Some Lambda Calculus and Type Theory Formalized*

James McKinna
jhm@dcs.ed.ac.uk

Robert Pollack
rap@dcs.ed.ac.uk

Laboratory for Foundations of Computer Science
The King's Buildings, University of Edinburgh, EH9 3JZ, Scotland

1 Introduction

"This paper is about our hobby." That is the first sentence of [MP93], the first report on our formal development of lambda calculus and type theory, written in autumn 1992. We have continued to pursue this hobby on and off ever since, and have developed a substantial body of formal knowledge, including Church-Rosser and standardization theorems for beta reduction, and the basic theory of Pure Type Systems (PTS) leading to the strengthening theorem and type checking algorithms for PTS. Some of this work is reported in [MP93, vBJMP94, Pol94b, Pol95]. In the present paper we survey this work, including some new proofs, and point out what we feel has been learned about the general issues of formalizing mathematics. On the technical side, we describe an abstract, and simplified, proof of standardization for beta reduction, not previously published, that does not mention redex positions or residuals. On the general issues, we emphasize the search for formal definitions that are convenient for formal proof and convincingly represent the intended informal concepts.

The LEGO Proof Development System [LP92] was used to check the work in an implementation of the Extended Calculus of Constructions (ECC) with inductive types [Luo94]. LEGO is a refinement style proof checker, publicly available by ftp and WWW, with a User's Manual [LP92] and a large collection of examples. Section 1.3 contains information on accessing the formal development described in

*Submitted to Journal of Automated Reasoning

this paper. Other interesting examples formalized in LEGO include program specification and data refinement [Luo91], strong normalization of System F [Alt93], synthetic domain theory [Reu95, Reu96], and operational semantics for imperative programs [Sch97].

1.1 Why?

PTS have a beautiful meta-theory, developed informally in [Bar92, Ber90, GN91, vBJ93, Geu93]. These papers are unusually clear and mathematical, and there is little doubt about the correctness of their results, so why write a machine-checked development? The informal presentations leave many decisions unspecified and many facts unproved. They are far from the detail of representation needed to write a computer program for typechecking PTS, and the lemmas needed to prove correctness of such a program. At the start, our long-term goal was to fill these gaps in order to increase confidence in proofchecking programs (such as LEGO) based on type theory. That goal is largely met in [Pol95]. Also, while the basic informal theory of PTS is well understood, the difficulties of formalization suggested reformulations which clarify the presentation.

Another goal of the project is to develop a realistic example of formal mathematics. In mathematics and computer science we do not prove one big theorem and then throw away all the work leading up to that theorem; we want to build a body of formal knowledge that can continually be extended. This suggests some design criteria for formalization. Representations and definitions must be suitable for the whole development, not specialized for a single theorem. The theory should be structured, like computer programs, by abstraction, providing "isolation of components" so that several parts of the theory can be worked on simultaneously, perhaps by several workers, and so that the inevitable wrong decisions in underlying representations can later be fixed without affecting too seriously a large theory that depends on them. The body of knowledge we want to formalize is itself still growing, e.g. [vBJMP94] reports advances on typechecking for PTS done later than our original formalization, that became part of our formal development. The work on typechecking benefited from the basic formalization of PTS, since proofs about several related systems could be easily adapted from proofs already done for PTS. Further, new subjects were included; e.g. the standardization theorem, not used in the type theory, was formalized by the first author. On the other hand, we do not claim that type theory is a realistic example for all formal mathematics: it is especially suitable for formalization because the objects are inductively constructed, their properties are proved by induction over structure, and there is little equality reasoning.

Perhaps the most compelling reason for our continuing fascination with this work is the lure of completely concrete, yet simple, proofs of results whose conventional presentation seems to require some notions that are "messy" to formalize, e.g. the standardization theorem discussed in section 3.4. We see such proofs as beautiful, both by their simplicity and their concreteness. There is a tendency in formalization to throw simplicity to the winds in frustration to get the proof to work at all; but once it is checked, it can be beautified relatively easily, as improved definitions and arguments are mechanically checked, easily pointing out new glitches and suggesting how to fix them. Also, a formal development is easy to come back to a year later, as all the details you would not otherwise have written down are explicit, and don't have to be rediscovered.

1.2 Related Work

There are many formalizations of the Church–Rosser theorem [Sha85, Hue94, Nip96, Pfe92]; the only formalization of a standardization theorem we know of is [Coq96a], for lazy combinator expressions. Formalizations of type theory include [DB93, Bar95]; both of these address limited aspects of very special type theories (essentially the Calculus of Constructions), although [Bar95] is very interesting work in which the program extraction mechanism of Coq is used to extract an executable typechecker from a proof of decidability of typechecking. In contrast to all the cited work except [Bar95], our development hasn't been terminated by reaching one specified theorem, but continues to grow in various directions guided by our interests, and by other work we come across that we feel needs checking. For example, both authors have checked parts of type theory papers we were asked to referee.

A novelty in our presentation is the use of named variables. Most of the formalizations of type theory or lambda calculus that we know of use de Bruijn indices ("nameless variables") [Sha85, Alt93, Hue94, Nip96, Bar95] or higher order abstract syntax [Pfe92] to avoid formalizing the renaming of variables to prevent unintended capture during substitution. While de Bruijn notation is concrete and suitable for formalization, there are reasons to formalize the theory with named variables. For one thing, implementations must use names at some level, whether internally or only for parsing and printing; in either case this use of names must be formally explained. More interesting is the insight to be gained into the meaning of binding. Many researchers agree that de Bruijn representation “really is” what we informally mean by lambda terms, in the sense that there is no need to quotient terms by alpha-conversion, i.e. intensional equality on de Bruijn terms corresponds with what is informally meant by identity of terms. Nonetheless, de Bruijn representation is a coding of the informal notion of binding, and doesn't address at all the relationship between free and bound variables, namely how free variables become bound. In our formalization, syntactic terms using named variables are themselves concrete: the names of bound variables actually occur (parametrically) in meta-formulas containing them, just as the names of free variables do. This is done using a formulation suggested by Coquand [Coq91], based on syntactically distinguishing free from bound variables[1]. Other work on formalization of binding and substitution using names includes [Coq96b, GM96, Owe95, Sat83, Sto88], but these do not work out any large examples using their binding notions. It would be interesting to compare our development with some similar example using the terms up-to alpha conversion of [GM96]. A presentation of type theory based on treating terms with named variables concretely is Martin-Löf's calculus of explicit substitutions [Tas93], but this presentation is not closed under alpha-conversion, as our presentation is (section 5.5.3), and we view this as a failure of concreteness of Martin-Löf's system.

1.3 This paper and the formal development

The source files for the development described in this paper, along with a README file explaining how to check them, is available on the LEGO WWW homepage `http://www.dcs.ed.ac.uk/home/lego/`.

LEGO uses a module system (described in [JP93]) based on Cardelli's *mock modules* [Car91]. Each source file is a module, and each module has a header saying which modules it depends on. Thus the directory of modules associated with this paper contains parallel, and even incompatible, developments. If you type `Load strengthening`, the file `strengthening.l` (which contains the proof of strengthening for PTS) will be loaded, preceeded by every module it depends on[2].

There are over 70 proof source files with extension `.l`[3] containing

[1] This distinction is already present in Gentzen [Gen69, pages 71–2, 116–7, 141, 216–7] and Prawitz [Pra65]

[2] The dependencies are determined from the module headers, not by examining the actual dependencies in the files.

[3] The `.l` files are the ones we wrote; LEGO generates “compiled” files with a `.o`

over 1500 definitions and lemmas. This is a large amount of formal knowledge, which we can only survey here. This paper uses informal mathematical notation, but almost every definition and lemma that we mention is given with its formal name in `typewriter font` (often in parentheses). You can then use *grep* to find the file in which it is defined and the files in which it is used. This is not particularly elegant, but it's how we do it too. Keeping track of a large amount of formal knowledge is a serious problem that we have not addressed very well.

1.3.1 About notation

As mentioned, this paper uses informal notation, which is arrived at by manually translating from the formal LEGO notation into LaTeX. Further, the translation is not purely syntactical; we chose to surpress some technical details to have a readable presentation. Errors are quite likely, arising from both our translation and your interpretation. This paper may be an informative outline of the formal work, but if you want to *believe* one of our results you must read its formal statement, and all the formal definitions used in its statement; see [Pol96] for discussion of believing a large formal development.

In [MP93] we used formal notation, verbatim text manually extracted from LEGO source files; no translation errors occur, but there is no reason to believe the verbatim text in the paper actually appears in the files. Indeed, the the document and the files drifted apart over time. In [Pol94b] we again used formal notation, mechanically extracting marked sections of the source files, following the idea of Knuth's WEB. We could rerun the extraction to update the document to the formal source, but many readers complained the document was as unreadable as the formal source. Presenting a formal development is a serious problem. Perhaps mechanical extraction with mechanical translation to informal notation is the right direction to pursue.

For better or worse, we have sanitised this presentation so that very little purely formal detail shows through. For example, we mostly surpress the distinction between boolean values and propositional values. However, we don't want to hide the fact that formalization requires many details that don't appear in informal presentations.

A few basic notations The development uses LEGO's built-in library of impredicative definitions for the usual logical connectives and

extension. These are the fully annotated λ-terms generated by the LEGO tactics called in the `.l` file.

their properties; we use standard notation for these connectives. Quantifiers are typed in ECC, but we reserve symbols to range over certain types, and drop the type labels almost everywhere; e.g. p will be reserved to range over parameters, $\mathtt{PP}$, so we write $\forall p \ldots$ instead of $\forall p{:}\mathtt{PP} \ldots$.

Well known computer science notations are used, e.g. $\mathtt{if}\,(\_,\_,\_)$ as *if-then-else*, $\mathsf{list}(\_)$ for the type of lists over $\_$, @ (or sometimes just concatenation) for list append. All funtions of ECC are total (as opposed to functions in the object theory of lambda terms and PTS), so some operations take extra arguments for a "failure value", e.g. $(\mathtt{assoc}\ a\ b\ l)$ returns b if a is not the first element of a pair occurring in l.

2 Pure Languages

In this section we discuss a formalization of the language of PTS, including terms, occurrences and substitution. We derive a strong induction principle for well-formed terms.

A Pure Language (PL) is a triple $(\mathtt{PP}, \mathtt{VV}, \mathtt{SS})$ where

- $\mathtt{PP}$ is an infinite set of *parameters*, ranged over by p, q, r; these are the global, or free, variables.
- $\mathtt{VV}$ is an infinite set of *variables*, ranged over by v, x, y; these are the local, or bound, variables.
- $\mathtt{SS}$ is a set of *sorts*, ranged over by s, t u; these are the constants.

$\mathtt{PP}$, $\mathtt{VV}$ and $\mathtt{SS}$ have decidable equality. That $\mathtt{PP}$ and $\mathtt{VV}$ are infinite is captured by the assumption that for every list of parameters (variables) there exists a parameter (variable) not occurring in the list; e.g.[4]:

$$\forall l \in \mathsf{list}(\mathtt{PP})\ .\ \exists p\ .\ \neg\mathtt{member}(p, l) \qquad (\mathtt{PPinf})$$

We are not assuming mathematical principles, but working parametrically in types $\mathtt{PP}$, $\mathtt{VV}$ and $\mathtt{SS}$ having the stated properties. These can be instantiated with particular types that provably do have these properties, e.g. the natural numbers, or lists over some finite enumeration type. By working parametrically we are preserving abstractness: only the stated properties are used in our proofs.

[4] In this formula, $\mathtt{member}(p, l)$ is decidable because $\mathtt{PP}$ has decidable equality.

2.1 Terms

The terms of a PL, Trm, ranged over by M, N, A, $\ldots$, E, a, b, are given by the grammar

$$
\begin{array}{lllll}
M & ::= & v \mid p \mid s & & \textit{atoms: variable, parameter, sort} \\
 & \mid & [v{:}M]M \mid \{v{:}M\}M & & \textit{binders: lambda, pi} \\
 & \mid & M\,M & & \textit{application}
\end{array}
$$

To be precise, `Trm` is inductively generated by six constructors: every term can be thought of as a well-founded tree whose leaves are variables, parameters and sorts, and whose interior nodes are lambda and pi (having three branches each) and application (having two branches). We often define functions on `Trm` by structural (primitive) recursion over this inductive definition. As usual, we intend $[v{:}A]B$ and $\{v{:}A\}B$ to bind v in B but not in A. However the intended binding structure is not determined by the definition of `Trm`, but is made explicit by the definitions of substitution and occurrence below. Equality on terms is defined by recursion over `Trm`; it inherits decidability from PP, VV and SS.

Remark 2.1 (Notation) *Often when doing case analysis by term structure, we want to say that the binders, lambda and pi, behave the same way. We introduce a notation $\langle v{:}A\rangle a$ to allow combining these cases. The actual formalization does not have such a notation, but this would have saved much cutting and pasting in developing the proofs.*

The *length* of a term is used as a measure for well-founded induction.

$$
\begin{array}{lcll}
\mathtt{lngth}(\alpha) & \triangleq & 1 & \alpha \in \mathtt{PP}, \mathtt{VV}, \mathtt{SS} \\
\mathtt{lngth}(\langle v{:}A\rangle a) & \triangleq & 1 + \mathtt{lngth}(A) + \mathtt{lngth}(a) & \\
\mathtt{lngth}(a\,b) & \triangleq & 1 + \mathtt{lngth}(a) + \mathtt{lngth}(b) &
\end{array}
$$

Two properties of this measure are used applications: if A is a proper subterm of B then $\mathtt{lngth}(A) < \mathtt{lngth}(B)$ (used in induction on the length of terms), and every term has positive length (used in reasoning about PTS by induction on the sum of the lengths of the terms in a context).

