# A Logiweb proof checker

Klaus Grue

GRD-2009-11-28.UTC:11:16:43.699221

## Contents

# 1 Introduction

*Logiweb* [1] 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) defines a general purpose proof checker.

## 1.1 Electronic appendices

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

- ../../../logiweb/014E93CEDBCA44EB611BC0974861950432277A602795E9B4F2BAD8BB0806/page/page.pdf
- ../../../logiweb/014E93CEDBCA44EB611BC0974861950432277A602795E9B4F2BAD8BB0806/page/chores.pdf [3]
- ../../../logiweb/014E93CEDBCA44EB611BC0974861950432277A602795E9B4F2BAD8BB0806/page/index.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 Referenced Logiweb pages

The present Logiweb page references the following other Logiweb page:

- ../../../logiweb/01AB1F51C8C17606A5C0331B5689B4858C796547B9A0A4AEF0BCB2BB0806/page/chores.pdf [2]

## 1.3 Bed page

The present page may be used as 'bed' page, i.e. as the first reference of other pages. Pages which use the present page as bed page and which do not define a verifier of their own use the verifier of the present page:

[**Verifier**:test1 $\wedge_c$ proofcheck]

The verifier is a conjunction of proofcheck which is the proof checker defined on the present page and test1 which is defined on the bed page of the present page. The test1 verifier executes test suites and supports testing of computable functions.

Pages which use the present page as bed page and which do not define a macro expander of their own use the macro expander of the present page:

[**Expander**:macro1]

The expander is nothing but a reexport of the expander defined on the bed page of the present page.

# 2 Terms

## 2.1 Aspect declarations

We shall use several user aspects for implementing the proof checker.

Logiweb has a list of predefined aspects like 'value', 'name', 'use', and so on. Predefined aspects are built up from small letters and space characters. The list of predefined aspects may grow with time, so user defined aspects must contain at least one character which is neither a small letter, nor a space character.

Furthermore, to reduce the risk of name conflicts, one should give any user aspect a personal touch. The aspects defined in the following all end with a slash and the initials of the present author. If two authors use the same aspect for different things, then a third author may run into trouble when referencing both of the two first authors.

Logiweb has means for defining guaranteed unique aspects, but we shall not use that here. The aspects to be used are:

[message stmt : 'statement/kg'] defines 'stmt' to represent the *statement aspect*. The present page uses the statement aspect for expressing lemmas, theories, axioms, inference rules, and conjectures.

[message proof : 'proof/kg'] defines 'proof' to represent the *proof* user aspect. The present page uses the proof aspect for expressing formal proofs.

[message meta : 'meta/kg'] defines 'meta' to represent the *meta* user aspect. The present page uses the meta aspect for declaring constructs to be meta-variables, meta-quantifiers and sequent operators.

[message math : 'math/kg'] defines 'math' to represent the *math* user aspect. The semantics of this aspect is defined by individual axiomatic systems, the intension being that definition of new function letters are done using this aspect.

[message tactic : 'tactic/kg'] defines 'tactic' to represent the *tactic* user aspect. The present page uses the tactic aspect for specifying how proofs should be translated into sequent terms.

[message unitac : 'unitac/kg'] defines 'unitac' to represent the *unitac* user aspect. The unitac aspect defines a special kind of tactic which may occur inside arguments of the 'unification tactic'. The unification tactic is described later.

[message locate : 'locate/kg'] defines 'locate' to represent the *locate* user aspect. The present page uses the locate aspect for locating the source of errors for the sake of diagnostic messages.

## 2.2 Definition constructs

The following constructs allow to define various aspects of a construct $x$.

$[[x \overset{\text{stmt}}{=} y] \doteq [x \overset{\text{stmt}}{\rightarrow} y]]$.

$[[x \overset{\text{proof}}{=} y] \doteq [x \overset{\text{proof}}{\rightarrow} y]]$.

$[[x \overset{\text{meta}}{=} y] \doteq [x \overset{\text{meta}}{\rightarrow} y]]$.

$[[x \overset{\text{math}}{=} y] \doteq [x \overset{\text{math}}{\rightarrow} y]]$.

$[[x \overset{\text{tactic}}{=} y] \doteq [x \overset{\text{tactic}}{\rightarrow} y]]$.

$[[x \overset{\text{unitac}}{=} y] \doteq [x \overset{\text{unitac}}{\rightarrow} y]]$.

$[[x \overset{\text{locate}}{=} y] \doteq [\text{root protect}(x) \overset{\text{locate}}{\rightarrow} y]]$.

In 'locate' definitions, the construct being defined is protected against macro expansion. This is important because errors are located in the body of a page, i.e. in the page before macro expansion whereas locate definitions are interpreted after macro expansion. So protection is needed in cases where one needs to assign a locate aspect to some construct which also has a macro definition.

## 2.3 Axioms, rules, theories, and lemmas

Axioms, inference rules, theories, and lemmas are all represented by statement definitions. As an example, a lemma $L$ saying that any statement infers itself could read

**Lemma** $L$: $\mathsf{a} \vdash \mathsf{a}$ □

which is simply shorthand for $[L \overset{\text{stmt}}{=} \mathsf{a} \vdash \mathsf{a}]$. The lemma construct is defined thus: [**Lemma** $x$: $y$ □ $\ddot{=}$ $[x \overset{\text{stmt}}{=} y]$].

A theory $T$ comprising an inference rule named MP and three axioms A1, A2, and A3 could read

**Theory** $T$: MP $\oplus$ A1 $\oplus$ A2 $\oplus$ A3 □

which is shorthand for $[T \overset{\text{stmt}}{=} \text{MP} \oplus \text{A1} \oplus \text{A2} \oplus \text{A3}]$. The theory construct is defined thus: [**Theory** $x$: $y$ □ $\ddot{=}$ $[x \overset{\text{stmt}}{\rightarrow} y]$].

An axiom A1 which says that $\mathsf{a} \Rightarrow \mathsf{b} \Rightarrow \mathsf{a}$ for all terms $\mathsf{a}$ and $\mathsf{b}$ could read

**Axiom** A1: $\Pi \mathsf{a}, \mathsf{b}$: $A \Rightarrow B \Rightarrow A$ □

This is shorthand for $[\text{A1} \overset{\text{stmt}}{=} \Pi \mathsf{a}, \mathsf{b}\text{:}\ \mathsf{a} \Rightarrow \mathsf{b} \Rightarrow \mathsf{a}]$ plus one more definition. The second definition is a unitac definition which explains how to prove the axiom. This may seem strange since, from a human point of view, axioms are assumed rather than proved. But in the present system, axioms actually do have to be proved, and the proof even depends on context. As an example, if axiom A1 occurs in a proof which is relative to the theory $T$ above, then the proof of A1 will be one which accesses the second element of $T$. In a proof relative to A1 $\oplus$ A2 $\oplus$ A3 $\oplus$ MP, the proof would be one which accesses the first element.

A working logician need not care about how axioms are proved backstage. And need not even know that axioms do have to be proved. The working logician just needs to state proofs in a style close to that of [4] and the definitions on the present page will take care of the rest.

The axiom construct is defined thus:

$$[\textbf{Axiom}\ x\text{:}\ y\ \square \ddot{=} [x \overset{\text{stmt}}{\rightarrow} y][x \overset{\text{unitac}}{=} \lambda u.\text{unitac-rule}\ (\ u\ )]]$$

When an axiom like A1 occurs in a proof line which is expanded using the unification tactic, then the unification tactic evaluates unitac-rule ( $\langle t, s, c \rangle$ ) where $t$ is A1, $s$ is a 'state' which contains local information, and $c$ is the cache of the page on which the proof occurs. Among other, the state $s$ contains information about which theories may be used and which hypotheses have been assumed. The value of unitac-rule ( $\langle t, s, c \rangle$ ) is a proof of $\Pi \mathsf{a}, \mathsf{b}\text{:}\ \mathsf{a} \Rightarrow \mathsf{b} \Rightarrow \mathsf{a}$ which the unification tactic will then try to instantiate suitably.

From the point of view of the proof checker, there is no difference between axioms and inference rules:

$$[\textbf{Rule}\ x\text{:}\ y\ \square \ddot{=} [x \overset{\text{stmt}}{\rightarrow} y][x \overset{\text{unitac}}{=} \lambda u.\text{unitac-rule}\ (\ u\ )]]$$

## 2.4 Meta variables

We use the meta aspect to declare that a construct is a meta variable. In general, several aspects are used when finding out what a construct means: Constructs which have a meta aspect mean whatever that aspect says they mean. Constructs without a meta aspect mean whatever the value aspect says they mean. Constructs with neither meta nor value aspect mean whatever the def aspect says they mean. Constructs with no meta, value, or def aspect are considered to be object variables. So e.g. $a$, $X$, and $\beta$ are object variables.

A construct is a meta variable if its meta aspect equals 'var'. The following construct allows to declare meta variables:

[meta $x \doteq [x \overset{\text{meta}}{=} \text{'var'}]$]

To have an arsenal of meta variables, we declare $\mathsf{a}$, $\mathsf{b}$, and so on to be meta variables: meta $\mathsf{a}$, meta $\mathsf{b}$, meta $\mathsf{c}$, meta $\mathsf{d}$, meta $\mathsf{e}$, meta $\mathsf{f}$, meta $\mathsf{g}$, meta $\mathsf{h}$, meta $\mathsf{i}$, meta $\mathsf{j}$, meta $\mathsf{k}$, meta $\mathsf{l}$, meta $\mathsf{m}$, meta $\mathsf{n}$, meta $\mathsf{o}$, meta $\mathsf{p}$, meta $\mathsf{q}$, meta $\mathsf{r}$, meta $\mathsf{s}$, meta $\mathsf{t}$, meta $\mathsf{u}$, meta $\mathsf{v}$, meta $\mathsf{w}$, meta $\mathsf{x}$, meta $\mathsf{y}$, and meta $\mathsf{z}$.

A term is a meta variable, if the meta aspect of the root of the term equals 'var'. As an example of use, we declare the unary construct $x^{\#}$ to be a meta variable:

meta $x^{\#}$

The construct above allows to use e.g. $a^{\#}$, $\beta^{\#}$, $c'^{\#}$, and $d_5{}^{\#}$ as meta variables.

Two meta variable are identical if they are identical as terms (i.e. w.r.t. $x \overset{\text{t}}{=} y$). As a peculiarity this entails that $(2+2)^{\#}$ and $4^{\#}$ are distinct meta variables.

We shall need two more indexed meta variables: meta tacfresh ( $x$ ) and meta unifresh ( $x$ , $y$ ). The unification tactic defined later uses these indexed meta variables to generate variables that are supposed to be 'fresh'. If you use these indexed meta variables in proofs, you ask for trouble.

## 2.5 Cache accessors

Given a construct $x$ and a cache $c$, the following functions return various information about $x$. The last construct below decides whether or not a construct $x$ is a meta variable.

[$x$ plist ( $c$ ) $\overset{\bullet}{=}$ $c[x^{\mathrm{r}}][\text{codex}][x^{\mathrm{r}}][x^{\mathrm{i}}][0]$]. Given a symbol $x$, this construct looks up an array which maps strings to the associated aspects.

[$x$ def ( $c$ , $a$ ) $\overset{\bullet}{=}$ $x$ plist ( $c$ )$[a]$]. Look up aspect $a$ of symbol $x$.

[$x$ lhs ( $c$ , $a$ ) $\overset{\bullet}{=}$ $x$ def ( $c$ , $a$ )$^2$]. Look up the left hand side of the aspect $a$ definition of symbol $x$.

[$x$ rhs ( $c$ , $a$ ) $\overset{\bullet}{=}$ $x$ def ( $c$ , $a$ )$^3$]. Look up the right hand side of the aspect $a$ definition of symbol $x$.

[$x$ metadef ( $c$ ) $\overset{\bullet}{=} x$ rhs ( $c$ , meta )$^{\mathrm{i}}$]. Look up the meta definition of symbol $x$.

[$x$ metavar ( $c$ ) $\overset{\bullet}{=} x$ metadef ( $c$ ) = var]. True if $x$ is a meta variable.

[$x$ stmt-rhs ( $c$ ) $\overset{\bullet}{=} x$ rhs ( $c$ , stmt )]. Look up the statement definition of symbol $x$.

[$x$ proof-rhs ( $c$ ) $\overset{\bullet}{=} x$ rhs ( $c$ , proof )]. Look up the statement definition of symbol $x$.

[$x$ tactic-rhs ( $c$ ) $\overset{\bullet}{=} x$ rhs ( $c$ , tactic )]. Look up the statement definition of symbol $x$.

[$x$ unitac-rhs ( $c$ ) $\overset{\bullet}{=} x$ rhs ( $c$ , unitac )]. Look up the statement definition of symbol $x$.

[$x$ locate-rhs ( $c$ ) $\overset{\bullet}{=} x$ rhs ( $c$ , locate )]. Look up the statement definition of symbol $x$.

## 2.6 Meta operators

In addition to meta variables, the five operators for building meta terms of the Logiweb sequent calculus are:

[$x \vdash y \overset{\text{meta}}{=}$ 'infer']. States that provability of $x$ implies provability of $y$.

[$x \Vdash y \overset{\text{meta}}{=}$ 'endorse'] States that if $x$ evaluates to $\mathsf{T}$ then $y$ is provable.

[$x \oplus y \overset{\text{meta}}{=}$ 'plus'] States that both $x$ and $y$ are provable.

[$\Pi x\colon y \overset{\text{meta}}{=}$ 'all']. States that $y$ is provable for all $x$

[$\perp\!\!\!\perp \overset{\text{meta}}{=}$ 'false']. Expresses absurdity. One may disprove $x$ is by proving $x \vdash \perp\!\!\!\perp$

## 2.7 Meta constructs

We shall say that a construct is a *meta construct* iff the construct has a meta definition. In other words we shall say that a construct $x$ is a meta construct iff $x$ metadef ( self ) $\neq \mathsf{T}$.

We shall say that a meta construct is a *meta variable* iff it is meta defined as 'var'. In other words, $x$ is a meta variable iff $x$ metadef ( self ) = var.

We shall say that a meta construct $x$ is a *meta binder* iff $x$ metadef ( self ) = all.

We shall say that a meta construct is a *meta operator* if it is neither a meta variable nor a meta binder.

## 2.8 Object constructs

We shall say that a construct is an *object construct* iff it is not a meta construct. So $x$ is an object construct iff $x$ metadef ( self ) = $\mathsf{T}$

We shall say that an object construct is an *object binder* if the construct is declared to denote lambda abstraction. So $x$ is an object binder iff self$[x^{\mathrm{r}}]$[code]$[x^{\mathrm{i}}] = 0$.

We shall say that an object construct $x$ is a *quote* if the construct is declared to denote quoting. So $x$ is a quote iff self$[x^{\mathrm{r}}]$[code]$[x^{\mathrm{i}}] = 1$.

We shall say that an object construct is an *object variable* if it neither has a value nor a math definition.

We shall say that object constructs other than object variables, object binders, and quotes are *object operators*.

## 2.9 Object terms

We shall say that a term is an *object term* if it is built up from the following:

- object operators applied to object terms.
- quotes applied to arbitrary terms.
- object variables applied to arbitrary terms (i.e. object variables with arbitrary indices).
- object binders applied to an object or meta variable and an object term in that order.
- meta variables applied to arbitrary terms (i.e. meta variables with arbitrary indices).

In other words, a term $x$ is an object term iff $x$ objectterm ( self ) where we define $x$ objectterm ( $c$ ) in the following.

[$x$ valuedef ( $c$ ) $\stackrel{\bullet}{=}$ $c[x^{\mathrm{r}}]$[code]$[x^{\mathrm{i}}]$]. The value definition of $x$. This is 0 for lambda abstractions, 1 for quotes, and $\mathsf{T}$ for constructs with no value definition.

[$x$ mathdef ( $c$ ) $\stackrel{\bullet}{=}$ $x$ rhs ( $c$ , math )$^{\mathrm{i}}$]. The math definition of $x$. This is $\mathsf{T}$ for constructs with no math definition.

[$x$ objectvar ( $c$ ) $\stackrel{\bullet}{=}$ $x$ metadef ( $c$ ) = $\mathsf{T}$ **and** $x$ valuedef ( $c$ ) = $\mathsf{T}$ **and** $x$ mathdef ( $c$ ) = $\mathsf{T}$]. This is true if $x$ is an object variable.

[$x$ objectlambda ( $c$ ) $\stackrel{\bullet}{=}$ $x$ metadef ( $c$ ) = $\mathsf{T}$ **and** $x$ valuedef ( $c$ ) = 0]. This is true if $x$ is a lambda abstraction.

[$x$ objectquote ( $c$ ) $\stackrel{\bullet}{=}$ $x$ metadef ( $c$ ) = $\mathsf{T}$ **and** $x$ valuedef ( $c$ ) = 1]. This is true if $x$ is a lambda abstraction.

[$x$ objectterm ( $c$ ) $\stackrel{\bullet}{=}$
**let** $d = x$ metadef ( $c$ ) **in**
**if** $d =$ var **then** $\mathsf{T}$ **else**
**if** $d \neq \mathsf{T}$ **then** $\mathsf{F}$ **else**
**let** $d = x$ valuedef ( $c$ ) **in**
**if** $d = 0$ **then** ($x^1$ objectvar ( $c$ ) **or** $x^1$ metavar ( $c$ )) **and** $x^2$ objectterm ( $c$ ) **else**
**if** $d = 1$ **then** $\mathsf{T}$ **else if** $d$ **and** $x$ mathdef ( $c$ ) **then** $\mathsf{T}$ **else** $x^{\mathrm{t}}$ objectterm$*$ ( $c$ )]

[$x$ objectterm$*$ ( $c$ ) $\stackrel{\bullet}{=} x \in \mathbf{A}$ **or** $x^{\mathrm{h}}$ objectterm ( $c$ ) **and** $x^{\mathrm{t}}$ objectterm$*$ ( $c$ )].

## 2.10 Meta terms

We shall say that a term is a *meta term* if it is built up from the following:

- object terms.
- meta operators applied to meta terms.
- meta binders applied to a meta variable and a meta term in that order.

In other words, a term $x$ is a meta term iff $x$ metaterm ( self ) where we define $x$ metaterm ( $c$ ) in the following.

[$x$ metaterm ( $c$ ) $\stackrel{\bullet}{=}$
**let** $d = x$ metadef ( $c$ ) **in**
**if** $d =$ var **then** $\mathsf{T}$ **else**
**if** $d =$ all **then** $x^1$ metavar ( $c$ ) **and** $x^2$ metaterm ( $c$ ) **else**
**if** $d \neq \mathsf{T}$ **then** $x^{\mathrm{t}}$ metaterm$*$ ( $c$ ) **else** $x$ objectterm ( $c$ )]

[$x$ metaterm$*$ ( $c$ ) $\stackrel{\bullet}{=} x \in \mathbf{A}$ **or** $x^{\mathrm{h}}$ metaterm ( $c$ ) **and** $x^{\mathrm{t}}$ metaterm$*$ ( $c$ )]

## 2.11 Avoidance

We shall say that a variable $x$ *avoids* a term $t$ if $x$ does not occur free in $t$. The meta version of avoidance is defined thus:

[$x$ metaavoid ( $c$ ) $y \stackrel{\bullet}{=} x$ metavar ( $c$ ) **and** metaavoid1 ( $x$ , $y$ , $c$ )]. True if $x$ is a meta variable and $x$ does not occur free in the meta term $y$.

