in**
$\quad$ **let** stateexpand $(\ v\ ,\ s\ ,\ c\ )$ **be** v **in**
$\quad$ **let** stateexpand $(\ w\ ,\ s\ ,\ c\ )$ **be** w **in**
$\quad$ **let** make-prime $(\ \text{make-var}\ (\ t\ )\ )$ **be** x **in**
$\quad$ **let** let2 $(\ x\ ,\ u\ ,\ w\ ,\ c\ )$ **be** w **in**
$\quad$ make-let $(\ \lceil * \rceil\ ,\ v\ ,\ w\ )]$

$[\text{let2}\ (\ x\ ,\ p\ ,\ w\ ,\ c\ ) \overset{\text{val}}{\to} \text{norm}\ x{\in}\mathbf{V}: p{\in}\mathbf{V}: w{\in}\mathbf{V}: c{\in}\mathbf{V}$:
$\quad$ **let** p^{r} **be** r **in let** p^{i} **be** i **in**
$\quad$ **let** $c[r][\text{codex}][r][i][0]$ **be** c' **in**
$\quad$ **let** $c'[\text{destructure}]$ **be** d **in**
$\quad$ **if** $d = \mathsf{T}$ **then**
$\quad$ **if** $c'[\text{value}] \neq \mathsf{T}$ **then** w **else** make-let $(\ p\ ,\ \lceil * \rceil\ ,\ w\ )$ **else**
$\quad$ **let** let3 $(\ x\ ,\ p^{\mathrm{t}}\ ,\ w\ ,\ c\ )$ **be** w **in**
$\quad$ make-let $(\ x\ ,\ d^3\ ,\ w\ )]$

$[\text{let3}\ (\ x\ ,\ p\ \text{prime}\ ,\ w\ ,\ c\ ) \overset{\text{val}}{\to} \text{norm}\ x{\in}\mathbf{V}{:}\ p'{\in}\mathbf{V}{:}\ w{\in}\mathbf{V}{:}\ c{\in}\mathbf{V}{:}$
$\quad \textbf{if}\ p' = \mathsf{T}\ \textbf{then}\ w\ \textbf{else}$
$\quad \textbf{let}\ p'^{\mathrm{h}}\ \textbf{be}\ p\ \textbf{in}\ \textbf{let}\ p'^{\mathrm{t}}\ \textbf{be}\ p'\ \textbf{in}$
$\quad \textbf{let}\ \text{let3}\ (\ x\ ,\ p'\ ,\ w\ ,\ c\ )\ \textbf{be}\ w\ \textbf{in}$
$\quad \textbf{let}\ \text{let2}\ (\ \text{make-prime}\ (\ x\ )\ ,\ p\ ,\ w\ ,\ c\ )\ \textbf{be}\ w\ \textbf{in}$
$\quad \textbf{let}\ \text{make-let}\ (\ x\ ,\ \text{make-tail}\ (\ x\ )\ ,\ w\ )\ \textbf{be}\ w\ \textbf{in}$
$\quad \text{make-let}\ (\ \lceil * \rceil\ ,\ \text{make-head}\ (\ x\ )\ ,\ w\ )]$

$[\text{make-var}\ (\ t\ ) \overset{\text{val}}{\to} \text{norm}\ t{\in}\mathbf{V}{:}$
$\quad \text{substitute}\ (\ t\ ,\ \lceil * \rceil\ ,\ \mathsf{T}\ )]$

$[\text{make-let}\ (\ u\ ,\ v\ ,\ w\ ) \overset{\text{val}}{\to} \text{norm}\ u{\in}\mathbf{V}{:}\ v{\in}\mathbf{V}{:}\ w{\in}\mathbf{V}{:}$
$\quad \textbf{let}\ (\lceil u \rceil :: u) :: (\lceil v \rceil :: v) :: (\lceil w \rceil :: w) :: \mathsf{T}\ \textbf{be}\ s\ \textbf{in}$
$\quad \text{substitute}\ (\ u\ ,\ \lceil \textbf{let}\ v\ \textbf{be}\ u\ \textbf{in}\ w \rceil\ ,\ s\ )]$

$[\text{make-prime}\ (\ x\ ) \overset{\text{val}}{\to} \text{norm}\ x{\in}\mathbf{V}{:}$
$\quad \textbf{let}\ (\lceil x \rceil :: x) :: \mathsf{T}\ \textbf{be}\ s\ \textbf{in}$
$\quad \text{substitute}\ (\ x\ ,\ \lceil x' \rceil\ ,\ s\ )]$

$[\text{make-head}\ (\ x\ ) \overset{\text{val}}{\to} \text{norm}\ x{\in}\mathbf{V}{:}$
$\quad \textbf{let}\ (\lceil x \rceil :: x) :: \mathsf{T}\ \textbf{be}\ s\ \textbf{in}$
$\quad \text{substitute}\ (\ x\ ,\ \lceil x^{\mathrm{h}} \rceil\ ,\ s\ )]$

$[\text{make-tail}\ (\ x\ ) \overset{\text{val}}{\to} \text{norm}\ x{\in}\mathbf{V}{:}$
$\quad \textbf{let}\ (\lceil x \rceil :: x) :: \mathsf{T}\ \textbf{be}\ s\ \textbf{in}$
$\quad \text{substitute}\ (\ x\ ,\ \lceil x^{\mathrm{t}} \rceil\ ,\ s\ )]$

6.13 Backquoting

$[\lceil_u\ v \rceil \overset{\circ}{=} \lambda x.\text{backquote0}\ (\ x\ )]$

$[\text{make-root}\ (\ t\ ,\ x\ ) \overset{\text{val}}{\to} \text{norm}\ t{\in}\mathbf{V}{:}\ x{\in}\mathbf{V}{:}\ x^{\mathrm{r}} :: x^{\mathrm{i}} :: t^{\mathrm{d}}]$

$[\text{make-pair}\ (\ t\ ,\ x\ ,\ y\ ) \overset{\text{val}}{\to} \text{norm}\ t{\in}\mathbf{V}{:}\ x{\in}\mathbf{V}{:}\ y{\in}\mathbf{V}{:}\ \text{make-root}\ (\ t\ ,\ \lceil \mathsf{T} :: \mathsf{T} \rceil\ ) :: x :: y :: \mathsf{T}]$

$[\text{make-true}\ (\ t\ ) \overset{\text{val}}{\to} \text{norm}\ t{\in}\mathbf{V}{:}\ \text{make-root}\ (\ t\ ,\ \lceil \mathsf{T} \rceil\ ) :: \mathsf{T}]$

$[\text{make-quote}\ (\ t\ ,\ x\ ) \overset{\text{val}}{\to} \text{norm}\ t{\in}\mathbf{V}{:}\ x{\in}\mathbf{V}{:}\ \text{make-root}\ (\ t\ ,\ \lceil \lceil \mathsf{T} \rceil \rceil\ ) :: x :: \mathsf{T}]$

$[\text{make-make-root}\ (\ t\ ,\ x\ ,\ y\ ) \overset{\text{val}}{\to} \text{norm}\ t{\in}\mathbf{V}{:}\ x{\in}\mathbf{V}{:}\ y{\in}\mathbf{V}{:}\ \text{make-root}\ (\ t\ ,\ \lceil \text{make-root}\ (\ \mathsf{T}\ ,\ \mathsf{T}\ ) \rceil\ ) :: x :: y :: \mathsf{T}]$

$[\text{backquote0}\ (\ x\ ) \overset{\text{val}}{\to} \text{norm}\ x{\in}\mathbf{V}{:}\ \text{backquote1}\ (\ x^0\ ,\ x^1\ ,\ x^2\ )]$

$[\text{backquote1}\ (\ t\ ,\ s\ ,\ c\ ) \overset{\text{val}}{\to} \text{norm}\ t{\in}\mathbf{V}{:}\ s{\in}\mathbf{V}{:}\ c{\in}\mathbf{V}{:}\ \text{backquote2}\ (\ t\ ,\ \text{stateexpand}\ (\ t^1\ ,\ s\ ,\ c\ )\ ,\ \text{stateexpand}\ (\ t^2\ ,\ s\ ,\ c\ )\ ,\ s\ ,\ c\ )]$

[backquote2 (t , u , v , s , c) $\stackrel{\text{val}}{\to}$ norm $t{\in}\mathbf{V}$: $u{\in}\mathbf{V}$: $v{\in}\mathbf{V}$: $s{\in}\mathbf{V}$: $c{\in}\mathbf{V}$:
if $v \stackrel{\text{r}}{=} \lceil \underline{\mathrm{T}} \rceil$ **then** v^1 **else**
make-pair (t , make-make-root (t , u , make-quote (t , v)) ,
backquote2∗ (t , u , v^{t} , s , c))]

[backquote2∗ (t , u , v , s , c) $\stackrel{\text{val}}{\to}$ norm $t{\in}\mathbf{V}$: $u{\in}\mathbf{V}$: $v{\in}\mathbf{V}$: $s{\in}\mathbf{V}$: $c{\in}\mathbf{V}$:
if $v \in \mathbf{A}$ **then** make-true (t) **else**
make-pair (t , backquote2 (t , u , v^{h} , s , c) , backquote2∗ (t , u , v^{t} , s , c))]

6.14 Rendering

We define a number of constructs for rendering. To see how most of them are used, consult the source of the present Logiweb page.

text x : y end text. Store the text y in a file named x.

latex (x). Run latex on the file named x.

bibtex (x). Run bibtex on the file named x.

makeindex (x). Run makeindex on the file named x.

dvipdfm (x). Run dvipdfm on the file named x.

tex-file (c , x , y). Equivalent to text x : y end text when c is the string "text".

tex-command (c , x). Equivalent to latex (x), bibtex (x), makeindex (x), and dvipdfm (x), respectively, when c is one of the strings "latex", "bibtex", "makeindex", and "dvipdfm".

page(x, y, t, b, m, a). A construct for defining a document with title t, bib-file b, main text m, and appendix a. The main text is processed by latex, bibtex, makeindex, latex, and latex. The appendix is processed by latex three times. x and y must be Name and Priority, respectively, where the Logiweb compiler expands the two into name and priority definitions, respectively.

Only the 'page' construct needs a definition here. All the other ones just have tex use definitions which are stated later.

[page(x, y, t, b, m, a) $\stackrel{\circ}{=} \lambda z$.page1 (z)]

[page1 (z) $\stackrel{\text{val}}{\to}$ norm $z{\in}\mathbf{V}$:
let z^0 **be** t **in**
let z^1 **be** s **in**
let z^2 **be** c **in**
let t^2 **be** y **in**

$\textbf{let}$ stateexpand (t^5 , s , c) $\textbf{be}$ m $\textbf{in}$
$\textbf{let}$ stateexpand (t^6 , s , c) $\textbf{be}$ a $\textbf{in}$
$\textbf{let}$ $\lceil$protect(page(x, y, t, b, m, a))$\rceil$ $\textbf{be}$ p $\textbf{in}$
$(p^{\mathrm{r}} :: p^{\mathrm{i}} :: t^{\mathrm{d}}) :: t^1 :: y :: t^3 :: t^4 :: m :: a :: \mathsf{T}]$

7 The Logiweb machine

A *Logiweb machine* is an abstract machine which consists of

- a Logiweb computing engine,
- an *interface*,
- a *boot handler*, and
- a *cache*.

The engine and interface comprise the 'hardware' of the abstract machine whereas the handler and cache comprise software which is preloaded on the machine.

Each Logiweb page may define one or more Logiweb machines. When a Logiweb page is rendered, all machines defined on the page (if any) are output as executable files. The cache of each machine equals the cache of the page defining the machine whereas each machine has its own, individual handler.

The interface of a Logiweb machine performs an input-eval-output-loop in which it converts input to a list of *input messages* and applies the handler to that input list. The return value is supposed to be a list of *output messages* which the interface executes. We shall refer to output messages as *requests*. We shall refer to input messages as *replies* when they are direct responses to requests and as *events* otherwise.

The interface has the ability to load code dynamically into the running machine. Such code can improve the efficiency of the engine and can add new abilities to the interface. A Logiweb machine is *virgin* from it starts and until first time it loads code. The virgin Logiweb machine has very few abilities.

7.1 Messages

The virgin Logiweb machine knows the following output events:

$[\text{quit request } (\ x \ ) \stackrel{\bullet\bullet}{=} \langle\langle 0, \text{quit}\rangle, x\rangle]$
$[\text{write request } (\ s \ ) \stackrel{\bullet\bullet}{=} \langle\langle 0, \text{write}\rangle, s\rangle]$
$[\text{read request} \stackrel{\bullet\bullet}{=} \langle\langle 0, \text{read}\rangle\rangle]$
$[\text{exec request } (\ p \ , \ h \ ) \stackrel{\bullet\bullet}{=} \langle\langle 0, \text{exec}\rangle, p, h\rangle]$
$[\text{extend request } (\ r \ , \ s \ ) \stackrel{\bullet\bullet}{=} \langle\langle 0, \text{extend}\rangle, r, s\rangle]$

And the following input messages:

$$[\text{boot event } (\ a\ ,\ e\ ,\ c\ ,\ s\ ) \stackrel{\bullet\bullet}{=} \langle\langle 0, \text{boot}\rangle, a, e, c, s\rangle]$$
$$[\text{read reply } (\ c\ ) \stackrel{\bullet\bullet}{=} \langle\langle 0, \text{read}\rangle, c\rangle]$$
$$[\text{exec reply } (\ i\ ,\ p\ ) \stackrel{\bullet\bullet}{=} \langle\langle 0, \text{exec}\rangle, i, p\rangle]$$
$$[\text{extend reply } (\ x\ ) \stackrel{\bullet\bullet}{=} \langle\langle 0, \text{extend}\rangle, x\rangle]$$

Each exec reply contains an "interrupt". The following interrupts are predefined:

$$[\text{exit interrupt} \stackrel{\bullet\bullet}{=} \langle\langle 0, \text{exit}\rangle\rangle]$$
$$[\text{time interrupt} \stackrel{\bullet\bullet}{=} \langle\langle 0, \text{time}\rangle\rangle]$$
$$[\text{memory interrupt} \stackrel{\bullet\bullet}{=} \langle\langle 0, \text{memory}\rangle\rangle]$$

7.2 Machine invocation

A user may start a Logiweb machine by typing its name followed by arguments on a command line. The machine then forms the term $t = h$ " $\langle$boot event (a , e , c , s)$\rangle$ where h is the boot handler, a is `argv`, e is the current environment, c is the cache, and s is system specific information in the same format as e. Then the machine enters an input-eval-output-loop.