2.2 Occurrences of Parameters and Sorts

The list of parameters occurring in a term is computed by primitive recursion over term structure, and the boolean judgement whether or not a given parameter occurs in a given term is decided by the member function on this list of parameters.

$$
\begin{array}{lcll}
\texttt{params}(p) & \triangleq & [p] & \\
\texttt{params}(\alpha) & \triangleq & [] & \alpha \in \texttt{VV}, \texttt{SS} \\
\texttt{params}(\langle v{:}A\rangle a) & \triangleq & \texttt{params}(A) \ @ \ \texttt{params}(a) & \\
\texttt{params}(a\, b) & \triangleq & \texttt{params}(a) \ @ \ \texttt{params}(b) & \\
p \in A & \triangleq & \texttt{member}(p, \texttt{params}(A)) &
\end{array}
$$

Similarly $\texttt{sorts}(A)$ and $s \in A$ are defined.

2.3 Substitution

For the machinery on terms, we need two kinds of substitution, for parameters and for variables, both defined by primitive recursion over term structure. Write $[a/p]M$ (formally `psub`) for substitution of a for a parameter, p, in M. This is entirely textual, not preventing capture. Since parameters have no binding instances in terms, there is no hiding of a parameter name by a binder.

$$
\begin{array}{lcll}
[a/p]q & \triangleq & \texttt{if}(p{=}q, a, q) & \\
{[a/p]}\alpha & \triangleq & \alpha & \alpha \in \texttt{VV}, \texttt{SS} \\
{[a/p]}\langle v{:}B\rangle b & \triangleq & \langle v{:}[a/p]B\rangle [a/p]b & \\
{[a/p]}(M\, N) & \triangleq & [a/p]M\, [a/p]N &
\end{array}
$$

Substitution of a for a variable, v, in M, written $[a/v]M$ (formally `vsub`), does respect hiding of bound instances from substitution, but does not prevent capture.

$$
\begin{array}{lcll}
[a/v]x & \triangleq & \texttt{if}(v{=}x, a, x) & \\
{[a/v]}\alpha & \triangleq & \alpha & \alpha \in \texttt{PP}, \texttt{SS} \\
{[a/v]}\langle x{:}B\rangle b & \triangleq & \langle x{:}[a/v]B\rangle \texttt{if}(v{=}x, b, [a/v]b) & \\
{[a/v]}(M\, N) & \triangleq & [a/v]M\, [a/v]N &
\end{array}
$$

$[\_/p]\_$ and $[\_/v]\_$ will be used only in safe ways in the type theory and the theory of reduction and conversion, so as to prevent unintended capture of variables. Note that these operations are total functions, and do not rename variables. Also, occurrences of a in $[a/\_]\_$ are *shared*, regardless of whether they occur within different binding scopes, in contrast to the situation with de Bruijn indices.

Some important lemmas can now be proved:

$$p \notin M \ \Rightarrow \ [N/p]\,[p/v]M \ = \ [N/v]M \ , \qquad (\texttt{vsub\_is\_psub\_alpha})$$

VCL-ATOM	$\texttt{Vclosed}(\alpha)$	$\alpha \in \mathsf{PP} \cup \mathsf{SS}$
VCL-BIND	$\dfrac{\texttt{Vclosed}(A) \qquad \texttt{Vclosed}(\,[p/v]B\,)}{\texttt{Vclosed}(\,\langle v{:}A\rangle B\,)}$	
VCL-APP	$\dfrac{\texttt{Vclosed}(A) \qquad \texttt{Vclosed}(B)}{\texttt{Vclosed}(\,A\,B\,)}$	

Table 1: Inductive definition of the relation `Vclosed`.

and we have a ready supply of terms of the shape $[p/v]M$, with $p \notin M$:

$$\forall p, M. \exists v, M'.\ M \ = \ [p/v]M' \ \wedge p \notin M' \qquad (\texttt{shape\_lemma})$$

Many other properties of these operations are proved in the formal development.

2.4 No Free Occurrences of Variables

Intuitively parameters are the free names in terms; variables are intended to be the *bound* names, and we do not consider terms with free variables to be well formed. We define inductively a predicate `Vclosed` (*variable-closed*) over terms (table 1). This is analagous to the way a typing relation specifies another kind of well-formedness. (It will turn out that every PTS-typable term is `Vclosed`). Thus `Vclosed` is used as an induction principle over well formed terms. As this relation is a simple case of ideas that recur many times in what follows, we will discuss it at some length.

Of course all terms of form s and p are `Vclosed` (rule VCL-ATOM), and no terms of shape v are `Vclosed` (there is no rule to introduce $\texttt{Vclosed}(v)$), but how do we define `Vclosed` for binders? The approach to "going under binders" is a central idea of our formal handling of names: for $\langle v{:}A\rangle a$ to be `Vclosed`, we require $\texttt{Vclosed}(A)$ and $\texttt{Vclosed}(\,[p/v]a\,)$ for some parameter p. That is, to go under a binder, first fill the hole with parameter, p. But p doesn't appear in the conclusion of rule VCL-BIND; which parameter are we to use? In the definition of `Vclosed` we say that any parameter will do, but there is another possible choice: that $\texttt{Vclosed}(\,[p/v]B\,)$ be derivable for all

p. This is not a formal question; it is one of the tasks of a reader of formal mathematics to decide if the formalisation correctly captures her informal understanding. But a formaliser can help readers by pointing out alternatives, and formally proving some relationship between them. This is especially interesting when alternative definitions lead to easier proofs in some cases. We will see this below for `Vclosed`.

Remark 2.2 *`Vclosed` is equivalent to having no free variables (`Vclosed_vclosed`, `vclosed_Vclosed`). This observation may be of informal interest ("the definition of `Vclosed` is reasonable"), but we do not use it formally because `Vclosed` allows us to avoid all talk of free variables.*

`Vclosed` Generation Lemmas Suppose you have a proof of $\texttt{Vclosed}(\langle v{:}A\rangle B)$; without examining it you know it must be constructed by VCL-BIND from proofs of $\texttt{Vclosed}(A)$ and $\texttt{Vclosed}([p/v]B)$, because no other rule for `Vclosed` has a conclusion of shape $\texttt{Vclosed}(\langle v{:}A\rangle B)$. The very fact that a relation is inductively defined means that its judgements can only be derived by using its rules. This is often called *case analysis*, and more generally, the lemmas that express such properties are called *generation lemmas* [Bar92], or *inversion principles* [DFH$^+$93]. Note that inversion principles are determined by the *shape* of a definition, not by its extension. LEGO has new and very useful tactics to automate the use of inversion [McB96], but most of what we describe in this paper was done before the tactics were available. We will frequently use inversion on inductive definitions in the rest of this paper without further comment.

The generation lemmas from the definition of `Vclosed` are

$$\begin{array}{l}
\texttt{Vclosed}(v) \;\Rightarrow\; \texttt{absurd} \\
\texttt{Vclosed}(\langle v{:}A\rangle B) \;\Rightarrow\; \texttt{Vclosed}(A) \wedge \exists p \,.\, \texttt{Vclosed}([p/v]B) \\
\texttt{Vclosed}(A\,B) \;\Rightarrow\; \texttt{Vclosed}(A) \wedge \texttt{Vclosed}(B)
\end{array}$$

Notice how the existential quantifier in the case for binders expresses the failure of the subformula property in `Vclosed`.

2.4.1 A better induction principle for `Vclosed`.

Here are three "obvious" facts about `Vclosed` (`alpha_Vclosed_lem`, `Vclosed_alpha`).

$$\begin{array}{c}
\texttt{Vclosed}(\langle v{:}A\rangle B) \;\Rightarrow\; \texttt{Vclosed}(A) \wedge \forall p \,.\, \texttt{Vclosed}([p/v]B) \\
\forall M.\texttt{Vclosed}(M) \;\Rightarrow\; \forall q, v.[q/v]M \;=\; M \\
\forall B, p, v.\texttt{Vclosed}([p/v]B) \;\Rightarrow\; \forall q.\texttt{Vclosed}([q/v]B)
\end{array}$$

They are all directly provable, but appear to need length induction (which appeals to well-founded induction and then subsidiary case analysis; e.g. the proof of claim `aVclosed_alpha` below), for the usual reason that statements about change of names are proved by length induction rather than structural induction: e.g. $[q/v]M$ is not generally a subterm of $(M\,N)$, but it is shorter than $(M\,N)$. We will derive a new induction principle which packages up such arguments once and for all.

Consider an alternative definition, called `aVclosed`, differing only in the rule for binders, in which the right premise requires $\mathtt{aVclosed}([p/v]B)$ for *every* p:

$$\textsc{aVcl-bind}\ \frac{\mathtt{aVclosed}(A) \qquad \forall p\,.\ \mathtt{aVclosed}([p/v]B)}{\mathtt{aVclosed}(\langle v{:}A\rangle B)}$$

We will show that `Vclosed` and `aVclosed` derive the same judgements. Induction over `aVclosed` is the principle which Melham and Gordon rediscovered as a consequence of their Axiom 4 (Unique Iteration) [GM96, Section 3.2].

It is worth saying that `Vclosed` is a type of finitely branching well-founded trees; i.e. Vcl-atom are the leaves, and Vcl-bind and Vcl-app are binary branching nodes. On the other hand, `aVclosed` contains infinitely branching well-founded trees, where aVcl-bind creates a branch for each parameter p. Notice also that for any term, A, there is at most one derivation of $\mathtt{aVclosed}(A)$, while there may be many derivations of $\mathtt{Vclosed}(A)$, differing in the parameters used in the left premises of instances of aVcl-bind.

Equivalence of `Vclosed` and `aVclosed` (`aVclosed_Vclosed`, `Vclosed_aVclosed`)

$$\forall A\,.\ \mathtt{aVclosed}(A) \quad\Leftrightarrow\quad \mathtt{Vclosed}(A)\,.$$

Both directions follow easily by structural inductions once we have the following claim (`aVclosed_alpha`):

$$\forall B, p, v\,.\ \mathtt{aVclosed}([p/v]B) \ \Rightarrow\ \forall q\,.\ \mathtt{aVclosed}([q/v]B)$$

Proof. The claim is proved by induction on $\mathtt{lngth}(B)$. This works because every term appearing in a premise of a rule of `aVclosed` is shorter than the term appearing in its conclusion; the typing relations to be considered later do not have this property, and more subtle proofs will be required (section 5.2.1).

By well-founded induction on $\texttt{lngth}(B)$, we have the goal

$$\begin{array}{l}\forall A\ .\ (\forall X\ .\ \texttt{lngth}(X) < \texttt{lngth}(A) \Rightarrow \\ \qquad\qquad \forall p, v\ .\ \texttt{aVclosed}([p/v]X) \Rightarrow \forall q\ .\ \texttt{aVclosed}([q/v]X)) \Rightarrow \\ \qquad \forall p, v\ .\ \texttt{aVclosed}([p/v]A) \Rightarrow \forall q\ .\ \texttt{aVclosed}([q/v]A)\end{array}$$

Now using term structural induction on A, we have cases for sort, variable, parameter, binder and application (only case analysis is necessary here; we don't use the structural induction hypotheses). Consider the case for binder: we must show $\texttt{aVclosed}(\,[q/v]\,\langle n{:}A\rangle B\,)$, i.e.

$$\texttt{aVclosed}(\,\langle n{:}\,[q/v]A\,\rangle\texttt{if}\,(v{=}n, B,\ [q/v]B\,)\,)$$

under the assumptions

$$\begin{array}{lll}\mathsf{ih} & : & \forall X\ .\ \texttt{lngth}(X) < \texttt{lngth}(\langle n{:}A\rangle B) \Rightarrow \\ & & \qquad \forall p, v\ .\ \texttt{aVclosed}([p/v]X) \Rightarrow \forall q\ .\ \texttt{aVclosed}([q/v]X) \\ \mathsf{vclp} & : & \texttt{aVclosed}([p/v]\langle n{:}A\rangle B) \\ & & \qquad \text{(i.e. } \texttt{aVclosed}(\langle n{:}[p/v]A\rangle\texttt{if}(v{=}n, B, [p/v]B)))\end{array}$$

By $\texttt{aVclosed}$ inversion applied to assumption vclp we also know

$$\begin{array}{lll}\mathsf{h1} & : & \texttt{aVclosed}([p/v]A) \\ \mathsf{h2} & : & \forall r\ .\ \texttt{aVclosed}([r/n]\texttt{if}(v{=}n, B, ([p/v]B)))\end{array}$$

By aVcl-bind, it suffices to show

$$\texttt{aVclosed}(\,[q/v]A\,) \quad \text{and} \quad \forall r\ .\ \texttt{aVclosed}(\,[r/n]\texttt{if}\,(v{=}n, B,\ [q/v]B\,)\,)$$

Noticing that $[p/v]\_$ doesn't change $\texttt{lngth}$, the first of these holds by ih and $\mathsf{h1}$. For the second, let r be an arbitrary parameter, and consider cases. If $v = n$ then we are done by $\mathsf{h2}$; i.e. $[q/v]B$ doesn't actually appear in the goal, and $[p/v]B$ doesn't actually appear in $\mathsf{h2}$. Finally the interesting case: if $v \neq n$ we use a straightfoward lemma ($\texttt{alpha\_commutes\_alpha}$)

$$\forall v, w\ .\ v \neq w \Rightarrow \forall r, q, A\ .\ [r/v]\,[q/w]A \ = \ [q/w]\,[r/v]A$$

to rewrite the goal to $\texttt{aVclosed}(\,[q/v]\,[r/n]B\,)$. By ih it suffices to show $\texttt{aVclosed}(\,[p/v]\,[r/n]B\,)$, which follows by $\mathsf{h2}$ after again rewriting the order of substituting p and r. ■

What have we gained? By defining `aVclosed` and showing it to be *extensionally* equivalent to `Vclosed`, we can view the induction principle of `aVclosed` as an induction principle for the *extension* of `Vclosed`, and this is clearly stronger than the induction principle of `Vclosed`. We insist on *extension* to point out that `aVclosed`-induction may be used to prove statements about the *judgement* `Vclosed`, but not about *derivations* of the judgement.