[$x$ metaavoid$*$ ( $c$ ) $y \stackrel{\bullet}{=}$
**if** $y \in \mathbf{A}$ **then** $\mathsf{T}$ **else**
**if** $x$ metaavoid ( $c$ ) ($y^{\mathrm{h}}$) **then** $x$ metaavoid$*$ ( $c$ ) ($y^{\mathrm{t}}$) **else** $y^{\mathrm{h}}$]. True if $x$ does not occur free in any element of the list $y$ of meta terms. Otherwise returns the first element of $y$ in which $x$ occurs free.

[metaavoid1 ( $x$ , $y$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $d = y$ metadef ( $c$ ) **in**
**if** $d =$ var **then not** $x \stackrel{\mathrm{t}}{=} y$ **else**
**if** $d =$ all **then** $x \stackrel{\mathrm{t}}{=} y^1$ **or** metaavoid1 ( $x$ , $y^2$ , $c$ ) **else**
**if** $d \neq \mathsf{T}$ **then** metaavoid1∗ ( $x$ , $y^{\mathrm{t}}$ , $c$ ) **else**
**let** $d = y$ valuedef ( $c$ ) **in**
**if** $d = 1$ **then** $\mathsf{T}$ **else** metaavoid1∗ ( $x$ , $y^{\mathrm{t}}$ , $c$ )]

[metaavoid1∗ ( $x$ , $y$ , $c$ ) $\stackrel{\bullet}{=}$ $y \in$ **A or** metaavoid1 ( $x$ , $y^{\mathrm{h}}$ , $c$ ) **and** metaavoid1∗ ( $x$ , $y^{\mathrm{t}}$ , $c$ )]

The object version of avoidance is defined in the following. We define a construct $x$ objectavoid∗ ( $c$ ) $y$ which is true if $x$ is a list of distinct object variables all of which avoid the term $y$.

[distinctvar ( $x$ , $c$ ) $\stackrel{\bullet}{=}$ $x \in$ **A or** $x^{\mathrm{h}}$ objectvar ( $c$ ) **and not** $x^{\mathrm{h}} \in x^{\mathrm{t}}$ **and** distinctvar ( $x^{\mathrm{t}}$ , $c$ )]. True if $x$ is a list of distinct meta variables.

[$x$ objectavoid∗ ( $c$ ) $y \stackrel{\bullet}{=}$ distinctvar ( $x$ , $c$ ) **and** (objectavoid1 ( $x$ , $y$ , $c$ , $\mathsf{T}$ )$^{\circ \mathrm{t}}$)]. True if $x$ is a list of object variables none of which occur free in the meta term $y$. If some element of $x$ definitely occur free in $y$ then the function returns $\mathsf{F}$. If the term contains meta variables then the function returns a list of meta variables which the variables in $x$ must avoid in order to avoid $y$.

[$x$ objectavoid ( $c$ ) $y \stackrel{\bullet}{=} \langle x \rangle$ objectavoid∗ ( $c$ ) $y$]. Same as above except that $x$ is a term rather than a list of terms.

[objectavoid1 ( $x$ , $y$ , $c$ , $r$ ) $\stackrel{\bullet}{=}$
**if** $x = \langle\,\rangle$ **then** $r$ **else**
**let** $d = y$ metadef ( $c$ ) **in**
**if** $d =$ var **then** $r$ set+ $y$ **else**
**let** $d = y$ valuedef ( $c$ ) **in**
**if** $d = \mathsf{T}$ **then if** $y \in x$ **then** $\mathsf{F}^{\bullet}$ **else** $r$ **else**
**if** $d = 0$ **then** objectavoid1 ( $x$ set- $y^1$ , $y^2$ , $c$ , $r$ ) **else**
**if** $d = 1$ **then** $r$ **else**
objectavoid1∗ ( $x$ , $y^{\mathrm{t}}$ , $c$ , $r$ )]

[objectavoid1∗ ( $x$ , $y$ , $c$ , $r$ ) $\stackrel{\bullet}{=}$ **if** $y \in$ **A then** $r$ **else** objectavoid1∗ ( $x$ , $y^{\mathrm{t}}$ , $c$ , objectavoid1 ( $x$ , $y^{\mathrm{h}}$ , $c$ , $r$ ) )]

## 2.12 Substitution freeness

We shall say that a substitution is *free* if one can perform the substitution without renaming. The meta and object versions of freeness are defined in the following.

[metafree ( $x$ , $y$ , $z$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $d = x$ metadef ( $c$ ) **in**
**if** $d =$ var **then** $\mathsf{T}$ **else**
**if** $d = \mathsf{T}$ **then** $z$ objectterm ( $c$ ) **or** $y$ metaavoid ( $c$ ) $x$ **else**
**if** $d \neq$ all **then** metafree$*$ ( $x^{\mathrm{t}}$ , $y$ , $z$ , $c$ ) **else**
**if** $y \stackrel{\mathrm{t}}{=} x^1$ **then** $\mathsf{T}$ **else**
**if** $y$ metaavoid ( $c$ ) $x^2$ **then** $\mathsf{T}$ **else**
$x^1$ metaavoid ( $c$ ) $z$ **and** metafree ( $x^2$ , $y$ , $z$ , $c$ )]. True if one can freely replace the meta variable $y$ by the meta term $z$ in the meta term $x$.

[metafree$*$ ( $x$ , $y$ , $z$ , $c$ ) $\stackrel{\bullet}{=}$ $x \in$ **A or** metafree ( $x^{\mathrm{h}}$ , $y$ , $z$ , $c$ ) **and** metafree$*$ ( $x^{\mathrm{t}}$ , $y$ , $z$ , $c$ )] True if one can freely replace the meta variable $y$ by the meta term $z$ in any element of the list $x$ of meta terms.

[objectfree ( $x$ , $y$ , $z$ , $c$ ) $\stackrel{\bullet}{=}$
**if** $x$ metadef ( $c$ ) $\neq \mathsf{T}$ **then** $\mathsf{F}$ **else**
**let** $d = x$ valuedef ( $c$ ) **in**
**if** $d = \mathsf{T}$ **then** $\mathsf{T}$ **else**
**if** $d = 1$ **then** $\mathsf{T}$ **else**
**if** $d \neq 0$ **then** objectfree$*$ ( $x^{\mathrm{t}}$ , $y$ , $z$ , $c$ ) **else**
**if** $y \stackrel{\mathrm{t}}{=} x^1$ **then** $\mathsf{T}$ **else**
**if** $y$ objectavoid ( $c$ ) $x^2$ **then** $\mathsf{T}$ **else**
$x^1$ objectavoid ( $c$ ) $z$ **and** objectfree ( $x^2$ , $y$ , $z$ , $c$ )]. True if one can freely replace the object variable $y$ by the object term $z$ in the object term $x$.

[objectfree$*$ ( $x$ , $y$ , $z$ , $c$ ) $\stackrel{\bullet}{=}$ $x \in$ **A or** objectfree ( $x^{\mathrm{h}}$ , $y$ , $z$ , $c$ ) **and** objectfree$*$ ( $x^{\mathrm{t}}$ , $y$ , $z$ , $c$ )]. True if one can freely replace the object variable $y$ by the object term $z$ in any element of the list $x$ of object terms.

## 2.13 Substitution

The function metasubst ( $x$ , $s$ , $c$ ) defined in the following substitutes variables with terms as specified in the association list $s$. Only free occurences of variables in $x$ are replaced. The function performs substitution without renaming of free variables and does not check whether or not the substitution if free. Substitution is parallel so that one may use e.g. $s = \langle \lceil x \rceil :: \lceil y \rceil, \lceil y \rceil :: \lceil x \rceil \rangle$ to swap $x$ and $y$.

[metasubst ( $x$ , $s$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $d = x$ metadef ( $c$ ) **in**
**if** $d =$ var **then** lookup ( $x$ , $s$ , $x$ ) **else**
**if** $d \neq$ all **then** $x^{\mathrm{h}} ::$ metasubst$*$ ( $x^{\mathrm{t}}$ , $s$ , $c$ ) **else**
**let** $s =$ remove ( $x^1$ , $s$ ) **in**
$\langle x^{\mathrm{h}}, x^1,$ metasubst ( $x^2$ , $s$ , $c$ ) $\rangle$].

[metasubst∗ ( $x$ , $s$ , $c$ ) $\stackrel{\bullet}{=}$
**if** $x \in$ **A then** $\top$ **else**
metasubst ( $x^{\mathrm{h}}$ , $s$ , $c$ ) : : metasubst∗ ( $x^{\mathrm{t}}$ , $s$ , $c$ )]

The following function is like metasubst ( $x$ , $s$ , $c$ ) but also checks that the substitution is free, that $z$ is a metaterm, and that if $x$ is a metaterm then so is the return value. Returns $\top$ on error.

[metasubstc ( $x$ , $s$ , $c$ ) $\stackrel{\bullet}{=}$ metasubstc1 ( $x$ , $s$ , $\top$ , $c$ )]

[metasubstc1 ( $x$ , $s$ , $b$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $d = x$ metadef ( $c$ ) **in**
**if** $d =$ var **then** metasubstc3 ( $x$ , $s$ , $b$ , $c$ ) **else**
**if** $d$ **then** metasubstc2 ( $x$ , $s$ , $b$ , $c$ ) **else**
**if** $d \neq$ all **then** $x^{\mathrm{h}}$ : : metasubstc1∗ ( $x^{\mathrm{t}}$ , $s$ , $b$ , $c$ ) **else**
**let** $s =$ remove ( $x^1$ , $s$ ) **in**
$\langle x^{\mathrm{h}}, x^1,$ metasubstc1 ( $x^2$ , $s$ , $x^1 : : b$ , $c$ ) $\rangle$].

[metasubstc1∗ ( $x$ , $s$ , $b$ , $c$ ) $\stackrel{\bullet}{=}$
**if** $x \in$ **A then** $\top$ **else**
metasubstc1 ( $x^{\mathrm{h}}$ , $s$ , $b$ , $c$ ) : : metasubstc1∗ ( $x^{\mathrm{t}}$ , $s$ , $b$ , $c$ )]

[metasubstc2 ( $x$ , $s$ , $b$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $d = x$ metadef ( $c$ ) **in**
**if** $d$ **then** $x^{\mathrm{h}}$ : : metasubstc2∗ ( $x^{\mathrm{t}}$ , $s$ , $b$ , $c$ ) **else**
**let** $v =$ lookup ( $x$ , $s$ , $\top$ ) **in**
**if** $v$ **then** $x$ **else**
**let** $d = v$ metadef ( $c$ ) **in**
**if not** $d$ **and** $d \neq$ var **then**
error ( $x$ , LocateProofLine ( $x$ , $c$ ) diag ( 'Substitution of' ) disp ( $x$ ) diag ( 'with' ) disp ( $v$ ) diag ( 'produces non-meta-term' ) end diagnose ) **else**
**if** metaavoid2∗ ( $b$ , $v$ , $c$ ) **then** $v$ **else**
error ( $x$ , LocateProofLine ( $x$ , $c$ ) diag ( 'At-seqop used for non-free substitution. Attempt to instantiate' ) disp ( $x$ ) diag ( 'with' ) disp ( $v$ ) end diagnose )]

[metasubstc2∗ ( $x$ , $s$ , $b$ , $c$ ) $\stackrel{\bullet}{=}$
**if** $x \in$ **A then** $\top$ **else**
metasubstc2 ( $x^{\mathrm{h}}$ , $s$ , $b$ , $c$ ) : : metasubstc2∗ ( $x^{\mathrm{t}}$ , $s$ , $b$ , $c$ )]

[metaavoid2∗ ( $x$ , $y$ , $c$ ) $\stackrel{\bullet}{=}$ $x$ **or** $x^{\mathrm{h}}$ metaavoid ( $c$ ) $y$ **and** metaavoid2∗ ( $x^{\mathrm{t}}$ , $y$ , $c$ )]

[metasubstc3 ( $x$ , $s$ , $b$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $v =$ lookup ( $x$ , $s$ , $\top$ ) **in**
**if** $v$ **then** $x$ **else**
**if** metaavoid2∗ ( $b$ , $v$ , $c$ ) **then** $v$ **else**

error ( $x$ , LocateProofLine ( $x$ , $c$ ) diag ( 'At-seqop used for non-free substitution. Attempt to instantiate' ) disp ( $x$ ) diag ( 'with' ) disp ( $v$ ) end diagnose )]

[remove ( $x$ , $s$ ) $\overset{\bullet}{=}$
**if** $s \in$ **A then** $s$ **else**
**if** $x \overset{\mathrm{t}}{=} s^{\mathrm{hh}}$ **then** remove ( $x$ , $s^{\mathrm{t}}$ ) **else**
$s^{\mathrm{h}} :: \mathrm{remove}\ (\ x\ ,\ s^{\mathrm{t}}\ )$]

## 2.14 Quantifiers

The following macro definition allows to write e.g. Πx, y, z: a instead of Πx: Πy: Πz: a.

[$\Pi u\colon v \overset{\circ}{=} \lambda x.$expand-All ( $x$ )]

[expand-All ( $x$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = x$ **in**
**let** $\langle \mathsf{T}, u, v\rangle = t$ **in**
**let** $u =$ stateexpand ( $u$ , $s$ , $c$ ) **in**
**let** $v =$ stateexpand ( $v$ , $s$ , $c$ ) **in**
expand-All1 ( $t$ , $u$ , $v$ )]

[expand-All1 ( $t$ , $u$ , $v$ ) $\overset{\bullet}{=}$
**if not** $u \overset{\mathrm{r}}{=} \lceil x, y\rceil$ **then** $\lceil_t\, \Pi \underline{u}\colon \underline{v}\rceil$ **else**
expand-All1 ( $t$ , $u^1$ , expand-All1 ( $t$ , $u^2$ , $v$ ) )]

# 3 Sequents

## 3.1 Term sets

We represent sets of terms by tuples of terms without repetitions so that no two elements $x$ and $y$ of the tuple satisfy $x \overset{\mathrm{t}}{=} y$.

[$x \in y \overset{\bullet}{=}$ **if** $y \in$ **A then** F **else** $x \overset{\mathrm{t}}{=} y^{\mathrm{h}}$ **or** $x \in y^{\mathrm{t}}$]. Membership of a set of terms.

[$x$ set+ $y \overset{\bullet}{=}$ **if** $y \in x$ **then** $x$ **else** $y :: x$]. Add the term $y$ to the set $x$ of terms.

[$x$ set- $y \overset{\bullet}{=}$ **if** $y \in x$ **then** $x$ set-- $y$ **else** $x$]. Remove the term $y$ from the set $x$ of terms.

[$x$ set-- $y \overset{\bullet}{=}$ **if** $x \in$ **A then** T **else if** $x^{\mathrm{h}} \overset{\mathrm{t}}{=} y$ **then** $x^{\mathrm{t}}$ **else** $x^{\mathrm{h}} :: x^{\mathrm{t}}$ set-- $y$]

[$x$ subset $y \overset{\bullet}{=}$ **if** $x \in$ **A then** T **else** $x^{\mathrm{h}} \in y$ **and** $x^{\mathrm{t}}$ subset $y$]

[$x$ union $y \overset{\bullet}{=}$ **if** $x \in$ **A then** $y$ **else** $x^{\mathrm{t}}$ union ($y$ set+ $x^{\mathrm{h}}$)]

[$x$ set= $y \overset{\bullet}{=} x$ subset $y$ **and** $y$ subset $x$]

## 3.2 Sequents

The sequents of Logiweb sequent calculus have form $\langle P, C, R\rangle$ where $P$ is a pool of premisses, $C$ is a pool of condition (i.e. side conditions), and $R$ is a result which holds if the given premisses and conditions hold. $P$ and $C$ are sets of meta terms (c.f. Section 3.1) and $R$ is a meta term.

$[x \text{ sequent}= y \overset{\bullet}{=} x^0 \text{ set}= y^0 \textbf{ and } x^1 \text{ set}= y^1 \textbf{ and } x^2 \overset{\text{t}}{=} y^2]$

## 3.3 Sequent operators

The thirteen sequent operators of the Logiweb sequent calculus are:

$[x^{\text{I}} \overset{\text{meta}}{=}$ 'Init']. Allows to build a sequent with $x$ as both premise and result.

$[x^{\triangleright} \overset{\text{meta}}{=}$ 'Ponens']. Allows to move a premise from result to premise pool.

$[x^{\triangleright\!\!\triangleright} \overset{\text{meta}}{=}$ 'Probans']. Allows to move a condition from result to condition pool.

$[x^{*} \overset{\text{meta}}{=}$ 'Verify']. Allows to evaluate and remove a condition from the result.

$[x^{\text{C}} \overset{\text{meta}}{=}$ 'Curry']. Allows to change a result from $x \oplus y \vdash z$ to $x \vdash y \vdash z$.

$[x^{\text{U}} \overset{\text{meta}}{=}$ 'Uncurry']. Allows to change a result from $x \vdash y \vdash z$ to $x \oplus y \vdash z$

$[x^{\text{D}} \overset{\text{meta}}{=}$ 'Dereference']. Allows to change the name of a result to the result itself.

$[x \mathbin{@} y \overset{\text{meta}}{=}$ 'at']. Allows to instantiate a meta quantifier.

$[x \vdash y \overset{\text{meta}}{=}$ 'infer']. Allows to move a premise from premise pool to result.

$[x \Vdash y \overset{\text{meta}}{=}$ 'endorse']. Allows to move a condition from condition pool to result.

$[x \text{ ie } y \overset{\text{meta}}{=}$ 'id est']. Allows to change the result of $x$ to the name $y$ for the result.

$[\Pi x\colon y \overset{\text{meta}}{=}$ 'all']. Allows to perform meta generalization.

$[x; y \overset{\text{meta}}{=}$ 'cut']. Allows to perform a cut operation.

The operators $x \vdash y$, $x \Vdash y$, and $\Pi x\colon y$ may be used both as meta and as sequent operators. Whether they denote the one or the other depends on context.

## 3.4 Sequent pseudo operators

In addition to the genuine sequent operators the following pseudo operators allow to build up the body of a proof. The pseudo operators disappear during unification.

$[x \rhd y \overset{\text{meta}}{=}$ 'ponens']. Same as $x^{\rhd}$ but $y$ specifies what the moved premise looks like as an aid to unification.

$[x \Rrightarrow y \overset{\text{meta}}{=}$ 'probans']. Same as $x^{\Rrightarrow}$ but $y$ specifies what the moved condition looks like as an aid to unification.

$[x * y \overset{\text{meta}}{=}$ 'verify']. Same as $x^*$ but $y$ specifies what the verified condition looks like as an aid to unification.

$[x \gg y \overset{\text{meta}}{=}$ 'conclude']. Same as $x$ but $y$ specifies what $x$ is supposed to prove as an aid to unification.

$[x^{@} \overset{\text{meta}}{=}$ 'At']. Same as $x \mathbin{@} y$ but leaves it to unification to determine $y$.

$[x^{\vdash} \overset{\text{meta}}{=}$ 'Infer']. Same as $y \vdash x$ but leaves it to unification to determine $y$.

$[x^{\Vdash} \overset{\text{meta}}{=}$ 'Endorse']. Same as $y \Vdash x$ but leaves it to unification to determine $y$.

$[x^{\Pi} \overset{\text{meta}}{=}$ 'All']. Same as $\Pi y\colon x$ but leaves it to unification to determine $y$.

$[x^{\gg} \overset{\text{meta}}{=}$ 'Conclude']. Add as many $x \mathbin{@} y$, $x^{\rhd}$, and $x^*$ operations as possible in an attempt to make $x$ prove an object term.

## 3.5 Counting of sequent operators

The following function can be used for profiling tactics in that it returns the number of tactic operators in the term t.