7.3 Input-eval-output-loop

When the machine enters the input-eval-output-loop, the interface asks the engine to reduce t. The result of that is supposed to be a list of output messages. The interface then executes the output messages one at a time as follows:

A quit request (x) makes the machine exit immediately with return code x.

A write request (s) makes the interface write the strings in s to standard output. If s is a cardinal then the cardinal is interpretted as a list of bytes encoded little endian base 256. Bytes are written in little endian order, leaving any interpretation of the bytes the the receiver. If s has form $u :: v$ then the interface recursively outputs first u and then v. If s is neither a cardinal nor a pair then it is ignored.

A read request makes the interface read one byte from standard input. Usually, reading is blocking, but this may be system dependent. Each read request results in a read reply (c) input message. The value of c is a cardinal which represents the input byte as a one byte string. The value of c is the empty string if reading is non-blocking and no input is available. The value of c is zero if the end of standard input has been reached.

An extend request (r , s) makes the interface treat s as a piece of source code which the interface compiles and loads. The details are system dependent. Each extend request results in an extend reply (x) input message where the semantics of x is system dependent.

An exec request (p , h) makes the machine discard all messages following the Exec and asks the engine to reduce p. During execution of p there is an upper, system dependent limit on the time and memory the engine may use. Reduction of p may stop because of timeout, memory overflow, or external interupts. Reduction may also stop because the engine succeeds to reduce p. We shall refer to the latter situation as an *exit* interrupt so that reduction of p always ends with an interrupt. Then the interface forms the term

$$t = h \text{ " } \langle \text{exec reply } (\langle i, a_1, \ldots, a_m \rangle \ , \ p \ ), e_1, \ldots, e_n \rangle$$

where i identifies the interrupt, $a_1, \ldots, a_m$ are possible interrupt parameters, and $e_1, \ldots, e_n$ are input messages that have occurred since last. Then the interface reenters the input-eval-output-loop.

The predefined interrupts (exit, time, and memory) take no parameters. An exit interrupt indicates that p has been reduced. A time interrupt indicates that execution of p was stopped by a timer. A memory interrupt indicates that execution of p ran out of memory. As a minimum, Logiweb machines must support the exit interrupt.

On the virgin machine, the input messages comprise one extend reply for each extend request plus one read reply for each read request. Replies occur in the reverse order of their associated requests.

7.4 Handlers and processes

The exec request (p , h) event makes the machine evaluate the *process* p and, when that is interrupted, the *handler* h. Hence, h corresponds to an interrupt handler and p corresponds to a less privileged user, operating system, or driver process. The handler h has full control over the machine in that it receives all input, generates all output, and cannot be interrupted.

7.5 Character constants

We now define a number of often used character constants.

[NULL $\stackrel{\bullet\bullet}{=}$ bt2vector (0)]

[TAB $\stackrel{\bullet\bullet}{=}$ bt2vector (9)]

[LF $\stackrel{\bullet\bullet}{=}$ bt2vector (10)]

[FF $\stackrel{\bullet\bullet}{=}$ bt2vector (12)]

[CR $\stackrel{\bullet\bullet}{=}$ bt2vector (13)]

[SP $\stackrel{\bullet\bullet}{=}$ ‘ ’]

[QQ $\stackrel{\bullet\bullet}{=}$ ‘"’]

[V128 $\stackrel{\bullet\bullet}{=}$ bt2vector (128)]

[V255 $\stackrel{\bullet\bullet}{=}$ bt2vector (255)]

Some auxiliary definitions that use these constants read:

[CRLF $\stackrel{\bullet\bullet}{=}$ CR::LF]

ASCII Carriage Return followed by Linefeed.

[LFCR $\stackrel{\bullet\bullet}{=}$ LF::CR]

ASCII Linefeed followed by Carriage Return.

['' $\stackrel{\bullet\bullet}{=}$ vt2vector ($\top$)]

A convenience function for expressing the empty string.

[writeln request (x) $\stackrel{\bullet\bullet}{=}$ write request (x::LF)]

[println (x) $\stackrel{\bullet\bullet}{=}$ print (x::LF)]

7.6 Hello World

A machine with [Hello World $\stackrel{\bullet\bullet}{=}$ map ($\lambda x.\langle$writeln request (HelloWorld)$\rangle$)] as boot handler writes "Hello World" and exits. The example abuses the high privilege boot handler to do the writing. In the absence of Exec and Quit events, the machine quits with return code 0 after writing 'Hello World'.

7.7 Echo

A machine with Echo below as boot handler echos all characters typed except that it quits when the user types 'q':

[Echo $\stackrel{\bullet\bullet}{=}$ map (λx.Echo1 (x))]

[Echo1 (x) $\stackrel{\bullet\bullet}{=}$
if $x \in$ **A then**
$\langle$read request, exec request ($\top$, Echo)$\rangle$ **else**
let x^{h} **be** e **in**
let Echo1 (x^{t}) **be** r **in**
if not $e^{\mathrm{h}} = \langle 0, \mathrm{read}\rangle$ **then** r **else**
if $e^1 = \mathrm{q}$ **then** $\langle$writeln request ($\top$), quit request (0)$\rangle$ **else** write request (e^1)::r]

7.8 Eecho

A machine with Eecho below as boot handler echos all characters typed twice except that it quits when the user types 'q':

[Eecho $\overset{\bullet\bullet}{=}$ map (λx.Eecho1 (x))]

[Eecho1 (x) $\overset{\bullet\bullet}{=}$
if $x \in$ **A then**
$\langle$read request, exec request ($\top$, Eecho)$\rangle$ **else**
let x^{h} **be** e **in**
let Eecho1 (x^{t}) **be** r **in**
if not $e^{\mathrm{h}} = \langle 0, \mathrm{read} \rangle$ **then** r **else**
if $e^1 = \mathrm{q}$ **then** $\langle$writeln request ($\top$), quit request (0)$\rangle$ **else** write request ($e^1 :: e^1$) $:: r$]

7.9 Execution constructs

A definition like [‘eecho’ $\overset{\text{exec}}{=}$ $\langle$Eecho$\rangle$] makes the Logiweb compiler generate a machine named ‘eecho’ whose handler is Eecho.

After macro expansion, the principal operator of the right hand side of the definition must have arity at least two. The first subtree of the principal operator is used as handler. The system specific information s of the boot event is derived from the second subtree of the principal operator.

Actually, generation of machines is part of the rendering process which may be customized by the user. For more on this see the Logiweb page which defines the Logiweb compiler.

In some cases one may want a page to represent one and only one anonymous machine. In that case it is suggested to name that machine ‘main’ (so that it is not *really* anonymous). As an example, if one wants to construct a Logiweb browser which can display pages with dynamic behavior, then one could let the anonymous machine of the page define the behavior.

Each machine defined on a page needs its own definition:

[‘hello’ $\overset{\text{exec}}{=}$ $\langle$Hello World$\rangle$]

[‘echo’ $\overset{\text{exec}}{=}$ $\langle$Echo$\rangle$]

[‘lgciotest’ $\overset{\text{exec}}{=}$ $\langle$lgcio1$\rangle$]

8 The lgcio interface

The lgcio interface provides all the input/output facilities needed by the lgc compiler. These features are:

FileWrite Write given contents to given file. If the file does not exist, create it in mode 0666 (read/write access to everybody). For the typical value of umask (0022) this results in mode 0644 (read/write to user, read to everybody).

FileWriteExec Same as above, but using mode 0777 instead of mode 0666. Hence, created files become executable.

FileRead Read all contents of a given file.

FileRm Remove given file. If the file does not exist, do nothing.

FileSymlink Create symbolic link.

FileReadLink Read symbolic link. In contrast, a FileRead on a symbolic link reads the contents of the file pointed to by the symbolic link.

FileMkdir Create directories containing given path. Examples: Given path abc/def/ghi create abc/ and abc/def/. Given path abc/def/ghi/ create abc/, abc/def/, and abc/def/ghi/.

FileRmdir Remove given, empty directory. If the directory does not exist, do nothing.

FileDir Read all entries of given directory.

FileType Read the type of the given file (regular, directory, symbolic link, non-existent, or other).

FileTypeRead Combination of FileRead and FileType: Return $t :: c$ where t is the type of the given file (non-existent or regular) and c is the contents of the file (if any).

TextWrite Same as FileWrite, but changes newline sequences to given sequence.

TextWriteExec Same as FileWriteExec, but changes newline sequences to given sequence.

FileGetCwd Return current working directory.

UnixTime Read the clock.

Demonize Turn the running process into a demon (i.e. get rid of the controlling terminal). Redirect stdout and stderr. Optionally try to change user id. Optionally write the process id of the process after demonization to a file.

Execlp1 In a given directory, run a given command with one argument. The search path is used to locate the command.

TcpQuery Send given bytes to given port of given domain.

8.1 The lgcio request

The lgcio (c) request carries communication between the computing engine and the lgcio interface.

To load the lgcio interface and link it to the lgcio (c) request, place an event of form

extend request (lgcio (⊤) , lgcio-interface)

on the list of output events from the computing engine.

Afterwards, to invoke a feature of the lgcio interface, place an lgcio (c) request in the list of output events from the computing engine. That will make the lgcio interface execute the command encoded by c and place an lgcio (c) reply in the list of input events to the computing engine.

The command c of an lgcio (c) request is interpreted as a string tree. The response c of an lgcio (c) reply is a list of one element strings. Actually, this holds true for all requests and replies that are defined using extend events.

We define lgcio (c) such that $\lceil \text{lgcio} (c) \rceil \stackrel{\text{t}}{=} \text{lgcio} (\lceil c \rceil)$:

$$[\text{lgcio} (c) \stackrel{\bullet\bullet}{=} \langle \lceil \text{lgcio} (c) \rceil^{\text{h}}, c \rangle]$$

8.2 Constants used by the lgcio interface

Each feature provided by the lgcio interface is identified by a command identifier. A command identifier is a single byte. Those command identifiers are defined in the following. Additionally, null bytes are used as end markers for strings. For that reason we also define EOS to denote a null byte.

$[\text{EOS} \stackrel{\bullet\bullet}{=} \text{bt2vector} (0)]$

$[\text{FileWrite} \stackrel{\bullet\bullet}{=} \text{bt2vector} (1)]$

$[\text{FileWriteExec} \stackrel{\bullet\bullet}{=} \text{bt2vector} (2)]$

$[\text{FileRead} \stackrel{\bullet\bullet}{=} \text{bt2vector} (3)]$

$[\text{FileRm} \stackrel{\bullet\bullet}{=} \text{bt2vector} (4)]$

$[\text{FileSymlink} \stackrel{\bullet\bullet}{=} \text{bt2vector} (5)]$

$[\text{FileReadLink} \stackrel{\bullet\bullet}{=} \text{bt2vector} (6)]$

$[\text{FileMkdir} \stackrel{\bullet\bullet}{=} \text{bt2vector} (7)]$

$[\text{FileRmdir} \stackrel{\bullet\bullet}{=} \text{bt2vector} (8)]$

$[\text{FileDir} \stackrel{\bullet\bullet}{=} \text{bt2vector} (9)]$

$[\text{FileType} \stackrel{\bullet\bullet}{=} \text{bt2vector} (10)]$

$[\text{FileTypeRead} \stackrel{\bullet\bullet}{=} \text{bt2vector} (11)]$

$[\text{TextWrite} \stackrel{\bullet\bullet}{=} \text{bt2vector} (12)]$

$[\text{TextWriteExec} \stackrel{\bullet\bullet}{=} \text{bt2vector} (13)]$

$[\text{FileGetCwd} \stackrel{\bullet\bullet}{=} \text{bt2vector} (14)]$

$[\text{UnixTime} \stackrel{\bullet\bullet}{=} \text{bt2vector} (20)]$

[Demonize $\stackrel{\bullet\bullet}{=}$ bt2vector (30)]

[Execlp1 $\stackrel{\bullet\bullet}{=}$ bt2vector (31)]

[TcpQuery $\stackrel{\bullet\bullet}{=}$ bt2vector (40)]

The reply from a FileType request represents the type of the queried file thus:

[FileTypeNonexistent $\stackrel{\bullet\bullet}{=}$ bt2vector (0)]

[FileTypeOther $\stackrel{\bullet\bullet}{=}$ bt2vector (1)]

[FileTypeRegular $\stackrel{\bullet\bullet}{=}$ bt2vector (2)]

[FileTypeDirectory $\stackrel{\bullet\bullet}{=}$ bt2vector (3)]

[FileTypeLink $\stackrel{\bullet\bullet}{=}$ bt2vector (4)]

The reply from a FileTypeRead request is of form $t :: c$ where t is FileTypeNonexistent or FileTypeRegular and c is the contents of the file (if any).

8.3 Data conversion

The following functions allow to construct arguments of request and analyse arguments of replies.