Notice that we could directly prove the analogue of claim `aVclosed_alpha` for `Vclosed` (the proof outlined above works), but it is not just the stronger *premises* of `aVclosed` we are after (i.e. the generation lemmas), it is the stronger *induction hypotheses*.

2.5 A Technical Digression: Renamings

$[\_/p]\_$ and $[\_/v]\_$ are sequential operations; we have not used a notion of simultaneous substitution, except in the following special case. A *renaming* is a finite function from parameters to parameters. Renamings are represented formally by their graphs as lists of ordered pairs.

$$\begin{array}{lll} \texttt{rp} & \triangleq & \texttt{PP} \times \texttt{PP} \qquad \text{(renaming pair)} \\ \texttt{Renaming} & \triangleq & \mathsf{list}(\texttt{rp}) \end{array}$$

ρ and σ range over renamings. The action of a renaming (`renTrm`) on parameters is by lookup in the representing list, and is extended compositionally to all terms.

$$\begin{array}{lll} \rho p & \triangleq & (\texttt{assoc}\ p\ p\ \rho) \\ \rho\alpha & \triangleq & \alpha \qquad\qquad\qquad (\alpha \in \texttt{VV}, \texttt{SS}) \\ \rho\langle v{:}A\rangle a & \triangleq & \langle v{:}\rho A\rangle \rho a \\ \rho(M\,N) & \triangleq & \rho(M)\,\rho(N) \end{array}$$

This is a "tricky" representation. First, if there is no pair (p,q) in ρ, $(\texttt{assoc}\ p\ p\ \rho)$ returns p, so the action of a renaming is always total, with finite support. Also, while there is no assumption that renamings are the graphs of functional relations, the action of a renaming is functional, because `assoc` finds the *first* matching pair. Conversely, consing a new pair to the front of a renaming will "shadow" any old pair with the same first component. We do not formalize these observations.

Renamings commute with substitution in a natural way:

$$\forall \rho, M, N, v.\, \rho(\,[N/v]M\,) \;=\; [\rho N/v]\rho M\;. \qquad (\texttt{vsub\_renTrm\_commutes})$$

Renaming is iterated substitution. We can analyse the action of a renaming in terms of substitution (`renTrm_is_conjugated_psub`):

$$\forall r, M, \rho, p, q \;.\; r \notin M \wedge r \notin \rho \;\Rightarrow\; ((p,q)::\rho)M \;=\; [q/r](\rho([r/p]M))$$

From this lemma it is easy to show that renaming respects any relation that substitution of parameters respects (`psub_resp_renTrm_resp`):

$$\begin{array}{l} \forall R : \mathtt{Trm}{\rightarrow}\mathtt{Trm}{\rightarrow}\mathtt{Prop}\;. \\ \qquad (\forall A, B \;.\; R(A,B) \;\Rightarrow\; \forall q, p \;.\; R([q/p]A, [q/p]B)) \quad \Rightarrow \\ \qquad \forall A, B \;.\; R(A,B) \;\Rightarrow\; \forall \rho \;.\; R(\rho A, \rho B) \end{array}$$

Similar results hold for n-ary relations R.

Injective and Surjective Renamings It is useful to have bijective renamings (e.g. in Section 5.2.1). The definitions are standard:

$$\mathtt{inj}(\rho) \;\triangleq\; \forall p, q \;.\; \rho p = \rho q \;\Rightarrow\; p = q\,, \qquad \mathtt{sur}(\rho) \;\triangleq\; \forall p\, \exists q \;.\; \rho q = p$$

It is surprisingly difficult to construct bijective renamings in general because of the trickiness of the representation mentioned above. However it's clear that any renaming that only swaps parameters is bijective (`swap_sur`, `swap_inj`), and this is enough for our purposes:

$$\begin{array}{c} \mathtt{swap}(p,q) \;\triangleq\; [(p,q),(q,p)] \\ \forall p, q \;.\; \mathtt{sur}(\mathtt{swap}(p,q)) \wedge \mathtt{inj}(\mathtt{swap}(p,q)) \end{array}$$

3 Reduction and Conversion

In this section we outline the theory of reduction and conversion of Pure Languages. The main results are the Church-Rosser and standardization theorems.

As in the definition of `Vclosed` (Section 2.4), the interesting point in defining reduction is how the relation goes under binders. To understand how reduction works, consider informally one-step beta-reduction of untyped lambda calculus. In our style the β and ξ rules are:

$$\beta \qquad (\lambda x.M)\,N \rightarrow [N/x]M \qquad \mathtt{Vclosed}(N)$$

$$\xi \qquad \frac{[q/x]M \;\rightarrow\; [q/y]N}{\lambda x.M \rightarrow \lambda y.N} \qquad q \notin M,\; q \notin N$$

The substitution $[N/x]M$ on the RHS of β does *not* prevent capture, so some restriction is required. It is obvious that no capture can occur if N is closed in the usual informal sense, but because we distinguish between parameters and variables it is enough that N be `Vclosed`. This is no actual restriction: we will only want to reason about `Vclosed` terms anyway, as these are the "well-formed" terms.

To use β under a binder, as allowed by ξ, we must preserve the invariant that β is only applied to `Vclosed` terms: we fill the "holes" left by stripping off the binder with a fresh parameter. Here is an instance of ξ where incorrect capture might occur (contracting the underlined redex):

$$\frac{[q/x]\underline{(\lambda v.\lambda x.v)\ x} = \underline{(\lambda v.\lambda x.v)\ q} \quad \rightarrow \quad \lambda x.q = [q/y]\lambda x.y}{\lambda x.(\underline{(\lambda v.\lambda x.v)\ x}) \quad \rightarrow \quad \lambda y.\lambda x.y}$$

After removing the outer binder λx, replacing its bound instances by a fresh parameter, q, and contracting the `Vclosed` redex thus obtained, we must re-bind the hole now occupied by q. (Since q was fresh, all instances of q mark holes that should be re-bound). According to ξ, we require a variable, y, and a term, N, such that $[q/y]N$ is the contractum of the `Vclosed` redex, $\lambda x.q$ in the example. Such a pair is y, $\lambda x.y$ (the one we have used above), as is z, $\lambda x.z$ for any $z \neq x$. However ξ does not derive the incorrect judgement

$$\lambda x.(\underline{(\lambda v.\lambda x.v)\ x}) \quad \rightarrow \quad \lambda x.\lambda x.x$$

because

$$[q/x]\lambda x.x \quad = \quad \lambda x.x \quad \neq \quad \lambda x.q.$$

Thus incorrect capture is avoided.

3.1 Parallel Reduction

Rather than use ordinary β-reduction, we take *parallel reduction* (à la Tait–Martin-Löf) as the basic reduction relation. Parallel reduction is convenient for the Church-Rosser and standardization theorems, as emphasised by Takahashi in her beautiful account [Tak95]. Our development follows that of [Tak95], with some refinements.

3.1.1 One-step parallel reduction

This relation, $\twoheadrightarrow\!\!\!\!\!\!\!\;\rightarrow$ (`par_red1`), is defined in Table 2. As in `Vclosed` above, the dependence of the congruence rule for binders on the choice of a parameter p is only apparent. However, something new arises here, namely the side conditions $p \notin B$, $p \notin B'$. These are *eigenvariable*

PR1-ATOM	$\alpha \twoheadrightarrow \alpha$	$\alpha \in \mathtt{PP} \cup \mathtt{SS}$
PR1-BETA	$\dfrac{A \twoheadrightarrow A' \qquad [p/u]B \twoheadrightarrow [p/v]B'}{(\,[u{:}U]B\,)\,A \twoheadrightarrow [A'/v]B'}$	$p \notin B, p \notin B'$ $\mathtt{Vclosed}(U)$
PR1-BIND	$\dfrac{A \twoheadrightarrow A' \qquad [p/u]B \twoheadrightarrow [p/v]B'}{\langle u{:}A\rangle B \twoheadrightarrow \langle v{:}A'\rangle B'}$	$p \notin B\,,\, p \notin B'$
PR1-APP	$\dfrac{A \twoheadrightarrow A' \qquad B \twoheadrightarrow B'}{A\,B \twoheadrightarrow A'\,B'}$	

Table 2: 1-Step Parallel Reduction

conditions[5], which ensure that the parameter p correctly indicates the position of the bound variables in the compound terms.

Only `Vclosed` terms participate in $\twoheadrightarrow$ (`par_red1_Vclosed`)

$$\forall A, B.\; A \twoheadrightarrow B \;\Rightarrow\; \mathtt{Vclosed}(A) \wedge \mathtt{Vclosed}(B)\;,$$

and $\twoheadrightarrow$ is reflexive on `Vclosed` terms (`par_red1_refl`)

$$\forall A\;.\; \mathtt{Vclosed}(A) \;\Rightarrow\; A \twoheadrightarrow A\;.$$

A stronger induction principle for $\twoheadrightarrow$ The rules PR1-ATOM, PR1-BIND and PR1-APP are the *congruence* rules for our language. As with `Vclosed` (section 2.4.1), we introduce a strong congruence rule for binders

$$\widetilde{\text{PR1}}\text{-BIND} \qquad \frac{A \tilde{\twoheadrightarrow} A' \qquad \forall p.\,[p/u]B \tilde{\twoheadrightarrow} [p/v]B'}{\langle u{:}A\rangle B \tilde{\twoheadrightarrow} \langle v{:}A'\rangle B'}$$

and prove that $\twoheadrightarrow$ and $\tilde{\twoheadrightarrow}$ are extensionally equivalent, giveing us stronger induction and inversion principles. Because of the eigenvariable conditions in PR1-BIND, a technique using renamings is required

[5] Kleene [Kle52, §78, on the notion of "pure variable" proof] explains how to treat such conditions; however, to do so he must explicitly consider operations on derivations, hence *dependent* elimination, whereas our methods require only rule induction, *i.e.* non-dependent elimination. The second author is grateful to N. Shankar for this reference.

to show the equivalence. We omit the details, but a similar argument is used in section 5.2.1.

The strong induction principle is used to show that $\twoheadrightarrow\!\!\!\!\!+$ is closed under substitution (`par_red1_psub`):

$$\frac{M \mathrel{+\!\!\!\rightarrow} M' \qquad N \mathrel{+\!\!\!\rightarrow} N'}{[N/p]M \mathrel{+\!\!\!\rightarrow} [N'/p]M'}$$

Many-step parallel reduction $\mathrel{+\!\!\!\twoheadrightarrow}$ (`par_redn`), is the transitive closure of $\mathrel{+\!\!\!\rightarrow}$. It inherits properties `par_redn_Vclosed` and `par_redn_refl` from the corresponding properties of $\mathrel{+\!\!\!\rightarrow}$ mentioned above.

3.1.2 Alpha-Conversion

We define α-conversion, $\overset{\alpha}{\sim}$, to be the least congruence, *i.e.* $\overset{\alpha}{\sim}$ is exactly $\mathrel{+\!\!\!\rightarrow}$ without the rule PR1-BETA, so $\overset{\alpha}{\sim} \subseteq \mathrel{+\!\!\!\rightarrow}$. This definition is symmetric, by inspection. To show that it is transitive requires the stronger induction principle for $\overset{\alpha}{\sim}$, which we prove in the same way as above. Hence $\overset{\alpha}{\sim}$ is an equivalence relation[6]. It is decidable for `Vclosed` terms (`decide_alpha_conv`):

$$\forall A, B.\, \texttt{Vclosed}(A) \;\Rightarrow\; \texttt{Vclosed}(B) \;\Rightarrow\; \texttt{decidable}(\, A \overset{\alpha}{\sim} B\,)$$

with a straightfoward but messy proof, by double induction on $\texttt{aVclosed}(A)$ and $\texttt{aVclosed}(B)$.

Informally, alpha-conversion is used for changing the names of variables. We do not have $\langle x{:}A\rangle B \overset{\alpha}{\sim} \langle y{:}A\rangle(\,[y/x]B\,)$ because $[y/x]B$ does not prevent capture. However, we do have (`true_alpha_conv_pi`):

$$\forall v, A, B\;.\; \texttt{Vclosed}(\,\langle v{:}A\rangle B\,) \;\Rightarrow\; \forall x \exists C\;.\; \langle v{:}A\rangle B \overset{\alpha}{\sim} \langle x{:}A\rangle C$$

Closure under α-conversion One of Coquand's original motivations for distinguishing between variables and parameters was to avoid the need to reason about α-conversion; many of the arguments below (Church-Rosser, standardisation, subject reduction) achieve this goal.