[seqcnt ( $t$ , $c$ ) $\overset{\bullet}{=}$
**if** $t \overset{\text{r}}{=} \lceil x^{\text{I}} \rceil$ **then** 1 **else**
**if** $t \overset{\text{r}}{=} \lceil x^{\rhd} \rceil$ **then** $1 +$ seqcnt ( $t^1$ , $c$ ) **else**
**if** $t \overset{\text{r}}{=} \lceil x^{\Rrightarrow} \rceil$ **then** $1 +$ seqcnt ( $t^1$ , $c$ ) **else**
**if** $t \overset{\text{r}}{=} \lceil x^{*} \rceil$ **then** $1 +$ seqcnt ( $t^1$ , $c$ ) **else**
**if** $t \overset{\text{r}}{=} \lceil x^{\text{C}} \rceil$ **then** $1 +$ seqcnt ( $t^1$ , $c$ ) **else**
**if** $t \overset{\text{r}}{=} \lceil x^{\text{U}} \rceil$ **then** $1 +$ seqcnt ( $t^1$ , $c$ ) **else**
**if** $t \overset{\text{r}}{=} \lceil x^{\text{D}} \rceil$ **then** $1 +$ seqcnt ( $t^1$ , $c$ ) **else**
**if** $t \overset{\text{r}}{=} \lceil x \mathbin{@} y \rceil$ **then** $1 +$ seqcnt ( $t^1$ , $c$ ) **else**
**if** $t \overset{\text{r}}{=} \lceil x \vdash y \rceil$ **then** $1 +$ seqcnt ( $t^2$ , $c$ ) **else**
**if** $t \overset{\text{r}}{=} \lceil x \Vdash y \rceil$ **then** $1 +$ seqcnt ( $t^2$ , $c$ ) **else**
**if** $t \overset{\text{r}}{=} \lceil x \text{ ie } y \rceil$ **then** $1 +$ seqcnt ( $t^1$ , $c$ ) **else**
**if** $t \overset{\text{r}}{=} \lceil \Pi x\colon y \rceil$ **then** $1 +$ seqcnt ( $t^2$ , $c$ ) **else**
**if** $t \overset{\text{r}}{=} \lceil x; y \rceil$ **then** $1 +$ seqcnt ( $t^1$ , $c$ ) $+$ seqcnt ( $t^2$ , $c$ ) **else**
error ( $t$ , LocateProofLine ( $t$ , $c$ ) diag ( 'In seqcnt: Unknown seqop in root of' ) disp ( $t$ ) end diagnose )]

## 3.6 Error message construct

The $[\text{error}\ (\ x\ ,\ y\ ) \stackrel{\bullet}{=} (\text{dbug}\ (\ x\ )\ y)^{\bullet}]$ construct is useful for signaling errors. The construct signals that an error has occurred by raising an exception. Exceptions take one parameter, and error ( $x$ , $y$ ) raises an exception with parameter $y$. That parameter is supposed to be a term which, when typeset, explains to a human reader what went wrong.

If the term $y$ contains no useful "debugging information" then error ( $x$ , $y$ ) takes the debugging information from $x$. At the time of writing, the debugging information is not used. As an example of use, a Logiweb browser could have a button for moving to the proof line in error. As another example, a custom renderer could add a hypertarget to the line in error such that one could navigate to it using an ordinary web browser.

## 3.7 Pattern recognition

The 'mismatch' construct defined in the following is such that e.g. mismatch ( $\ulcorner x \vdash y \urcorner$ , $R$ , $c$ ) is false if $R$ has form $x \vdash y$.

$[\text{mismatch}\ (\ x\ ,\ y\ ,\ c\ ) \stackrel{\bullet}{=}$
**let** $a = x$ metadef ( $c$ ) **in**
**if** $a = \mathsf{T}$ **then** $\mathsf{F}$ **else**
**if** $a \neq y$ metadef ( $c$ ) **then** $\mathsf{T}$ **else** mismatch$*$ ( $x^{\mathrm{t}}$ , $y^{\mathrm{t}}$ , $c$ )]

$[\text{mismatch}*\ (\ x\ ,\ y\ ,\ c\ ) \stackrel{\bullet}{=}$
**if** $x \in$ **A and** $y \in$ **A then** $\mathsf{F}$ **else**
**if** $x \in$ **A or** $y \in$ **A then** $\mathsf{T}$ **else**
mismatch ( $x^{\mathrm{h}}$ , $y^{\mathrm{h}}$ , $c$ ) **or** mismatch$*$ ( $x^{\mathrm{t}}$ , $y^{\mathrm{t}}$ , $c$ )]

## 3.8 Sequent evaluator

seqeval ( $t$ , $c$ ) evaluates the sequent term $t$. The result is either an exception or a valid sequent $s$. In the latter case, $t$ is said to be a sequent proof of $s$.

$[\text{seqeval}\ (\ t\ ,\ c\ ) \stackrel{\bullet}{=}$
**if** $t \stackrel{\mathrm{r}}{=} \ulcorner x^{\mathrm{I}} \urcorner$ **then** eval-Init ( $t^1$ , $c$ ) **else**
**if** $t \stackrel{\mathrm{r}}{=} \ulcorner x^{\triangleright} \urcorner$ **then** eval-Ponens ( seqeval ( $t^1$ , $c$ ) , $t^1$ , $c$ ) **else**
**if** $t \stackrel{\mathrm{r}}{=} \ulcorner x^{\Rrightarrow} \urcorner$ **then** eval-Probans ( seqeval ( $t^1$ , $c$ ) , $t^1$ , $c$ ) **else**
**if** $t \stackrel{\mathrm{r}}{=} \ulcorner x^{*} \urcorner$ **then** eval-Verify ( seqeval ( $t^1$ , $c$ ) , $t^1$ , $c$ ) **else**
**if** $t \stackrel{\mathrm{r}}{=} \ulcorner x^{\mathrm{C}} \urcorner$ **then** eval-Curry ( seqeval ( $t^1$ , $c$ ) , $t^1$ , $c$ ) **else**
**if** $t \stackrel{\mathrm{r}}{=} \ulcorner x^{\mathrm{U}} \urcorner$ **then** eval-Uncurry ( seqeval ( $t^1$ , $c$ ) , $t^1$ , $c$ ) **else**
**if** $t \stackrel{\mathrm{r}}{=} \ulcorner x^{\mathrm{D}} \urcorner$ **then** eval-Deref ( seqeval ( $t^1$ , $c$ ) , $t^1$ , $c$ ) **else**
**if** $t \stackrel{\mathrm{r}}{=} \ulcorner x\ @\ y \urcorner$ **then** eval-at ( $t$ , $\mathsf{T}$ , $c$ ) **else**
**if** $t \stackrel{\mathrm{r}}{=} \ulcorner x \vdash y \urcorner$ **then** eval-infer ( $t^1$ , seqeval ( $t^2$ , $c$ ) , $t^2$ , $c$ ) **else**
**if** $t \stackrel{\mathrm{r}}{=} \ulcorner x \Vdash y \urcorner$ **then** eval-endorse ( $t^1$ , seqeval ( $t^2$ , $c$ ) , $t^2$ , $c$ ) **else**
**if** $t \stackrel{\mathrm{r}}{=} \ulcorner x\ \text{ie}\ y \urcorner$ **then** eval-ie ( seqeval ( $t^1$ , $c$ ) , $t^2$ , $t^1$ , $c$ ) **else**

**if** $t \stackrel{\mathrm{r}}{=} \lceil \Pi x{:}\, y \rceil$ **then** eval-all ( $t^1$ , seqeval ( $t^2$ , $c$ ) , $t^2$ , $c$ ) **else**
**if** $t \stackrel{\mathrm{r}}{=} \lceil x; y \rceil$ **then** eval-cut ( seqeval ( $t^1$ , $c$ ) , seqeval ( $t^2$ , $c$ ) , $c$ ) **else**
error ( $t$ , LocateProofLine ( $t$ , $c$ ) diag ( 'Unknown seqop in root of' ) disp ( $t$ ) end diagnose )]

## 3.9 Definitions of sequent operators

The individual sequent operators of the Logiweb sequent calculus are defined as follows:

[eval-Init ( $x$ , $c$ ) $\stackrel{\bullet}{=}$
**if** $x$ metaterm ( $c$ ) **then** $\langle\langle x\rangle, \langle\,\rangle, x\rangle$ **else**
error ( $x$ , LocateProofLine ( $x$ , $c$ ) diag ( 'Init-seqop used on non-meta term:' ) disp ( $x$ ) end diagnose )]

[eval-Ponens ( $x$ , $d$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $\langle P, C, R\rangle = x$ **in**
**if** mismatch ( $\lceil x \vdash y \rceil$ , $R$ , $c$ ) **then**
error ( $d$ , LocateProofLine ( $d$ , $c$ ) diag ( 'Ponens-seqop used on non-inference:' ) disp ( $R$ ) end diagnose ) **else**
**let** $\langle \mathsf{T}, P', R'\rangle = R$ **in**
$\langle P \text{ set+ } P', C, R'\rangle$]

[eval-Probans ( $x$ , $d$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $\langle P, C, R\rangle = x$ **in**
**if** mismatch ( $\lceil x \Vdash y \rceil$ , $R$ , $c$ ) **then**
error ( $d$ , LocateProofLine ( $d$ , $c$ ) diag ( 'Probans-seqop used on non-endorsement:' ) disp ( $R$ ) end diagnose ) **else**
**let** $\langle \mathsf{T}, C', R'\rangle = R$ **in**
$\langle P, C \text{ set+ } C', R'\rangle$]

[eval-Verify ( $x$ , $d$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $\langle P, C, R\rangle = x$ **in**
**if** mismatch ( $\lceil x \Vdash y \rceil$ , $R$ , $c$ ) **then**
error ( $d$ , LocateProofLine ( $d$ , $c$ ) diag ( 'Verify-seqop used on non-endorsement:' ) disp ( $R$ ) end diagnose ) **else**
**let** $\langle \mathsf{T}, C', R'\rangle = R$ **in**
**if** (eval ( $C'$ , $\mathsf{T}$ , $c$ ) " $c^{\mathrm{M}})^{\mathrm{U}\circ} = \mathsf{F} :: \mathsf{T}$ **then** $\langle P, C, R'\rangle$ **else**
error ( $d$ , LocateProofLine ( $d$ , $c$ ) diag ( 'Verify-seqop used on false condition:' ) disp ( $C'$ ) end diagnose )]

[eval-Curry ( $x$ , $d$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $\langle P, C, R\rangle = x$ **in**
**if** mismatch ( $\lceil (x \oplus y) \vdash z \rceil$ , $R$ , $c$ ) **then**
error ( $d$ , LocateProofLine ( $d$ , $c$ ) diag ( 'Curry-seqop used on unfit argument:' ) disp ( $R$ ) end diagnose ) **else**
**let** $\lceil_{\mathsf{T}} (\underline{X} \oplus \underline{Y}) \vdash \underline{Z} \rceil = R$ **in**
$\langle P, C, \lceil_R \underline{X} \vdash \underline{Y} \vdash \underline{Z} \rceil\rangle$]

[eval-Uncurry ( $x$ , $d$ , $c$ ) $\overset{\bullet}{=}$
**let** $\langle P, C, R\rangle = x$ **in**
**if** mismatch ( $\lceil x \vdash y \vdash z \rceil$ , $R$ , $c$ ) **then**
error ( $d$ , LocateProofLine ( $d$ , $c$ ) diag ( 'Uncurry-seqop used on unfit argument:' ) disp ( $R$ ) end diagnose ) **else**
**let** $\lceil_{\mathsf{T}} \underline{X} \vdash \underline{Y} \vdash \underline{Z} \rceil = R$ **in**
$\langle P, C, \lceil_R (\underline{X} \oplus \underline{Y}) \vdash \underline{Z} \rceil \rangle$]

[eval-Deref ( $x$ , $d$ , $c$ ) $\overset{\bullet}{=}$
**let** $\langle P, C, R\rangle = x$ **in**
**let** $R' = R$ stmt-rhs ( $c$ ) **in**
**if** $R' = \mathsf{T}$ **then**
error ( $d$ , LocateProofLine ( $d$ , $c$ ) diag ( 'Deref-seqop used on undefined statement:' ) disp ( $R$ ) end diagnose ) **else**
**if** $R'$ metaterm ( $c$ ) **then** $\langle P, C, R'\rangle$ **else**
error ( $d$ , LocateProofLine ( $d$ , $c$ ) diag ( 'Deref-seqop applied to' ) disp ( $R$ ) diag ( 'produced non-meta term:' ) disp ( $R'$ ) end diagnose )]

[eval-at ( $x$ , $v$ , $c$ ) $\overset{\bullet}{=}$
**if not** $x \overset{\mathrm{r}}{=} \lceil x \mathbin{@} y \rceil$ **then let** $\langle P, C, R\rangle =$ seqeval ( $x$ , $c$ ) **in**
$\langle P, C,$ eval-at1 ( $R$ , $v$ , $\mathsf{T}$ , $x$ , $c$ )$\rangle$ **else**
**if** $x^2$ metaterm ( $c$ ) **then** eval-at ( $x^1$ , $x^2 :: v$ , $c$ ) **else**
error ( $x$ , LocateProofLine ( $x$ , $c$ ) diag ( 'At-seqop used on non-meta-term:' ) disp ( $x^2$ ) end diagnose )]

[eval-at1 ( $R$ , $v$ , $s$ , $d$ , $c$ ) $\overset{\bullet}{=}$
**if** $v$ **then** metasubstc ( $R$ , $s$ , $c$ ) **else**
**if** mismatch ( $\lceil \Pi x{:}\, y \rceil$ , $R$ , $c$ ) **then** error ( $d$ , LocateProofLine ( $d$ , $c$ ) diag ( 'At-seqop used on non-quantifier:' ) disp ( $R$ ) end diagnose ) **else**
eval-at1 ( $R^2$ , $v^{\mathrm{t}}$ , $(R^1 :: v^{\mathrm{h}}) :: s$ , $d$ , $c$ )]

[eval-infer ( $x$ , $y$ , $d$ , $c$ ) $\overset{\bullet}{=}$
**if not** $x$ metaterm ( $c$ ) **then**
error ( $d$ , LocateProofLine ( $d$ , $c$ ) diag ( 'Infer-seqop used on non-meta term:' ) disp ( $x$ ) end diagnose ) **else**
**let** $\langle P, C, R\rangle = y$ **in**
$\langle P$ set- $x, C, \lceil_R \underline{x} \vdash \underline{R} \rceil \rangle$]

[eval-endorse ( $x$ , $y$ , $d$ , $c$ ) $\overset{\bullet}{=}$
**if not** $x$ metaterm ( $c$ ) **then**
error ( $d$ , LocateProofLine ( $d$ , $c$ ) diag ( 'Endorse-seqop used on non-meta term:' ) disp ( $x$ ) end diagnose ) **else**
**let** $\langle P, C, R\rangle = y$ **in**
$\langle P, C$ set- $x, \lceil_R \underline{x} \Vdash \underline{R} \rceil \rangle$]

[eval-ie ( $x$ , $y$ , $d$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $\langle P, C, R\rangle = x$ **in**
**if** $R \stackrel{\mathrm{t}}{=} y$ stmt-rhs ( $c$ ) **then**
$\langle P, C, y\rangle$ **else**
error ( $d$ , LocateProofLine ( $d$ , $c$ ) diag ( 'IdEst-seqop used on non-matching result. Attempt to match' ) disp ( $y$ ) diag ( 'with' ) disp ( $R$ ) end diagnose )]

[eval-all ( $x$ , $y$ , $d$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $\langle P, C, R\rangle = y$ **in**
**if not** $x$ metaavoid$*$ ( $c$ ) $P$ **then**
error ( $d$ , LocateProofLine ( $d$ , $c$ ) diag ( 'All-seqop catches variable' ) disp ( $x$ ) diag ( 'which is free in the following premise:' ) disp ( $x$ metaavoid$*$ ( $c$ ) $P$ ) end diagnose ) **else**
**if not** $x$ metaavoid$*$ ( $c$ ) $C$ **then**
error ( $d$ , LocateProofLine ( $d$ , $c$ ) diag ( 'All-seqop catches variable' ) disp ( $x$ ) diag ( 'which is free in the following condition:' ) disp ( $x$ metaavoid$*$ ( $c$ ) $C$ ) end diagnose ) **else**
$\langle P, C, \lceil_R \Pi \underline{x}\colon \underline{R}\rceil\rangle$]

[eval-cut ( $x$ , $y$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $\langle P, C, R\rangle = x$ **in**
**let** $\langle P', C', R'\rangle = y$ **in**
$\langle P$ union $(P'$ set- $R), C$ union $C', R'\rangle$]

# 4 Proof construction

## 4.1 Location information

[**Proof of** $x\colon y \stackrel{\text{locate}}{=}$ proof : : 1]. State that the given construct defines a proof and that the name of the proof is the first argument.

[$x$ **proof of** $y\colon z \stackrel{\text{locate}}{=}$ proof : : 2]. State that the given construct defines a proof and that the name of the proof is the second argument.

[$[x \stackrel{\text{proof}}{=} y] \stackrel{\text{locate}}{=}$ proof : : 1]. State that the given construct defines a proof and that the name of the proof is the first argument.

[L$*\colon * \gg *; *ˆ{\text{locate}}{=}$ line : : 1]. State that the given construct defines a proof line and that the name of the proof line is the first argument.

[L$*\colon * \gg *; \stackrel{\text{locate}}{=}$ line : : 1]

[L$*\colon * \gg *\square \stackrel{\text{locate}}{=}$ line : : 1]

[L$*\colon$ Premise $\gg *; * \stackrel{\text{locate}}{=}$ line : : 1]

[L$*$: Condition $\gg *; * \overset{\text{locate}}{=}$ line : : 1]

[L$*$: Arbitrary $\gg *; * \overset{\text{locate}}{=}$ line : : 1]

[L$*$: Local $\gg * \ddot{=} *; * \overset{\text{locate}}{=}$ line : : 1]

[L$*$: Block $\gg$ Begin; $\quad *$ L$*$: Block $\gg$ End; $* \overset{\text{locate}}{=}$ line : : 3]

## 4.2 Error message generation

[make-string ( $d$ , $x$ ) $\overset{\bullet}{=} \langle\langle 0, x, d\rangle\rangle$]. Convert the string $x$ to a tree which represents the term

[dbug ( $x$ ) $y \overset{\bullet}{=}$
**if** $y^{\mathrm{d}} \neq \mathsf{T}$ **then** $y$ **else**
make-root ( $x$ , $y$ ) : : dbug$*$ ( $x$ ) $y^{\mathrm{t}}$]

[dbug$*$ ( $x$ ) $y \overset{\bullet}{=}$ **if** $y$ **then** $\mathsf{T}$ **else** (dbug ( $x$ ) $y^{\mathrm{h}}$) : : dbug$*$ ( $x$ ) $y^{\mathrm{t}}$]

[glue ( $x$ ) $y \overset{\bullet}{=} \langle$make-root ( $\mathsf{T}$ , $\lceil$glue' ( $x$ ) $y\rceil$ ), make-string ( $\mathsf{T}$ , $x$ ), $y\rangle$]

[diag ( $x$ ) $y \overset{\bullet}{=} \langle$make-root ( $\mathsf{T}$ , $\lceil$diag' ( $x$ ) $y\rceil$ ), make-string ( $\mathsf{T}$ , $x$ ), $y\rangle$]

[disp ( $x$ ) $y \overset{\bullet}{=} \langle$make-root ( $\mathsf{T}$ , $\lceil$disp' ( $x$ ) $y\rceil$ ), $x, y\rangle$]

[form ( $x$ ) $y \overset{\bullet}{=} \langle$make-root ( $\mathsf{T}$ , $\lceil$form' ( $x$ ) $y\rceil$ ), $x, y\rangle$]

[end diagnose $\overset{\bullet}{=}$ make-string ( $\mathsf{T}$ , '' )]

## 4.3 Error location

[Locate ( $t$ , $s$ , $c$ ) $\overset{\bullet}{=}$
**let** $\mathsf{T} : : d =$ reverse ( $t^{\mathrm{dh}}$ )$^\circ$ **in**
**let** $e : : v =$ Locate1 ( $d^{\mathrm{t}}$ , $c[d^{\mathrm{h}}][\text{body}]$ , $s$ , $\mathsf{T}$ , $c$ )$^\circ$ **in**
**if** $e$ **then** $\bullet$ **else** $v$]. $t$ is supposed to be a term which has been macro expanded. If macro expansion has been set up properly, then it is possible to locate where $t$ came from before macro expansion. $d$ is set to that location encoded as a list $\langle r, i_0, \cdots, i_n\rangle$ where $r$ is the reference of the page where the term occurred and $i_0, \cdots, i_n$ is a path from the root of that page to the term. That path is traversed and for each symbol on the path, the locate aspect is looked up. The locate aspect is supposed to be a pair of a string and a value. If the string equals $s$ then a pair of the symbol with subtrees and the value is returned. If more than one node matches, the last one (the one closest to $t$) is returned. If no nodes match, then $\mathsf{T}$ is returned.