[septet∗2card (r , s) $\stackrel{\bullet\bullet}{=}$
if $s \in$ **A then** r **else** septet∗2card (s^{h} + septet-base $\cdot$ r , s^{t})]

Convert the big endian list s of septets to a cardinal. Septets are numbers in the range from 0 to 127, inclusive. Lists of septets represent numbers base 128. The result is accumulated in r

[septet-base $\stackrel{\bullet\bullet}{=}$ 128]

[parse-card (v) $\stackrel{\bullet\bullet}{=}$ parse-card1 (T , v)]

Parse the cardinal at the beginning of the singleton list v and return $c :: v'$ where c is the cardinal and v' is the unparsed part of v. The cardinal is expressed as a little endian sequence of septets. Bytes b in the range from 128 to 255, inclusive, are treated as *middle septets*; they represent the septet $v - 128$ and also indicate that the given septet is not the last one in the cardinal. Bytes b in the range from 0 to 127 are treated as *end septets*; they represent the septet v and also indicate that the given septet is the last one in the cardinal.

[parse-card1 (r , v) $\stackrel{\bullet\bullet}{=}$
if $v \in$ **A then** $\bullet$ **else**
let $b :: v = v$ **in**
if $b <$ NULL **or** $b >$ V255 **then** $\bullet$ **else**
if $b <$ V128 **then** septet∗2card (0 , $b -$ NULL $:: r$) $:: v$ **else**
parse-card1 ($b -$ V128 $:: r$, v)]

[exp10 (e) $\stackrel{\bullet\bullet}{=}$ **if** $e = 0$ **then** 1 **else** $10 \cdot$ exp10 ($e - 1$)]

Raise 10 to power e.

[parse-unixTime (v) $\stackrel{\bullet\bullet}{=}$
let $f :: v =$ parse-card (v^1) **in**
let $s :: v =$ parse-card (v) **in**
let $e :: v =$ parse-card (v) **in**
$\langle s \cdot$ exp10 (e) $+ f, e \rangle$]

Parse the reply from unixTime of form lgcio (c) into a number of seconds s, a fraction of seconds f, and an exponent e. Those values represent the value $\mathrm{s} + \mathrm{f} \cdot 10^{-\mathrm{e}}$. Then return $m :: e$ such that $\mathrm{m} \cdot 10^{-\mathrm{e}} = \mathrm{s} + \mathrm{f} \cdot 10^{-\mathrm{e}}$.

[make-card (c) $\stackrel{\bullet\bullet}{=}$
if $c < 128$ **then** bt2vector (c) **else**
bt2vector ((c mod 128) + 128) :: make-card (c div 128)]

Convert the cardinal c into a little endian sequence of singletons base 128. The end byte (the last byte) is in the range from 0 to 127. The middle bytes (the other bytes) are represented offset 128.

8.4 Individual lgcio requests

Constructors for the individual commands understood by the lgcio interface are defined in the following.

[fileWrite (p , c) $\stackrel{\bullet\bullet}{=}$ lgcio (FileWrite $:: p ::$ EOS $:: c$)]

Write the contents c to a file with pathname p. If the file does not exist, create it with file permissions 666 which assigns read and write access to everybody. Most users will have a file-creation mask (umask) which restricts write access for others than the user. The reply has form lgcio ($\langle\rangle$). In other words, the reply contains an empty byte vector. On errors, an error message is written to standard output and the lgcio interface makes the Logiweb machine exit.

[fileWriteExec (p , c) $\stackrel{\bullet\bullet}{=}$ lgcio (FileWriteExec $:: p ::$ EOS $:: c$)]

Same as fileWrite (p , c) but uses file permissions 777 (read, write, and execute).

[fileRead (p) $\stackrel{\bullet\bullet}{=}$ lgcio (FileRead : : p : : EOS)]

Read the file with pathname p. The reply has form lgcio (c) where c is the contents of the file encoded as a byte vector.

[fileRm (p) $\stackrel{\bullet\bullet}{=}$ lgcio (FileRm : : p : : EOS)]

Remove the file or symbolic link with pathname p. The reply has form lgcio ($\langle\rangle$).

[fileSymlink (p , l) $\stackrel{\bullet\bullet}{=}$ lgcio (FileSymlink : : p : : EOS : : l : : EOS)]

Create a symbolic link at pathname l which points to pathname p. The reply has form lgcio ($\langle\rangle$).

[fileReadLink (l) $\stackrel{\bullet\bullet}{=}$ lgcio (FileReadLink : : l : : EOS)]

Read the symbolic link with pathname l. The reply has form lgcio (p) where p is the contents of the symbolic link. Note that p is not zero terminated.

[fileMkdir (p) $\stackrel{\bullet\bullet}{=}$ lgcio (FileMkdir : : p : : EOS)]

Create a directory with pathname p. The reply has form lgcio ($\langle\rangle$).

[fileRmdir (p) $\stackrel{\bullet\bullet}{=}$ lgcio (FileRmdir : : p : : EOS)]

Remove the directory with pathname p. The directory must be empty. The reply has form lgcio ($\langle\rangle$).

[fileDir (p) $\stackrel{\bullet\bullet}{=}$ lgcio (FileDir : : p : : EOS)]

Read the directory with pathname p. The reply has form lgcio (c) where c is a list of strings each ended by EOS. The list of strings may or may not include "." and "..".

[fileType (p) $\stackrel{\bullet\bullet}{=}$ lgcio (FileType : : p : : EOS)]

Stat the given pathname. The result is of form lgcio (t) where t is a byte vector containing a single byte. The value of t is:

FileTypeNonexistent File does not exist

FileTypeRegular File is regular

FileTypeDirectory File is directory

FileTypeLink File is link

FileTypeOther Otherwise

[fileTypeRead (p) $\stackrel{\bullet\bullet}{=}$ lgcio (FileTypeRead : : p : : EOS)]

Read the file with pathname p. The reply has form lgcio (t : : c) where t is FileTypeNonexistent or FileTypeRegular. c is the contents, if any, of the file encoded as a byte vector.

[textWrite (p , n , c) $\stackrel{\bullet\bullet}{=}$ lgcio (TextWrite : : p : : EOS : : n : : EOS : : c)]

Same as fileWrite (p , c) but changes newline sequences in c to n.

[textWriteExec (p , n , c) $\stackrel{\bullet\bullet}{=}$ lgcio (TextWriteExec : : p : : EOS : : n : : EOS : : c)]

[fileGetCwd $\stackrel{\bullet\bullet}{=}$ lgcio (FileGetCwd)]

Same as fileWriteExec (p , c) but changes newline sequences in c to n.

[unixTime $\stackrel{\bullet\bullet}{=}$ lgcio (UnixTime)]

Return the number of seconds since the Unix epoch. Ignore leap seconds in the sense that the count stops during leap seconds.

[demonize (l , p , u) $\stackrel{\bullet\bullet}{=}$ lgcio (Demonize : : l : : EOS : : p : : EOS : : u : : EOS)]

Turn the running Logiweb machine into a demon by getting rid of the controlling terminal. Redirect standard output and standard error to a log file with pathname l. During demonization, the Logiweb machine changes process identifier (pid). Write the pid to the file with pathname p. If p is the empty string, do not write the pid to any file. Change user and group identifier (uid and gid) to the user with login name u. If u is the empty string, do not change uid and gid.

[execlp1 (d , p , a) $\stackrel{\bullet\bullet}{=}$ lgcio (Execlp1 : : d : : EOS : : p : : EOS : : a : : EOS)]

In directory d, run the program p with argument a. Search the PATH and use the current environment. Wait for the program to exit. Return the return value.

[tcpQuery (d , p , m , e , q) $\stackrel{\bullet\bullet}{=}$
let floor (m , exp10 (e)) **be** v **in**
lgcio (TcpQuery : : d : : EOS : : make-card (p) : : make-card (v^{h}) : : make-card (v^{t}) : : make-card (e) : : q)]

Send the query q to domain d port p and wait up to $\mathrm{m} \cdot 10^{-\mathrm{e}}$ seconds for a reply. The reply has form lgcio (c) where c is the response. In case of error/timeout, chop the response, i.e. just return all bytes received. As an example, if the domain d cannot be resolved, the response is lgcio ($\langle\rangle$).