Name-carrying syntax is regarded as an abbreviation for a *quotient* modulo α-conversion, so that when we formalise a relation R such as parallel reduction above, we really intend R modulo the quotient structure, i.e. $\overset{\alpha}{\sim} \circ R \circ \overset{\alpha}{\sim}$. We say a relation R is:

[6] This should be contrasted with Gallier's meticulous but long-winded treatment in [Gal90].

closed under α if $\alpha\circ R \subseteq R\circ\alpha$; **strongly closed**, if $\alpha\circ R \subseteq R$; **full wrt** α if $R\circ\alpha \subseteq \alpha\circ R$; **strongly full**, if $R\circ\alpha \subseteq R$.

Remark 3.1 *⇻ is strongly closed under α-conversion: the proof is the same as that for transitivity of $\stackrel{\alpha}{\sim}$, with the additional case of a redex handled by observing that an α-variant of a redex is a redex. However ⇻ is* not *full w.r.t. $\stackrel{\alpha}{\sim}$-classes. For example*

$$([x{:}q]\,x\,x\,)\,([w{:}q]w\,) \mathrel{\text{⇻}} ([y{:}q]y\,)\,([y{:}q]y\,) \qquad \textit{for every } y\,,$$

but no α-variant of the LHS ⇻-reduces to $([y_1{:}q]y_1\,)\,([y_2{:}q]y_2\,)$ where $y_1 \neq y_2$, although this is an α-variant of the RHS.

3.1.3 A Church-Rosser Theorem

Using the argument of Tait and Martin-Löf, as modernized in [Tak95], we prove the first CR theorem (`par_redn_DP`):

$$\forall M, M', M''.\ M \twoheadrightarrow M' \wedge M \twoheadrightarrow M'' \;\Rightarrow\; \exists N.\ M' \twoheadrightarrow N \wedge M'' \twoheadrightarrow N$$

by the usual strip lemma argument and the diamond property of ⇻ (`comp_dev_par_red1_DP`).

To do so, we introduce an inductive characterisation of complete development, $\longrightarrow\!\!\!\!\!\!\text{⇻}$, (`comp_dev`)[7]. It is given by the same rules as ⇻, except for the application rule:

$$\text{CD-APP} \qquad \frac{A \longrightarrow\!\!\!\!\!\!\text{⇻}\; A' \qquad B \longrightarrow\!\!\!\!\!\!\text{⇻}\; B'}{A\,B \longrightarrow\!\!\!\!\!\!\text{⇻}\; A'\,B'} \qquad A \text{ is not a lambda}$$

The side condition in CD-APP forces contraction of all redexes: we have a deterministic sub-relation of ⇻ .

The theorem on finiteness of developments now becomes the combination of:

- induction on the definition of $\longrightarrow\!\!\!\!\!\!\text{⇻}$, which we may think of as a partial correctness assertion;
- the existence (for `Vclosed` terms) of complete developments, (`comp_dev_exists`), which we may think of as a termination argument.

[7] *cf.* the definition of $\longrightarrow\!\!\!\!\!\!\text{⇻}$ as a function by structural recursion on terms [Tak95]

This separation of concerns gives us an advantage over Takahashi's informal proofs, in that we do not have to consider, in each proof about $\multimap\!\!\!\!\!\rightarrow$, a subsidiary induction (case-analysis) to resolve the redex/non-redex distinction in the case of an application. This is handled once and for all in the existence proof, while induction on the definition of $\multimap\!\!\!\!\!\rightarrow$ already delineates the redex/non-redex distinction. The price we pay is that we no longer work with an object-level function, but rather a functional relation.

Of course, we have simplified matters by considering developments of the entire set of redexes in a term: this is sufficient for our purposes, but a more refined analysis (e.g. [Hue94]) would take us beyond our simple datastructure of terms.

The diamond property (`comp_dev_par_red1_DP`) follows easily from the following "Takahashi" lemma (`comp_dev_preCR`):

$$\forall M, N, M'.\ M \,\twoheadrightarrow\!\!\!\!\!\!\;\rightarrow N\,,\ M \longrightarrow\!\!\!\!\!\!\rightarrow M' \ \Rightarrow\ N \,\twoheadrightarrow\!\!\!\!\!\!\;\rightarrow M'$$

whose proof is by induction on $M \longrightarrow\!\!\!\!\!\!\rightarrow M'$, with inversion of $M \,\twoheadrightarrow\!\!\!\!\!\!\;\rightarrow N$. As usual, the interesting case, of a redex/redex, appeals to `par_red1_psub`.

Remark 3.2 *These proofs do not make any appeal to α-conversion. This is because both the $\twoheadrightarrow\!\!\!\!\!\!\;\rightarrow$ and $\longrightarrow\!\!\!\!\!\!\rightarrow$ relations are strongly closed under α-conversion. Indeed, we may show the following two properties, which strengthen* `comp_dev_exists`*, namely* `comp_dev_unique`*:*

$$M \longrightarrow\!\!\!\!\!\!\rightarrow M'\,,\ M \longrightarrow\!\!\!\!\!\!\rightarrow M'' \ \Rightarrow\ M' \overset{\alpha}{\sim} M''\ ;$$

and `comp_dev_exists_unique`*:*

$$M \overset{\alpha}{\sim} M' \ \Rightarrow\ \exists M''.\, M \longrightarrow\!\!\!\!\!\!\rightarrow M'\,,\ M \longrightarrow\!\!\!\!\!\!\rightarrow M''$$

3.2 Conversion

We define conversion, $\simeq$ (`conv`), as the symmetric and transitive closure of $\twoheadrightarrow\!\!\!\!\!\twoheadrightarrow$. It inherits properties `conv_Vclosed` and `conv_refl` from those of $\twoheadrightarrow\!\!\!\!\!\twoheadrightarrow$ mentioned above.

The second Church-Rosser theorem is now straightfoward to prove for conversion (`convCR`):

$$\forall M, M'.\ M \simeq M' \ \Rightarrow\ \exists N.\ M \twoheadrightarrow\!\!\!\!\!\twoheadrightarrow N \wedge M' \twoheadrightarrow\!\!\!\!\!\twoheadrightarrow N$$

3.3 Normal Forms

A term is *beta normal* (`beta_norm`), if it has no beta redexes . This may be defined with the same rules as `aVclosed`, except the rule for application, which is

$$\text{BN-APP} \qquad \frac{\texttt{beta\_norm}(A) \qquad \texttt{beta\_norm}(B)}{\texttt{beta\_norm}(A\,B)} \quad A \text{ is not a lambda}$$

All `beta_norm` terms are `Vclosed` (`beta_norm_Vclosed`). A relation of reduction to normal form is defined:

$$A \downarrow N \triangleq \quad \texttt{beta\_norm}(N) \wedge A \twoheadrightarrow\!\!\!\!\!\!\!\!\:\shortmid\;\; N \qquad\qquad (\texttt{normal\_form})$$

$\twoheadrightarrow\!\!\!\!\!\!\!\!\:\shortmid\;\;$ is reflexive, so there is reduction from a normal form, but every reduct of a normal form is a normal form (`par_redn_bnorm_is_bnorm`)

$$\forall A, B\,.\; A \twoheadrightarrow\!\!\!\!\!\!\!\!\:\shortmid\;\; B \;\Rightarrow\; \texttt{beta\_norm}(A) \;\Rightarrow\; \texttt{beta\_norm}(B)$$

Any reduct of a normal form alpha-converts with that normal form (`par_redn_bnorm_is_alpha_conv`)

$$\forall A, B\,.\; A \twoheadrightarrow\!\!\!\!\!\!\!\!\:\shortmid\;\; B \;\Rightarrow\; \texttt{beta\_norm}(A) \;\Rightarrow\; A \overset{\alpha}{\sim} B\;.$$

Hence, by Church-Rosser, normal forms of a term are unique up to alpha-conversion (`nf_unique`). Since $\overset{\alpha}{\sim} \subseteq \rightarrow\!\!\!\!\!\shortmid\;$, the converse also holds (`nf_alpha_class`)

$$\forall A, M, N.\; A \downarrow M \;\Rightarrow\; (\;A \downarrow N \quad\Leftrightarrow\quad M \overset{\alpha}{\sim} N\;).$$

Thus the class of normal forms of a (`Vclosed`) term is either empty or exactly an alpha-conversion equivalence class.

Deciding conversion Conversion is decidable for normalizing terms. The proof of this depends on Church-Rosser; since normal forms are unique only up to alpha-conversion, it also depends on decidability of alpha-conversion (Section 3.1.2).

3.4 The Standardization Theorem

Our work on type-checking requires us to go beyond theorems such as Church-Rosser in the analysis of reduction. In particular, to talk of syntax-directed systems, we must consider *deterministic* reduction relations, of which weak-head reduction is the simplest. A typical

WH1-APP	$\dfrac{A \to_{wh} A'}{A\,B \to_{wh} A'\,B}$
WH1-BETA	$(\,[v{:}V]B\,)\,A \to_{wh} [A/v]B$

Table 3: One step of weak-head reduction

property required of such a relation is the following counterpart to the quasi-normalisation theorem (`wh_standardisation_lemma`):

$$A \twoheadrightarrow\!\!\!\!\!\!+\;\, B,\ \texttt{whnf}(B) \;\Rightarrow\; \exists C.A \twoheadrightarrow_{wh} C,\ \texttt{whnf}(C),\ C \twoheadrightarrow\!\!\!\!\!\!+\;\, B$$

Takahashi showed how to approach such theorems with an analysis of parallel reduction into head reduction followed by internal reduction, a so-called *semi-standardization* lemma [Mit79]. We adapted her methods to the case of *weak-head* reduction, and the corresponding modified notion of internal reduction. In doing so we simplify them somewhat, in particular by removing the need for the complex invariant $M \star N$. Moreover, the arguments we employed can be replicated in the context of head reduction and internal reduction in their classical senses.

Recently, we rounded off this line of development by proving a standardization theorem for pure languages. The main novelty here is the removal of any mention of residuals (so the reader may be unconvinced that we have formalised *the* standardization theorem). The other thing to observe is that all the desirable consequences of standardization, which we required to analyse type-checking, such as the lemma above, are already corollaries to the semi-standardization lemma.

There are three main ingredients to the theorem: weak-head reduction, internal parallel reduction, and standard reduction itself.

3.4.1 Weak-head reduction

One step of weak-head reduction (`wh_red1`) is shown in Table 3. By inversion, we see that there are no weak-head reducts of a lambda, so we may assume without loss of generality that A is not a lambda in the rule WH1-APP. We have not built any `Vclosed` assumptions into the definition, as this will always be used in a context in which all terms are `Vclosed`.

The reader may validate such a definition by considering the weak-head normal forms (Table 4), and various lemmas relating $\to_{wh}$ and

WH-ATOM	$\texttt{whnf}(\alpha)$	$\alpha \in \texttt{PP} \cup \texttt{SS}$
WH-BIND	$\texttt{whnf}(\langle v{:}A\rangle B)$	
WH-APP	$\dfrac{\texttt{whnf}(A)}{\texttt{whnf}(A\,B)}$	A is not a lambda

Table 4: Weak-head normal forms

IP1-ATOM	$\alpha \twoheadrightarrow\!\!\!\!\!\shortmid\;_i\,\alpha$	$\alpha \in \texttt{PP} \cup \texttt{SS}$
IP1-BIND	$\dfrac{A \twoheadrightarrow\!\!\!\!\!\shortmid\;\, A' \qquad [p/u]B \twoheadrightarrow\!\!\!\!\!\shortmid\;\, [p/v]B'}{\langle u{:}A\rangle B \twoheadrightarrow\!\!\!\!\!\shortmid\;_i\, \langle v{:}A'\rangle B'}$	$p \notin B\,,\, p \notin B'$
IP1-APP	$\dfrac{A \twoheadrightarrow\!\!\!\!\!\shortmid\;_i\, A' \qquad B \twoheadrightarrow\!\!\!\!\!\shortmid\;\, B'}{A\,B \twoheadrightarrow\!\!\!\!\!\shortmid\;_i\, A'\,B'}$	

Table 5: One step of internal parallel reduction

`whnf` (`wh_red1_determin`, `wh_nf_is_nf1`, `alpha_conv_resp_wh_nf`).

Many-step weak-head reduction, $\twoheadrightarrow_{wh}$, (`wh_redn`) is defined as the reflexive transitive closure of $\rightarrow_{wh}$. It is closed under renamings, substitution (`psub_resp_wh_redn`), and application on the right (`wh_redn_app`):

$$\begin{array}{l} \forall M, M'.M \twoheadrightarrow_{wh} M' \;\Rightarrow\; [N/p]M \twoheadrightarrow_{wh} [N/p]M' \\ \forall M, M'.M \twoheadrightarrow_{wh} M' \;\Rightarrow\; M\,N \twoheadrightarrow_{wh} M'\,N \end{array}$$

3.4.2 Internal parallel reduction

The classical notion of head reduction leads to a notion of "internal" redex, as any non-head redex. We adapt such a notion to weak-head reduction, which gives us the definition of internal parallel reduction, $\twoheadrightarrow\!\!\!\!\!\shortmid\;_i$, (`ipar_red1`) as shown in Table 5. We allow arbitrary parallel reduction in each compound term, except in the rator position of applications, where we restrict to internal reduction.

It is immediate by structural induction that internal parallel reductions are parallel reductions. We also have the important abstract property (`ipar_red1_refl_wh_nf`) that $\twoheadrightarrow\!\!\!\!\!+_i$ preserves *and reflects* weak-head normal forms, and *a fortiori*, the shape (outermost constructor) of a term. This reflection of weak-head normal forms, together with the lemma below, is the key to the proof of the quasi-normalisation result with which we opened this discussion.