[Locate1 ( $d$ , $t$ , $s$ , $r$ , $c$ ) $\stackrel{\bullet}{=}$
**if** $t$ **then** $r$ **else**
**let** $v = t$ locate-rhs ( $c$ ) **in**
**let** $v =$ **if** $v$ **then** $\mathsf{T}$ **else** eval ( $v$ , $\mathsf{T}$ , $c$ )$^{\mathrm{U}}$ **in**
**let** $r =$ **if** $v^{\mathrm{h}} = s$ **then** $t :: v^{\mathrm{t}}$ **else** $r$ **in**
**if** $d \in$ **A then** $r$ **else**
Locate1 ( $d^{\mathrm{t}}$ , nth ( $d^{\mathrm{h}}$ , $t^{\mathrm{t}}$ ) , $s$ , $r$ , $c$ )]

[LocateProofLine ( $v$ , $c$ ) $y \stackrel{\bullet}{=}$
**let** $p =$ Locate ( $v$ , proof , $c$ ) **in**
**let** $l =$ Locate ( $v$ , line , $c$ ) **in**
**if not** $p$ **and not** $l$ **then**
dbug ( $v$ ) diag ( 'Line' ) form ( nth ( $l^{\mathrm{t}}$ , $l^{\mathrm{h}}$ ) ) diag ( 'in proof of' ) form ( nth ( $p^{\mathrm{t}}$ , $p^{\mathrm{h}}$ ) ) glue ( ':\newline ' ) $y$ **else**
**if not** $p$ **then**
dbug ( $v$ ) diag ( 'In proof of' ) form ( nth ( $p^{\mathrm{t}}$ , $p^{\mathrm{h}}$ ) ) glue ( ':\newline ' ) $y$ **else**
dbug ( $v$ ) diag ( 'Cannot locate error.' ) diag ( 'Look at ''macro'' and ''locate'' definitions.' ) diag ( '' ) $y$]

## 4.4 Proof checker

[claiming ( $x$ , $y$ ) $\stackrel{\bullet}{=}$
**if not** $y \stackrel{\mathrm{r}}{=} \lceil u \wedge_{\mathrm{c}} v \rceil$ **then** $x \stackrel{\mathrm{t}}{=} y$ **else**
claiming ( $x$ , $y^1$ ) **or** claiming ( $x$ , $y^2$ )].

True if $x$ is among the conjuncts of a term $y$ built up from conjunctions and $u \wedge_{\mathrm{c}} v$.

[proofcheck $\doteq \lambda c$.proofcheck1 ( $c$ )].

Suited as conjunct in the claim of a page.

[**Verifier**:test1 $\wedge_{\mathrm{c}}$ proofcheck].

Use both proofcheck and test1 as verifiers.

[proofcheck1 ( $c$ ) $\stackrel{\bullet}{=}$
**let** $e :: v =$ proofcheck2 ( $c$ )$^{\circ}$ **in**
**if** $v \neq \mathsf{T}$ **then** $v$ **else**
**if not** $e$ **then** $\mathsf{T}$ **else**
$\lceil$'In proof checker: unprocessed exception'$\rceil$].

Proofcheck page. Errors are reported as exceptions whose value is a term to be typeset as diagnose and for which the debugging information of the root indicates the location of the error.

[proofcheck2 ( $c$ ) $\stackrel{\bullet}{=}$
**let** $r = c[0]$ **in**

**let** $a$ = seqeval$*$ ( $c$[$r$][codex][$r$] , $c$ ) **in**
**let** $x$ = circularitycheck1 ( $r$ , $c$ , $a$ , $a$ ) **in** $\mathsf{T}$].

Check all proofs on the page using seqeval$*$ ( $a$ , $c$ ), then check that the proofs do not reference each other in a circular way.

[seqeval$*$ ( $a$ , $c$ ) $\stackrel{\bullet}{=}$
**if** $a = \mathsf{T}$ **then** $\mathsf{T}$ **else**
**let** $x :: y = a$ **in**
**if not** $x \in \mathbf{Z}$ **then** seqeval$*$ ( $x$ , $c$ ) :: seqeval$*$ ( $y$ , $c$ ) **else**
**let** $d = y$[0][proof] **in if** $d = \mathsf{T}$ **then** $\mathsf{T}$ **else**
print ( $d^2$ rhs ( $c$ , 'name' )$^{\mathrm{i}}$ :: LF ) .then.
**let** $e :: p$ = (eval ( $d^3$ , $\mathsf{T}$ , $c$ ) " $d^{\mathrm{M}}$ " $c^{\mathrm{M}}$)$^{\mathrm{U}}$[hook-arg]$^\circ$ **in**
**if** $e$ **and** $p$ **then** error ( $d^2$ , diag ( 'Malformed proof of' ) form ( $d^2$ ) end diagnose ) **else**
**if** $e$ **then** $p^\bullet$ **else**
**let** $e :: S$ = seqeval ( $p$ , $c$ )$^\circ$ **in**
**if** $e$ **and** $S$ **then** error ( $d^2$ , diag ( 'Malformed sequent proof of' ) form ( $d^2$ ) end diagnose ) **else**
**if** $e$ **then** $S^\bullet$ **else**
**let** $\langle P, C, R\rangle = S$ **in**
**if** $C \neq \langle\rangle$ **then** error ( $d^2$ , diag ( 'In proof of' ) form ( $d^2$ ) glue ( ':\newline ' ) diag ( 'Unchecked sidecondition:' ) disp ( $C^{\mathrm{h}}$ ) end diagnose ) **else**
**let** $l = y$[0][stmt] **in**
**if** $l = \mathsf{T}$ **then** error ( $d^2$ , diag ( 'Proof of non-existent lemma:' ) disp ( $d^2$ ) end diagnose ) **else**
**if not** $l^3 \stackrel{\mathrm{t}}{=} R$ **then**
error ( $d$ , diag ( 'The proof of' ) disp ( $d^2$ ) diag ( 'does not prove what the lemma says.' ) diag ( 'After macro expansion, the lemma says:' ) disp ( $l^3$ ) diag ( 'After macro, tactic, and sequent expansion, the proof concludes:' ) disp ( $R$ ) end diagnose ) **else**
premisecheck$*$ ( $P$ , $c$ ) **and** $x :: S$].

Sequent evaluate all proofs in the array $a$ and return an array of sequents. Proofs are tactic evaluated first end the resulting sequents are checked for everything except circularities.

[premisecheck$*$ ( $P$ , $c$ ) $\stackrel{\bullet}{=}$
**if** $P \in \mathbf{A}$ **then** $\mathsf{T}$ **else** premisecheck ( $P^{\mathrm{h}}$ , $c$ ) **and** premisecheck$*$ ( $P^{\mathrm{t}}$ , $c$ )].

Check each premise in the list $P$ of premisses.

[checked ( $r$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $x = c$[$r$][codex][$r$][0][0][claim]$^3$ **in**
**if** claiming ( ⌈proofcheck⌉ , $x$ ) **then** $\mathsf{T}$ **else**
**if not** $x$ **then** $\mathsf{F}$ **else**
**let** $r = c$[$r$][bibliography]$^1$ **in**

**if** $r$ **then** $\mathsf{F}$ **else**
**let** $x = c[r][\text{codex}][r][0][0][\text{claim}]^3$ **in**
claiming ( $\lceil$proofcheck$\rceil$ , $x$ )]. True if the page with reference $r$ has been checked by proofcheck.

[premisecheck ( $P$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $r = P^{\text{r}}$ **in let** $i = P^{\text{i}}$ **in**
**if** $r \neq c[0]$ **and** $c[r][\text{diagnose}]^{\text{U}} \neq \mathsf{T}$ **then**
error ( $P$ , diag ( 'Lemma' ) disp ( $P$ ) diag ( 'occurs on a page with a non-empty diagnose.' ) diag ( 'Avoid referencing that lemma.' ) end diagnose ) **else**
**if not** checked ( $r$ , $c$ ) **then**
error ( $P$ , diag ( 'Lemma' ) disp ( $P$ ) diag ( 'occurs on a page which has not been checked' ) diag ( 'by the present proof checker.' ) diag ( 'Avoid referencing that lemma.' ) end diagnose ) **else**
**if** $P$ proof-rhs ( $c$ ) **then**
error ( $P$ , LocateProofLine ( $P$ , $c$ ) diag ( 'Lemma' ) disp ( $P$ ) diag ( 'has no proof. Avoid referencing that lemma.' ) end diagnose ) **else** $\mathsf{T}$].

Check that a referenced lemma occurs on a correct page (diagnose = $\mathsf{T}$), has been checked (claim contains verifier), and has been proved (has non-empty proof aspect).

[circularitycheck1 ( $r$ , $c$ , $a$ , $q$ ) $\stackrel{\bullet}{=}$
**if** $a = \mathsf{T}$ **then** $q$ **else**
**let** $x :: y = a$ **in**
**if** $x \in \mathbf{Z}$ **then** circularitycheck2 ( $r$ , $c$ , $x$ , $q$ ) **else** circularitycheck1 ( $r$ , $c$ , $y$ , circularitycheck1 ( $r$ , $c$ , $x$ , $q$ ) )].

Check all indices in $a$ for circularities. $q$ is an array of sequents. The elements of $q$ are set to 0 while they are being checked for circularities and are set to 1 when it has been verified that they are not circular.

[circularitycheck2 ( $r$ , $c$ , $i$ , $q$ ) $\stackrel{\bullet}{=}$
**let** $v = q[i]$ **in**
**if** $v = 0$ **then**
**let** $n = c[r][\text{codex}][r][i][0][\text{proof}]^2$ **in**
error ( $n$ , diag ( 'Circular proof. The vicious circle includes lemma' ) disp ( $n$ ) end diagnose ) **else**
**if** $v = 1$ **then** $q$ **else**
circularitycheck2$*$ ( $r$ , $c$ , $v^{\text{h}}$ , $q[i \rightarrow 0]$ )$[i \rightarrow 1]$].

Check the proof with index $i$ for circularities.

[circularitycheck2$*$ ( $r$ , $c$ , $l$ , $q$ ) $\stackrel{\bullet}{=}$
**if** $l \in \mathbf{A}$ **then** $q$ **else**
**let** $p :: l = l$ **in**

**let** $q = \text{circularitycheck2}*\ (\ r\ ,\ c\ ,\ l\ ,\ q\ )$ **in**
**if** $r \neq p^{\mathrm{r}}$ **then** $q$ **else** $\text{circularitycheck2}\ (\ r\ ,\ c\ ,\ p^{\mathrm{i}}\ ,\ q\ )$].

Check all proofs in the list $l$ of terms for circularity.

[lemma1 $\overset{\text{stmt}}{=}$ $\Pi\mathsf{x}\colon \Pi\mathsf{y}\colon \mathsf{x} \vdash \mathsf{y} \vdash \mathsf{x}$].

Manually stated lemma for testing.

[lemma1 $\overset{\text{proof}}{=}$ $\lambda p.\lambda c.\mathsf{T}[\text{hook-arg}\rightarrow\lceil\Pi\mathsf{x}\colon \Pi\mathsf{y}\colon \mathsf{x} \vdash \mathsf{y} \vdash \mathsf{x}^{\mathrm{I}}\rceil]$].

Manually stated proof for testing.

## 4.5 Conversions from values to terms

The following are useful for constructing indexed variables:

[$\text{int2string}\ (\ t\ ,\ v\ ) \overset{\bullet}{=}$
**if** $v > 0$ **then** $\langle 0 :: \text{int2string1}\ (\ v\ ,\ \ )\ ::\ t^{\mathrm{d}}\rangle$ **else**
**if** $v = 0$ **then** $\langle 0 :: 0 :: t^{\mathrm{d}}\rangle$ **else**
$\langle 0 :: 256 \cdot \text{int2string1}\ (\ -v\ ,\ \ ) + \text{logand}\ (\ -\ ,\ 255\ ) :: t^{\mathrm{d}}\rangle$]

[$\text{int2string1}\ (\ v\ ,\ r\ ) \overset{\bullet}{=}$
**if** $v = 0$ **then** $r$ **else**
**let** $z = \text{logand}\ (\ 0\ ,\ 255\ )$ **in**
**let** $x :: y = \text{floor}\ (\ v\ ,\ 10\ )$ **in**
$\text{int2string1}\ (\ x\ ,\ 256 \cdot r + z + y\ )$]

[$\text{val2term}\ (\ t\ ,\ v\ ) \overset{\bullet}{=}$
**if** $v = \mathsf{T}$ **then** $\lceil_t\, \mathsf{T}\rceil$ **else**
**if** $v = \mathsf{F}$ **then** $\lceil_t\, \mathsf{F}\rceil$ **else**
**if** $v \in \mathbf{P}$ **then** $\lceil_t\, \underline{\text{val2term}\ (\ t\ ,\ v^{\mathrm{h}}\ )} :: \underline{\text{val2term}\ (\ t\ ,\ v^{\mathrm{t}}\ )}\rceil$ **else**
**if** $v \in \mathbf{M}$ **then** $\lceil_t\, \text{map}\ (\ \cdots\ )\rceil$ **else**
**if** $v \in \mathbf{O}$ **then** $\lceil_t\, \text{object}\ (\ \underline{\text{val2term}\ (\ t\ ,\ \text{destruct}\ (\ v\ )\ )}\ )\rceil$ **else if**
$v = 0$ **then** $\lceil_t\, 0\rceil$ **else**
**if** $v < 0$ **then** $\lceil_t\, -\underline{\text{card2term}\ (\ t\ ,\ -v\ )}\rceil$ **else**
$\text{card2term}\ (\ t\ ,\ v\ )$]

[$\text{card2term}\ (\ t\ ,\ v\ ) \overset{\bullet}{=}$
**if** $v = 0$ **then** $\lceil_t\, \rceil$ **else**
**if** $\text{evenp}\ (\ v\ )$ **then** $\lceil_t\, \underline{\text{card2term}\ (\ t\ ,\ \text{half}\ (\ v\ )\ )}0\rceil$ **else** $\lceil_t\, \underline{\text{card2term}\ (\ t$

## 4.6 Unification

[$\text{univar}\ (\ t\ ,\ v\ ,\ i\ ) \overset{\bullet}{=} \lceil_t\, \text{unifresh}\ (\ \underline{v}\ ,\ \underline{\text{int2string}\ (\ t\ ,\ i\ )}\ )\rceil$]

[$\text{pterm}\ (\ t\ ,\ c\ ) \overset{\bullet}{=} \text{pterm1}\ (\ t\ ,\ \mathsf{T}\ ,\ 1\ ,\ c\ )$].

We shall refer to terms in which some of the bound meta variables are replaced by fresh variables indexed by numbers as *parameter terms*.

pterm ( $t$ , $c$ ) replaces meta variables bound at the top level in $t$. The function is used e.g. on instances of lemmas in proofs before unification. That allows to recognize instantiable meta variables after their associated quantifiers have been removed. $c$ should be the cache of the page.

[pterm1 ( $t$ , $s$ , $n$ , $c$ ) $\overset{\bullet}{=}$
**let** $d = t$ metadef ( $c$ ) **in**
**if** $d \neq$ all **then** pterm2 ( $t$ , $s$ , $c$ ) **else**
**let** $v =$ univar ( $t^1$ , $t^1$ , $n$ ) **in**
**let** $t' =$ pterm1 ( $t^2$ , $(t^1 :: v) :: s$ , $n + 1$ , $c$ ) **in**
$\lceil_t \Pi \underline{v}: \underline{t'} \rceil$].

[pterm2 ( $t$ , $s$ , $c$ ) $\overset{\bullet}{=}$
**let** $d = t$ metadef ( $c$ ) **in**
**let** $s =$ **if** $d =$ all **then** $(t^1 :: t^1) :: s$ **else** $s$ **in**
**if** $d \neq$ var **then** $t^{\mathrm{h}} ::$ pterm2$*$ ( $t^{\mathrm{t}}$ , $s$ , $c$ ) **else**
**let** $n =$ lookup ( $t$ , $s$ , $\mathsf{T}$ ) **in**
**if** $n = \mathsf{T}$ **then** $t$ **else** $n$]

[pterm2$*$ ( $t$ , $s$ , $c$ ) $\overset{\bullet}{=}$
**if** $t \in$ **A then** $\mathsf{T}$ **else**
pterm2 ( $t^{\mathrm{h}}$ , $s$ , $c$ ) :: pterm2$*$ ( $t^{\mathrm{t}}$ , $s$ , $c$ )]

[inst ( $t$ , $d$ , $a$ ) $\overset{\bullet}{=}$
**if not** $t \overset{\mathrm{r}}{=} \lceil$unifresh ( $\mathsf{T}$ , $\mathsf{T}$ )$\rceil$ **then** $(t^{\mathrm{r}} :: t^{\mathrm{i}} :: d^{\mathrm{d}}) ::$ inst$*$ ( $t^{\mathrm{t}}$ , $d$ , $a$ ) **else**
**let** $u = a[t^{2\mathrm{i}}]$ **in if** $u \neq \mathsf{T}$ **then** inst ( $u$ , $d$ , $a$ ) **else**
error ( $d$ , diag ( 'Incomplete unification. Uninstantiated variable:' ) disp ( $t$ ) end diagnose )].

Instantiate the parameter term $t$ as specified by the array $a$. May loop indefinitely. As an example, a substitution which maps x to x::x will keep expanding x forever. We shall say that a substitution is *circular* if there exists a term $t$ for which inst ( $t$ , $d$ , $x$ ) loops indefinitely.

[inst$*$ ( $t$ , $d$ , $a$ ) $\overset{\bullet}{=}$ **if** $t \in$ **A then** $\mathsf{T}$ **else** inst ( $t^{\mathrm{h}}$ , $d$ , $a$ ) :: inst$*$ ( $t^{\mathrm{t}}$ , $d$ , $a$ )].

[occur ( $t$ , $u$ , $s$ ) $\overset{\bullet}{=}$
**if** $u \overset{\mathrm{r}}{=} \lceil$unifresh ( $\mathsf{T}$ , $\mathsf{T}$ )$\rceil$ **then** $t \overset{\mathrm{t}}{=} u$ **or** occur ( $t$ , $s[u^{2\mathrm{i}}]$ , $s$ ) **else**
occur$*$ ( $t$ , $u^{\mathrm{t}}$ , $s$ )].