8.5 The lgcio interface definition

lgcio-interface $\stackrel{\bullet}{=}$

```
/*
The lgcio interface provides all the input/output facilities
needed by the Logiweb lgc compiler.
```

```
*/

#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <pwd.h>
#include <dirent.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <sys/select.h>
#include <sys/wait.h>
#ifdef __CYGWIN__
#include <process.h>
#include <cygwin/in.h>
#endif /* __CYGWIN__ */

#define TRUE                    1
#define FALSE                   0

#define PUT0                    0
#define PUT1                    1
#define GET0                    2
#define GET1                    3

#define BUFSIZE                 1048576
#define S_rw_r_r                S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH

/* PATHSIZE is _POSIX_PATH_MAX, so a path of this size is supported
 * all POSIX platforms.
 * We do not want to depend on _POSIX_PATH_MAX being defined, howeve
#define PATHSIZE                256

#define FileWrite               1
#define FileWriteExec           2
#define FileRead                3
#define FileRm                  4
#define FileSymlink             5
#define FileReadLink            6
#define FileMkdir               7
#define FileRmdir               8
```

```
#define FileDir                 9
#define FileType                10
#define FileTypeRead            11
#define TextWrite               12
#define TextWriteExec           13
#define FileGetCwd              14
#define UnixTime                20
#define Demonize                30
#define Execlp1                 31
#define TcpQuery                40

#define FILE_TYPE_NONEXISTENT  0
#define FILE_TYPE_OTHER        1
#define FILE_TYPE_REGULAR      2
#define FILE_TYPE_DIRECTORY    3
#define FILE_TYPE_LINK         4



/* Error handling */

void die(char *msg){
  printf("%s\n",msg);
  exit(0);}

void perror1(char* msg){
  printf("%s: %s\n",msg,strerror(errno));}

void pdie(char *msg){
  perror1(msg);
  exit(0);}

void Pdie(char *msg){
  perror(msg);
  exit(-1);}

#define PDIE2(x) strcpy(pdiemsg2,x)
char pdiemsg2[BUFSIZE+1];
void pdie2(char *msg1,char *msg2){
  perror1(msg1);
  printf("%s: %s\n",msg2,pdiemsg2);
  exit(0);}

#define PDIE3(x,y) strcpy(pdiemsg2,x);strcpy(pdiemsg3,y)
char pdiemsg3[BUFSIZE+1];
void pdie3(char *msg1,char *msg2,char *msg3){
```

```
  perror1(msg1);
  printf("%s: %s\n",msg2,pdiemsg2);
  printf("%s: %s\n",msg3,pdiemsg3);
  exit(0);}

#define PDIE4(x,y,z) strcpy(pdiemsg2,x);strcpy(pdiemsg3,y);strcpy(pd
char pdiemsg4[BUFSIZE+1];
void pdie4(char *msg1,char *msg2,char *msg3,char *msg4){
  perror1(msg1);
  printf("%s: %s\n",msg2,pdiemsg2);
  printf("%s: %s\n",msg3,pdiemsg3);
  printf("%s: %s\n",msg4,pdiemsg4);
  exit(0);}



/* Argument handling */

int skipString(char buffer[],int bufsize,int bufpnt,char *msg){
  char *pnt=memchr(buffer+bufpnt,0,bufsize-bufpnt);
  if(pnt==NULL) die(msg);
  return (pnt-buffer)+1;}

int skipInt(char buffer[],int bufsize,int bufpnt,char *msg){
  char *pnt;
  for(pnt=buffer+bufpnt;(pnt-buffer)<bufsize;pnt++)
  if(((*pnt)&128)==0) return (pnt-buffer)+1;
  die(msg);}

int getInt(char *buffer){
  if(((*buffer)&128)==0) return *buffer;
  return *buffer&127+128*getInt(buffer+1);}



/* Write file */

int fileWrite2(int fd,char *pnt,char *end){
  int length;
  while(pnt!=end){
    length=write(fd,pnt,end-pnt);
    if(length<0) pdie2("fileWrite()/write()","path");
    pnt+=length;}}

int fileWrite1(int state,char buffer[],int bufsize,mode_t mode){
  static int fd;
```

```
  int bufpnt=0;
  switch(state){
  case PUT0:
    bufpnt=skipString(buffer,bufsize,1,
      "fileWrite(): invalid path name");
    fd=creat(buffer+1,mode);
    PDIE2(buffer+1);
    if(fd<0) pdie2("fileWrite()/open()","path");
  case PUT1:
    fileWrite2(fd,buffer+bufpnt,buffer+bufsize);
/*
    while(bufpnt<bufsize){
      length=write(fd,buffer+bufpnt,bufsize-bufpnt);
      if(length<0) pdie2("fileWrite()/write()","path");
      bufpnt+=length;}
*/
    return 0;
  case GET0:
    if(close(fd)!=0) pdie2("fileWrite()/close()","path");
  case GET1:
    return 0;}}

int fileWrite(int state,char buffer[],int bufsize){
  return fileWrite1(state,buffer,bufsize,0666);}

int fileWriteExec(int state,char buffer[],int bufsize){
  return fileWrite1(state,buffer,bufsize,0777);}



/* Read file */

int fileRead(int state,char buffer[],int bufsize){
  static int fd=-1;
  ssize_t length;
  switch(state){
  case PUT0:
    skipString(buffer,bufsize,1,
      "fileRead(): invalid path name");
    if(fd>=0) close(fd);
    fd=open(buffer+1,O_RDONLY);
    PDIE2(buffer+1);
    if(fd<0) pdie2("fileRead()/open()","path");
  case PUT1:
    return 0;
  case GET0:
```

```
  case GET1:
    if(fd<0) return 0;
    length=read(fd,buffer,BUFSIZE);
    if(length<0) pdie2("fileRead()/read()","path");
    if(length>0) return length;
    if(close(fd)!=0) pdie2("fileRead()/close()","path");
    fd=-1;
    return 0;}}



/* Remove file */

int fileRm(int state,char buffer[],int bufsize){
  switch(state){
  case PUT0:
    skipString(buffer,bufsize,1,"fileRm(): invalid path name");
    PDIE2(buffer+1);
    if(unlink(buffer+1)&&errno!=ENOENT) pdie2("fileRm()/unlink()","p
  case PUT1:
  case GET0:
  case GET1:
    return 0;}}



/* Symlink */

int fileSymlink(int state,char buffer[],int bufsize){
  int arg1,arg2;
  switch(state){
  case PUT0:
    arg1=1;
    arg2=skipString(buffer,bufsize,arg1,
      "fileSymlink(): invalid arguments");
    skipString(buffer,bufsize,arg2,
      "fileSymlink(): invalid arguments");
    PDIE3(buffer+arg1,buffer+arg2);
    if(symlink(buffer+arg1,buffer+arg2))
    pdie3("fileSymlink()/symlink()","dest","link");
  case PUT1:
  case GET0:
  case GET1:
    return 0;}}
```

```
/* Read link */

int fileReadLink(int state,char buffer[],int bufsize){
  int length;
  static char path[PATHSIZE];
  switch(state){
  case PUT0:
    length=skipString(buffer,bufsize,1,
      "fileReadLink(): invalid path name");
    if(length-1>PATHSIZE) die("fileReadLink(): invalid path");
    PDIE2(buffer+1);
    strcpy(path,buffer+1);
  case PUT1:
  case GET1:
    return 0;
  case GET0:
    length=readlink(path,buffer,bufsize);
    if(length<0) pdie2("fileReadLink()/readlink()","path");
    if(length>=bufsize) die("fileReadLink(): link too long");
    return length;}}



/* Make directory */

int fileMkdir(int state,char buffer[],int bufsize){
  char *bufpnt;
  int result;
  switch(state){
  case PUT0:
    skipString(buffer,bufsize,1,
      "fileMkdir(): invalid path name");
    PDIE2(buffer+1);
    result=0;
    if(buffer[1]==0) return 0;
    for(bufpnt=buffer+2;*bufpnt!=0;bufpnt++)
    if(*bufpnt=='/'){*bufpnt=0;result=mkdir(buffer+1,0777);*bufpnt='
    if(result<0&&errno!=EEXIST) pdie2("fileMkdir()/mkdir()","path");
  case PUT1:
  case GET0:
  case GET1:
    return 0;}}
```

```
/* Remove directory */

int fileRmdir(int state,char buffer[],int bufsize){
  switch(state){
  case PUT0:
    skipString(buffer,bufsize,1,
      "fileRmdir(): invalid path name");
    PDIE2(buffer+1);
    if(rmdir(buffer+1)<0&&errno!=ENOENT){
      pdie2("fileRmdir()/rmdir()","path");}
  case PUT1:
  case GET0:
  case GET1:
    return 0;}}



/* Read directory */

int fileDir(int state,char buffer[],int bufsize){
  static DIR *dir=NULL;
  struct dirent *entry;
  size_t length;
  switch(state){
  case PUT0:
    skipString(buffer,bufsize,1,
      "fileDir(): invalid path name");
    if(dir!=NULL){if(closedir(dir)<0) pdie("fileDir()/closedir()");}
    dir=opendir(buffer+1);
    PDIE2(buffer+1);
    if(dir==NULL) pdie2("fileDir()/opendir()","path");
  case PUT1:
    return 0;
  case GET0:
  case GET1:
    if(dir==NULL) return 0;
    errno=0;
    entry=readdir(dir);
    if(errno!=0) pdie2("fileDir()/readdir()","path");
    if(entry==NULL){
      if(closedir(dir)<0) pdie2("fileDir()/closedir()","path");
      dir=NULL;
      return 0;}
    length=strlen(entry->d_name);
    if(length+1>bufsize)
    die("fileDir(): invalid directory entry");
```

```
    strcpy(buffer,entry->d_name);
    return length+1;}}



/* Query file existence and type */

int fileType(int state,char buffer[],int bufsize){
  struct stat buf;
  static char result;
  switch(state){
  case PUT0:
    skipString(buffer,bufsize,1,
      "fileType(): invalid path name");
    PDIE2(buffer+1);
    if(lstat(buffer+1,&buf)<0){
      if(errno==ENOENT) result=FILE_TYPE_NONEXISTENT; else
      pdie2("fileType()/lstat()","path");
      return 0;}
    if(S_ISREG(buf.st_mode)) result=FILE_TYPE_REGULAR; else
    if(S_ISDIR(buf.st_mode)) result=FILE_TYPE_DIRECTORY; else
    if(S_ISLNK(buf.st_mode)) result=FILE_TYPE_LINK; else
    result=FILE_TYPE_OTHER;
  case PUT1:
  case GET1:
    return 0;
  case GET0:
    buffer[0]=result;
    return 1;}}



/* Read file if it exists */

int fileTypeRead(int state,char buffer[],int bufsize){
  struct stat buf;
  static int fd=-1;
  ssize_t length;
  switch(state){
  case PUT0:
    skipString(buffer,bufsize,1,
      "fileTypeRead(): invalid path name");
    if(fd>=0) close(fd);
    fd=-1;
    PDIE2(buffer+1);
    if(stat(buffer+1,&buf)<0){
```

```
      if(errno==ENOENT) return 0; else
      pdie2("fileType()/lstat()","path");}
    if(S_ISDIR(buf.st_mode)) return 0;
    fd=open(buffer+1,O_RDONLY);
  case PUT1:
    return 0;
  case GET0:
    if(fd<0) {buffer[0]=FILE_TYPE_NONEXISTENT; return 1;}
    buffer[0]=FILE_TYPE_REGULAR;
    length=read(fd,buffer+1,BUFSIZE-1);
    if(length<0) pdie2("fileTypeRead()/read()","path");
    if(length>0) return length+1;
    if(close(fd)!=0) pdie2("fileTypeRead()/close()","path");
    fd=-1;
    return 1;
  case GET1:
    if(fd<0) return 0;
    length=read(fd,buffer,BUFSIZE);
    if(length<0) pdie2("fileTypeRead()/read()","path");
    if(length>0) return length;
    if(close(fd)!=0) pdie2("fileTypeRead()/close()","path");
    fd=-1;
    return 0;}}



/* Write text file */

int textWrite1(int state,char buffer[],int bufsize,mode_t mode){
  int arg1,arg2;
  static int fd;
  int bufpnt=0;
  int length;
  static char newlineseq[BUFSIZE+1];
  static int newlinelength;
  static int ignore;
  int p;
  switch(state){
  case PUT0:
    arg1=1;
    arg2=skipString(buffer,bufsize,arg1,
      "textWrite(): invalid path name");
    bufpnt=skipString(buffer,bufsize,arg2,
      "textWrite(): invalid newline sequence");
    strcpy(newlineseq,buffer+arg2);
    newlinelength=strlen(newlineseq);
```

```
    ignore=256;
    fd=creat(buffer+arg1,mode);
    PDIE2(buffer+arg1);
    if(fd<0) pdie2("textWrite()/open()","path");
  case PUT1:
    while(bufpnt<bufsize){
      if(buffer[bufpnt]==ignore){
        bufpnt++;
        ignore=256;}
      if(buffer[bufpnt]==10){
        fileWrite2(fd,newlineseq,newlineseq+newlinelength);
        bufpnt++;
        ignore=13;}
      else if(buffer[bufpnt]==13){
        fileWrite2(fd,newlineseq,newlineseq+newlinelength);
        bufpnt++;
        ignore=10;}
      else{
        for(p=bufpnt+1;p<bufsize&&buffer[p]!=10&&buffer[p]!=13;p++);
        fileWrite2(fd,buffer+bufpnt,buffer+p);
        bufpnt=p;
        ignore=256;}}
    return 0;
  case GET0:
    if(close(fd)!=0) pdie2("textWrite()/close()","path");
  case GET1:
    return 0;}}

int textWrite(int state,char buffer[],int bufsize){
  return textWrite1(state,buffer,bufsize,0666);}

int textWriteExec(int state,char buffer[],int bufsize){
  return textWrite1(state,buffer,bufsize,0777);}



/* Get Current Working Directory */

int fileGetCwd(int state,char buffer[],int bufsize){
  char *dir;
  if(state!=GET0) return 0;
  dir=getcwd(buffer,BUFSIZE);
  if(dir==0) pdie("fileGetCwd");
  return strlen(dir);}
```

```
/* Time */

#define putInt(x) {\
  if(x<0) die("putInt(): Negative integer");\
  for(;x>=128;x/=128){\
    buffer[bufpnt++]=x%128+128;\
    if(bufpnt>=bufsize) die("Buffer overflow");}\
  buffer[bufpnt++]=x;}

int unixTime(int state,char buffer[],int bufsize){
  struct timeval tv;
  int exponent=6;
  int bufpnt=0;
  if(state!=GET0) return 0;
  if(gettimeofday(&tv,NULL)) pdie("time()/gettimeofday()");
  putInt(tv.tv_usec);
  putInt(tv.tv_sec);
  putInt(exponent);
  return bufpnt;}



/* Demonize */

void demon_dief(FILE* file,char* msg){
  fprintf(file,"demonize()/%s: %s\n",msg,strerror(errno));
  exit(0);}

void demon_die(char* msg){
  demon_dief(stderr,msg);}

void open_log_file(int fd0,char* path,int flags){
  int fd1;
  FILE* file=(fd0==2)?stdout:stderr;
  if(close(fd0)==-1) demon_dief(file,"close()");
  fd1=open(path,flags,S_rw_r_r);
  if(fd1<0) demon_dief(file,"open()");
  if(fd1==fd0) return;
  fprintf(file,
    "Opened fd=%d, should have opened fd=%d\n",fd1,fd0);
  exit(0);}

void demonize1(char* log_file,char* pid_file,char* usr_name){
  pid_t pid;
  FILE *pid_stream;
```

```
  struct passwd *user;
  int fd;
  pid=fork();
  if(pid==-1) demon_die("first fork()");
  if(pid) exit(0);
  if(setsid()==-1) demon_die("setsid()");
  pid=fork();
  if(pid==-1) demon_die("second fork()");
  if(pid) exit(0);
  if(pid_file[0]){
    fd=open(pid_file,O_WRONLY|O_CREAT|O_EXCL,S_rw_r_r);
    if(fd<0) demon_die("open(pid_file)");
    if(!(pid_stream=fdopen(fd,"w"))) demon_die("fdopen()");
    if(fprintf(pid_stream,"%d\n",getpid())<0)
    demon_die("fprintf(pid_stream)");
    if(fclose(pid_stream)) demon_die("fclose(pid_stream)");}
  if(usr_name[0]){
    user=getpwnam(usr_name);
    if(!user) demon_die("getpwnam()");
    if(setgid(user->pw_uid)) demon_die("setgid()");
    if(setuid(user->pw_uid)) demon_die("setuid()");}
  if(chdir("/")==-1) demon_die("chdir()");
  umask(022);
  if(close(0)==-1) demon_die("close(0)");
  if(open("/dev/null",O_RDONLY)!=0) demon_die("open(0)");
  open_log_file(1,log_file,O_WRONLY|O_CREAT|O_TRUNC);
  open_log_file(2,log_file,O_WRONLY|O_CREAT|O_TRUNC);}

int demonize(int state,char buffer[],int bufsize){
  int arg1,arg2,arg3;
  switch(state){
  case PUT0:
    arg1=1;
    arg2=skipString(buffer,bufsize,arg1,
      "demonize(): invalid arguments");
    arg3=skipString(buffer,bufsize,arg2,
      "demonize(): invalid arguments");
    skipString(buffer,bufsize,arg3,
      "demonize(): invalid arguments");
    demonize1(buffer+arg1,buffer+arg2,buffer+arg3);
  case PUT1:
  case GET0:
  case GET1:
    return 0;}}
```

```
/* execlp1 */

int execlp1(int state,char buffer[],int bufsize){
  int arg1,arg2,arg3;
  int fd,fd0,fd1,fd2;
  pid_t pid;
  static int status;
  switch(state){
  case PUT0:
    arg1=1;
    arg2=skipString(buffer,bufsize,arg1,
      "execlp1(): invalid argument 1");
    arg3=skipString(buffer,bufsize,arg2,
      "execlp1(): invalid argument 2");
    skipString(buffer,bufsize,arg3,
      "execlp1(): invalid argument 3");
    PDIE4(buffer+arg1,buffer+arg2,buffer+arg3);
#ifndef __CYGWIN__
    pid=fork();
    if(pid<0) pdie("fork()");
    if(pid==0){
      if(chdir(buffer+arg1)) pdie2("execlp1()/chdir()","path");
      fd=open("/dev/null",O_RDWR);
      if(fd<0) Pdie("execlp1()/open(): Cannot open /dev/null");
      if(close(STDIN_FILENO)<0) Pdie("execlp1()/close() failed for s
      if(dup2(fd,STDIN_FILENO)<0) Pdie("execlp1()/dup2() failed for
      if(close(STDOUT_FILENO)<0)
      Pdie("execlp1()/close() failed for stdout");
      if(dup2(fd,STDOUT_FILENO)<0)
      Pdie("execlp1()/dup2() failed for stdout");
      if(close(STDERR_FILENO)<0)
      Pdie("execlp1()/close() failed for stderr");
      if(dup2(fd,STDERR_FILENO)<0) exit(-1);
      execlp(buffer+arg2,buffer+arg2,buffer+arg3,(char *)NULL);
      exit(-1);}
    if(wait(&status)<0)
    pdie4("execlp1()/wait()","path","prog","arg");
#else
    if(chdir(buffer+arg1)) pdie2("execlp1()/chdir()","path");
    fd=open("/dev/null",O_RDWR);
    if(fd<0) Pdie("execlp1()/open(): Cannot open /dev/null");
    fd0=dup(STDOUT_FILENO);
    if(fd0<0) Pdie("execlp1()/dup(): Cannot duplicate stdin");
    fd1=dup(STDOUT_FILENO);
    if(fd1<0) Pdie("execlp1()/dup(): Cannot duplicate stdout");
```

```
    fd2=dup(STDOUT_FILENO);
    if(fd2<0) Pdie("execlp1()/dup(): Cannot duplicate stderr");
    if(dup2(fd,STDIN_FILENO)<0) Pdie("execlp1()/dup2() failed for st
    if(dup2(fd,STDOUT_FILENO)<0) Pdie("execlp1()/dup2() failed for s
    if(dup2(fd,STDERR_FILENO)<0) Pdie("execlp1()/dup2() failed for s
    status=spawnlp(_P_WAIT,buffer+arg2,buffer+arg2,buffer+arg3,(char
    if(dup2(fd2,STDERR_FILENO)<0) Pdie("execlp1()/dup2() failed for
    if(dup2(fd1,STDOUT_FILENO)<0) Pdie("execlp1()/dup2() failed for
    if(dup2(fd0,STDIN_FILENO)<0) Pdie("execlp1()/dup2() failed for f
    if(close(fd)) Pdie("execlp1()/close() failed for fd");
    if(close(fd0)) Pdie("execlp1()/close() failed for fd1");
    if(close(fd1)) Pdie("execlp1()/close() failed for fd1");
    if(close(fd2)) Pdie("execlp1()/close() failed for fd1");
#endif /* __CYGWIN__ */
  case PUT1:
  case GET1:
    return 0;
  case GET0:
    buffer[0]=(char)(-1);
    if(WIFEXITED(status)) buffer[0]=(char)(WEXITSTATUS(status));
    return 1;}}

/* tcpQuery */

void print_herror(void){
  switch(h_errno){
    case HOST_NOT_FOUND:
    fprintf(stderr,"The specified host is unknown.\n");
    break;
    case NO_ADDRESS:
    fprintf(stderr,
      "The requested name is valid but does not have\n");
    fprintf(stderr,"an IP address.\n");
    break;
    case NO_RECOVERY:
    fprintf(stderr,
      "A non-recoverable name server error occurred.\n");
    break;
    case TRY_AGAIN:
    fprintf(stderr,
      "A temporary error occurred on an authoritative\n");
    fprintf(stderr,"name server. Try again later.\n");
    break;
    default:
    fprintf(stderr,"Unknown error.\n");}}
```

```
int set_addr(struct sockaddr_in *addr,char* domain,int port){
  struct hostent *host=gethostbyname(domain);
  if(!host){
    fprintf(stderr,"Unix call: gethostbyname(%s)\n",domain);
    print_herror();
    return -1;}
  addr->sin_family=AF_INET;
  addr->sin_port=htons(port);
  memcpy(&(addr->sin_addr),host->h_addr_list[0],host->h_length);
  return 0;}

#define GIVE_UP(x) {close(fd);fd=-1;return 0;}
int tcpQuery(int state,char* buffer,int bufsize){
  int arg1,arg2,arg3,arg4,arg5,arg6;
  char *domain;
  int port,secs,frac,expo;
  struct timeval tv;
  static int fd=-1;
  fd_set set;
  static long secs0,frac0;
  struct sockaddr_in to;
  ssize_t length1;
  int result;
  int flags;
  switch(state){
  case PUT0:
    arg1=1;
    arg2=skipString(buffer,bufsize,arg1,
      "tcpQuery(): invalid argument 1");
    arg3=skipInt(buffer,bufsize,arg2,
      "tcpQuery(): invalid argument 2");
    arg4=skipInt(buffer,bufsize,arg3,
      "tcpQuery(): invalid argument 3");
    arg5=skipInt(buffer,bufsize,arg4,
      "tcpQuery(): invalid argument 4");
    arg6=skipInt(buffer,bufsize,arg5,
      "tcpQuery(): invalid argument 5");
    domain=buffer+arg1;
    port=getInt(buffer+arg2);
    secs=getInt(buffer+arg3);
    frac=getInt(buffer+arg4);
    expo=getInt(buffer+arg5);
    PDIE3(buffer+arg1,buffer+arg3);
    for(;expo>6;expo--) frac/=10;
    for(;expo<6;expo++) frac*=10;
    if(gettimeofday(&tv, NULL))
```

```
  pdie3("tcpQuery()/gettimeofday()","domain","seconds");
  secs0=secs+tv.tv_sec-1;
  frac0=frac+tv.tv_usec+1000000;
  if(fd>=0) close(fd);
  fd=socket(AF_INET,SOCK_STREAM,0);
  if(fd<0) pdie3("tcpQuery()/socket()","domain","seconds");
  if(set_addr(&to,domain,port)) GIVE_UP(1)
  flags=fcntl(fd,F_GETFL,0);
  fcntl(fd,F_SETFL,flags|O_NONBLOCK);
  if(connect(fd,(void *)&to,sizeof(to))){
    if(errno!=EINPROGRESS) GIVE_UP(2);
    FD_ZERO(&set);
    FD_SET(fd,&set);
    if(gettimeofday(&tv,NULL))
    pdie3("tcpQuery()/gettimeofday()","domain","seconds");
    secs=secs0-tv.tv_sec;
    frac=frac0-tv.tv_usec;
    secs=secs+frac/1000000;
    frac=frac%1000000;
    if(secs<0) GIVE_UP(3)
    tv.tv_sec=secs;
    tv.tv_usec=frac;
    result=select(fd+1,0,&set,0,&tv);
    if(result<0) pdie3("tcpQuery()/select()","domain","seconds");
    if(result==0) GIVE_UP(4)
    result=select(fd+1,0,&set,0,&tv);
    if(result<0) pdie3("tcpQuery()/select()","domain","seconds");
    if(result==0) GIVE_UP(5)}
  bufsize=bufsize-arg6;
  buffer=buffer+arg6;
case PUT1:
  do{
    FD_ZERO(&set);
    FD_SET(fd,&set);
    if(gettimeofday(&tv,NULL))
    pdie3("tcpQuery()/gettimeofday()","domain","seconds");
    secs=secs0-tv.tv_sec;
    frac=frac0-tv.tv_usec;
    secs=secs+frac/1000000;
    frac=frac%1000000;
    if(secs<0) GIVE_UP(6)
    tv.tv_sec=secs;
    tv.tv_usec=frac;
    result=select(fd+1,0,&set,0,&tv);
    if(result<0) pdie3("tcpQuery()/select()","domain","seconds");
    if(result==0) GIVE_UP(7)
```

```
      length1=send(fd,buffer,bufsize,0);
      if(length1<0) GIVE_UP(8)
      buffer+=length1;
      bufsize-=length1;}
    while(bufsize>0);
    return 0;
  case GET0:
  case GET1:
    if(fd<0) return 0;
    FD_ZERO(&set);
    FD_SET(fd,&set);
    if(gettimeofday(&tv,NULL))
    pdie3("tcpQuery()/gettimeofday()","domain","seconds");
    secs=secs0-tv.tv_sec;
    frac=frac0-tv.tv_usec;
    secs=secs+frac/1000000;
    frac=frac%1000000;
    if(secs<0) GIVE_UP(9)
    tv.tv_sec=secs;
    tv.tv_usec=frac;
    result=select(fd+1,&set,0,&set,&tv);
    if(result<0) pdie3("tcpQuery()/select()","domain","seconds");
    if(result==0) GIVE_UP(10)
    length1=recv(fd,buffer,bufsize,0);
    if(length1<=0) GIVE_UP(11)
    return length1;}}



/* Dispatch on command */

#define CASE(x,y) case x:return y(state,buffer,bufsize)
int trigger2(int state,char command,char buffer[],int bufsize){
  switch(command){
    CASE(FileWrite    ,fileWrite);
    CASE(FileWriteExec,fileWriteExec);
    CASE(FileRead     ,fileRead);
    CASE(FileRm       ,fileRm);
    CASE(FileSymlink  ,fileSymlink);
    CASE(FileReadLink ,fileReadLink);
    CASE(FileMkdir    ,fileMkdir);
    CASE(FileRmdir    ,fileRmdir);
    CASE(FileDir      ,fileDir);
    CASE(FileType     ,fileType);
    CASE(FileTypeRead ,fileTypeRead);
    CASE(TextWrite    ,textWrite);
```

```
    CASE(TextWriteExec,textWriteExec);
    CASE(FileGetCwd   ,fileGetCwd);
    CASE(UnixTime     ,unixTime);
    CASE(Demonize     ,demonize);
    CASE(Execlp1      ,execlp1);
    CASE(TcpQuery     ,tcpQuery);
    default: return 0;}}



/* Transfer one buffer */

int trigger1(int push1,char buffer[],int bufsize){
  static int push0=FALSE;
  static char command;
  if(push1){
    if(push0) return trigger2(PUT1,command,buffer,bufsize);
    push0=TRUE;
    command=buffer[0];
    return trigger2(PUT0,command,buffer,bufsize);}
  if(!push0) return trigger2(GET1,command,buffer,bufsize);
  push0=FALSE;
  return trigger2(GET0,command,buffer,bufsize);}



/* Transfer one byte */

int trigger(int c){
  static int push0=FALSE;
  static char buffer[BUFSIZE+1];
  static int bufpnt=0;
  static int bufsize;
  int push1=(c>=0);
  if(push1){
    if(!push0){push0=TRUE;bufpnt=0;}
    if(bufpnt>=BUFSIZE){trigger1(TRUE,buffer,bufpnt);bufpnt=0;}
    buffer[bufpnt++]=c;
    return -1;}
  if(push0){push0=FALSE;trigger1(TRUE,buffer,bufpnt);bufsize=0;}
  if(bufpnt>=bufsize){
    bufsize=trigger1(FALSE,buffer,BUFSIZE);
    bufpnt=0;}
  if(bufpnt>=bufsize) return -1;
  return (unsigned char)buffer[bufpnt++];}
```