Semi-standardization (`par_red1_wh_redn_ipar_red1`)

$$\forall M, M'.\, M \twoheadrightarrow\!\!\!\!\!+ M' \;\Rightarrow\; \exists M_w.\, M \twoheadrightarrow_{wh} M_w \twoheadrightarrow\!\!\!\!\!+_i M'$$

Proof. The proof proceeds by induction on $M \twoheadrightarrow\!\!\!\!\!+ M'$. The only tricky case is that of the parallel β step. By inductive hypothesis, we obtain (introducing Skolem constants A_w, B_w)

$$\begin{array}{lcl} \mathsf{ihA} & : & A \twoheadrightarrow_{wh} A_w \twoheadrightarrow\!\!\!\!\!+_i A' \\ \mathsf{ihB} & : & [p/u]B \twoheadrightarrow_{wh} [p/w]B_w \twoheadrightarrow\!\!\!\!\!+_i [p/v]B' \end{array}$$

and we are required to show that there exists some M_w such that

$$(\,[u{:}U]B\,)\,A \twoheadrightarrow_{wh} M_w \twoheadrightarrow\!\!\!\!\!+_i [A'/v]B'$$

Since

$$(\,[u{:}U]B\,)\,A \rightarrow_{wh} [A/u]B \;=\; [A/p]\,[p/u]B \twoheadrightarrow_{wh} [A/p]\,[p/w]B_w$$

by `psub_resp_wh_redn` and `psub_is_vsub_alpha`, we may conclude the result by stitching weak-head reduction sequences together, provided we can establish the following claim, which is the easy base case of Lemma 2.4 in [Tak95] (`wh_ipar_red1_psub`):

$$\frac{M \twoheadrightarrow\!\!\!\!\!+_i M' \qquad N \twoheadrightarrow_{wh} N_w \twoheadrightarrow\!\!\!\!\!+_i N' \qquad N \twoheadrightarrow\!\!\!\!\!+ N'}{\exists P.\, [N/p]M \twoheadrightarrow_{wh} P \twoheadrightarrow\!\!\!\!\!+_i [N'/p]M'}$$

This is proved in the same way as we showed closure of parallel reduction under substitution, by induction on $M \twoheadrightarrow\!\!\!\!\!+_i M'$. A detail to observe here is that we must explicitly assume that the reduction from N to N' is parallel. Takahashi builds this into her $\star$ invariant, whereas in the use of `wh_ipar_red1_psub`, we obtain this assumption for free as the *premise* associated with reduction in A.

We show the case of an application $M = A\,B$, where $A \twoheadrightarrow\!\!\!\!\!+_i A'$ and $B \twoheadrightarrow\!\!\!\!\!+ B'$. By induction hypothesis, there exists some P_A such that $[N/p]A \twoheadrightarrow_{wh} P_A \twoheadrightarrow\!\!\!\!\!\!+_i [N'/p]A'$. The proof of the claim, and hence of the whole lemma, is completed by taking $P =_{\mathrm{def}} P_A\,([N/p]B)$, and appealing once more to `psub_resp_wh_redn`, `ipar_red1_app` and `par_red1_psub`. ∎

To establish the full semi-standardization result for $\twoheadrightarrow\!\!\!\!\!\!+\!\!\!\!\to$, we must also show the commutation result (`ipar_wh_redn_commutes`):

$$M \twoheadrightarrow\!\!\!\!\!\!+_i M_w \twoheadrightarrow_{wh} M' \;\;\Rightarrow\;\; \exists M'_w.\, M \twoheadrightarrow_{wh} M'_w \twoheadrightarrow\!\!\!\!\!\!+_i M'$$

This is a corollary (by induction on $M_w \twoheadrightarrow_{wh} M'$) of the following lemma (`ipar_wh_red1_commutes`):

$$\forall M, M_w, M'.\; M \twoheadrightarrow\!\!\!\!\!\!+_i M_w \rightarrow_{wh} M' \;\;\Rightarrow\;\; \exists M'_w.\, M \twoheadrightarrow_{wh} M'_w \twoheadrightarrow\!\!\!\!\!\!+_i M'$$

Proof. Induction on $M \twoheadrightarrow\!\!\!\!\!\!+_i M_w$, and inversion of the ancillary hypothesis $M_w \rightarrow_{wh} M'$. All cases are trivial, except that of application $M = A\,B$, where $M_w = A'\,B'$, $A \twoheadrightarrow\!\!\!\!\!\!+_i A'$ and $B \twoheadrightarrow\!\!\!\!\!\!+ B'$. We show the case of a wh-redex, where we have: $M_w = ([v{:}V]A'')\,B'$ and $M' = [B'/v]A''$. We use the fact that `ipar_red1` reflects the weak-head normal form $A' = [v{:}V]A''$ to infer that $A = [u{:}U]A'''$. Moreover, since $A \twoheadrightarrow\!\!\!\!\!\!+_i A'$, we obtain by inversion that $U \twoheadrightarrow\!\!\!\!\!\!+ V$ and $\forall p.\, [p/u]A''' \twoheadrightarrow\!\!\!\!\!\!+ [p/v]A''$. Choosing $p \notin A''$, $p \notin A'''$, and applying PR1-SUB, we obtain $[B/u]A''' \twoheadrightarrow\!\!\!\!\!\!+ [B'/v]A''$. Now we appeal to Lemma 3.4.2, finally to conclude that for some P, we have

$$M \;=\; ([u{:}U]A''')\,B \;\rightarrow_{wh}\; [B/u]A''' \;\twoheadrightarrow_{wh}\; P \twoheadrightarrow\!\!\!\!\!\!+_i [B'/v]A'' \;=\; M'$$

as required. ∎

Throughout we used induction on the definitions of the various reduction relations to establish these lemmas. This is by contrast with Takahashi's treatment, where induction is on the term structure (with inversion of the relational hypotheses). This leads to slightly weaker arguments, and consequently to the need for stronger inductive invariants. Such refinements to these arguments would be inconceivable without machine support.

3.4.3 Standard Reduction

The property of being a standard reduction, $\longrightarrow_s$, is usually stated (e.g. by Mitschke [Mit79]) in terms of a highly intensional geometric definition on λ-terms. To formalise this definition directly, we would have to enrich the datatype of terms in order to be able to speak of

STD-ATOM	$\alpha \longrightarrow_s \alpha$	$\alpha \in \mathtt{PP} \cup \mathtt{SS}$
STD-BIND	$\dfrac{A \longrightarrow_s A' \qquad [p/u]B \longrightarrow_s [p/v]B'}{\langle u{:}A\rangle B \longrightarrow_s \langle v{:}A'\rangle B'}$	$p \notin B\,,\, p \notin B'$
STD-APP	$\dfrac{A \longrightarrow_s A' \qquad B \longrightarrow_s B'}{A\,B \longrightarrow_s A'\,B'}$	
STD-WH	$\dfrac{A \twoheadrightarrow_{wh} B \qquad B \longrightarrow_s C}{A \longrightarrow_s C}$	

Table 6: Standard reduction, `standard` (adapted from Plotkin)

redex *positions* in a term. Such an approach has been taken in [Hue94]; we have chosen instead a presentation (table 6), adapted from Plotkin's notion of *standard sequence* [Plo75]. The essence of this presentation is to define standard reduction as the least congruence closed under prefixing by weak-head reductions. We leave implicit the sequence of redexes contracted, as this may be *computed* by recursion, and its "left-to-right" character, and thus avoid mention of residuals or redex positions.

Remark 3.3 *We have defined standard reductions of arbitrary length once and for all, and without any considerations of reduction to normal form; note furthermore, however, that we may easily achieve this end by adding the side condition $A' \neq \lambda$ to the* STD-APP *rule, and* ***whnf***(B) *to the* STD-WH *rule. Also, this definition is both strongly closed under α-conversion,* and *strongly full.*

Our final aim is the following standardization theorem (**`the_standardisation_lemma`**):

$$\forall A, B.\ A \twoheadrightarrow\!\!\!\!\twoheadrightarrow B \ \Rightarrow\ A \longrightarrow_s B$$

Induction on $A \twoheadrightarrow\!\!\!\!\twoheadrightarrow B$ suffices if we can show that our notion of standard reduction *absorbs* single steps of parallel reduction:

$$A \twoheadrightarrow\!\!\!\!\rightarrow B \longrightarrow_s C \ \Rightarrow\ A \longrightarrow_s C\ .$$

By our Lemma 3.4.2, and STD-WH, it suffices to prove the following lemma, that standard reduction absorbs single steps of *internal* parallel reduction (`standard_absorbs_ipar_red1`):

$$A \twoheadrightarrow\!\!\!\!\!\!\!\;\mapsto_i B \longrightarrow_s C \;\;\Rightarrow\;\; A \longrightarrow_s C \; .$$

Proof. The proof is by induction on $B \longrightarrow_s C$. This avoids reconsidering the tricky application case of the commutation lemma above, and we may exploit the fact that $\twoheadrightarrow\!\!\!\!\!\!\!\;\mapsto_i$ preserves and reflects the shape of terms. The price we pay is the need for strong induction on $B \longrightarrow_s C$, as the ancillary hypothesis $A \twoheadrightarrow\!\!\!\!\!\!\!\;\mapsto_i B$ is then an *expansion* step. In each non-atomic case, we make a subsidiary appeal to semi-standardization, in order to be able to exploit the induction hypotheses. This gives a rather mechanical and "abstract nonsense" flavour to the argument, emphasising once more that the real complexity lies in the the proof of Lemma 3.4.2.

We focus on the case of a binder, where we have as hypotheses:

$$\begin{array}{lcl}
\mathsf{premA} & : & A \longrightarrow_s A' \\
\mathsf{premB} & : & B \longrightarrow_s B' \\
\mathsf{ihA} & : & \forall C. C \twoheadrightarrow\!\!\!\!\!\!\!\;\mapsto_i A \;\Rightarrow\; C \longrightarrow_s A' \\
\mathsf{ihB} & : & \forall p. \forall C. C \twoheadrightarrow\!\!\!\!\!\!\!\;\mapsto_i [p/u]B \;\Rightarrow\; C \longrightarrow_s [p/v]B' \\
\mathsf{H} & : & C \twoheadrightarrow\!\!\!\!\!\!\!\;\mapsto_i \langle u{:}A\rangle B
\end{array}$$

By inversion of H, we conclude that $C = \langle w{:}A_c\rangle B_c$, where $A_c \twoheadrightarrow\!\!\!\!\!\!\!\;\mapsto A$ and $\forall p.\, [p/w]B_c \twoheadrightarrow\!\!\!\!\!\!\!\;\mapsto [p/u]B$. But now by semi-standardization, $A_c \twoheadrightarrow_{wh} A_w \twoheadrightarrow\!\!\!\!\!\!\!\;\mapsto_i A$ and $[p/w]B_c \twoheadrightarrow_{wh} B_w \twoheadrightarrow\!\!\!\!\!\!\!\;\mapsto_i [p/u]B$. So by induction hypothesis ihA, we have $A_w \longrightarrow_s A'$, and using STD-WH to fold back the weak-head reductions $A_c \twoheadrightarrow_{wh} A_w$, we finally obtain $A_c \longrightarrow_s A'$. In exactly the same way, modulo the choice of a parameter $p \notin B_c, B, B'$, we obtain $[p/w]B_c \longrightarrow_s [p/v]B'$. But now $C \longrightarrow_s \langle v{:}A'\rangle B'$ by construction.

This concludes the proof that standard reduction absorbs internal reduction, and hence parallel reduction, and so finally we may conclude that every (parallel) β-reduction sequence may be standardized. ∎

4 Pure Type Systems

PTS is a class of type theories given by a set of derivation rules (table 7) parameterized by a PL, $(\mathtt{PP}, \mathtt{VV}, \mathtt{SS})$, and by two relations

- *axioms*, $\mathtt{ax} \subseteq \mathtt{SS} \times \mathtt{SS}$, written informally as $\mathtt{ax}(s_1{:}s_2)$

$$\textsc{Ax} \qquad \bullet \vdash s_1 : s_2 \qquad\qquad \mathtt{ax}(s_1{:}s_2)$$

$$\textsc{Start}\ \frac{\Gamma \vdash A : s}{\Gamma[p{:}A] \vdash p : A} \qquad\qquad p \notin \Gamma$$

$$\textsc{Weak}\ \frac{\Gamma \vdash \alpha : C \qquad \Gamma \vdash A : s}{\Gamma[p{:}A] \vdash \alpha : C} \qquad\qquad \alpha \in \mathtt{PP} \cup \mathtt{SS}\,,\ p \notin \Gamma$$

$$\textsc{Pi}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma[p{:}A] \vdash\ [p/x]B\ : s_2}{\Gamma \vdash\ \{x{:}A\}B\ : s_3} \qquad\qquad \begin{array}{r} p \notin B \\ \mathtt{rl}(s_1, s_2, s_3) \end{array}$$

$$\textsc{Lda}\ \frac{\Gamma[p{:}A] \vdash\ [p/x]M\ :\ [p/y]B \qquad \Gamma \vdash\ \{y{:}A\}B\ : s}{\Gamma \vdash\ [x{:}A]M\ :\ \{y{:}A\}B} \qquad\qquad \begin{array}{r} p \notin M \\ p \notin B \end{array}$$

$$\textsc{App}\ \frac{\Gamma \vdash M : \{x{:}A\}B \qquad \Gamma \vdash N : A}{\Gamma \vdash\ M\,N\ :\ [N/x]B}$$

$$\textsc{tConv}\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s}{\Gamma \vdash M : B} \qquad\qquad A \leq B$$

Table 7: The Typing Rules of PTS (formal name `gts`)

- *rules*, $\mathtt{rl} \subseteq \mathtt{SS} \times \mathtt{SS} \times \mathtt{SS}$, written informally as $\mathtt{rl}(s_1, s_2, s_3)$.