True if $t$ occurs in inst ( $u$ , $d$ , $s$ ). May loop indefinitely if $s$ is circular.

[occur$*$ ( $t$ , $u$ , $s$ ) $\overset{\bullet}{=}$
**if** $u \in$ **A then** F **else** occur ( $t$ , $u^{\mathrm{h}}$ , $s$ ) **or** occur$*$ ( $t$ , $u^{\mathrm{t}}$ , $s$ )].

[unify ( $t$ , $u$ , $s$ ) $\stackrel{\bullet}{=}$
**if** $t \stackrel{\mathrm{r}}{=} \lceil$unifresh ( $\mathsf{T}$ , $\mathsf{T}$ )$\rceil$ **then** unify2 ( $t$ , $u$ , $s$ ) **else**
**if** $u \stackrel{\mathrm{r}}{=} \lceil$unifresh ( $\mathsf{T}$ , $\mathsf{T}$ )$\rceil$ **then** unify2 ( $u$ , $t$ , $s$ ) **else**
**if** $t \stackrel{\mathrm{r}}{=} u$ **then** unify$*$ ( $t^{\mathrm{t}}$ , $u^{\mathrm{t}}$ , $s$ ) **else**
error ( $t$ , diag ( 'Unable to unify' ) disp ( $t$ ) diag ( 'with' ) disp ( $u$ ) end diagnose )].

If possible, return the 'smallest' substitution which contains $s$ and unifies the parameter terms $t$ and $u$, i.e. a substitution which, if applied to $t$ and $u$ yield identical terms. Raise an exception if unification is impossible.

[unify$*$ ( $t$ , $u$ , $s$ ) $\stackrel{\bullet}{=}$
**if** $t \in \mathbf{A}$ **then** $s$ **else**
unify$*$ ( $t^{\mathrm{t}}$ , $u^{\mathrm{t}}$ , unify ( $t^{\mathrm{h}}$ , $u^{\mathrm{h}}$ , $s$ ) )].

[unify2 ( $t$ , $u$ , $s$ ) $\stackrel{\bullet}{=}$
**if** $t \stackrel{\mathrm{t}}{=} u$ **then** $s$ **else**
**let** $t' = s[t^{2\mathrm{i}}]$ **in**
**if** $t' \neq \mathsf{T}$ **then** unify ( $t'$ , $u$ , $s$ ) **else**
**if** occur ( $t$ , $u$ , $s$ ) **then** $\bullet$ **else** $s[t^{2\mathrm{i}} \rightarrow u]$].

Does the same as unify ( $t$ , $u$ , $s$ ) but only when $t$ is a number.

## 4.7 Hooks for the tactic state

[hook-arg $\stackrel{\bullet}{=}$ arg]

Argumentation hook. States returned from tactic or unitac evaluation contain the returned argumentation on this hook.

[hook-res $\stackrel{\bullet}{=}$ con]

Result hook. States returned from tactic or unitac evaluation contain the expected conclusion on this hook.

[hook-idx $\stackrel{\bullet}{=}$ idx]

Index hook. During unitac evaluation, the index hook contains an accumulating index used for generating fresh variables.

[hook-uni $\stackrel{\bullet}{=}$ uni]

Unification hook. During unitac evaluation, the result of unification is accumulated here.

[hook-pre $\stackrel{\bullet}{=}$ pre]

Premise hook. Premises are stacked here during tactic evaluation. Premises may be explicitly assumed premises, the conclusion of the left hand side of a cut, or a theory assumed at the beginning of a proofs. Elements of the premise hook have form $l :: p :: x$ where $l$

is a label for referring to the premise, $p$ is the premise itself, and $x$ may be used by individual tactics.

[hook-cond $\stackrel{\bullet}{=}$ cond]

Condition hook. Side conditions are stacked here during tactic evaluation.

[hook-parm $\stackrel{\bullet}{=}$ parm]

Parameter hook. Tactic evaluation of $x$ @ $y$ stacks $y$ on the parameter hook. Tactic evaluation of $\Pi x\colon y$ unstacks. Useful for parameterized tactics.

## 4.8 Tactic evaluation

[taceval ( $t$ , $s$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $d = t$ tactic-rhs ( $c$ ) **in**
**if not** $d$ **then** (eval ( $d$ , $\mathsf{T}$ , $c$ ) " $\langle t, s, c\rangle^{\mathrm{M}})^{\mathrm{U}}$ **else**
**let** $\langle r, a\rangle$ = lookup ( $t$ , $s$[hook-pre] , $\mathsf{T}$ ) **in**
**if not** $a$ **then** $s$[hook-res$\rightarrow r$][hook-arg$\rightarrow a$] **else**
error ( $t$ ,
diag ( 'Unknown tactic operator in root of argumentation:' )
disp ( $t$ ) end diagnose )].

Evaluate the 'tactic' aspect of $t$.

[tactic-push ( $n$ , $v$ , $s$ ) $\stackrel{\bullet}{=}$
$s[n\rightarrow v :: s[n]]]$

Push the value $v$ onto the stack named $n$ in the tactic state $s$.

[tactic-pop ( $n$ , $s$ ) $\stackrel{\bullet}{=}$
$s[n\rightarrow s[n]^{\mathrm{t}}]]$

Pop a value from the stack named $n$ in the tactic state $s$ and return the new state (not the popped value).

[taceval1 ( $t$ , $p$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $d = t$ tactic-rhs ( $c$ ) **in**
**let** $t = \lceil_t \underline{t} \vdash \underline{p} \rceil$ **in**
**if** $d$ **then** taceval ( $t$ , tacstate0 , $c$ ) **else**
(eval ( $d$ , $\mathsf{T}$ , $c$ ) " $\langle t, \text{tacstate0}, c\rangle^{\mathrm{M}})^{\mathrm{U}}$]

Evaluate the 'tactic' aspect of $t \vdash p$ using the tactic definition of $t$. If $t$ has no tactic definition then use taceval.

## 4.9 Proof macros

[$x$ **lemma** $y\colon z$ $\square \doteq$ [$y \stackrel{\text{stmt}}{=} x \vdash z$][$y \stackrel{\text{unitac}}{=} \lambda u$.unitac-lemma ( $u$ )]].

A lemma declaration is the same as a statement definition plus a unitac definition which says how to sequent-prove a lemma plus what the lemma concludes.

[**Proof of** $x$: $y \doteqdot [x \overset{\text{proof}}{\rightarrow} \lambda p.\lambda c.\text{taceval}\ (\ \lceil y \rceil\ ,\ \text{tacstate0}\ ,\ c\ )]$].

A proof is the same as a proof definition.

[$x$ **proof of** $y$: $z \doteqdot [y \overset{\text{proof}}{\rightarrow} \lambda p.\lambda c.\text{taceval1}\ (\ \lceil x \rceil\ ,\ \lceil z \rceil\ ,\ c\ )]$].

Assume $x$, then prove $y$ using the proof $z$. If $x$ has a tactic definition then that tactic is applied to the entire proof.

### 4.10 Proof line macros

[L$l$: $a \gg x$; $\doteqdot a \gg x$]

[L$l$: $a \gg x\Box \doteqdot a \gg x$]

[L$l$: Arbitrary $\gg x$; $n \doteqdot \Pi x$: $n$]

[L$l$: Local $\gg x \doteqdot y$; $n \doteqdot$ **let** $x \doteqdot y$ **in** $n$]

## 5 Tactic definitions of sequent operators

### 5.1 Init tactic

[$x^{\text{I}} \overset{\text{tactic}}{=} \lambda u.\text{tactic-Init}\ (\ u\ )$].

[tactic-Init ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $\langle \mathsf{T}, x\rangle = t$ **in**
$s$[hook-arg$\rightarrow t$][hook-res$\rightarrow x$]]

### 5.2 Ponens tactic

[$x^{\triangleright} \overset{\text{tactic}}{=} \lambda u.\text{tactic-Ponens}\ (\ u\ )$].

[tactic-Ponens ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $\langle \mathsf{T}, x\rangle = t$ **in**
**let** $s = \text{taceval}\ (\ x\ ,\ s\ ,\ c\ )$ **in**
**let** $a = \lceil_t\, \underline{s[\text{hook-arg}]^{\triangleright}} \rceil$ **in**
**let** $r = \underline{s[\text{hook-res}]}$ **in**
**if** mismatch ( $\lceil x \vdash y \rceil$ , $r$ , $c$ ) **then**
error ( $t$ , diag ( 'Ponens tactic used on non-inference:' ) disp ( $r$ )
end diagnose ) **else**
$s$[hook-arg$\rightarrow a$][hook-res$\rightarrow r^2$]]

## 5.3 Probans tactic

$[x^{\Vdash} \overset{\text{tactic}}{=} \lambda u.\text{tactic-Probans ( } u \text{ )}]$.

$[\text{tactic-Probans ( } u \text{ )} \overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $\langle \mathsf{T}, x\rangle = t$ **in**
**let** $s = \text{taceval ( } x \text{ , } s \text{ , } c \text{ )}$ **in**
**let** $a = \lceil_t\, s[\underline{\text{hook-arg}}]^{\Vdash} \rceil$ **in**
**let** $r = s[\overline{\text{hook-res}}]$ **in**
**if** mismatch ( $\lceil x \Vdash y \rceil$ , $r$ , $c$ ) **then**
error ( $t$ , diag ( 'Probans tactic used on non-endorsement:' ) disp ( $r$ ) end diagnose ) **else**
$s[\text{hook-arg}{\rightarrow}a][\text{hook-res}{\rightarrow}r^2]]$

## 5.4 Verify tactic

$[x^{*} \overset{\text{tactic}}{=} \lambda u.\text{tactic-Verify ( } u \text{ )}]$.

$[\text{tactic-Verify ( } u \text{ )} \overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $\langle \mathsf{T}, x\rangle = t$ **in**
**let** $s = \text{taceval ( } x \text{ , } s \text{ , } c \text{ )}$ **in**
**let** $a = \lceil_t\, s[\underline{\text{hook-arg}}]^{*} \rceil$ **in**
**let** $r = s[\overline{\text{hook-res}}]$ **in**
**if** mismatch ( $\lceil x \Vdash y \rceil$ , $r$ , $c$ ) **then**
error ( $t$ , diag ( 'Verify tactic used on non-endorsement:' ) disp ( $r$ ) end diagnose ) **else**
$s[\text{hook-arg}{\rightarrow}a][\text{hook-res}{\rightarrow}r^2]]$

## 5.5 Curry tactic

$[x^{\mathrm{C}} \overset{\text{tactic}}{=} \lambda u.\text{tactic-Curry ( } u \text{ )}]$.

$[\text{tactic-Curry ( } u \text{ )} \overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $\langle \mathsf{T}, x\rangle = t$ **in**
**let** $s = \text{taceval ( } x \text{ , } s \text{ , } c \text{ )}$ **in**
**let** $a = \lceil_t\, s[\underline{\text{hook-arg}}]^{\mathrm{C}} \rceil$ **in**
**let** $r = s[\overline{\text{hook-res}}]$ **in**
**if** mismatch ( $\lceil (x \oplus y) \vdash z \rceil$ , $r$ , $c$ ) **then**
error ( $t$ , diag ( 'Curry tactic used on unfit argument:' ) disp ( $r$ ) end diagnose ) **else**
**let** $r = \lceil_r\, \underline{r^{11}} \vdash \underline{r^{12}} \vdash \underline{r^2} \rceil$ **in**
$s[\text{hook-arg}{\rightarrow}a][\text{hook-res}{\rightarrow}r]]$

## 5.6 Uncurry tactic

$[x^{\mathrm{U}} \overset{\text{tactic}}{=} \lambda u.\text{tactic-Uncurry ( } u \text{ )}]$.

[tactic-Uncurry ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $\langle \mathsf{T}, x\rangle = t$ **in**
**let** $s =$ taceval ( $x$ , $s$ , $c$ ) **in**
**let** $a = \lceil_t s[\overline{\text{hook-arg}}]^{\mathrm{U}} \rceil$ **in**
**let** $r = s[\overline{\text{hook-res}}]$ **in**
**if** mismatch ( $\lceil x \vdash y \vdash z\rceil$ , $r$ , $c$ ) **then**
error ( $t$ , diag ( 'Uncurry tactic used on unfit argument:' ) disp ( $r$ ) end diagnose ) **else**
**let** $r = \lceil_r (\underline{r^1} \oplus \underline{r^{21}}) \vdash \underline{r^{22}}\rceil$ **in**
$s$[hook-arg→$a$][hook-res→$r$]]

## 5.7 Deref tactic

$[x^{\mathrm{D}} \overset{\text{tactic}}{=} \lambda u.\text{tactic-Deref ( } u \text{ )}]$.

[tactic-Deref ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $\langle \mathsf{T}, x\rangle = t$ **in**
**let** $s =$ taceval ( $x$ , $s$ , $c$ ) **in**
**let** $a = \lceil_t s[\overline{\text{hook-arg}}]^{\mathrm{D}} \rceil$ **in**
**let** $r = s[\overline{\text{hook-res}}]$ **in**
**let** $r' = r$ stmt-rhs ( $c$ ) **in**
**if** $r' = \mathsf{T}$ **then**
error ( $t$ , diag ( 'Deref tactic used on undefined statement:' ) disp ( $r$ ) end diagnose ) **else**
$s$[hook-arg→$a$][hook-res→$r'$]]

## 5.8 At tactic

Accumulate instantiations on the "parameters" stack. To do multiple instantiations in parallel, tactic-at1 ( $t$ , $s$ , $v$ , $c$ ) accumulates instantiations on the $v$ list and then tactic-at2 ( $t$ , $s$ , $v$ , $s'$ , $c$ ) accumulates instantiations on the $s'$ association list of substitutions to be performed in parallel.

$[x \mathbin{@} y \overset{\text{tactic}}{=} \lambda u.\text{tactic-at ( } u \text{ )}]$.

[tactic-at ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in** tactic-at1 ( $t$ , $s$ , $\mathsf{T}$ , $c$ )]

[tactic-at1 ( $t$ , $s$ , $v$ , $c$ ) $\overset{\bullet}{=}$
**if** mismatch ( $\lceil x \mathbin{@} y\rceil$ , $t$ , $c$ ) **then**
tactic-at2 ( $t$ , taceval ( $t$ , $s$ , $c$ ) , $v$ , $\mathsf{T}$ , $c$ ) **else**

**let** $\langle \mathsf{T}, x, y \rangle = t$ **in**
**let** $s =$ tactic-push ( hook-parm , $y$ , $s$ ) **in**
tactic-at1 ( $x$ , $s$ , $y :: v$ , $c$ )]

[tactic-at2 ( $t$ , $s$ , $v$ , $s$ prime , $c$ ) $\overset{\bullet}{=}$
**if** $v$ **then** $s$[hook-res→metasubst ( $s$[hook-res] , $s'$ , $c$ )] **else**
**let** $y :: v = v$ **in**
**let** $a = \lceil_t \, s[\underline{\text{hook-arg}}] \mathbin{@} \underline{y} \rceil$ **in**
**let** $r = s[\overline{\text{hook-res}}]$ **in**
**if** mismatch ( $\lceil \Pi x \colon y \rceil$ , $r$ , $c$ ) **then**
error ( $t$ , diag ( 'At tactic used on non-quantifier:' ) disp ( $r$ ) end
diagnose ) **else**
**let** $s = s$[hook-arg→$a$][hook-res→$r^2$] **in**
tactic-at2 ( $t$ , $s$ , $v$ , $(r^1 :: y) :: s'$ , $c$ )]

## 5.9 Infer tactic

Accumulate premises on the "premises" stack.

$[x \vdash y \overset{\text{tactic}}{=} \lambda u.\text{tactic-infer} \; ( \; u \; )]$.

[tactic-infer ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c \rangle = u$ **in**
**let** $\langle \mathsf{T}, x, y \rangle = t$ **in**
**let** $s =$ tactic-push ( hook-pre , $\langle \mathsf{T}, x, \lceil_x \underline{x}^{\mathrm{I}} \rceil \rangle$ , $s$ ) **in**
**let** $s =$ taceval ( $y$ , $s$ , $c$ ) **in**
**let** $a = \lceil_t \underline{x} \vdash s[\underline{\text{hook-arg}}] \rceil$ **in**
**let** $r = \lceil_t \underline{x} \vdash \overline{s[\overline{\text{hook-res}}]} \rceil$ **in**
$s$[hook-arg→$a$][hook-res→$r$]]

## 5.10 Endorse tactic

Accumulate side conditions on the "conditions" stack.

$[x \Vdash y \overset{\text{tactic}}{=} \lambda u.\text{tactic-endorse} \; ( \; u \; )]$.

[tactic-endorse ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c \rangle = u$ **in**
**let** $\langle \mathsf{T}, x, y \rangle = t$ **in**
**let** $s =$ taceval ( $y$ , tactic-push ( hook-cond , $\langle \mathsf{T}, x \rangle$ , $s$ ) , $c$ ) **in**
**let** $a = \lceil_t \underline{x} \Vdash \underline{s[\text{hook-arg}]} \rceil$ **in**
**let** $r = \lceil_t \underline{x} \Vdash \overline{s[\overline{\text{hook-res}}]} \rceil$ **in**
$s$[hook-arg→$a$][hook-res→$r$]]

## 5.11 Id est tactic

$[x \text{ ie } y \overset{\text{tactic}}{=} \lambda u.\text{tactic-ie ( } u \text{ )}]$.

$[\text{tactic-ie ( } u \text{ )} \overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $\langle \mathsf{T}, x, y\rangle = t$ **in**
**let** $s = \text{taceval ( } x \text{ , } s \text{ , } c \text{ )}$ **in**
**let** $a = \lceil_t \underline{s[\text{hook-arg}]} \text{ ie } \underline{y} \rceil$ **in**
$s[\text{hook-arg} \rightarrow a][\text{hook-res} \rightarrow y]]$

## 5.12 All tactic

Pop instantiations from the "parameters" stack.

$[\Pi x\colon y \overset{\text{tactic}}{=} \lambda u.\text{tactic-all ( } u \text{ )}]$.

$[\text{tactic-all ( } u \text{ )} \overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $\langle \mathsf{T}, x, y\rangle = t$ **in**
**let** $s = \text{taceval ( } y \text{ , tactic-pop ( hook-parm , } s \text{ ) , } c \text{ )}$ **in**
**let** $a = \lceil_t \Pi \underline{x}\colon \underline{s[\text{hook-arg}]} \rceil$ **in**
**let** $r = \lceil_t \Pi \underline{x}\colon \underline{s[\text{hook-res}]} \rceil$ **in**
$s[\text{hook-arg} \rightarrow a][\text{hook-res} \rightarrow r]]$

## 5.13 Cut tactic

Accumulate conclusions of left hand sides of cuts on the "premises" stack.

$[x; y \overset{\text{tactic}}{=} \lambda u.\text{tactic-cut ( } u \text{ )}]$.

$[\text{tactic-cut ( } u \text{ )} \overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $\langle \mathsf{T}, x, y\rangle = t$ **in**
**let** $s' = \text{taceval ( } x \text{ , } s \text{ , } c \text{ )}$ **in**
**let** $a = s'[\text{hook-arg}]$ **in**
**let** $r = s'[\text{hook-res}]$ **in**
**let** $s = \text{tactic-push ( hook-pre , } \langle \mathsf{T}, r, \lceil_r \underline{r}^{\mathrm{I}} \rceil\rangle \text{ , } s \text{ )}$ **in**
**let** $s = \text{taceval ( } y \text{ , } s \text{ , } c \text{ )}$ **in**
**let** $a = \lceil_t \underline{a}; \underline{s[\text{hook-arg}]} \rceil$ **in**
$s[\text{hook-arg} \rightarrow a]]$

# 6 Unification tactic

## 6.1 Main definitions

$[a \gg a \text{ prime} \overset{\text{tactic}}{=} \lambda x.\text{unitac ( } x \text{ )}]$.