8.6 Test of the lgcio interface

In Section 7.9 we defined that the lgciotest machine uses lgcio1 as handler. That handler is defined in the following. When executed, the lgciotest machine invokes all the features of the lgcio interface.

[lgcio1 $\stackrel{\bullet\bullet}{=}$ map (λx.lgcio2 (x))]

[lgcio2 (x) $\stackrel{\bullet\bullet}{=}$

```
extend request ( lgcio ( ⊤ ) , lgcio-interface ) : :
fileMkdir ( testdir/ ) : :
fileWrite ( testdir/testfile1 , abc ) : :
fileWriteExec ( testdir/testfile2 , def ) : :
fileRead ( testdir/testfile1 ) : :
fileRead ( testdir/testfile2 ) : :
fileSymlink ( testfile2 , testdir/testfile3 ) : :
fileSymlink ( testfile3 , testdir/testfile4 ) : :
fileRead ( testdir/testfile4 ) : :
fileReadLink ( testdir/testfile4 ) : :
fileType ( testdir/testfile0 ) : :
fileType ( /dev/tty ) : :
fileType ( testdir/testfile1 ) : :
fileType ( testdir ) : :
fileType ( testdir/testfile3 ) : :
fileTypeRead ( testdir/testfile0 ) : :
fileTypeRead ( testdir/testfile1 ) : :
fileTypeRead ( testdir ) : :
fileTypeRead ( testdir/testfile3 ) : :
fileDir ( testdir ) : :
textWrite ( testdir/testtext1 , 12 , abc ) : :
textWriteExec ( testdir/testtext2 , 12 , def ) : :
fileRead ( testdir/testtext1 ) : :
fileRead ( testdir/testtext2 ) : :
unixTime : :
write request ( tcpQuery ) : :
tcpQuery ( logiweb.eu , 80 , 21 , 1 , GET/test.html : : CRLF ) : :
unixTime : :
fileGetCwd : :
fileRm ( testdir/testfile1 ) : :
fileRm ( testdir/testfile2 ) : :
fileRm ( testdir/testfile3 ) : :
fileRm ( testdir/testfile4 ) : :
fileRm ( testdir/testfile5 ) : :
fileRm ( testdir/testtext1 ) : :
fileRm ( testdir/testtext2 ) : :
fileRmdir ( testdir ) : :
```

fileRmdir (testdir2) : :
exec request ($\top$, lgcio3) : : $\top$]

[lgcio3 $\stackrel{\bullet\bullet}{=}$ map (λx.lgcio4 (x))]

[lgcio4 (x) $\stackrel{\bullet\bullet}{=}$
if
println (‘exec reply’) .then.
let e : : x = x **in** $e \stackrel{\mathrm{r}}{=}$ exec reply ($\top$, $\top$) **and**
let x = reverse (x) **in**
println (‘extend reply lgcio-interface’) .then.
let e : : x = x **in** $e \stackrel{\mathrm{r}}{=}$ extend reply ($\top$) **and**
println (‘fileMkdir testdir/’) .then.
let e : : x = x **in** e = lgcio ($\top$) **and**
println (‘fileWrite testdir/testfile1’) .then.
let e : : x = x **in** e = lgcio ($\top$) **and**
println (‘fileWriteExec testdir/testfile2’) .then.
let e : : x = x **in** e = lgcio ($\top$) **and**
println (‘fileRead testdir/testfile1’) .then.
let e : : x = x **in** vt2vector (e^1) = ‘abc
’ **and**
println (‘fileRead testdir/testfile2’) .then.
let e : : x = x **in** vt2vector (e^1) = ‘def
’ **and**
println (‘fileSymlink testfile2 testdir/testfile3’) .then.
let e : : x = x **in** e = lgcio ($\top$) **and**
println (‘fileSymlink testfile3 testdir/testfile4’) .then.
let e : : x = x **in** e = lgcio ($\top$) **and**
println (‘fileRead testdir/testfile4’) .then.
let e : : x = x **in** vt2vector (e^1) = ‘def
’ **and**
println (‘fileReadLink testdir/testfile4’) .then.
let e : : x = x **in** vt2vector (e^1) = testfile3 **and**
println (‘fileType testdir/testfile0’) .then.
let e : : x = x **in** e^1 = FileTypeNonexistent : : $\top$ **and**
println (‘fileType /dev/tty’) .then.
let e : : x = x **in** e^1 = FileTypeOther : : $\top$ **and**
println (‘fileType testdir/testfile1’) .then.
let e : : x = x **in** e^1 = FileTypeRegular : : $\top$ **and**
println (‘fileType testdir’) .then.
let e : : x = x **in** e^1 = FileTypeDirectory : : $\top$ **and**
println (‘fileType testdir/testfile3’) .then.
let e : : x = x **in** e^1 = FileTypeLink : : $\top$ **and**
println (‘fileTypeRead testdir/testfile0’) .then.
let e : : x = x **in** e^1 = FileTypeNonexistent : : $\top$ **and**
println (‘fileTypeRead testdir/testfile1’) .then.