We usually intend `ax` and `rl` to be decidable, but this assumption is not used in the basic theory of PTS. When we are interested in algorithms for typechecking ([Pol94b, Pol95]), even stronger assumptions about decidability are needed.

The typing judgement has the shape $\Gamma \vdash M : A$, meaning that in context Γ, term M has type A. (The formal name of this relation is `gts`, from the old name "Generalized Type Systems".) We call M (or (Γ, M)) the *subject* and A the *predicate* of judgement. Contexts, ranged over by Γ, Δ, bind parameters to types:

$$\Gamma \quad ::= \quad \bullet \ \mid\ \Gamma[p{:}A] \qquad\qquad \text{(null, cons)}.$$

Contexts are formalized as lists over $\mathtt{PP} \times \mathtt{Trm}$. If Γ participates in some derivable judgement, $\Gamma \vdash M : A$, we say $\mathtt{Valid}(\Gamma)$.

4.1 A Generalization: Abstract Conversion

We have further generalized PTS by parameterizing the rules of table 7 on another relation, $\leq$ (called *abstract conversion*), occurring in the side condition of rule tCONV. In conventional presentations of PTS [Bar92], the actual relation of beta-conversion ($\simeq$) is used for $\leq$.

There are several reasons to be interested in parameterizing PTS on its conversion relation. For one thing, the type theory ECC, implemented in LEGO, is not actually a PTS because it uses a generalized notion of conversion called *cumulativity*. ECC is of special interest to us, so we formalize an extension of PTS which includes ECC. Our formal development includes a typechecking algorithm for ECC [Pol94b]. Even for PTS, there is a notorious open problem, the *Expansion Postponement* problem [vBJMP94, Pol94b], which asks if the conversion relation in table 7 can be replaced by beta-reduction without changing the typability of any terms. We know of one other work on PTS using an abstract conversion relation: [BM].

The only properties of $\leq$ necessary to prove the substitution lemma (section 5.4) are reflexivity, transitivity, and invariance under substitution:

`cnv_refl`	$\forall A \,.\, \texttt{Vclosed}(A) \;\Rightarrow\; A \leq A$
`cnv_trans`	$\forall A, D, B \,.\, A \leq D \;\Rightarrow\; D \leq B \;\Rightarrow\; A \leq B$
`psub_resp_cnv`	$\forall N, A, B, p \,.\, \texttt{Vclosed}(N) \Rightarrow$ $A \leq B \;\Rightarrow\; [N/p]A \leq [N/p]B$

To prove the subject reduction theorem (section 5.5), we finally need that conversion is related to contraction of redexes, and has an "internal" Church-Rosser property:

`cnv_conv`	$\simeq \subseteq \leq$
`cnvCR_pi`	$\forall u, v, A, a, B, b \,.$ $\{u{:}A\}B \leq \{v{:}a\}b \;\Rightarrow\; a \leq A \wedge \forall q \,.\, [q/u]B \leq [q/v]b$

Notice the contravariance in the last property. It is easy to prove that $\simeq$ has these properties, so we can formally instantiate $\leq$ by $\simeq$ (or by the cumulativity of ECC) and discharge all these assumptions; we are working abstractly, not making assumptions.

$\simeq$ is an equivalence relation, but $\leq$ is a partial order (which is why we use an asymmetric symbol for $\leq$). Other significant properties of $\simeq$ that do not generally hold for $\leq$ include:

$$s_1 \simeq s_2 \;\Rightarrow\; s_1 = s_2\,, \qquad A \simeq s \;\Rightarrow\; A \twoheadrightarrow s\,, \qquad A \twoheadrightarrow s \;\Rightarrow\; s \in A\,.$$

Some differences between abstract-conversion-PTS and β-conversion-PTS are detailed in [Pol94b]. This kind of analysis, of which properties are actually used in some body of work, is greatly aided by formalization.

4.2 Are the rules a formalization of PTS?

Leaving aside abstract conversion, the rules of table 7 differ from the standard informal presentation [Bar92] in several ways. First, the handling of parameters and variables in the PI and LDA rules, is similar to that in the rules of table 2. Other differences are restriction of weakening to atomic subjects, and the LDA rule.

Binding and substitution The treatment of operating under binders in table 7 is analogous to that in the reduction relations considered above. (See discussion of the rule LDA below.) As the substitution used in rule APP may cause capture of variables, we must show that N is `Vclosed`. In fact we have (`gts_Vclosed_lem`)

$$\forall \Gamma, M, A \,.\; \Gamma \vdash M : A \;\Rightarrow\; \texttt{Vclosed}(M) \wedge \texttt{Vclosed}(A)$$

by structural induction. It also follows easily that a `Valid` context is `Vclosed` in an obvious sense.

Atomic Weakening The standard presentation of weakening in PTS allows any term as subject

$$(\texttt{WEAKENING}) \qquad \frac{\Gamma \vdash M : C \qquad \Gamma \vdash A : s}{\Gamma[p{:}A] \vdash M : C} \qquad p \notin \Gamma$$

where we restrict to atomic terms, α (see rule WEAK). Our rules derive the same judgements (`weakening` is seen to be admissible in section 5.3), but allow fewer derivations (those derivations where weakening is pushed to the leaves). This gives a cleaner meta theory, as induction over derivations treats fewer cases. For example, given a judgement, $\Gamma \vdash M\,N : B$, with an application as its subject, there is no confusion whether it is derived by APP or WEAK. Thus, with atomic weakening, any judgement may only be derived by TCONV or by exactly one of the remaining rules.

The Lambda Rule For the rule for typing a lambda in informal presentations [Bar92, Geu93] is

$$\lambda \qquad \frac{\Gamma[x{:}A] \vdash M : B \qquad \Gamma \vdash \{x{:}A\}B : s}{\Gamma \vdash [x{:}A]M : \{x{:}A\}B}$$

The conventional understanding is that the bound variable, x, doesn't really occur in the conclusion of rule λ, as the notations "$[x{:}A]M$" and "$\{x{:}A\}B$" refer to alpha-equivalence classes[8]. Thus, in concrete notation, the subject and predicate of the lambda rule may bind different variables, which we formalize in our rule LDA. One might, instead, formalize rule λ as

$$\text{LDA'} \qquad \frac{\Gamma[p{:}A] \vdash [p/x]M : [p/x]B \qquad \Gamma \vdash \{x{:}A\}B : s}{\Gamma \vdash [x{:}A]M : \{x{:}A\}B} \qquad \begin{matrix} p \notin M \\ p \notin B \end{matrix}$$

This was our first attempt, and surprisingly, this system derives the same judgements as the system of table 7 (lemmas `rlts_gts` and `gts_rlts`). However, using LDA', the subject reduction theorem is difficult to prove, and derivations are distorted by the need to use the conversion rule for alpha-conversion. See [Pol94b] for more details.

5 PTS With Abstract Conversion

We survey the development leading to the subject reduction theorem. The main difference between this section and the presentation in [Bar92] is our use of the atomic weakening rules (section 4.2), and our simpler proof of subject reduction (section 5.5).

5.1 Some basic facts

Here is a sample of the many small facts that had to be established, usually by simple structural induction.

Parameter lemmas All parameter occurrences in a judgement are bound in the context, and the binding instances in a `Valid` context are distinct parameters:

$$(\Gamma \vdash M : A \wedge p \notin \Gamma) \;\Rightarrow\; p \notin M \wedge p \notin A \qquad (\texttt{free\_params\_lem1})$$
$$\Gamma[p{:}B] \vdash M : A \;\Rightarrow\; p \notin \Gamma \qquad (\texttt{CxtCorrect0})$$

[8]However, in the left premise of rule λ, all free instances of the actual symbol "x" in M and B are intended to refer to the context entry $[x{:}A]$. Thus the conventional reading of this rule doesn't make sense as concrete notation.

Start Lemmas Every axiom is derivable in every valid context, and the global bindings of a valid context are all derivable:

$$(\Gamma \vdash M : A \wedge \mathtt{ax}(s_1{:}s_2)) \;\Rightarrow\; \Gamma \vdash s_1 : s_2 \qquad (\mathtt{sStartLem})$$
$$\Gamma[p{:}B]\Delta \vdash M : A \;\Rightarrow\; \Gamma[p{:}B]\Delta \vdash p : B \qquad (\mathtt{vStartLem})$$

5.2 A Better induction principle for PTS

As for previous relations, we define an alternative typing relation, $\vdash_{\mathsf{a}}$ (`apts`), that identifies all those derivations of each $\vdash$ judgement that are inessentially different because of parameters occurring in the derivation but not in its conclusion. $\vdash_{\mathsf{a}}$ differs from $\vdash$ only in the right premise of the Pi rule and the left premise of the Lda rule.

$$\text{aPi} \quad \frac{\Gamma \vdash_{\mathsf{a}} A : s_1 \qquad \forall p \notin \Gamma\ .\ \Gamma[p{:}A] \vdash_{\mathsf{a}} [p/x]B : s_2}{\Gamma \vdash_{\mathsf{a}} \{x{:}A\}B : s_3}\,\mathtt{rl}(s_1, s_2, s_3)$$

$$\text{aLda} \quad \frac{\forall p \notin \Gamma\ .\ \Gamma[p{:}A] \vdash_{\mathsf{a}} [p/x]M : [p/y]B \qquad \Gamma \vdash_{\mathsf{a}} \{y{:}A\}B : s}{\Gamma \vdash_{\mathsf{a}} [x{:}A]M : \{y{:}A\}B}$$

In these premises we avoid choosing a particular parameter by requiring the premise to hold for all parameters for which there is no reason it should not hold, that is, for all "sufficiently fresh" parameters. As before, we will show that $\vdash$ and $\vdash_{\mathsf{a}}$ derive the same judgements.

It is interesting to compare the side conditions of Pi with those of aPi. In Pi we need the side condition $p \notin B$ so that no unintended occurrences of p (i.e. those not arising from occurrences of the variable x) are bound in the right premise; we do not need $p \notin \Gamma$ because the validity of $\Gamma[p{:}A]$ is obvious from the right premise. In aPi, we cannot require the right premise for all p, but only for those such that $\Gamma[p{:}A]$ remains valid, i.e. those p not occurring in Γ. However the condition $p \notin B$ is not required because of "genericity", that is, the right premise of aPi must hold for the infinitely many parameters not occurring in Γ, while only finitely many of these instances can occur in B.

5.2.1 $\vdash_{\mathsf{a}}$ is equivalent to $\vdash$

As with previous relations, this equivalence will give us a stronger induction principle, and stronger generation (inversion) lemmas for $\vdash$.

$$\forall \Gamma, M, A\ .\ \Gamma \vdash_{\mathsf{a}} M : A \;\Leftrightarrow\; \Gamma \vdash M : A \qquad (\mathtt{apts\_gts},\ \mathtt{gts\_apts})$$

Proof. Direction $\Rightarrow$ is straightfoward by structural induction on $\Gamma \vdash_{\mathsf{a}} M : A$.

To prove direction $\Leftarrow$, first prove a lemma that bijective renamings respect $\vdash_{\mathsf{a}}$ (`bij_ren_resp_apts`)

$$\forall \rho, \Gamma, M, A\ .\ \mathtt{Bij}(\rho) \ \Rightarrow\ \Gamma \vdash_{\mathsf{a}} M : A \ \Rightarrow\ \rho\Gamma \vdash_{\mathsf{a}} \rho M : \rho A$$

by $\vdash_{\mathsf{a}}$-structural induction[9].