Proof tactic which tries to modify the argumentation $a$ so that it proves the term $a'$. The tactic adds missing $x$ @ $y$, $x^{\triangleright}$, and $x^*$ operations and determines the $y$ in $x$ @ $y$ using unification.

```
[unitac ( x ) ≐
    let ⟨t, s, c⟩ = x in
    let e :: v = unitac1 ( t , s , c )° in
    if not e then v else
    if not v then error ( t , LocateProofLine ( t , c ) v ) else
    error ( t ,
    LocateProofLine ( t , c )
    diag ( 'During unification: Uncaught exception' )
    end diagnose )]
```

Hand the term $t$ of form $a \gg a'$, the tactic state $s$, and the codex $c$ to unitac1. Locate any errors found by unitac1.

```
[unitac1 ( t , s , c ) ≐
    let s = s[hook-idx→1][hook-uni→⊤] in
    let s = unieval ( t , s , c ) in
    let t = inst ( s[hook-arg] , t , s[hook-uni] ) in
    let e :: v = taceval ( t , s , c )° in if not e then v else
    if not v then v• else
    error ( t ,
    diag ( 'Post unification tactic evalutation: Uncaught exception' )
    end diagnose )]
```

Call unieval to unitac expand the proof term $t$ to a unification and an uninstantiated argumentation. The unification and argumentation are added to the state $s$. $c$ is the cache. Then instantiate the argumentation and do usual tactic expansion.

## 6.2 Adaption

```
[unitac-adapt ( t , s , c ) ≐
    if t = var then s else
    let a = s[hook-arg] in
    let r = s[hook-res] in
    let d = r metadef ( c ) in
    if d = var then s else
    if d = t then s else
    if d = infer then
    unitac-adapt ( t , s[hook-arg→⌈a a▷⌉][hook-res→r²] , c ) else
    if d = endorse then
    unitac-adapt ( t , s[hook-arg→⌈a a*⌉][hook-res→r²] , c ) else
    if d = all then
    let i = s[hook-idx] in
    let s = s[hook-idx→i + 1] in
```

**let** $v$ = univar ( $a$ , $r^1$ , $i$ ) **in**
**let** $r'$ = metasubst ( $r^2$ , $\langle r^1 :: v \rangle$ , $c$ ) **in**
unitac-adapt ( $t$ , $s$[hook-arg→$\lceil_a \underline{a} \mathbin{@} \underline{v} \rceil$][hook-res→$r'$] , $c$ ) **else**
**if** $t$ **then** $s$ **else**
error ( $a$ ,
diag ( 'Cannot convert' ) disp ( $r$ )
diag ( 'to type "' ) diag ( $t$ ) diag ( '"' ) end diagnose )].

Add Ponens, Verify, and At operations to the argumentation inside $s$ until the root of the conclusion is of type $t$. As examples, $t$ could be "infer", "endorse", "all", "var" or $\mathsf{T}$. In the latter case, a maximum of Ponens, Verify, and At operations are added. When $t$ is "var", a minimum of operators are added (meaning that $s$ is returned unchanged).

## 6.3 Unitac evaluation

[unieval ( $t$ , $s$ , $c$ ) $\overset{\bullet}{=}$
**let** $d = t$ unitac-rhs ( $c$ ) **in**
**if** $d$ **then**
**let** $\langle r, a \rangle$ = lookup ( $t$ , $s$[hook-pre] , $\mathsf{T}$ ) **in**
**if not** $a$ **then** $s$[hook-arg→$a$][hook-res→$r$] **else**
error ( $t$ ,
diag ( 'Unknown unitac operator in root of argumentation:' )
disp ( $t$ ) end diagnose ) **else**
**let** $e :: s$ = (eval ( $d$ , $\mathsf{T}$ , $c$ ) " $\langle t, s, c \rangle^{\mathrm{M}})^{\mathrm{U}\circ}$ **in**
**if not** $e$ **then** $s$ **else**
**if not** $s$ **then** $s^\bullet$ **else**
error ( $t$ ,
diag ( 'Exception raised by the unitac aspect of:' ) disp ( $t$ ) end diagnose )]

Unitac expand the argumentation $t$ to a unification, an uninstantiated argumentation, and an expected conclusion. The unification, argumentation, and conclusion are added to the state $s$ as $s$ [hook-uni], $s$[hook-arg], and $s$[hook-res], respectively. $c$ is the cache.

During unitac expansion, fresh variables are generated using univar ( $t'$ , $v$ , $s$[hook-idx] ) where $s$[hook-idx] is supposed to be a unique index. $v$ is the varible which receives the unique index and debugging information is taken from $t'$. The index is supposed to guarantee uniqueness in itself. The purpose of $v$ is to make error messages more readable.

# 7 Unitac definitions

## 7.1 Init

$[x^{\mathrm{I}} \overset{\text{unitac}}{=} \lambda u.\text{unitac-Init ( } u \text{ )}]$

[unitac-Init ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
$s$[hook-res$\rightarrow t^1$][hook-arg$\rightarrow t$]]

If $t$ has form $r^{\mathrm{I}}$ then the conclusion is $r$ and the argumentation is $r^{\mathrm{I}}$.

## 7.2 Ponens

$[x^{\triangleright} \overset{\text{unitac}}{=} \lambda u.\text{unitac-Ponens ( } u \text{ )}]$

$[x \triangleright y \overset{\text{unitac}}{=} \lambda u.\text{unitac-ponens ( } u \text{ )}]$

[unitac-Ponens ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $s$ = unieval ( $t^1$ , $s$ , $c$ ) **in**
**let** $s$ = unitac-adapt ( infer , $s$ , $c$ ) **in**
**let** $a = s$[hook-arg] **in**
**let** $r = s$[hook-res] **in**
$s$[hook-arg$\rightarrow\lceil_t \underline{a}^{\triangleright}\rceil$][hook-res$\rightarrow r^2$]]

If the argumentation has form $a^{\triangleright}$ then convert the argumentation $a$ into something whose conclusion $r$ is expected to be of form $x \vdash y$.

[unitac-ponens ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $s$ = unieval ( $t^1$ , $s$ , $c$ ) **in**
**let** $s$ = unitac-adapt ( infer , $s$ , $c$ ) **in**
**let** $a = s$[hook-arg] **in**
**let** $r = s$[hook-res] **in**
**let** $s$ = unieval ( $t^2$ , $s$ , $c$ ) **in**
**let** $s$ = unitac-adapt ( $r^1$ metadef ( $c$ ) , $s$ , $c$ ) **in**
**let** $a' = s$[hook-arg] **in**
**let** $r' = s$[hook-res] **in**
**let** $u = s$[hook-uni] **in**
**let** $u$ = unify ( $r^1$ , $r'$ , $u$ ) **in**
**let** $s = s$[hook-uni$\rightarrow u$] **in**
$s$[hook-arg$\rightarrow\lceil_t \underline{a'}; \underline{a}^{\triangleright}\rceil$][hook-res$\rightarrow r^2$]]

If the argumentation has form $a \triangleright a'$ then convert $a$ into something whose result is expected to be of form $x \vdash y$. Convert $a'$ into something whose result is expected to be of form $r'$. Unify $x$ with $r'$ and return argumentation $(a'; a^{\triangleright})$ and conclusion $y$.

## 7.3 Probans

$[x^{\Rrightarrow} \overset{\text{unitac}}{=} \lambda u.\text{unitac-Probans}\ (\ u\ )]$

$[x \Rrightarrow y \overset{\text{unitac}}{=} \lambda u.\text{unitac-probans}\ (\ u\ )]$

$[\text{unitac-Probans}\ (\ u\ ) \overset{\bullet}{=}$
$\quad \textbf{let}\ \langle t, s, c\rangle = u\ \textbf{in}$
$\quad \textbf{let}\ s = \text{unieval}\ (\ t^1\ ,\ s\ ,\ c\ )\ \textbf{in}$
$\quad \textbf{let}\ s = \text{unitac-adapt}\ (\ \text{endorse}\ ,\ s\ ,\ c\ )\ \textbf{in}$
$\quad \textbf{let}\ a = s[\text{hook-arg}]\ \textbf{in}$
$\quad \textbf{let}\ r = s[\text{hook-res}]\ \textbf{in}$
$\quad s[\text{hook-arg}{\rightarrow}\lceil_t\, \underline{a}^{\Rrightarrow}\rceil][\text{hook-res}{\rightarrow}r^2]]$

If the argumentation has form $a \Rrightarrow p$ then convert the argumentation $a$ into something whose conclusion $r$ is expected to be of form $x \Vdash y$.

$[\text{unitac-probans}\ (\ u\ ) \overset{\bullet}{=}$
$\quad \textbf{let}\ \langle t, s, c\rangle = u\ \textbf{in}$
$\quad \textbf{let}\ s = \text{unieval}\ (\ t^1\ ,\ s\ ,\ c\ )\ \textbf{in}$
$\quad \textbf{let}\ s = \text{unitac-adapt}\ (\ \text{endorse}\ ,\ s\ ,\ c\ )\ \textbf{in}$
$\quad \textbf{let}\ a = s[\text{hook-arg}]\ \textbf{in}$
$\quad \textbf{let}\ r = s[\text{hook-res}]\ \textbf{in}$
$\quad \textbf{let}\ u = s[\text{hook-uni}]\ \textbf{in}$
$\quad \textbf{let}\ \langle p\rangle = \text{lookup}\ (\ t^2\ ,\ s[\text{hook-cond}]\ ,\ \langle t^2\rangle\ )\ \textbf{in}$
$\quad \textbf{let}\ u = \text{unify}\ (\ r^1\ ,\ p\ ,\ u\ )\ \textbf{in}$
$\quad \textbf{let}\ s = s[\text{hook-uni}{\rightarrow}u]\ \textbf{in}$
$\quad s[\text{hook-arg}{\rightarrow}\lceil_t\, \underline{a}^{\Rrightarrow}\rceil][\text{hook-res}{\rightarrow}r^2]]$

If the argumentation has form $a \Rrightarrow s$ then convert $a$ into something whose result is expected to be of form $x \Vdash y$. Unify $x$ with $s$ and return argumentation $a^{\Rrightarrow}$ and conclusion $y$.

## 7.4 Verify

$[x^* \overset{\text{unitac}}{=} \lambda u.\text{unitac-Verify}\ (\ u\ )]$

$[x * y \overset{\text{unitac}}{=} \lambda u.\text{unitac-verify}\ (\ u\ )]$

$[\text{unitac-Verify}\ (\ u\ ) \overset{\bullet}{=}$
$\quad \textbf{let}\ \langle t, s, c\rangle = u\ \textbf{in}$
$\quad \textbf{let}\ s = \text{unieval}\ (\ t^1\ ,\ s\ ,\ c\ )\ \textbf{in}$
$\quad \textbf{let}\ s = \text{unitac-adapt}\ (\ \text{endorse}\ ,\ s\ ,\ c\ )\ \textbf{in}$
$\quad \textbf{let}\ a = s[\text{hook-arg}]\ \textbf{in}$
$\quad \textbf{let}\ r = s[\text{hook-res}]\ \textbf{in}$
$\quad s[\text{hook-arg}{\rightarrow}\lceil_t\, \underline{a}^*\rceil][\text{hook-res}{\rightarrow}r^2]]$

If the argumentation has form $a^*$ then convert the argumentation $a$ into something whose conclusion $r$ is expected to be of form $x \Vdash y$.

[unitac-verify ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $s =$ unieval ( $t^1$ , $s$ , $c$ ) **in**
**let** $s =$ unitac-adapt ( endorse , $s$ , $c$ ) **in**
**let** $a = s$[hook-arg] **in**
**let** $r = s$[hook-res] **in**
**let** $u = s$[hook-uni] **in**
**let** $u =$ unify ( $r^1$ , $t^2$ , $u$ ) **in**
**let** $s = s$[hook-uni$\rightarrow u$] **in**
$s$[hook-arg$\rightarrow\lceil_t \underline{a}^*\rceil$][hook-res$\rightarrow r^2$]]

If the argumentation has form $a * s$ then convert $a$ into something whose result is expected to be of form $x \Vdash y$. Unify $x$ with $s$ and return argumentation $a^*$ and conclusion $y$.

## 7.5 Curry and decurry

[$x^{\mathrm{C}} \overset{\text{unitac}}{=} \lambda u.$unitac-Curry ( $u$ )]

[$x^{\mathrm{U}} \overset{\text{unitac}}{=} \lambda u.$unitac-Uncurry ( $u$ )]

[unitac-Curry ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $s =$ unieval ( $t^1$ , $s$ , $c$ ) **in**
**let** $s =$ unitac-adapt ( infer , $s$ , $c$ ) **in**
**let** $a = s$[hook-arg] **in**
**let** $r = s$[hook-res] **in**
**if not** $r^1 \overset{\mathrm{r}}{=} \lceil x \oplus y\rceil$ **then**
error ( $r$ , diag ( 'Unsuited for currying:' ) disp ( $r$ ) end diagnose ) **else**
**let** $r = \lceil_r \underline{r^{11}} \vdash \underline{r^{12}} \vdash \underline{r^2}\rceil$ **in**
$s$[hook-arg$\rightarrow\lceil_t \underline{a}^{\mathrm{C}}\rceil$][hook-res$\rightarrow r$]]

If the argumentation has form $a^{\mathrm{C}}$ then convert the argumentation $a$ into something with conclusion $x \oplus y \vdash z$ and return conclusion $x \vdash y \vdash z$.

[unitac-Uncurry ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $s =$ unieval ( $t^1$ , $s$ , $c$ ) **in**
**let** $s =$ unitac-adapt ( infer , $s$ , $c$ ) **in**
**let** $a = s$[hook-arg] **in**
**let** $r = s$[hook-res] **in**
**if not** $r^2 \overset{\mathrm{r}}{=} \lceil x \vdash y\rceil$ **then**
error ( $r$ , diag ( 'Unsuited for uncurrying:' ) disp ( $r$ ) end diagnose ) **else**
**let** $r = \lceil_r (\underline{r^1} \oplus \underline{r^{21}}) \vdash \underline{r^{22}}\rceil$ **in**
$s$[hook-arg$\rightarrow\lceil_t \underline{a}^{\mathrm{U}}\rceil$][hook-res$\rightarrow r$]]

If the argumentation has form $a^{\mathrm{U}}$ then convert the argumentation $a$ into something with conclusion $x \vdash y \vdash z$ and return conclusion $(x \oplus y) \vdash z$.

## 7.6 At

$[x^{@} \overset{\text{unitac}}{=} \lambda u.\text{unitac-At}\ (\ u\ )]$

$[x\ @\ y \overset{\text{unitac}}{=} \lambda u.\text{unitac-at}\ (\ u\ )]$

$[\text{unitac-At}\ (\ u\ ) \overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $s = \text{unieval}\ (\ t^1\ ,\ s\ ,\ c\ )$ **in**
**let** $s = \text{unitac-adapt}\ (\ \text{all}\ ,\ s\ ,\ c\ )$ **in**
**let** $a = s[\text{hook-arg}]$ **in**
**let** $r = s[\text{hook-res}]$ **in**
**let** $i = s[\text{hook-idx}]$ **in**
**let** $s = s[\text{hook-idx}{\rightarrow}i + 1]$ **in**
**let** $v = \text{univar}\ (\ a\ ,\ r^1\ ,\ i\ )$ **in**
**let** $a = \lceil_t\ \underline{a}\ @\ \underline{v} \rceil$ **in**
**let** $r = \text{metasubst}\ (\ r^2\ ,\ \langle r^1 :: v\rangle\ ,\ c\ )$ **in**
$s[\text{hook-arg}{\rightarrow}a][\text{hook-res}{\rightarrow}r]]$

If the argumentation has form $a^{@}$ then convert $a$ into something whose conclusion is expected to be of form $\Pi x\colon y$. Then replace $x$ by a fresh variable $v$ in $y$ and return argumentation $a\ @\ v$ and conclusion $y$.

$[\text{unitac-at}\ (\ u\ ) \overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $s = \text{unieval}\ (\ t^1\ ,\ s\ ,\ c\ )$ **in**
**let** $s = \text{unitac-adapt}\ (\ \text{all}\ ,\ s\ ,\ c\ )$ **in**
**let** $a = s[\text{hook-arg}]$ **in**
**let** $r = s[\text{hook-res}]$ **in**
**let** $i = s[\text{hook-idx}]$ **in**
**let** $s = s[\text{hook-idx}{\rightarrow}i + 1]$ **in**
**let** $v = \text{univar}\ (\ a\ ,\ r^1\ ,\ i\ )$ **in**
**let** $a = \lceil_t\ \underline{a}\ @\ \underline{v} \rceil$ **in**
**let** $r = \text{metasubst}\ (\ r^2\ ,\ \langle r^1 :: v\rangle\ ,\ c\ )$ **in**
**let** $u = s[\text{hook-uni}]$ **in**
**let** $u = \text{unify}\ (\ t^2\ ,\ v\ ,\ u\ )$ **in**
**let** $s = s[\text{hook-uni}{\rightarrow}u]$ **in**
$s[\text{hook-arg}{\rightarrow}a][\text{hook-res}{\rightarrow}r]]$

If the argumentation has form $a\ @\ a'$ then convert $a$ into something whose conclusion is expected to be of form $\Pi x\colon y$. Then replace $x$ by a fresh variable $v$ in $y$, unify $v$ with $a'$, and return argumentation $a\ @\ v$ and conclusion $y$.

## 7.7 Reference and dereference

$[x^{\mathrm{D}} \overset{\text{unitac}}{=} \lambda u.\text{unitac-Deref ( } u \text{ )}]$

$[\text{unitac-Deref ( } u \text{ )} \overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $s = \text{unieval ( } t^1 \text{ , } s \text{ , } c \text{ )}$ **in**
**let** $s = \text{unitac-adapt ( } \top \text{ , } s \text{ , } c \text{ )}$ **in**
**let** $r = s[\text{hook-res}]$ **in**
**let** $l = r \text{ stmt-rhs ( } c \text{ )}$ **in**
**if** $l$ **then** $s$ **else**
**let** $a = s[\text{hook-arg}]$ **in**
**let** $a = \lceil_t \underline{a}^{\mathrm{D}} \rceil$ **in**
$s[\text{hook-arg}{\rightarrow}a][\text{hook-res}{\rightarrow}l]]$

Let $r$ be the conclusion of $t^1$. $r$ is supposed to be a lemma. Let $l$ be the contents of that lemma. Use $l$ as conclusion of $t^{1\mathrm{D}}$.

$[x \text{ ie } y \overset{\text{unitac}}{=} \lambda u.\text{unitac-idest ( } u \text{ )}]$

$[\text{unitac-idest ( } u \text{ )} \overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $s = \text{unieval ( } t^1 \text{ , } s \text{ , } c \text{ )}$ **in**
**let** $l = t^2 \text{ stmt-rhs ( } c \text{ )}$ **in**
**let** $s = \text{unitac-adapt ( } l \text{ metadef ( } c \text{ ) , } s \text{ , } c \text{ )}$ **in**
**let** $r = s[\text{hook-res}]$ **in**
**let** $u = s[\text{hook-uni}]$ **in**
**let** $u = \text{unify ( } r \text{ , } l \text{ , } u \text{ )}$ **in**
**let** $s = s[\text{hook-uni}{\rightarrow}u]$ **in**
**let** $a = s[\text{hook-arg}]$ **in**
**let** $a = \lceil_t \underline{a} \text{ ie } \underline{t^2} \rceil$ **in**
$s[\text{hook-arg}{\rightarrow}a][\text{hook-res}{\rightarrow}t^2]]$

Let $l$ be the contents of the lemma named $t^2$. Unify the conclusion $r$ of $t^1$ with $l$ and take $t^2$ to be the conclusion of $t^1$ ie $t^2$.