let $e :: x = x$ **in** $e^1 =$ FileTypeRegular :: a :: b :: c :: LF :: T **and**
println ('fileTypeRead testdir') .then.
let $e :: x = x$ **in** $e^1 =$ FileTypeNonexistent :: T **and**
println ('fileTypeRead testdir/testfile3') .then.
let $e :: x = x$ **in** $e^1 =$ FileTypeRegular :: d :: e :: f :: LF :: T **and**
println ('fileDir testdir') .then.
let $e :: x = x$ **in** $e \overset{\mathrm{r}}{=}$ lgcio (T) **and**
println ('textWrite testdir/testtext1') .then.
let $e :: x = x$ **in** $e =$ lgcio (T) **and**
println ('textWriteExec testdir/testtext2') .then.
let $e :: x = x$ **in** $e =$ lgcio (T) **and**
println ('fileRead testdir/testtext1') .then.
let $e :: x = x$ **in** vt2vector (e^1) = 'a12b12c12' **and**
println ('fileRead testdir/testtext2') .then.
let $e :: x = x$ **in** vt2vector (e^1) = 'd12e12f12' **and**
println ('unixTime') .then.
let $e :: x = x$ **in** $e \overset{\mathrm{r}}{=}$ lgcio (T) **and** trace (parse-unixTime (e))
.then.
println ('tcpQuery logiweb.eu 80 GET /test.html') .then.
let $e :: x = x$ **in** $e \overset{\mathrm{r}}{=}$ lgcio (T) **and** trace (vt2vector (e^1)) .then.
println ('unixTime') .then.
let $e :: x = x$ **in** $e \overset{\mathrm{r}}{=}$ lgcio (T) **and** trace (parse-unixTime (e))
.then.
println ('fileGetCwd') .then.
let $e :: x = x$ **in** $e \overset{\mathrm{r}}{=}$ lgcio (T) **and** println (e^1) .then.
println ('fileRm testdir/testfile1') .then.
let $e :: x = x$ **in** $e =$ lgcio (T) **and**
println ('fileRm testdir/testfile2') .then.
let $e :: x = x$ **in** $e =$ lgcio (T) **and**
println ('fileRm testdir/testfile3') .then.
let $e :: x = x$ **in** $e =$ lgcio (T) **and**
println ('fileRm testdir/testfile4') .then.
let $e :: x = x$ **in** $e =$ lgcio (T) **and**
println ('fileRm testdir/testfile5') .then.
let $e :: x = x$ **in** $e =$ lgcio (T) **and**
println ('fileRm testdir/testtext1') .then.
let $e :: x = x$ **in** $e =$ lgcio (T) **and**
println ('fileRm testdir/testtext2') .then.
let $e :: x = x$ **in** $e =$ lgcio (T) **and**
println ('fileRmdir testdir') .then.
let $e :: x = x$ **in** $e =$ lgcio (T) **and**
println ('fileRmdir testdir2') .then.
let $e :: x = x$ **in** $e =$ lgcio (T) **and**
$x = \mathsf{T}$ **then**
⟨write request (Testsucceeded.)⟩ **else**

$\langle$write request (Testfailed.), quit request (1)$\rangle$]

9 Compiler support functions

9.1 General functions

[x last $\stackrel{\bullet\bullet}{=}$ **if** x^{t} **then** x^{h} **else** x^{t} last]

[default (x , y) $\stackrel{\bullet\bullet}{=}$ **if** y **then** x **else** y]

[repeat (n , x) $\stackrel{\bullet\bullet}{=}$
if $n \leq 0$ **then** $\top$ **else** x :: repeat ($n - 1$, x)]

9.2 Conversion to mixed endian hexadecimal

[lgc-string2mixed (a) $\stackrel{\bullet\bullet}{=}$ lgc-string2mixed1 (vt2vector$*$ (a))]

[lgc-string2mixed1 (a) $\stackrel{\bullet\bullet}{=}$
if a **then** $\top$ **else**
let $c :: a = a$ **in**
let $m :: n =$ floor ($c -$ NULL , 16) **in**
lgc-hexdigit (m) :: lgc-hexdigit (n) :: lgc-string2mixed1 (a)]

[lgc-hexdigit (m) $\stackrel{\bullet\bullet}{=}$
if $m \leq 9$ **then** $m +$ '0' **else** $m +$ 'A' $-$ Base]

9.3 Rack conversion

Recall that the rack R of a page is an array which is non-uniform in the sense that $R[i]$ has different types for different values of i. For the sake of efficiency, the lgc-compiler is able to dump the rack of a page to a file so that it can read it back again later.

We refer to a directory which contains rack files as a cache directory. Cache directories typically do not contain rack files directly. Rather, they contain directories which in turn contain the rack files. Rack files are typically named `page.lgr`. The directory containing the rack file of a page is typically named a where a is the reference of the page expressed in mixed endian hexadecimal.

The rack2sl (R) function converts the rack R on internal form to a list of singleton strings suited for writing to a rack file.

The sl2rack (r) converts the other way. For racks R we have sl2rack (rack2sl (R)) $= R$.

A rack is expressed as a sequence $\langle s_3, s_4, s_5, \cdots \rangle$ of items. Each item can be:
Each element s_k can represent one of the following:

- A zero byte followed by a cardinal c expressed base 128 represents that cardinal.

- A one byte represents T.
- A two byte followed by a cardinal c expressed base 128 followed by c bytes represents the vector comprising those bytes.
- Two cardinals $i, j < k$ expressed base 128 represents $s_i :: s_j$.
- One cardinal $i = k$ expressed base 128 marks the end of the rack.

[rack2sl (R) $\overset{\text{Late}}{\to\to}$ norm $R{\in}\mathbf{V}$:
if R **then** bt2vector∗ (3) **else**
let s = rack2sl1 (R , $\mathsf{T}[0{\to}3]$) **in**
vt2vector∗ ($s[2]$ :: make-card ($s[0]$))]

Convert the rack R to external form.

[rack2sl1 (R , s) $\overset{\bullet\bullet}{=}$
if R **then** $s[1{\to}1]$ **else**
if $R \in \mathbf{Z}$ **then** rack2sl-card (R , s) **else**
if $R \in \mathbf{P}$ **then** rack2sl-pair (R , s) **else** $\bullet$]

Convert the rack R to external form and store it in $s[2]$. During conversion, $s[0]$ is the value of the next unused index, $s[1]$ is the index of the converted rack, $s[3][i]$ is the index of the cardinal i, and $s[4][i][j]$ is the index of $v_i :: v_j$ where v_i is the value with index i.

[rack2sl-card (R , s) $\overset{\bullet\bullet}{=}$
if $R < 0$ **then** $\bullet$ **else**
let $i = s[3][R]$ **in**
if not i **then** $s[1{\to}i]$ **else**
let $n = s[0]$ **in**
let $s = s[0{\to}n+1]$ **in**
let $s = s[1{\to}n]$ **in**
let $s = s[\langle 3, R\rangle{\Rightarrow}n]$ **in**
let $d :: q$ = floor (integer-length (R) , 8) **in**
let $r = s[2]$ **in**
if $q \neq 1$ **then** $s[2{\to}r$ :: make-card (0) :: make-card (R)] **else**
$s[2{\to}r$ :: make-card (2) :: make-card (d) :: vector (R)]]

Same as rack2sl1 (R , s) except that R is assumed to be a cardinal.

[rack2sl-pair (R , s) $\overset{\bullet\bullet}{=}$
let s = rack2sl1 (R^{h} , s) **in**
let $i = s[1]$ **in**
let s = rack2sl1 (R^{t} , s) **in**
let $j = s[1]$ **in**
let $k = s[4][i][j]$ **in**
if not k **then** $s[1{\to}k]$ **else**
let $n = s[0]$ **in**
let $s = s[0{\to}n+1]$ **in**

$\textbf{let}\ s = s[1 \rightarrow n]\ \textbf{in}$
$\textbf{let}\ s = s[\langle 4, i, j \rangle \Rightarrow n]\ \textbf{in}$
$\textbf{let}\ r = s[2]\ \textbf{in}$
$s[2 \rightarrow r :: \text{make-card}\ (\ i\ ) :: \text{make-card}\ (\ j\ )]]$

$[\text{sl2rack}\ (\ r\ ) \overset{\text{Late}}{\rightarrow\rightarrow} \text{norm}\ r \in \mathbf{V}:$
$\text{sl2rack1}\ (\ r\ ,\ 3\ ,\ \top\ )]$

$[\text{sl2rack1}\ (\ r\ ,\ n\ ,\ s\ ) \overset{\bullet\bullet}{=}$
$\textbf{let}\ c :: r = \text{parse-card}\ (\ r\ )\ \textbf{in}$
$\textbf{if}\ c = n\ \textbf{then}\ s[n-1]\ \textbf{else}$
$\textbf{let}\ C :: r = \text{parse-card}\ (\ r\ )\ \textbf{in}$
$\textbf{if}\ c = 0\ \textbf{then}\ \text{sl2rack1}\ (\ r\ ,\ n+1\ ,\ s[n \rightarrow C]\ )\ \textbf{else}$
$\textbf{if}\ c = 2\ \textbf{then}\ \text{sl2rack-vector}\ (\ C\ ,\ r\ ,\ n\ ,\ s\ ,\ \top\ )\ \textbf{else}$
$\text{sl2rack1}\ (\ r\ ,\ n+1\ ,\ s[n \rightarrow s[c] :: s[C]]\ )]$

Convert the rack r to internal form. The n parameter is the next unused index and s must be an array which maps indices into r to values.

$[\text{sl2rack-vector}\ (\ C\ ,\ r\ ,\ n\ ,\ s\ ,\ v\ ) \overset{\bullet\bullet}{=}$
$\textbf{if}\ C = 0\ \textbf{then}\ \text{sl2rack1}\ (\ r\ ,\ n+1\ ,\ s[n \rightarrow \text{vt2vector}\ (\ v\ )]\ )\ \textbf{else}$
$\textbf{let}\ b :: r = r\ \textbf{in}$
$\textbf{if}\ b < \text{NULL}\ \textbf{or}\ b > \text{V255}\ \textbf{then}\ \bullet\ \textbf{else}$
$\text{sl2rack-vector}\ (\ C - 1\ ,\ r\ ,\ n\ ,\ s\ ,\ v :: b\ )]$

Accumulate the first C bytes of r in v. When C reaches zero, convert v to a vector, put it in s under the next unused index, and continue conversion.

9.4 Rack cleaning and restoring

Not all information in a rack is suited for dumping to a file. The branches of a rack which are not suited for dumping are:

- R['code'] This branch is the only branch which contains compiled code. Since rack files are platform independent, the code branch has to be reconstructed each time a rack file is read.
- R['cluster'] This branch is very big, contains redundant information, and is very fast to reconstruct. For that reason, it is omitted from the rack file and reconstructed each time a rack file is read.
- R['diagnose'] This branch is map-tagged and, hence, its value is computed lazily. However, when rack files are written, the value is forced to be computed and the map-tag is removed. The map-tag is added again when each time the rack file is read.

Functions for removing and restoring the three branches above are defined in the following.

[lgr-rack-clean (R) $\stackrel{\bullet\bullet}{=}$

let $R = R[\text{`code'} \rightarrow \mathsf{T}]$ **in**
let $R = R[\text{`cluster'} \rightarrow \mathsf{T}]$ **in**
let $d = R[\text{`diagnose'}]^{\mathrm{U}}$ **in**
$R[\text{`diagnose'} \rightarrow d]]$

Remove the code and cluster branches and untag the diagnose branch of the rack R.

[lgr-cache-restore (c) $\stackrel{\bullet\bullet}{=}$

let $r = c[0]$ **in**
let $d = c[r][\text{`diagnose'}]^{\mathrm{M}}$ **in**
let $c = \text{compile}\ (\ c\ )$ **in**
$c[\langle r, \text{`diagnose'}\rangle \Rightarrow d]]$

Restore the diagnose and code branches in the cache c. Note that the call to compile (c) verifies the page lazily, leading to a maptagged, unevaluated diagnose, but that diagnose is overwritten by the known diagnose so it is never evaluated.

[lgr-cluster-closure (r , C) $\stackrel{\bullet\bullet}{=}$

let $c = C[r]$ **in**
let $\mathsf{T} :: b = c[r][\text{`bibliography'}]$ **in**
let $C' = \text{lgr-cluster-closure1}\ (\ b\ ,\ C\ )$ **in**
let $c = c[\langle r, \text{`cluster'}\rangle \Rightarrow C']$ **in**
let $c = \text{lgr-cluster-closure2}\ (\ c\ ,\ C'\ )$ **in**
$C[r \rightarrow c]]$

Lookup the cache c in the cluster C. Set C' to the cluster of all transitively referenced pages and install it in the cluster branch of the rack. Then add racks of all transitively referenced pages to the cache c and install it in the cluster C.

[lgr-cluster-closure1 (b , C) $\stackrel{\bullet\bullet}{=}$

if $b \in \mathbf{A}$ **then** T **else**
let $r :: b = b$ **in**
let $C' = \text{lgr-cluster-closure1}\ (\ b\ ,\ C\ )$ **in**
let $C' = C'[r \rightarrow C[r]]$ **in**
lgr-array-add (C' , $C[r][r][\text{`cluster'}]$)]

Construct the subcluster C' of C which comprises all caches reflexively and transitively referenced from the bibliography b.