Now we proceed to prove $\Gamma \vdash M : A \ \Rightarrow\ \Gamma \vdash_{\mathsf{a}} M : A$ by structural induction on a derivation of $\Gamma \vdash M : A$. All cases are trivial except for the rules Pi and Lda. Consider the case for Pi: we must prove $\Gamma \vdash_{\mathsf{a}} \{n{:}A\}B : s_3$ under the assumptions

$$\begin{array}{lll} \text{sc} & : & \mathtt{rl}(s_1, s_2, s_3) \\ \text{noccB} & : & p \notin B \\ \text{l\_prem} & : & \Gamma \vdash A : s_1 \\ \text{r\_prem} & : & \Gamma[p{:}A] \vdash [p/n]B : s_2 \\ \text{l\_ih} & : & \Gamma \vdash_{\mathsf{a}} A : s_1 \\ \text{r\_ih} & : & \Gamma[p{:}A] \vdash_{\mathsf{a}} [p/n]B : s_2 \end{array}$$

By rule `aPi` (using l_ih) it suffices to show $\Gamma[r{:}A] \vdash_{\mathsf{a}} [r/n]B : s_2$ for arbitrary parameter $r \notin \Gamma$. Thus, using the free parameter lemmas of section 5.1, we know

$$\begin{array}{llll} \text{noccG} & : & r \notin \Gamma & \\ \text{norA} & : & r \notin A & \text{from l\_prem and noccG} \\ \text{nopG} & : & p \notin \Gamma & \text{from r\_prem} \\ \text{nopA} & : & p \notin A & \text{from r\_prem} \end{array}$$

Taking $\rho = \mathtt{swap}(r,p)$, we have $\rho(\Gamma[p{:}A]) \vdash_{\mathsf{a}} \rho(\,[p/n]B\,) : \rho s_2$ is derivable using `bij_ren_resp_apts` to rename r_ih. (Recall from section 2.5 that $\mathtt{swap}(p,q)$ is always bijective.) Thus we are finished if we can show

$$\rho(\Gamma[p{:}A]) = \Gamma[r{:}A] \qquad \wedge \qquad \rho(\,[p/n]B\,) = [r/n]B\ .$$

It is clear that the first equation holds from nopG, noccG, norA and nopA. For the second equation, notice that if $r = p$ then we are done trivially, so assume $r \neq p$, and hence $r \notin [p/n]B$ (from r_prem and noccG). Now, using `vsub_renTrm_commutes` (section 2.5) we have

$$\begin{array}{rcll} \rho([p/n]B) & = & \{p \mapsto r\}([p/n]B) & (r \notin [p/n]B) \\ & = & [\{p \mapsto r\}p/n](\{p \mapsto r\}B)) & (\mathtt{vsub\_renTrm\_commutes}) \\ & = & [r/n]B & (\text{noccB}) \end{array}$$

[9] Actually injectivity of a renaming is enough for it to preserve $\vdash_{\mathsf{a}}$, but we cannot prove this until after we know $\vdash_{\mathsf{a}} \ = \ \vdash$.

as required. ■

5.3 The Thinning Lemma and Weakening

The Thinning Lemma is important to our formulation because it shows that full weakening (`weakening`) is admissible in our system, justifying our use of atomic weakening in the definition of $\vdash$ (section 4.2).

The *subcontext* relation is defined

$$\Gamma \sqsubseteq \Delta \quad \triangleq \quad \forall b : \mathtt{PP} \times \mathtt{Trm} \,.\; b \in \Gamma \;\Rightarrow\; b \in \Delta$$

We also say Δ *extends* Γ. This is the definition used informally in [Bar92, GN91, vBJ93]; a much more complicated definition is required to express this property in a representation using de Bruijn indices for global variables. Now we can state (`thinning_lemma`):

$$\forall \Gamma, \Delta, M, A \,.\; \Gamma \vdash M : A \;\Rightarrow\; (\, \Gamma \sqsubseteq \Delta \wedge \mathtt{Valid}(\Delta) \,) \;\Rightarrow\; \Delta \vdash M : A$$

A naive attempt to prove the thinning lemma by structural induction on $\Gamma \vdash M : A$ encounters serious difficulties with parameter side conditions (see [MP93, Pol94b] for discussion), but a proof is straightfoward using $\vdash_{\mathtt{a}}$-induction, justified by the previous section. The full weakening rule is a corollary of `thinning_lemma`.

5.4 Cut and type correctness

The substitution lemma

$$\frac{\Gamma \vdash N : A \qquad \Gamma[p{:}A]\Delta \vdash M : B}{\Gamma(\,[N/p]\Delta\,) \vdash [N/p]M \;:\; [N/p]B} \qquad (\mathtt{substitution\_lemma})$$

is proved by induction on the derivation of $\Gamma[p : A]\Delta \vdash M : B$. From this we get the commonly used case (`cut_rule`) by instantiating Δ to the empty context.

Among the correctness criteria for type systems is that every type is itself well formed. In PTS we have the theorem (`type_correctness`):

$$\forall \Gamma, M, A \,.\; \Gamma \vdash M : A \;\Rightarrow\; \exists s \,.\; A = s \vee \Gamma \vdash A : s$$

The proof is by structural induction; the only non-trivial case is rule App, which uses the substitution lemma and `vsub_is_psub_alpha` (section 2.3).

5.5 Subject Reduction Theorem: Closure Under Reduction

An important property of type systems is that a term does not lose types under reduction, thus types are a classification of terms preserved by computation. In fact we will show entire $\vdash$-judgements are closed under reduction. We now need all five properties of abstract conversion (section 4.1).

5.5.1 Non-Overlapping Reduction

Our goal is to prove

$$\Gamma \vdash M : A \ \Rightarrow \ M \twoheadrightarrow M' \ \Rightarrow \ \Gamma \vdash M' : A \qquad (\texttt{gtsSR})$$

usually called *the* subject reduction theorem. Our naive strategy is to show that one step of reduction preserves typing, by induction on $\Gamma \vdash M : A$. The critical case is rule App, when the application is actually a redex that is contracted. In order to simplify that case, we want to avoid overlapping redexes, as allowed in the β-rule of $\not\rightarrow$. We want some reduction relation with no overlapping, whose transitive closure is equal to $\not\twoheadrightarrow$.

Another difficulty is that in rules Pi and Lda, a subterm of the subject of the conclusion (the type-label) appears in the context part of a premise; thus in these cases of an induction argument, a reduction in the subject of the conclusion may result in a reduction in the context of a premise. This suggests that the induction hypothesis should be strengthened to simultaneously treat reduction in the context and the subject, leading to the goal

$$\begin{aligned} &G \vdash M : A \quad \Rightarrow \\ &\qquad (M \rightarrow M' \ \Rightarrow \ G \vdash M' : A) \wedge (G \rightarrow G' \ \Rightarrow \ G' \vdash M : A) \end{aligned}$$

where $\rightarrow$ is ordinary one-step β-reduction. This approach is used in [GN91, Bar92], and produces a large number of case distinctions, based on which subterm of the subject contains the one redex which is contracted; all of these subcases are inessential except to isolate the one non-trivial case where the redex contracted is the application constructed by rule App. This simultaneous treatment of one reduction in either the context or the subject suggested to us that the proof would be smoother using a reduction relation that is congruent simultaneously in all branches, while forbidding overlapping of redexes. One step *non-overlapping* reduction, $\stackrel{\text{no}}{\rightarrow}$ (`no_red1`), is defined by the same rules as $\not\rightarrow$

(table 2) except for the β-rule, which is modified to prevent overlapping redexes:

$$\text{NOR1-BETA} \quad (\,[u{:}U]B\,)\,A \overset{\mathsf{no}}{\to} [A/u]B \qquad\qquad \texttt{Vclosed}(\,(\,[u{:}U]B\,)\,A\,)$$

Clearly $\overset{\mathsf{no}}{\to} \subseteq \twoheadrightarrow\!\!\!\!\!\!|\;$, so from the assumed property `cnv_conv` (section 4.1), we have

$$A \overset{\mathsf{no}}{\to} B \;\Rightarrow\; A \le B \wedge B \le A \;.$$

We extend $\overset{\mathsf{no}}{\to}$ compositionally to contexts (`red1Cxt`), and to pairs of a context and a term (`red1Subj`), writing $\Gamma \overset{\mathsf{no}}{\to} \Delta$ and $\langle\Gamma, M\rangle \overset{\mathsf{no}}{\to} \langle\Delta, N\rangle$. We also define $\overset{\mathsf{no}}{\twoheadrightarrow}$ (`no_redn`), the transitive closure of $\overset{\mathsf{no}}{\to}$, and show $\overset{\mathsf{no}}{\twoheadrightarrow} = \twoheadrightarrow\!\!\!\!\!\!|\;\;$ (`no_par_redn`, `par_no_redn`).

5.5.2 The Main Lemma (`subject_reduction_lem`)

$$\Gamma \vdash M : A \;\Rightarrow\; \langle\Gamma, M\rangle \overset{\mathsf{no}}{\twoheadrightarrow} \langle\Gamma', M\rangle' \;\Rightarrow\; \Gamma' \vdash M' : A$$

Proof. By structural induction on $\Gamma \vdash M : A$. We show the interesting case, from rule App. Given

$$\begin{array}{ll}
\text{l_prem} & : \Gamma \vdash M : \{x{:}A\}B \\
\text{r_prem} & : \Gamma \vdash L : A \\
\text{l_ih} & : \forall \Gamma', M' \,.\, (\langle\Gamma, M\rangle \overset{\mathsf{no}}{\to} \langle\Gamma', M'\rangle) \;\Rightarrow\; \Gamma' \vdash M' : \{x{:}A\}B \\
\text{r_ih} & : \forall \Gamma', L' \,.\, (\langle\Gamma, L\rangle \overset{\mathsf{no}}{\to} \langle\Gamma', L'\rangle) \;\Rightarrow\; \Gamma' \vdash L' : A \\
\text{red_subj} & : \langle\Gamma, M\,L\rangle \overset{\mathsf{no}}{\to} \langle\Delta, R\rangle
\end{array}$$

we must show $\Delta \vdash R : [L/x]B$. By induction hypotheses

$$\begin{array}{ll}
\text{gtsDM} & : \Delta \vdash M : \{x{:}A\}B \\
\text{gtsDL} & : \Delta \vdash L : A
\end{array}$$

By type correctness of gtsDM, for some s

$$\text{gtsDpi} : \Delta \vdash \{x{:}A\}B : s$$

By the pi-generation lemma, for some s_2 and $p \notin B$

$$\text{gtsDB} : \Delta[p{:}A] \vdash [p/x]B : s_2$$

By the cut rule on gtsDL and gtsDB (we also use `vsub_is_psub_alpha` (section 2.3) here, and several more times in this case)

$$\text{gtsDBsub} : \Delta \vdash [L/x]B : s_2.$$

Now there are two subcases

$\boldsymbol{R = M' L'}$ where $M \overset{\text{no}}{\to} M'$ and $L \overset{\text{no}}{\to} L'$.
The goal is $\Delta \vdash M' L' : [L/x]B$; by rule App and the induction hypotheses we easily have $\Delta \vdash M' L' : [L'/x]B$. Use rule tCnv and gtsDBsub to expand L' in the predicate back to L as required (this uses `cnv_conv`).

$\boldsymbol{R = [L/v]b}$ where $M = [v{:}A']b$.
The goal is $\Delta \vdash [L/v]b : [L/x]B$. By l_ih $\Delta \vdash [v{:}A']b : \{x{:}A\}B$ and by the lambda-generation lemma, for some w, B' and s'

gtsDpi' : $\Delta \vdash \{w{:}A'\}B' : s'$
gtsDb : $\forall p \notin \Delta \,.\, \Delta[p{:}A'] \vdash [p/v]b : [p/w]B'$
c' : $\{w{:}A'\}B' \leq \{x{:}A\}B$

By `(cnvCR_pi c')` (this is the only time it is used in this proof)

cnvA : $A \leq A'$
cnvB : $\forall q \,.\, [q/w]B' \leq [q/x]B$

By a generation lemma on gtsDpi'

gtsDA' : $\Delta \vdash A' : s1'$

By `(tCnv cnvA gtsDL gtsDA')`

gtsDL' : $\Delta \vdash L : A'$

By tCnv, cnvB and gtsDBsub, it suffices to show $\Delta \vdash [L/v]b : [L/x]B'$ which follows by cut on gtsDL' and gtsDb. ∎

5.5.3 Closure Under Reduction

It is now easy to show the subject reduction theorem, `gtsSR`, and a useful corollary, *predicate reduction*

$$\Gamma \vdash M : A \;\Rightarrow\; A \twoheadrightarrow A' \;\Rightarrow\; \Gamma \vdash M : A' \,. \qquad (\texttt{gtsPR})$$

Finally, extending $\twoheadrightarrow$ compositionally to contexts[10], $\vdash$ is closed under beta-reduction (`gtsAllRed`)

$$\Gamma \vdash M : A \;\Rightarrow\; (\, \Gamma \twoheadrightarrow \Gamma' \wedge M \twoheadrightarrow M' \wedge A \twoheadrightarrow A' \,) \;\Rightarrow\; \Gamma' \vdash M' : A' \,.$$

[10] This relation is equivalent to transitively closing the compositional extension of of $\overset{\text{no}}{\to}$ to contexts.

There is a trivial but useful lemma (`predicate_conv`):

$$(\,\Gamma \vdash M : A \wedge A \simeq s\,) \;\Rightarrow\; \Gamma \vdash M : s\,.$$

Unlike rule TCNV, we don't ask for evidence that s has a type, but the side condition uses $\simeq$, not $\leq$. To prove such a lemma with $\leq$ requires technical restrictions; e.g. ECC with its type hierarchy chopped off a finite level fails to have such a property because of the sort at the top of the hierarchy.

Closure Under Alpha-Conversion From `gtsAllRed` and the fact that $\overset{\alpha}{\sim} \subseteq \twoheadrightarrow$, it follows that $\vdash$ judgements are preserved by $\overset{\alpha}{\sim}$ (`gts_alpha_closed`). Hence an implementation may typecheck a judgement as stated by a user, rather than searching for an alpha-equivalent judgement which is derivable. See [Pol94b, Pol94a].

5.6 Another Presentations of PTS

In several rules of $\vdash$ the context Γ occurs more then once in the list of premises; in order to build a complete derivation, Γ must be constructed (by START and WEAK) in each branch in which it appears. It is much more efficient to assume that we start with a valid context, and only check that when rules extend the context (i.e. the right premise of PI and the left premise of LDA) they maintain validity. This is more in keeping with implementations which are actually used, where we work in a "current context" of mathematical assumptions. We present such a system in table 8, and show it is equivalent to $\vdash$. The idea is originally due to Martin-Löf [Mar71], and is used in [Hue89].

This system has two judgements, a type judgment of shape $\Gamma \vdash_{\mathsf{lv}} M : A$ (`lvtyp`), and a validity judgement of shape $\Gamma \vdash_{\mathsf{lv}}$ (`lvcxt`). Note that they are not mutually inductive: validity depends on typing, but not conversely.