## 7.8 Infer

$[x^{\vdash} \overset{\text{unitac}}{=} \lambda u.\text{unitac-Infer ( } u \text{ )}]$

$[\text{unitac-Infer ( } u \text{ )} \overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $s = \text{unieval ( } t^1 \text{ , } s \text{ , } c \text{ )}$ **in**
**let** $i = s[\text{hook-idx}]$ **in**
**let** $s = s[\text{hook-idx}{\rightarrow}i + 1]$ **in**
**let** $v = \text{univar ( } t \text{ , } \lceil\mathcal{A}\rceil \text{ , } i \text{ )}$ **in**
**let** $r = s[\text{hook-res}]$ **in**
**let** $a = s[\text{hook-arg}]$ **in**

$$\begin{array}{l}\quad\textbf{let } r = \lceil_t \underline{v} \vdash \underline{r} \rceil \textbf{ in}\\ \quad\textbf{let } a = \lceil_t \underline{v} \vdash \underline{a} \rceil \textbf{ in}\\ \quad s[\text{hook-arg}{\rightarrow}a][\text{hook-res}{\rightarrow}r]]\end{array}$$

$$[x \vdash y \overset{\text{unitac}}{=} \lambda u.\text{unitac-infer } (\ u\ )]$$

$$\begin{array}{l}[\text{unitac-infer } (\ u\ ) \overset{\bullet}{=}\\ \quad\textbf{let } \langle t, s, c\rangle = u \textbf{ in}\\ \quad\textbf{let } s = \text{unieval } (\ t^2\ ,\ s\ ,\ c\ ) \textbf{ in}\\ \quad\textbf{let } r = s[\text{hook-res}] \textbf{ in}\\ \quad\textbf{let } a = s[\text{hook-arg}] \textbf{ in}\\ \quad\textbf{let } r = \lceil_t \underline{t^1} \vdash \underline{r} \rceil \textbf{ in}\\ \quad\textbf{let } a = \lceil_t \underline{t^1} \vdash \underline{a} \rceil \textbf{ in}\\ \quad s[\text{hook-arg}{\rightarrow}a][\text{hook-res}{\rightarrow}r]]\end{array}$$

## 7.9 Endorse

$$[x^{\Vdash} \overset{\text{unitac}}{=} \lambda u.\text{unitac-Endorse } (\ u\ )]$$

$$\begin{array}{l}[\text{unitac-Endorse } (\ u\ ) \overset{\bullet}{=}\\ \quad\textbf{let } \langle t, s, c\rangle = u \textbf{ in}\\ \quad\textbf{let } s = \text{unieval } (\ t^1\ ,\ s\ ,\ c\ ) \textbf{ in}\\ \quad\textbf{let } i = s[\text{hook-idx}] \textbf{ in}\\ \quad\textbf{let } s = s[\text{hook-idx}{\rightarrow}i+1] \textbf{ in}\\ \quad\textbf{let } v = \text{univar } (\ t\ ,\ \lceil\mathcal{A}\rceil\ ,\ i\ ) \textbf{ in}\\ \quad\textbf{let } r = s[\text{hook-res}] \textbf{ in}\\ \quad\textbf{let } a = s[\text{hook-arg}] \textbf{ in}\\ \quad\textbf{let } r = \lceil_t \underline{v} \Vdash \underline{r} \rceil \textbf{ in}\\ \quad\textbf{let } a = \lceil_t \underline{v} \Vdash \underline{a} \rceil \textbf{ in}\\ \quad s[\text{hook-arg}{\rightarrow}a][\text{hook-res}{\rightarrow}r]]\end{array}$$

$$[x \Vdash y \overset{\text{unitac}}{=} \lambda u.\text{unitac-endorse } (\ u\ )]$$

$$\begin{array}{l}[\text{unitac-endorse } (\ u\ ) \overset{\bullet}{=}\\ \quad\textbf{let } \langle t, s, c\rangle = u \textbf{ in}\\ \quad\textbf{let } s = \text{unieval } (\ t^2\ ,\ s\ ,\ c\ ) \textbf{ in}\\ \quad\textbf{let } r = s[\text{hook-res}] \textbf{ in}\\ \quad\textbf{let } a = s[\text{hook-arg}] \textbf{ in}\\ \quad\textbf{let } r = \lceil_t \underline{t^1} \Vdash \underline{r} \rceil \textbf{ in}\\ \quad\textbf{let } a = \lceil_t \underline{t^1} \Vdash \underline{a} \rceil \textbf{ in}\\ \quad s[\text{hook-arg}{\rightarrow}a][\text{hook-res}{\rightarrow}r]]\end{array}$$

## 7.10 All

$$[x^{\Pi} \overset{\text{unitac}}{=} \lambda u.\text{unitac-All } (\ u\ )]$$

$$[\text{unitac-All}\ (\ u\ ) \overset{\bullet}{=}$$
$$\quad \textbf{let}\ \langle t, s, c\rangle = u\ \textbf{in}$$
$$\quad \textbf{let}\ s = \text{unieval}\ (\ t^1\ ,\ s\ ,\ c\ )\ \textbf{in}$$
$$\quad \textbf{let}\ i = s[\text{hook-idx}]\ \textbf{in}$$
$$\quad \textbf{let}\ s = s[\text{hook-idx}{\rightarrow}i+1]\ \textbf{in}$$
$$\quad \textbf{let}\ v = \text{univar}\ (\ t\ ,\ \lceil\mathcal{A}\rceil\ ,\ i\ )\ \textbf{in}$$
$$\quad \textbf{let}\ r = s[\text{hook-res}]\ \textbf{in}$$
$$\quad \textbf{let}\ a = s[\text{hook-arg}]\ \textbf{in}$$
$$\quad \textbf{let}\ r = \lceil_t\, \Pi\underline{v}\colon \underline{r}\rceil\ \textbf{in}$$
$$\quad \textbf{let}\ a = \lceil_t\, \Pi\underline{v}\colon \underline{a}\rceil\ \textbf{in}$$
$$\quad s[\text{hook-arg}{\rightarrow}a][\text{hook-res}{\rightarrow}r]]$$

$$[\Pi x\colon y \overset{\text{unitac}}{=} \lambda u.\text{unitac-all}\ (\ u\ )]$$

$$[\text{unitac-all}\ (\ u\ ) \overset{\bullet}{=}$$
$$\quad \textbf{let}\ \langle t, s, c\rangle = u\ \textbf{in}$$
$$\quad \textbf{let}\ s = \text{unieval}\ (\ t^2\ ,\ s\ ,\ c\ )\ \textbf{in}$$
$$\quad \textbf{let}\ r = s[\text{hook-res}]\ \textbf{in}$$
$$\quad \textbf{let}\ a = s[\text{hook-arg}]\ \textbf{in}$$
$$\quad \textbf{let}\ r = \lceil_t\, \Pi\underline{t^1}\colon \underline{r}\rceil\ \textbf{in}$$
$$\quad \textbf{let}\ a = \lceil_t\, \Pi\underline{t^1}\colon \underline{a}\rceil\ \textbf{in}$$
$$\quad s[\text{hook-arg}{\rightarrow}a][\text{hook-res}{\rightarrow}r]]$$

## 7.11 Cut

$$[x; y \overset{\text{unitac}}{=} \lambda u.\text{unitac-cut}\ (\ u\ )]$$

$$[\text{unitac-cut}\ (\ u\ ) \overset{\bullet}{=}$$
$$\quad \textbf{let}\ \langle t, s, c\rangle = u\ \textbf{in}$$
$$\quad \textbf{let}\ s = \text{unieval}\ (\ t^1\ ,\ s\ ,\ c\ )\ \textbf{in}$$
$$\quad \textbf{let}\ a' = s[\text{hook-arg}]\ \textbf{in}$$
$$\quad \textbf{let}\ s = \text{unieval}\ (\ t^2\ ,\ s\ ,\ c\ )\ \textbf{in}$$
$$\quad \textbf{let}\ r = s[\text{hook-res}]\ \textbf{in}$$
$$\quad \textbf{let}\ a = s[\text{hook-arg}]\ \textbf{in}$$
$$\quad \textbf{let}\ a = \lceil_t\, \underline{a'}; \underline{a}\rceil\ \textbf{in}$$
$$\quad s[\text{hook-arg}{\rightarrow}a][\text{hook-res}{\rightarrow}r]]$$

## 7.12 Unary conclude

The $x^{\gg}$ operator adds Ponens, Verify, and At operations to the argumentation inside $x$ until all occurrences of $u \vdash v$, $u \Vdash v$, and $\Pi u\colon v$ have been removed. One may think of $x^{\gg}$ as a unary version of $x \gg y$ in which $y$ is missing but for which it is known that $y$ is an object term.

$$[x^{\gg} \overset{\text{unitac}}{=} \lambda u.\text{unitac-Conclude}\ (\ u\ )]$$

[unitac-Conclude ( $u$ ) $\stackrel{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $s =$ unieval ( $t^1$ , $s$ , $c$ ) **in**
unitac-adapt ( $\mathsf{T}$ , $s$ , $c$ )]

If $t$ has form $a^{\gg}$ then find the conclusion $r$ of $a$, add as many $x$ @ $y$, $x^{\triangleright}$, and $x^*$ operations as possible, and return $a :: r :: u$.

# 8 Lemmas, rules, and proof lines

Lemmas, rules, and proof lines do not always say what they seem to say. As an example, a lemma stating that $x = x$ in Mendelsons System S does not say $x = x$. Rather, it says System S $\vdash x = x$. As another example, if one allows the use of deduction in FOL-based proofs then a proof line proving e.g. $x{+}1 = 0{+}(x{+}1)$ under the hypothesis $x = 0 + x$ really proves $x = 0 + x \Rightarrow x + 1 = 0 + (x + 1)$. Or, rather, it proves System S $\vdash x = 0 + x \Rightarrow x + 1 = 0 + (x + 1)$ if the proof is stated in System S.

No single way of handling lemmas, rules, and proofs will satisfy all theories. As an example, deduction in FOL is different from deduction in map theory, and for that reason those two theories need different ways of treating proof lines in hypothetical proofs.

For that reason, the way lemmas, rules, and proofs are treated is embedded in the tactic state.

## 8.1 The initial tactic state

The initial tactic state tacstate0 contains unitac0. unitac0 in turn defines how to treat lemmas, rules, and proof lines during unitac evaluation. The definition of tacstate0 reads:

[hook-unitac $\stackrel{\bullet}{=}$ unitac]

[tacstate0 $\stackrel{\bullet}{=}$ $\mathsf{T}$[hook-unitac$\rightarrow$unitac0]]

The unitac0 structure defines how to handle lemmas, rules, and proof lines during unitac evaluation. Handling of proof lines really means how to handle the conclude construct $x \gg y$.

[hook-lemma $\stackrel{\bullet}{=}$ lemma]

[hook-rule $\stackrel{\bullet}{=}$ rule]

[hook-conclude $\stackrel{\bullet}{=}$ conclude]

[unitac0 $\stackrel{\bullet}{=}$ $\mathsf{T}$[hook-lemma$\rightarrow\lceil\lambda u$.unitac-lemma-std ( $u$ )$\rceil$][hook-rule$\rightarrow\lceil\lambda u$.unitac-ru
( $u$ )$\rceil$][hook-conclude$\rightarrow\lceil\lambda u$.unitac-conclude-std ( $u$ )$\rceil$]].

## 8.2 Conclude

$[x \gg y \overset{\text{unitac}}{=} \lambda u.\text{unitac-conclude ( } u \text{ )}]$

[unitac-conclude ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $d = s$[hook-unitac][hook-conclude] **in**
(eval ( $d$ , $\mathsf{T}$ , $c$ )" $\langle t, s, c\rangle^{\mathrm{M}})^{\mathrm{U}}$]

[unitac-conclude-std ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $s =$ unieval ( $t^1$ , $s$ , $c$ ) **in**
**let** $s =$ unitac-adapt ( $t^2$ metadef ( $c$ ) , $s$ , $c$ ) **in**
**let** $u =$ unify ( $s$[hook-res] , $t^2$ , $s$[hook-uni] ) **in**
$s$[hook-uni$\rightarrow u$][hook-res$\rightarrow t^2$]]

If $t$ has form $t^1 \gg t^2$ then convert $t^1$ into something whose conclusion fits $t^2$. Unify the conclusion with $t^2$ and return the resulting state $s$.

## 8.3 Rules

We define two rule tactics: $x$ Rule for explicit use and unitac-rule ( $x$ ) which is implicitly attached to axioms and inference rules. The "implicit attachment" is done by the axiom and rule macros. As an example of use, A1 Rule explicitly proves A1 whereas A1 implicitly proves $\Pi \mathsf{a}, \mathsf{b}\colon A \Rightarrow B \Rightarrow A$. Note that the former proves the name and the latter proves the contents of A1.

$[r$ Rule $\overset{\text{unitac}}{=} \lambda u.\text{unitac-Rule ( } u \text{ )}]$

[unitac-Rule ( $x$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = x$ **in**
**let** $r = t^1$ **in**
**let** $a =$ unitac-theo ( $r$ , $s$ , $c$ ) **in**
$s$[hook-arg$\rightarrow a$][hook-res$\rightarrow r$]]

[unitac-rule ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = u$ **in**
**let** $d = s$[hook-unitac][hook-rule] **in**
(eval ( $d$ , $\mathsf{T}$ , $c$ )" $\langle t, s, c\rangle^{\mathrm{M}})^{\mathrm{U}}$]

[unitac-rule-std ( $x$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c\rangle = x$ **in**
**let** $r = t$ stmt-rhs ( $c$ ) **in**
**let** $a =$ unitac-theo ( $t$ , $s$ , $c$ ) **in**
**let** $a = \lceil_r \underline{a}^{\mathrm{D}} \rceil$ **in**
$s$[hook-arg$\rightarrow a$][hook-res$\rightarrow r$]]

[unitac-theo ( $t$ , $s$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $p$ = reverse ( $s$[hook-pre] ) **in**
unitac-rule0 ( $t$ , $p$ , $c$ )]

Construct a proof of the rule $t$ assuming the premisses contained in the tactic state $s$. Complain if no proof is found.

[unitac-rule0 ( $r$ , $P$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $p$ = unitac-rule1 ( $r$ , $P$ , $c$ ) **in**
**if not** $p$ **then** $p$ **else**
**if** $r$ metadef ( $c$ ) = plus **then** unitac-rule-plus ( $r$ , $P$ , $c$ ) **else**
**let** $d = r$ stmt-rhs ( $c$ ) **in**
**if not** $d$ **then** unitac-rule-stmt ( $d$ , $r$ , $P$ , $c$ ) **else**
error ( $r$ ,
diag ( 'No locally assumed theory includes the following rule:' ) disp ( $r$ )
end diagnose )]

Construct a proof of the rule $r$ assuming the premisses $P$. Complain if no proof is found. The function first tries to find $r$ verbatim among the premisses (where each premise is tree searched). If no proof is found, $r$ is decomposed.

[unitac-rule-plus ( $R$ , $P$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $\langle \mathsf{T}, r, r' \rangle = R$ **in**
**let** $p$ = unitac-rule0 ( $r$ , $P$ , $c$ ) **in**
**if** $p$ **then** $\mathsf{T}$ **else**
**let** $p'$ = unitac-rule0 ( $r'$ , $P$ , $c$ ) **in**
**if** $p'$ **then** $\mathsf{T}$ **else** $\lceil_R \underline{p}; \underline{p'}; ((\underline{r} \oplus \underline{r'}) \vdash (\underline{r} \oplus \underline{r'})^{\mathrm{I}})^{\mathrm{C} \triangleright \triangleright} \rceil]$

Construct a proof of the rule $R$ of form $r \oplus r'$ assuming the premisses $P$. Return $\mathsf{T}$ if no proof is found.

[unitac-rule-stmt ( $d$ , $r$ , $P$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $p$ = unitac-rule0 ( $d$ , $P$ , $c$ ) **in**
**if** $p$ **then** $\mathsf{T}$ **else**
$\lceil_r \underline{p}; \underline{d}^{\mathrm{I}}$ ie $\underline{r} \rceil]$

Construct a proof of the rule $r$ with definition $d$ assuming the premisses $P$. Return $\mathsf{T}$ if no proof is found.

[unitac-rule1 ( $r$ , $P$ , $c$ ) $\stackrel{\bullet}{=}$
**if** $P \in \mathbf{A}$ **then** $\mathsf{T}$ **else**
unitac-rule2 ( $r$ , $P^{\mathrm{h1}}$ , $c$ ) **and**
unitac-rule1 ( $r$ , $P^{\mathrm{t}}$ , $c$ )]

Construct a proof of the rule $r$ assuming the premisses $P$. Return $\mathsf{T}$ if the rule is not found among the premisses.

[unitac-rule2 ( $r$ , $T$ , $c$ ) $\stackrel{\bullet}{=}$
**let** $p$ = unitac-search ( $r$ , $T$ , $c$ ) **in**

**if** $p$ **then** $\mathsf{T}$ **else**
unitac-rule3 ( $r$ , $p$ )]

Construct a proof of the rule $r$ assuming the theory $T$. Sequent evaluating the proof gives $\langle\langle T\rangle, \mathsf{T}, r'\rangle$ where $r'$ is what the rule with name $r$ says. Return $\mathsf{T}$ if no proof is found (i.e. if the rule does not belong to the theory).

[unitac-search ( $r$ , $T$ , $c$ ) $\stackrel{\bullet}{=}$
unitac-search1 ( $r$ , $T$ , $\mathsf{T}$ , $\langle T\rangle$ , $c$ )$^{\circ\mathrm{t}}$].

Search for the rule $r$ in the theory $T$ and return the "path" of $r$. Return $\mathsf{T}$ is $r$ is not found.

A path is a non-empty list of "addresses". An address is a pair of a tree and a list of "edges". Each edge is one of the strings "head" and "tail". As an example, the address $\langle(x \oplus y) \oplus z, \mathrm{head}, \mathrm{tail}\rangle$ points out the $y$ in the tree $(x \oplus y) \oplus z$.

A path $\langle a, b\rangle$ adresses some term $a'$ which, when dereferenced, is supposed to be the starting point of $b$. As an example, $\langle\langle \mathrm{Prop} \oplus \mathrm{A3}, \mathrm{head}\rangle, \langle \mathrm{A1} \oplus \mathrm{A2} \oplus \mathrm{MP}, \mathrm{tail}, \mathrm{head}\rangle\rangle$ is a path to A2 within $\mathrm{Prop} \oplus \mathrm{A3}$.

The path is returned in reverse order.

[unitac-search1 ( $r$ , $T$ , $p$ , $a$ , $c$ ) $\stackrel{\bullet}{=}$
**if** $r \stackrel{\mathrm{t}}{=} T$ **then** (reverse ( $a$ ) :: $p$)$^{\bullet}$ **else**
**let** $T' = T$ stmt-rhs ( $c$ ) **in**
**if not** $T'$ **then** unitac-search1 ( $r$ , $T'$ , reverse ( $a$ ) :: $p$ , $\langle T'\rangle$ , $c$ ) **else**
**if** $T$ metadef ( $c$ ) $\neq$ plus **then** $\mathsf{T}$ **else**
unitac-search1 ( $r$ , $T^1$ , $p$ , head :: $a$ , $c$ ) **and**
unitac-search1 ( $r$ , $T^2$ , $p$ , tail :: $a$ , $c$ )].