[lgr-array-add (a , A) $\stackrel{\bullet\bullet}{=}$

if $A \in \mathbf{A}$ **then** a **else**
if $A^{\mathrm{h}} \in \mathbf{Z}$ **then** $a[A^{\mathrm{h}} \rightarrow A^{\mathrm{t}}]$ **else**
lgr-array-add (lgr-array-add (a , A^{h}) , A^{t})]

Add all elements of the array A to the array a.

[lgr-cluster-closure2 (c , C) $\stackrel{\bullet\bullet}{=}$
if $C \in$ **A then** c **else**
let $r = C^{\mathrm{h}}$ **in**
if $r \in$ **Z then** $c[r \rightarrow C[r][r]]$ **else**
lgr-cluster-closure2 (lgr-cluster-closure2 (c , C^{h}) , C^{t})]

Add all racks in the cluster C to the cache c.

10 Ripemd160

10.1 Introduction

Ripemd-160 is described in: H. Dobbertin, A. Bosselaers, B. Preneel, "RIPEMD-160, a strengthened version of RIPEMD", which is available on the net in ps and pdf. The article is an updated and corrected version of the article published in Fast Software Encryption, LNCS 1039, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 71-82.

The code in the following is an adaption of C code taken from http://homes.esat.lae/ripemd160.html. The software came with the following header:

```
    FILE:     rmd160.c

    CONTENTS: A sample C-implementation of the RIPEMD-160
              hash-function.
    TARGET:   any computer with an ANSI C compiler

    AUTHOR:   Antoon Bosselaers, ESAT-COSIC
    DATE:     1 March 1996
    VERSION:  1.0

    Copyright (c) Katholieke Universiteit Leuven
    1996, All Rights Reserved

Conditions for use of the RIPEMD-160 Software

The RIPEMD-160 software is freely available for use under the terms
conditions described hereunder, which shall be deemed to be accepted
any user of the software and applicable on any use of the software:

1. K.U.Leuven Department of Electrical Engineering-ESAT/COSIC shall
   all purposes be considered the owner of the RIPEMD-160 software a
   all copyright, trade secret, patent or other intellectual propert
   rights therein.
2. The RIPEMD-160 software is provided on an "as is" basis without
   warranty of any sort, express or implied. K.U.Leuven makes no
   representation that the use of the software will not infringe any
```

patent or proprietary right of third parties. User will indemnify K.U.Leuven and hold K.U.Leuven harmless from any claims or liabil which may arise as a result of its use of the software. In no circumstances K.U.Leuven R&D will be held liable for any deficien fault or other mishappening with regard to the use or performance the software.

3. User agrees to give due credit to K.U.Leuven in scientific public or communications in relation with the use of the RIPEMD-160 soft as follows: RIPEMD-160 software written by Antoon Bosselaers, available at http://www.esat.kuleuven.be/~cosicart/ps/AB-9601/.

10.2 Elementary 32 bit functions

[ripemd-mask0 $\stackrel{\bullet\bullet}{=}$
ash (1 , 32) − 1]

A cardinal whose 32 least significant bits are one bits.

[ripemd-mask1 (x) $\stackrel{\bullet\bullet}{=}$
logand (x , ripemd-mask0)]

[ripemd-rol (x , n) $\stackrel{\bullet\bullet}{=}$
ripemd-mask1 (logior (ash (x , n) , ash (x , $n - 32$)))]

ripemd-rol (x , n) cyclically rotates n bits to the left. x must be of a 32 bit cardinal and n must satisfy $0 \leq n < 32$.

10.3 Bit combination functions

[ripemd-f (x , y , z) $\stackrel{\bullet\bullet}{=}$
logxor (x , logxor (y , z))]

[ripemd-g (x , y , z) $\stackrel{\bullet\bullet}{=}$
logior (logand (x , y) , logand (lognot (x) , z))]

[ripemd-h (x , y , z) $\stackrel{\bullet\bullet}{=}$
ripemd-mask1 (logxor (logior (x , lognot (y)) , z))]

[ripemd-i (x , y , z) $\stackrel{\bullet\bullet}{=}$
logior (logand (x , z) , logand (y , lognot (z)))]

[ripemd-j (x , y , z) $\stackrel{\bullet\bullet}{=}$
ripemd-mask1 (logxor (x , logior (y , lognot (z))))]

10.4 Constants

[ripemd-gg-const $\stackrel{\bullet\bullet}{=}$ 1518500249]

[ripemd-hh-const $\stackrel{\bullet\bullet}{=}$ 1859775393]

[ripemd-ii-const $\stackrel{\bullet\bullet}{=}$ 2400959708]

[ripemd-jj-const $\stackrel{\bullet\bullet}{=}$ 2840853838]

[ripemd-ggg-const $\stackrel{\bullet\bullet}{=}$ 2053994217]

[ripemd-hhh-const $\stackrel{\bullet\bullet}{=}$ 1836072691]

[ripemd-iii-const $\stackrel{\bullet\bullet}{=}$ 1548603684]

[ripemd-jjj-constant $\stackrel{\bullet\bullet}{=}$ 1352829926]

10.5 Macros used in rounds

[ripemd-ff (a , b , c , d , e , x , s); $z \ddot{=}$
let a = ripemd-mask1 (a + ripemd-f (b , c , d) + x) **in**
let a = ripemd-mask1 (ripemd-rol (a , s) + e) **in**
let c = ripemd-rol (c , 10) **in** z]

[ripemd-gg (a , b , c , d , e , x , s); $z \ddot{=}$
let a = ripemd-mask1 (a+ripemd-g (b , c , d)+x+ripemd-gg-const) **in**
let a = ripemd-mask1 (ripemd-rol (a , s) + e) **in**
let c = ripemd-rol (c , 10) **in** z]

[ripemd-hh (a , b , c , d , e , x , s); $z \ddot{=}$
let a = ripemd-mask1 (a+ripemd-h (b , c , d)+x+ripemd-hh-const) **in**
let a = ripemd-mask1 (ripemd-rol (a , s) + e) **in**
let c = ripemd-rol (c , 10) **in** z]

[ripemd-ii (a , b , c , d , e , x , s); $z \ddot{=}$
let a = ripemd-mask1 (a+ripemd-i (b , c , d)+x+ripemd-ii-const) **in**
let a = ripemd-mask1 (ripemd-rol (a , s) + e) **in**
let c = ripemd-rol (c , 10) **in** z]

[ripemd-jj (a , b , c , d , e , x , s); $z \ddot{=}$
let a = ripemd-mask1 (a+ripemd-j (b , c , d)+x+ripemd-jj-const) **in**
let a = ripemd-mask1 (ripemd-rol (a , s) + e) **in**
let c = ripemd-rol (c , 10) **in** z]

[ripemd-fff (a , b , c , d , e , x , s); $z \ddot{=}$
let a = ripemd-mask1 (a + ripemd-f (b , c , d) + x) **in**
let a = ripemd-mask1 (ripemd-rol (a , s) + e) **in**
let c = ripemd-rol (c , 10) **in** z]

[ripemd-ggg (a , b , c , d , e , x , s); $z \ddot{=}$
let a = ripemd-mask1 (a+ripemd-g (b , c , d)+x+ripemd-ggg-const) **in**
let a = ripemd-mask1 (ripemd-rol (a , s) + e) **in**
let c = ripemd-rol (c , 10) **in** z]

[ripemd-hhh (a , b , c , d , e , x , s); $z \ddot{=}$
let a = ripemd-mask1 (a+ripemd-h (b , c , d)+x+ripemd-hhh-const) **in**
let a = ripemd-mask1 (ripemd-rol (a , s) + e) **in**
let c = ripemd-rol (c , 10) **in** z]

[ripemd-iii (a , b , c , d , e , x , s); $z \ddot{=}$
let a = ripemd-mask1 (a+ripemd-i (b , c , d)+x+ripemd-iii-const) **in**
let a = ripemd-mask1 (ripemd-rol (a , s) + e) **in**
let c = ripemd-rol (c , 10) **in** z]

[ripemd-jjj (a , b , c , d , e , x , s); $z \ddot{=}$
let a = ripemd-mask1 (a+ripemd-j (b , c , d)+x+ripemd-jjj-const) **in**
let a = ripemd-mask1 (ripemd-rol (a , s) + e) **in**
let c = ripemd-rol (c , 10) **in** z]

10.6 Initial quintuple

[ripemd-init $\bullet\bullet\!=$
$\langle 1732584193, 4023233417, 2562383102, 271733878, 3285377520 \rangle$]

10.7 Compress function

The ripemd-compress (Q , X) function combines the quintuple $Q = \langle a, b, c, d, e \rangle$ and the array A into a new quintuple. $X[i]$ is supposed to be a 32-bit cardinal for all i between 0 and 15, inclusive.

[ripemd-compress (Q , X) $\bullet\bullet\!=$
let $a :: b :: c :: d :: e :: \mathsf{T} = Q$ **in**
let $A :: B :: C :: D :: E :: \mathsf{T} = Q$ **in**
ripemd-ff (a , b , c , d , e , $X[0]$, 11);
ripemd-ff (e , a , b , c , d , $X[1]$, 14);
ripemd-ff (d , e , a , b , c , $X[2]$, 15);
ripemd-ff (c , d , e , a , b , $X[3]$, 12);
ripemd-ff (b , c , d , e , a , $X[4]$, 5);
ripemd-ff (a , b , c , d , e , $X[5]$, 8);
ripemd-ff (e , a , b , c , d , $X[6]$, 7);
ripemd-ff (d , e , a , b , c , $X[7]$, 9);
ripemd-ff (c , d , e , a , b , $X[8]$, 11);

ripemd-ff (b , c , d , e , a , $X[9]$, 13);
ripemd-ff (a , b , c , d , e , $X[10]$, 14);
ripemd-ff (e , a , b , c , d , $X[11]$, 15);
ripemd-ff (d , e , a , b , c , $X[12]$, 6);
ripemd-ff (c , d , e , a , b , $X[13]$, 7);
ripemd-ff (b , c , d , e , a , $X[14]$, 9);
ripemd-ff (a , b , c , d , e , $X[15]$, 8);
ripemd-gg (e , a , b , c , d , $X[7]$, 7);
ripemd-gg (d , e , a , b , c , $X[4]$, 6);
ripemd-gg (c , d , e , a , b , $X[13]$, 8);
ripemd-gg (b , c , d , e , a , $X[1]$, 13);
ripemd-gg (a , b , c , d , e , $X[10]$, 11);
ripemd-gg (e , a , b , c , d , $X[6]$, 9);
ripemd-gg (d , e , a , b , c , $X[15]$, 7);
ripemd-gg (c , d , e , a , b , $X[3]$, 15);
ripemd-gg (b , c , d , e , a , $X[12]$, 7);
ripemd-gg (a , b , c , d , e , $X[0]$, 12);
ripemd-gg (e , a , b , c , d , $X[9]$, 15);
ripemd-gg (d , e , a , b , c , $X[5]$, 9);
ripemd-gg (c , d , e , a , b , $X[2]$, 11);
ripemd-gg (b , c , d , e , a , $X[14]$, 7);
ripemd-gg (a , b , c , d , e , $X[11]$, 13);
ripemd-gg (e , a , b , c , d , $X[8]$, 12);
ripemd-hh (d , e , a , b , c , $X[3]$, 11);
ripemd-hh (c , d , e , a , b , $X[10]$, 13);
ripemd-hh (b , c , d , e , a , $X[14]$, 6);
ripemd-hh (a , b , c , d , e , $X[4]$, 7);
ripemd-hh (e , a , b , c , d , $X[9]$, 14);
ripemd-hh (d , e , a , b , c , $X[15]$, 9);
ripemd-hh (c , d , e , a , b , $X[8]$, 13);
ripemd-hh (b , c , d , e , a , $X[1]$, 15);
ripemd-hh (a , b , c , d , e , $X[2]$, 14);
ripemd-hh (e , a , b , c , d , $X[7]$, 8);
ripemd-hh (d , e , a , b , c , $X[0]$, 13);
ripemd-hh (c , d , e , a , b , $X[6]$, 6);
ripemd-hh (b , c , d , e , a , $X[13]$, 5);
ripemd-hh (a , b , c , d , e , $X[11]$, 12);
ripemd-hh (e , a , b , c , d , $X[5]$, 7);
ripemd-hh (d , e , a , b , c , $X[12]$, 5);
ripemd-ii (c , d , e , a , b , $X[1]$, 11);
ripemd-ii (b , c , d , e , a , $X[9]$, 12);
ripemd-ii (a , b , c , d , e , $X[11]$, 14);
ripemd-ii (e , a , b , c , d , $X[10]$, 15);
ripemd-ii (d , e , a , b , c , $X[0]$, 14);
ripemd-ii (c , d , e , a , b , $X[8]$, 15);
ripemd-ii (b , c , d , e , a , $X[12]$, 9);

ripemd-ii (a , b , c , d , e , $X[4]$, 8);
ripemd-ii (e , a , b , c , d , $X[13]$, 9);
ripemd-ii (d , e , a , b , c , $X[3]$, 14);
ripemd-ii (c , d , e , a , b , $X[7]$, 5);
ripemd-ii (b , c , d , e , a , $X[15]$, 6);
ripemd-ii (a , b , c , d , e , $X[14]$, 8);
ripemd-ii (e , a , b , c , d , $X[5]$, 6);
ripemd-ii (d , e , a , b , c , $X[6]$, 5);
ripemd-ii (c , d , e , a , b , $X[2]$, 12);
ripemd-jj (b , c , d , e , a , $X[4]$, 9);
ripemd-jj (a , b , c , d , e , $X[0]$, 15);
ripemd-jj (e , a , b , c , d , $X[5]$, 5);
ripemd-jj (d , e , a , b , c , $X[9]$, 11);
ripemd-jj (c , d , e , a , b , $X[7]$, 6);
ripemd-jj (b , c , d , e , a , $X[12]$, 8);
ripemd-jj (a , b , c , d , e , $X[2]$, 13);
ripemd-jj (e , a , b , c , d , $X[10]$, 12);
ripemd-jj (d , e , a , b , c , $X[14]$, 5);
ripemd-jj (c , d , e , a , b , $X[1]$, 12);
ripemd-jj (b , c , d , e , a , $X[3]$, 13);
ripemd-jj (a , b , c , d , e , $X[8]$, 14);
ripemd-jj (e , a , b , c , d , $X[11]$, 11);
ripemd-jj (d , e , a , b , c , $X[6]$, 8);
ripemd-jj (c , d , e , a , b , $X[15]$, 5);
ripemd-jj (b , c , d , e , a , $X[13]$, 6);
ripemd-jjj (A , B , C , D , E , $X[5]$, 8);
ripemd-jjj (E , A , B , C , D , $X[14]$, 9);
ripemd-jjj (D , E , A , B , C , $X[7]$, 9);
ripemd-jjj (C , D , E , A , B , $X[0]$, 11);
ripemd-jjj (B , C , D , E , A , $X[9]$, 13);
ripemd-jjj (A , B , C , D , E , $X[2]$, 15);
ripemd-jjj (E , A , B , C , D , $X[11]$, 15);
ripemd-jjj (D , E , A , B , C , $X[4]$, 5);
ripemd-jjj (C , D , E , A , B , $X[13]$, 7);
ripemd-jjj (B , C , D , E , A , $X[6]$, 7);
ripemd-jjj (A , B , C , D , E , $X[15]$, 8);
ripemd-jjj (E , A , B , C , D , $X[8]$, 11);
ripemd-jjj (D , E , A , B , C , $X[1]$, 14);
ripemd-jjj (C , D , E , A , B , $X[10]$, 14);
ripemd-jjj (B , C , D , E , A , $X[3]$, 12);
ripemd-jjj (A , B , C , D , E , $X[12]$, 6);
ripemd-iii (E , A , B , C , D , $X[6]$, 9);
ripemd-iii (D , E , A , B , C , $X[11]$, 13);
ripemd-iii (C , D , E , A , B , $X[3]$, 15);
ripemd-iii (B , C , D , E , A , $X[7]$, 7);
ripemd-iii (A , B , C , D , E , $X[0]$, 12);

ripemd-iii (E , A , B , C , D , $X[13]$, 8);
ripemd-iii (D , E , A , B , C , $X[5]$, 9);
ripemd-iii (C , D , E , A , B , $X[10]$, 11);
ripemd-iii (B , C , D , E , A , $X[14]$, 7);
ripemd-iii (A , B , C , D , E , $X[15]$, 7);
ripemd-iii (E , A , B , C , D , $X[8]$, 12);
ripemd-iii (D , E , A , B , C , $X[12]$, 7);
ripemd-iii (C , D , E , A , B , $X[4]$, 6);
ripemd-iii (B , C , D , E , A , $X[9]$, 15);
ripemd-iii (A , B , C , D , E , $X[1]$, 13);
ripemd-iii (E , A , B , C , D , $X[2]$, 11);
ripemd-hhh (D , E , A , B , C , $X[15]$, 9);
ripemd-hhh (C , D , E , A , B , $X[5]$, 7);
ripemd-hhh (B , C , D , E , A , $X[1]$, 15);
ripemd-hhh (A , B , C , D , E , $X[3]$, 11);
ripemd-hhh (E , A , B , C , D , $X[7]$, 8);
ripemd-hhh (D , E , A , B , C , $X[14]$, 6);
ripemd-hhh (C , D , E , A , B , $X[6]$, 6);
ripemd-hhh (B , C , D , E , A , $X[9]$, 14);
ripemd-hhh (A , B , C , D , E , $X[11]$, 12);
ripemd-hhh (E , A , B , C , D , $X[8]$, 13);
ripemd-hhh (D , E , A , B , C , $X[12]$, 5);
ripemd-hhh (C , D , E , A , B , $X[2]$, 14);
ripemd-hhh (B , C , D , E , A , $X[10]$, 13);
ripemd-hhh (A , B , C , D , E , $X[0]$, 13);
ripemd-hhh (E , A , B , C , D , $X[4]$, 7);
ripemd-hhh (D , E , A , B , C , $X[13]$, 5);
ripemd-ggg (C , D , E , A , B , $X[8]$, 15);
ripemd-ggg (B , C , D , E , A , $X[6]$, 5);
ripemd-ggg (A , B , C , D , E , $X[4]$, 8);
ripemd-ggg (E , A , B , C , D , $X[1]$, 11);
ripemd-ggg (D , E , A , B , C , $X[3]$, 14);
ripemd-ggg (C , D , E , A , B , $X[11]$, 14);
ripemd-ggg (B , C , D , E , A , $X[15]$, 6);
ripemd-ggg (A , B , C , D , E , $X[0]$, 14);
ripemd-ggg (E , A , B , C , D , $X[5]$, 6);
ripemd-ggg (D , E , A , B , C , $X[12]$, 9);
ripemd-ggg (C , D , E , A , B , $X[2]$, 12);
ripemd-ggg (B , C , D , E , A , $X[13]$, 9);
ripemd-ggg (A , B , C , D , E , $X[9]$, 12);
ripemd-ggg (E , A , B , C , D , $X[7]$, 5);
ripemd-ggg (D , E , A , B , C , $X[10]$, 15);
ripemd-ggg (C , D , E , A , B , $X[14]$, 8);
ripemd-fff (B , C , D , E , A , $X[12]$, 8);
ripemd-fff (A , B , C , D , E , $X[15]$, 5);
ripemd-fff (E , A , B , C , D , $X[10]$, 12);

ripemd-fff (D , E , A , B , C , $X[4]$, 9);
ripemd-fff (C , D , E , A , B , $X[1]$, 12);
ripemd-fff (B , C , D , E , A , $X[5]$, 5);
ripemd-fff (A , B , C , D , E , $X[8]$, 14);
ripemd-fff (E , A , B , C , D , $X[7]$, 6);
ripemd-fff (D , E , A , B , C , $X[6]$, 8);
ripemd-fff (C , D , E , A , B , $X[2]$, 13);
ripemd-fff (B , C , D , E , A , $X[13]$, 6);
ripemd-fff (A , B , C , D , E , $X[14]$, 5);
ripemd-fff (E , A , B , C , D , $X[0]$, 15);
ripemd-fff (D , E , A , B , C , $X[3]$, 13);
ripemd-fff (C , D , E , A , B , $X[9]$, 11);
ripemd-fff (B , C , D , E , A , $X[11]$, 11);
let $a' :: b' :: c' :: d' :: e' :: \mathsf{T} = Q$ **in**
let $A' =$ ripemd-mask1 ($b' + c + D$) **in**
let $B' =$ ripemd-mask1 ($c' + d + E$) **in**
let $C' =$ ripemd-mask1 ($d' + e + A$) **in**
let $D' =$ ripemd-mask1 ($e' + a + B$) **in**
let $E' =$ ripemd-mask1 ($a' + b + C$) **in**
$\langle A', B', C', D', E' \rangle$]

10.8 Conversion of buffer to array

[ripemd-buffer2array (B) $\stackrel{\bullet\bullet}{=}$
ripemd-buffer2array1 (B , 0 , T)]

[ripemd-buffer2array1 (B , n , X) $\stackrel{\bullet\bullet}{=}$
if $n \geq 16$ **then** $X :: B$ **else**
let $c =$ ripemd-mask1 (vt2vector (list-prefix (B , 4))) **in**
let $B =$ list-suffix (B , 4) **in**
let $X = X[n \rightarrow c]$ **in**
ripemd-buffer2array1 (B , $n + 1$, X)]

[ripemd-chunk $\stackrel{\bullet\bullet}{=}$ 64]

[ripemd-pad $\stackrel{\bullet\bullet}{=}$ repeat (ripemd-chunk , NULL)]

10.9 Conversion of buffer to Ripemd-160 code

[ripemd (B) $\stackrel{\text{Late}}{\rightarrow\rightarrow}$ norm $B \in \mathbf{V}$:
let $B =$ vt2vector$*$ (B) **in**
let $l =$ length (B) **in**
let $Q =$ ripemd1 (l , l , ripemd-init , B) **in**
ripemd2 (Q)]

Convert the singleton list B to a Ripemd-160 hash code. The hash code is also expressed as a singleton list.

[ripemds (s) $\stackrel{\bullet\bullet}{=}$
vt2vector (lgc-string2mixed (ripemd (s)))]

Same as above but takes a string as input and delivers a string in mixed endian hexadecimal as output.

[ripemd1 (L , l , Q , B) $\stackrel{\bullet\bullet}{=}$
if $l <$ ripemd-chunk **then** ripemd-finish (L , l , Q , B) **else let**
$X :: B =$ ripemd-buffer2array (B) **in**
let $Q =$ ripemd-compress (Q , X) **in**
ripemd1 (l , $l -$ ripemd-chunk , Q , B)]

Successively compress the buffer B into the quintuple Q. L and l are the original and remaining length of B, respectively.

[ripemd-finish (L , l , Q , B) $\stackrel{\bullet\bullet}{=}$
let $B =$ append (B , V128 :: ripemd-pad) **in**
let $X :: B =$ ripemd-buffer2array (B) **in**
if $l < 56$ **then** ripemd-compress (Q , ripemd-length (X , L)) **else**
let $Q =$ ripemd-compress (Q , X) **in**
let $X :: B =$ ripemd-buffer2array (ripemd-pad) **in**
ripemd-compress (Q , ripemd-length (X , L))]

Pad the buffer B with an end byte followed by zeros, put the original length of B in $X[14]$ and $X[15]$, and do a final compress. If B plus end byte occupies all or part of $X[14]$ and $X[15]$ then B is padded with an additional chunk of zeros.

[ripemd-length (X , L) $\stackrel{\bullet\bullet}{=}$
let $X = X[14 \rightarrow$ ripemd-mask1 (ash (L , 3))] **in** $X[15 \rightarrow$ ripemd-mask1 (ash (L , $^{-}29$))]]

Place the original length L of the buffer in $X[14]$ and $X[15]$.

[ripemd2 (Q) $\stackrel{\bullet\bullet}{=}$
if Q **then** $\top$ **else**
let $C :: Q = Q$ **in**
let $C =$ vt2vector$*$ ($C +$ ash (1 , 32)) **in**
append (C , ripemd2 (Q))]

Convert quintuple Q to a list of singleton strings.

Index

abstraction, lambda, 4
application, functional, 4
apply, 4
aspect, 5
aspect declaration, 5
aspect, claim, 7
aspect, execute, 8
aspect, logiweb charge, 6
aspect, logiweb name, 6
aspect, macro, 7
aspect, message, 6
aspect, priority, 7
aspect, render, 7
aspect, tex show, 5, 7
aspect, tex use, 5, 6
aspect, unpack, 7
aspect, user, 8
aspect, value, 5, 6

base page, 3
Boolean type, 16
boot handler, 52
bottom term, 10
byte, 30
byte tree, 23

cache, 12, 52
charge aspect, logiweb, 6
claim, 7
claim aspect, 7
compiler, Logiweb, 5, 9
computing engine, Logiweb, 10
construct, if-then-else, 4
construct, page, 7, 31

debugging information, 30
declaration, aspect, 5
define, 5
definition, 5

eager, 13
engine, Logiweb computing, 10
event, 52
exception type, 16
execute, 8
execute aspect, 8

falsehood, 12
form, normal, 10
function normal form, 10
function term, 10
functional application, 4

handler, boot, 52
harvesting, 5
hide, 5
home page, 30

if, 4
if-then-else construct, 4
index, 8, 16, 30
information, debugging, 30
input message, 52
integer type, 16
interface, 52
introduce, 5

lambda, 4
lambda abstraction, 4
lgccharge, 6
lgcname, 6
Logiweb, 3
logiweb charge aspect, 6
Logiweb compiler, 5, 9
Logiweb computing engine, 10
Logiweb machine, 52
logiweb name aspect, 6
Logiweb page, 3
Logiweb server, 9
Logiweb vector, 7

machine, Logiweb, 52
macro, 7
macro aspect, 7
map, 11
map theory, 11
map type, 16
message, 6

message aspect, 6
message, index, 52
message, output, 52

name, 6
name aspect, logiweb, 6
natural number, 14
natural, untagged, 15
naturals, 14
normal form, 10
number, natural, 14

output message, 52

page construct, 7, 31
page, home, 30
page, Logiweb, 3
pair type, 16
pairing, 12
post, 9
postassociative, 9
pre, 9
preassociative, 9
predefined type, 16
priority, 7
priority aspect, 7
proclaim, 4
proclamation, 4

quote, 4
quoting, 30

reduce, 10
reference, 8, 16, 30
render, 7
render aspect, 7
reply, 52
representation, untagged, 14
request, 52
revelation, 5
root normal form, 10

server, Logiweb, 9
show aspect, tex, 5, 7
show, tex, 7
stack, 31
strict, 13

tag, 16
tagged value, 16
tex show, 7
tex show aspect, 5, 7
tex use, 6
tex use aspect, 5, 6
theory, map, 11
tree, byte, 23
tree, vector, 23
true, 4
true term, 10
truth, 4
truth normal form, 10
type, boolean, 16
type, exception, 16
type, integer, 16
type, map, 16
type, pair, 16
type, predefined, 16

unpack, 7
unpack aspect, 7
untagged natural, 15
untagged representation, 14
use aspect, tex, 5, 6
use, tex, 6
user aspect, 8

value, 6
value aspect, 5, 6
value, tagged, 16
vector, 22
vector tree, 23
vector, Logiweb, 7
visibility, 47

References

[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] 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.

[3] K. Grue. A logiweb base page - chores. Technical report, Logiweb, 2006. ../../../logiweb/01AB1F51C8C17606A5C0331B5689B4858C796547B9A0A4AEF0BCB2BB0806/page/appendix.pdf.