Search for $r$ in $T$ and throw the path of $r$ when found. Return $\mathsf{T}$ if $r$ is not found. Accumulate the path in $p$ in reverse order and the addresses in $a$ in reverse order. The path is returned in reverse order.

[unitac-rule3 ( $r$ , $p$ ) $\stackrel{\bullet}{=}$
**let** $q$ = unitac-rule4 ( $r$ , $p^{\mathrm{h}}$ ) **in**
**if** $p^{\mathrm{t}}$ **then** $q$ **else let** $p$ = unitac-rule3 ( $r$ , $p^{\mathrm{t}}$ ) **in**
$\lceil_r \underline{p}^{\mathrm{D}}; \underline{q}\rceil$]

[unitac-rule4 ( $r$ , $a$ ) $\stackrel{\bullet}{=}$
**let** $p$ = unitac-rule5 ( $r$ , $a^{\mathrm{h}}$ , $a^{\mathrm{t}}$ , $\mathsf{T}$ ) **in**
$\lceil_r \underline{p}^{\triangleright}\rceil$].

Given an address $T :: a$, construct a proof which generates the sequent $\langle\langle T\rangle, \mathsf{T}, a'\rangle$ where $a'$ is the conjunct at address $a$ in conjunction $T$.

[unitac-rule5 ( $r$ , $T$ , $a$ , $s$ ) $\overset{\bullet}{=}$
**if** $a \in \mathbf{A}$ **then**
**let** $p =$ unitac-stack ( $r$ , $s$ , $\lceil_r \underline{T^{\mathrm{I}}} \rceil$ ) **in** $\lceil_r \underline{T} \vdash \underline{p} \rceil$ **else let** $e :: a = a$ **in**
**if** $e =$ head **then**
$\lceil_r \underline{\text{unitac-rule5 ( } r \text{ , } T^1 \text{ , } a \text{ , } T^2 :: s \text{ )}^{\mathrm{U}}} \rceil$ **else**
$\lceil_r (\underline{T^1} \vdash \underline{\text{unitac-rule5 ( } r \text{ , } T^2 \text{ , } a \text{ , } s \text{ )}})^{\mathrm{U}} \rceil$].

Given a conjunction $T$ and a list $a$ of edges, prove $T \vdash a'$ where $a'$ is the conjunct at address $a$.

[unitac-stack ( $r$ , $s$ , $t$ ) $\overset{\bullet}{=}$
**if** $s \in \mathbf{A}$ **then** $t$ **else**
**let** $p :: s = s$ **in**
**let** $t =$ unitac-stack ( $r$ , $s$ , $t$ ) **in**
$\lceil_r \underline{p} \vdash \underline{t} \rceil$].

Add the premisses on the stack $s$ to the term $t$.

## 8.4 Lemmas

[unitac-lemma ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c \rangle = u$ **in**
**let** $d = s$[hook-unitac][hook-lemma] **in**
(eval ( $d$ , $\mathsf{T}$ , $c$ ) " $\langle t, s, c \rangle^{\mathrm{M}}$)$^{\mathrm{U}}$]

[unitac-lemma-std ( $x$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c \rangle = x$ **in**
**let** $\langle \mathsf{T}, T, r \rangle = t$ stmt-rhs ( $c$ ) **in**
**let** $a' =$ unitac-theo ( $T$ , $s$ , $c$ ) **in**
**let** $a = \lceil_t \underline{a'}; \underline{t}^{\mathrm{ID}\triangleright} \rceil$ **in**
$s$[hook-arg$\rightarrow a$][hook-res$\rightarrow r$]].

Define how to sequent prove a lemma and what the lemma concludes.

# 9 Tactic definitions of proof line constructs

## 9.1 Conclude-cut lines

[L$l$: $a \gg x; n \mathrel{\ddot{=}}$ Line $l$ : $a$ >> $x$ ; $n$]

[Line $l$ : $a$ >> $x$ ; $n \overset{\text{tactic}}{=} \lambda u$.tactic-conclude-cut ( $u$ )]

[tactic-conclude-cut ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c \rangle = u$ **in**
**let** $\langle \mathsf{T}, l, a, x, n \rangle = t$ **in**
**let** $s' =$ taceval ( $\lceil_t \underline{a} \gg \underline{x} \rceil$ , $s$ , $c$ ) **in**
**let** $a' = s'$[hook-arg] **in**

**let** $r = s'$[hook-res] **in**
**let** $s =$ tactic-push ( hook-pre , $\langle l, r, \lceil_r \underline{r}^{\mathrm{I}} \rceil \rangle$ , $s$ ) **in**
**let** $s =$ taceval ( $n$ , $s$ , $c$ ) **in**
**let** $a = s$[hook-arg] **in**
$s$[hook-arg$\rightarrow\lceil_t \underline{a'}; \underline{a} \rceil$]]

## 9.2 Premise line

[L$l$: Premise $\gg x; n \doteqdot$ Line $l$ : Premise >> $x$ ; $n$]

[Line $l$ : Premise >> $x$ ; $n \overset{\text{tactic}}{=} \lambda u$.tactic-premise-line ( $u$ )]

[tactic-premise-line ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c \rangle = u$ **in**
**let** $\langle \mathsf{T}, l, x, n \rangle = t$ **in**
**let** $s =$ taceval ( $n$ , tactic-push ( hook-pre , $\langle l, x, \lceil_x \underline{x}^{\mathrm{I}} \rceil \rangle$ , $s$ ) , $c$ ) **in**
**let** $a = \lceil_t \underline{x} \vdash \overline{s[\text{hook-arg}]} \rceil$ **in**
**let** $r = \lceil_t \underline{x} \vdash \overline{s[\text{hook-res}]} \rceil$ **in**
$s$[hook-arg$\rightarrow\overline{a}$][hook-res$\rightarrow r$]]

## 9.3 Condition line

[L$l$: Condition $\gg x; n \doteqdot$ Line $l$ : Condition >> $x$ ; $n$]

[Line $l$ : Condition >> $x$ ; $n \overset{\text{tactic}}{=} \lambda u$.tactic-condition-line ( $u$ )]

[tactic-condition-line ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c \rangle = u$ **in**
**let** $\langle \mathsf{T}, l, x, n \rangle = t$ **in**
**let** $s =$ taceval ( $n$ , tactic-push ( hook-cond , $\langle l, x \rangle$ , $s$ ) , $c$ ) **in**
**let** $a = \lceil_t \underline{x} \Vdash \overline{s[\text{hook-arg}]} \rceil$ **in**
**let** $r = \lceil_t \underline{x} \Vdash \overline{s[\text{hook-res}]} \rceil$ **in**
$s$[hook-arg$\rightarrow\overline{a}$][hook-res$\rightarrow r$]]

## 9.4 Blocks

[L$l$: Block $\gg$ Begin; $x$ L$k$: Block $\gg$ End; $n \doteqdot$ Line $l$ : Block >> Begin ; $x$ line $k$ : Block >> End ; $n$]

[Line $l$ : Block >> Begin ; $x$ line $k$ : Block >> End ; $n \overset{\text{tactic}}{=} \lambda u$.tactic-block ( $u$ )]

[tactic-block ( $u$ ) $\overset{\bullet}{=}$
**let** $\langle t, s, c \rangle = u$ **in**
**let** $\langle \mathsf{T}, l, x, k, n \rangle = t$ **in**
**let** $s' =$ taceval ( $x$ , $s$ , $c$ ) **in**

**let** $a' = s'$[hook-arg] **in**
**let** $r = s'$[hook-res] **in**
**let** $s =$ tactic-push ( hook-pre , $\langle k, r, \lceil_r \underline{r}^{\mathrm{I}} \rceil \rangle$ , $s$ ) **in**
**let** $s =$ taceval ( $n$ , $s$ , $c$ ) **in**
**let** $a = s$[hook-arg] **in**
$s$[hook-arg$\rightarrow\lceil_t \underline{a'}; \underline{a} \rceil$]]

# 10 Sample proofs

## 10.1 Propositional Calculus

We now define Propositional Calculus (Prop) as in [4]. We also define Intuitionistic Propositional Calculus (IProp).

**Axiom** A1: $\Pi \mathsf{x}, \mathsf{y}: \mathsf{x} \Rightarrow \mathsf{y} \Rightarrow \mathsf{x}$ □

**Axiom** A2: $\Pi \mathsf{x}, \mathsf{y}, \mathsf{z}: (\mathsf{x} \Rightarrow \mathsf{y} \Rightarrow \mathsf{z}) \Rightarrow (\mathsf{x} \Rightarrow \mathsf{y}) \Rightarrow \mathsf{x} \Rightarrow \mathsf{z}$ □

**Axiom** A3: $\Pi \mathsf{x}, \mathsf{y}: (\neg \mathsf{y} \Rightarrow \neg \mathsf{x}) \Rightarrow (\neg \mathsf{y} \Rightarrow \mathsf{x}) \Rightarrow \mathsf{y}$ □

**Rule** MP: $\Pi \mathsf{x}, \mathsf{y}: \mathsf{x} \Rightarrow \mathsf{y} \vdash \mathsf{x} \vdash \mathsf{y}$ □

**Theory** IProp: A1 $\oplus$ A2 $\oplus$ MP □

**Theory** Prop: IProp $\oplus$ A3 □

The inference rule of Modus Ponens (MP) is typically applied to two arguments, so we introduce a macro for that:

$$[x \trianglerighteq y \doteqdot \mathrm{MP} \rhd x \rhd y^{\gg}]$$

In the macro, $y^{\gg}$ is argument $y$ from which all occurences of $\Pi \mathsf{u}: \mathsf{v}$, $\mathsf{u} \vdash \mathsf{v}$, and $\mathsf{u} \Vdash \mathsf{v}$ are stripped by applications of $\mathsf{a}$ @ $\mathsf{b}$, $\mathsf{a}^{\rhd}$, and $\mathsf{a}^{*}$, respectively. Stripping is done during tactic evaluation. The macro does not strip $x$ since that happens automatically: During tactic evaluation, the MP rules forces $x$ to be of form $\mathsf{x} \Rightarrow \mathsf{y}$ which causes $x$ to be stripped.

As an example of a proof, take the first lemma proved in [4]:

Prop **lemma** lemma2: $\Pi \mathsf{x}: \mathsf{x} \Rightarrow \mathsf{x}$ □

Prop **proof of** lemma2:

| | | | |
|---|---|---|---|
| L01: | Arbitrary $\gg$ | $\mathsf{x}$ | ; |
| L02: | A2 $\gg$ | $(\mathsf{x} \Rightarrow (\mathsf{y} \Rightarrow \mathsf{x}) \Rightarrow \mathsf{x}) \Rightarrow$ $(\mathsf{x} \Rightarrow \mathsf{y} \Rightarrow \mathsf{x}) \Rightarrow \mathsf{x} \Rightarrow \mathsf{x}$ | ; |
| L03: | A1 $\gg$ | $\mathsf{x} \Rightarrow (\mathsf{y} \Rightarrow \mathsf{x}) \Rightarrow \mathsf{x}$ | ; |
| L04: | L02 $\trianglerighteq$ L03 $\gg$ | $(\mathsf{x} \Rightarrow \mathsf{y} \Rightarrow \mathsf{x}) \Rightarrow \mathsf{x} \Rightarrow \mathsf{x}$ | ; |
| L05: | A1 $\gg$ | $\mathsf{x} \Rightarrow \mathsf{y} \Rightarrow \mathsf{x}$ | ; |
| L06: | L04 $\trianglerighteq$ L05 $\gg$ | $\mathsf{x} \Rightarrow \mathsf{x}$ | □ |

We may leave more work to unification simply by omitting line L02 and L03:

Prop **lemma** lemma2a: $\Pi x\colon x \Rightarrow x$ □

Prop **proof of** lemma2a:

| | | | |
|---|---|---|---|
| L01: | Arbitrary $\gg$ | $x$ | ; |
| L04: | A2 $\trianglerighteq$ A1 $\gg$ | $(x \Rightarrow y \Rightarrow x) \Rightarrow x \Rightarrow x$ | ; |
| L05: | A1 $\gg$ | $x \Rightarrow y \Rightarrow x$ | ; |
| L06: | L04 $\trianglerighteq$ L05 $\gg$ | $x \Rightarrow x$ | □ |

We may state the proof even shorter as follows:

Prop **lemma** lemma2b: $\Pi x\colon x \Rightarrow x$ □

Prop **proof of** lemma2b:

| | | | |
|---|---|---|---|
| L01: | Arbitrary $\gg$ | $x$ | ; |
| L05: | A1 $\gg$ | $x \Rightarrow y \Rightarrow x$ | ; |
| L06: | A2 $\trianglerighteq$ A1 $\trianglerighteq$ L05 $\gg$ | $x \Rightarrow x$ | □ |

It would be tempting to change the argumentation of line L06 to A2 $\trianglerighteq$ A1 $\trianglerighteq$ A1, but then unification would be unable to determine how to instantiate the second quantifier of A1. We may specify that instantiation explicitly, though:

Prop **lemma** lemma2c: $\Pi x\colon x \Rightarrow x$ □

Prop **proof of** lemma2c:

| | | | |
|---|---|---|---|
| L01: | Arbitrary $\gg$ | $x$ | ; |
| L06: | A2 $\trianglerighteq$ A1 $\trianglerighteq$ (A1$^{@}$ @ $y$) $\gg$ | $x \Rightarrow x$ | □ |

In the proof above, A1$^{@}$ tells unification to instantiate the first quantifier of A1 suitably. A1$^{@}$ @ $y$ tells unification to instantiate the first quantifier suitably and to instantiate the second one to $y$.

Line L01 above applies the generalization sequent operator $\Pi x\colon y$ to the rest of the proof. The same effect may be achieved by putting that operator in the argumentation of line L06. But then one must also add a meta-quantifier to the conclusion of line L06. The $\Pi x\colon y$ construct happens to denote both the generalization sequent operator and the meta-quantifier:

Prop **lemma** lemma2d: $\Pi x\colon x \Rightarrow x$ □

Prop **proof of** lemma2d:

| | | | |
|---|---|---|---|
| L06: | $\Pi x\colon$ A2 $\trianglerighteq$ A1 $\trianglerighteq$ (A1$^{@}$ @ $y$) $\gg$ | $\Pi x\colon x \Rightarrow x$ | □ |

One may even leave to unification to guess the quantified variable:

Prop **lemma** lemma2e: $\Pi x\colon x \Rightarrow x$ □

Prop **proof of** lemma2e:

| | | | |
|---|---|---|---|
| L06: | (A2 $\trianglerighteq$ A1 $\trianglerighteq$ (A1$^{@}$ @ $y$))$^{\Pi}$ $\gg$ | $\Pi x\colon x \Rightarrow x$ | □ |

The theory part of a lemma and a proof may be stated as a conjunction in an arbitrary order. Each conjunct may be a rule or a theory, where each theory recursively may be an arbitrary conjunction of rules and theories. As an example, IProp $\oplus$ A3 contains the rules A1, A2, A3, and MP which makes the following lemma and proof correct:

IProp $\oplus$ A3 **lemma** lemma2f: $\Pi x\colon x \Rightarrow x$ $\Box$

IProp $\oplus$ A3 **proof of** lemma2f:

| | | | |
|---|---|---|---|
| L01: | Arbitrary $\gg$ | $x$ | ; |
| L06: | A2 $\trianglerighteq$ A1 $\trianglerighteq$ (A1$^{@}$ @ $y$) $\gg$ | $x \Rightarrow x$ | $\Box$ |

## 10.2 Blocks

Blocks allow subproofs inside other proofs. The following example stress test the block concept in that it first proves that any statement $x$ implies itself, then applies that to the statement itself. The proof is much like applying the identity function to itself which of course yeilds the identity function.

Prop **lemma** lemma3: $\Pi x\colon x \vdash x$ $\Box$

Prop **proof of** lemma3:

| | | | |
|---|---|---|---|
| L01: | Block $\gg$ | Begin | ; |
| L02: | Arbitrary $\gg$ | $x$ | ; |
| L03: | Premise $\gg$ | $x$ | ; |
| L04: | L03 $\gg$ | $x$ | ; |
| L05: | Block $\gg$ | End | ; |
| L06: | L05 $\triangleright$ L05 $\gg$ | $\Pi x\colon x \vdash x$ | $\Box$ |

## 10.3 System S

For the sake of testing we introduce a small subset of Mendelsons System S. For historical reasons, this system is used in the Logiweb test suites.

**Theory** System S: S.S1 $\oplus$ S.S5 $\Box$

**Rule** S.S1: $\Pi x, y, z\colon x = y \vdash x = z \vdash y = z$ $\Box$

**Rule** S.S5: $\Pi x\colon x + 0 = x$ $\Box$

System S **lemma** S.reflexivity: $\Pi x\colon x = x$ $\Box$

System S **proof of** S.reflexivity:

| | | | |
|---|---|---|---|
| L01: | Arbitrary $\gg$ | $x$ | ; |
| L02: | S.S5 $\gg$ | $x + 0 = x$ | ; |
| L03: | S.S1 $\triangleright$ L02 $\triangleright$ L02 $\gg$ | $x = x$ | $\Box$ |

## 10.4 Rules

Prop **lemma** lemma4a: IProp □

Prop **proof of** lemma4a:

| | | | |
|---|---|---|---|
| L01: | IProp Rule ≫ | IProp | □ |

Prop **lemma** lemma4b: A1 □

Prop **proof of** lemma4b:

| | | | |
|---|---|---|---|
| L01: | A1 Rule ≫ | A1 | □ |

Prop **lemma** lemma4c: A2 □

Prop **proof of** lemma4c:

| | | | |
|---|---|---|---|
| L01: | A2 Rule ≫ | A2 | □ |

Prop **lemma** lemma4d: A3 □

Prop **proof of** lemma4d:

| | | | |
|---|---|---|---|
| L01: | A3 Rule ≫ | A3 | □ |

IProp ⊕ A3 **lemma** lemma4e: Prop □

IProp ⊕ A3 **proof of** lemma4e:

| | | | |
|---|---|---|---|
| L01: | Prop Rule ≫ | Prop | □ |

IProp **lemma** lemma4f: A3 ⊢ Prop □

IProp **proof of** lemma4f:

| | | | |
|---|---|---|---|
| L01: | Premise ≫ | A3 | ; |
| L02: | Prop Rule ≫ | Prop | □ |

IProp **lemma** lemma4g: A3 ⊢ $x \Rightarrow x$ □

IProp **proof of** lemma4g:

| | | | |
|---|---|---|---|
| L01: | Premise ≫ | A3 | ; |
| L02: | lemma2 ≫ | $x \Rightarrow x$ | □ |

# Index

# References


[1] 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.

[2] K. Grue. A logiweb base page. Technical report, Logiweb, 2006. 01AB1F51C8C17606A5C0331B5689B4858C796547B9A0A4AEF0BCB2BB0806/page/index.html.

[3] K. Grue. A logiweb proof checker - chores. Technical report, Logiweb, 2006. ../../../logiweb/014E93CEDBCA44EB611BC0974861950432277A602795E9B4F2BAD8BB0806/page/appendix.pdf.

[4] E. Mendelson. *Introduction to Mathematical Logic*. Wadsworth and Brooks, 3. edition, 1987.