We have proved that $\vdash_{\mathsf{lv}}$ characterizes $\vdash$ (`iff_gts_lvcxt_lvtyp`):

$$\forall \Gamma, M, A\,.\; \Gamma \vdash M : A \quad\Leftrightarrow\quad (\,\Gamma \vdash_{\mathsf{lv}} M : A \wedge \Gamma \vdash_{\mathsf{lv}}\,)$$

Direction $\Leftarrow$ of the proof is subtle. Formally, it uses an auxiliary mutual inductive definition, and a well-founded induction requiring dependent elimination; this is the only place in the entire development that either mutual induction or dependent elimination are used. More abstractly, direction $\Leftarrow$ claims termination of a function that replaces all the proof annotations omitted by $\vdash_{\mathsf{lv}}$. As this is a fast-growing function, its termination is a strong result. The proof is described in [Pol94b].

lvSrt	$\Gamma \vdash_{\mathsf{lv}} s_1 : s_2$	$\mathtt{ax}(s_1{:}s_2)$
lvPar	$\Gamma \vdash_{\mathsf{lv}} p : A$	$[p{:}A] \in \Gamma$
lvPi	$\dfrac{\Gamma \vdash_{\mathsf{lv}} A : s_1 \qquad \Gamma[p{:}A] \vdash_{\mathsf{lv}} [p/x]B : s_2}{\Gamma \vdash_{\mathsf{lv}} \{x{:}A\}B : s_3}$	$\mathtt{rl}(s_1, s_2, s_3)$ $p \notin B,\ p \notin \Gamma$
lvLda	$\dfrac{\Gamma[p{:}A] \vdash_{\mathsf{lv}} [p/x]M : [p/y]B \qquad \Gamma \vdash_{\mathsf{lv}} \{y{:}A\}B : s}{\Gamma \vdash_{\mathsf{lv}} [x{:}A]M : \{y{:}A\}B}$	$p \notin M$ $p \notin B,\ p \notin \Gamma$
lvApp	$\dfrac{\Gamma \vdash_{\mathsf{lv}} M : \{x{:}A\}B \qquad \Gamma \vdash_{\mathsf{lv}} N : A}{\Gamma \vdash_{\mathsf{lv}} M\,N : [N/x]B}$	
lvConv	$\dfrac{\Gamma \vdash_{\mathsf{lv}} M : A \qquad \Gamma \vdash_{\mathsf{lv}} B : s}{\Gamma \vdash_{\mathsf{lv}} M : B}$	$A \leq B$
lvNil	$\bullet \vdash_{\mathsf{lv}}$	
lvCons	$\dfrac{\Gamma \vdash_{\mathsf{lv}}}{\Gamma[p{:}A] \vdash_{\mathsf{lv}}}$	$p \notin \Gamma,\ \ \Gamma \vdash_{\mathsf{lv}} A : s$

Table 8: The system of locally valid contexts (`lvtyp`, `lvcxt`)

6 PTS with β-conversion

It is remarkable how little about $\leq$ has been needed for the theory described so far (section 4.1). In [Pol94b] we pursue the theory of PTS with abstract conversion to a correct typechecking algorithm for *cumulative* PTS, including Luo's system ECC. Here, we point out a more standard theory of PTS with β-conversion, i.e. we instantiate $\leq$ in the preceeding with the actual relation $\simeq$. (In LEGO the command `Cut` executes the admissible rule `substitution_lemma` of section 5.4.) This theory, leading to the strengthening theorem and typechecking algorithms for classes of PTS, is detailed in [vBJMP94].

6.1 Strengthening

Strengthening is a tricky result about PTS, first proved by Jutting [vBJ93]:

$$\begin{array}{ll}\forall\Gamma,\Delta,C,d,D,q\ . & \\ \quad (q\notin d \wedge q \notin D \wedge q \notin \Delta) \ \Rightarrow & \qquad (\texttt{gts\_strengthening}) \\ \qquad \Gamma[q{:}C]\Delta \vdash d : D \ \Rightarrow\ \Gamma\Delta \vdash d : D & \end{array}$$

The development we formalize, in which strengthening is a corollary to work on typechecking, is described in detail in [vBJMP94]. We were particularly interested to prove strengthening because LEGO uses it in the `Discharge` command.

6.2 Functional PTS

Functional PTS are well behaved and are, perhaps, the only ones that are interesting in practice.

$$\texttt{Functional} \triangleq \begin{cases} \texttt{ax}(s{:}t) \wedge \texttt{ax}(s{:}u) \ \Rightarrow\ t = u, & \text{and} \\ \texttt{rl}(s_1,s_2,t) \wedge \texttt{rl}(s_1,s_2,u) \ \Rightarrow\ t = u. & \end{cases}$$

In a functional PTS, `ax` and `rl` are the graphs of partial functions, but we do not necessarily have procedures to compute these functions.

Uniqueness of Types The definition of functional PTS makes sense for abstract-conversion PTS, and is useful in that setting, as it gives a kind of uniqueness: when building a derivation of a typing judgement guided by the syntax of its subject, it is deterministic which axiom to use at each instance of AX, and which rule to use at each instance of PI [Pol94b]. The idea behind the definition of *functional* is that the uniqueness just mentioned propagates through whole derivations to give a property that types are unique up to conversion:

$$\begin{array}{l}\texttt{conv\_unique\_types} \ \triangleq \\ \qquad \forall\Gamma,M,A,B\ .\ \Gamma\vdash M : A \wedge \Gamma \vdash M : B \ \Rightarrow\ A \simeq B\end{array}$$

For β-conversion PTS we prove

$$\texttt{Functional} \ \Rightarrow\ \texttt{conv\_unique\_types} \qquad (\texttt{types\_unicity})$$

by structural induction on $\Gamma\vdash M : A$ and inversion of $\Gamma \vdash M : B$. However, this proof uses properties of $\simeq$, and cannot be modified to prove any similar property of abstract-conversion PTS.

`types_unicity` is too linear for $\leq$, which is only a partial order; the correct generalization is a *principal types* lemma, saying that any type is above some principal type, but we cannot hope that every two types are comparable [Pol94b].

Subject Expansion Any β-PTS with uniqueness of types also has a *subject expansion* property (`subject_expansion`):

$$\begin{array}{l} \texttt{conv\_unique\_types} \Rightarrow \\ \quad \forall \Gamma, M, N, A, B \,.\, (M \twoheadrightarrow N \wedge \Gamma \vdash N : A \wedge \Gamma \vdash M : B) \Rightarrow \\ \qquad \Gamma \vdash M : A \end{array}$$

While subject reduction says that terms don't lose types under reduction, this lemma says terms don't gain types under reduction. In this reading, $\Gamma \vdash N : A$ is the principal premise, and $\Gamma \vdash M : B$ is a well-formedness premise. There are examples of two different ways subject expansion can fail for non-functional PTS in [vBJMP94].

References

[Alt93] Thorsten Altenkirch. A formalization of the strong normalization proof for System F in LEGO. In *Proceedings of the International Conference on Typed Lambda Calculi and Applications, TLCA'93.* Springer-Verlag, LNCS 664, March 1993.

[Bar92] Henk Barendregt. Lambda calculi with types. In Abramsky, Gabbai, and Maibaum, editors, *Handbook of Logic in Computer Science*, volume II. Oxford University Press, 1992.

[Bar95] Bruno Barras. Coq en coq. Master's thesis, DEA Informatique, Mathématiques et Applications, INRIA–Rocquencourt, October 1995.

[Ber90] Stefano Berardi. *Type Dependence and Constructive Mathematics.* PhD thesis, Dipartimento di Informatica, Torino, Italy, 1990.

[BM] Gilles Barthe and Paul-Andre Melliès. On the subject reduction property for algebraic type systems. Presented at CSL'96; submitted for the proceedings.

[Car91] Luca Cardelli. F-sub, the system. Technical report, DEC Systems Research Centre, 1991.

[Coq91] Thierry Coquand. An algorithm for testing conversion in type theory. In G. Huet and G. D. Plotkin, editors, *Logical Frameworks.* Cambridge University Press, 1991.

[Coq96a] Caterina Coquand. Combinator shared reduction and infinite objects in type theory. Manuscript obtained from `http://www.cs.chalmers.se`, March 1996.

[Coq96b] Thierry Coquand. An algorithm for type-checking dependent types. submitted, 1996.

[DB93] Gilles Dowek and Robert Boyer. Towards checking proof checkers. In Herman Geuvers, editor, *Informal Proceedings of the Nijmegen Workshop on Types for Proofs and Programs*, May 1993.

[DFH+93] Dowek, Felty, Herbelin, Huet, Murthy, Parent, Paulin-Mohring, and Werner. The Coq proof assistant user's guide, version 5.8. Technical report, INRIA-Rocquencourt, February 1993.

[Gal90] Jean Gallier. On Girard's *'Candidats de reductibilité'*. In P. Odifreddi, editor, *Logic and Computer Science*, volume 31 of *APIC Studies in Data Processing*, pages 123–203. Academic Press, 1990.

[Gen69] Gerhard Gentzen. *The Collected Papers of Gerhard Gentzen*. Studies in Logic and the Foundations of Mathematics. North-Holland, 1969. editor M. Szabo.

[Geu93] Herman Geuvers. *Logics and Type Systems*. PhD thesis, Department of Mathematics and Computer Science, University of Nijmegen, 1993.

[GM96] Andrew Gordon and Tom Melham. Five axioms of alpha conversion. In *International Conference on Theorem Proving in Higher Order Logics, Turku, Finland*, 1996. To appear.

[GN91] Herman Geuvers and Mark-Jan Nederhof. A modular proof of strong normalization for the calculus of constructions. *Journal of Functional Programming*, 1(2):155–189, April 1991.

[Hue89] Gérard Huet. The constructive engine. In R. Narasimhan, editor, *A Perspective in Theoretical Computer Science*. World Scientific Publishing, 1989. Commemorative Volume for Gift Siromoney.

[Hue94] Gérard Huet. Residual theory in λ-calculus: A formal development. *Journal of Functional Programming*, 4(3):371–394, July 1994.

[JP93] Claire Jones and Randy Pollack. Incremental changes in LEGO: 1993. See `http://www.dcs.ed.ac.uk/home/lego/`, May 1993.

[Kle52] Stephen C. Kleene. *Introduction to Metamathematics*. van Nostrand, Princeton, 1952.

[LP92] Zhaohui Luo and Robert Pollack. LEGO proof development system: User's manual. Technical Report ECS-LFCS-92-211, LFCS, Computer Science Dept., University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ, Scotland, May 1992. Updated version. See `http://www.dcs.ed.ac.uk/home/lego/`

[Luo91] Zhaohui Luo. Program specification and data refinement in type theory. In *TAPSOFT '91 (Volume 1)*, number 493 in Lecture Notes in Computer Science, pages 143–168. Springer-Verlag, 1991.

[Luo94] Z. Luo. *Computation and Reasoning: A Type Theory for Computer Science*. International Series of Monographs on Computer Science. Oxford University Press, 1994.

[Mar71] Per Martin-Löf. A theory of types. Technical Report 71-3, University of Stockholm, 1971.

[McB96] Conor McBride. Inverting inductively defined predicates in LEGO. Technical report, The University of Edinburgh, 1996. forthcoming.

[Mit79] Gerd Mitschke. The standardisation theorem for λ-calculus. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 25:29–31, 1979.

[MP93] James McKinna and Robert Pollack. Pure Type Systems formalized. In M.Bezem and J.F.Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications, TLCA'93, Utrecht*, number 664 in LNCS, pages 289–305. Springer-Verlag, March 1993.

[Nip96] Tobias Nipkow. More Church-Rosser proofs (in isabelle/hol). In *Automated Deduction – CADE-13*, volume 1104 of *LNCS*, pages 733–747. Springer-Verlag, 1996.

[Owe95] Chris Owens. Coding binding and substitution explicitly in Isabelle. In *Proceedings of the Isabelle Users' Workshop*, 1995.

[Pfe92] Frank Pfenning. A proof of the Church-Rosser theorem and its representation in a logical framework. Technical Report CMU-CS-92-186, Carnegie Mellon University, September 1992.

[Plo75] Gordon Plotkin. Call-by-name, call-by-value, and the λ-calculus. *Theoretical Computer Science*, 1, 1975.

[Pol94a] Robert Pollack. Closure under alpha-conversion. In Henk Barendregt and Tobias Nipkow, editors, *Types for Proofs and Programs: International Workshop TYPES'93, Nijmegen, May 1993, Selected Papers*, volume 806 of *LNCS*, pages 313–332. Springer-Verlag, 1994.

[Pol94b] Robert Pollack. *The Theory of LEGO: A Proof Checker for the Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, 1994. `ftp://ftp.dcs.ed.ac.uk/pub/lego/thesis-pollack.ps.Z`.

[Pol95] Robert Pollack. A verified typechecker. In M.Dezani-Ciancaglini and G.Plotkin, editors, *Proceedings of the Second International Conference on Typed Lambda Calculi and Applications, TL-CA'95, Edinburgh*. Springer-Verlag, LNCS 902, April 1995.

[Pol96] Robert Pollack. How to believe a machine-checked proof. Talk given at TYPES'96. Submitted for publication, 1996.

[Pra65] Dag Prawitz. *Natural Deduction; A Proof-Theoretical Study*. Stockholm Studies in Philosophy 3. Almqvist and Wiksell, 1965.

[Reu95] Bernhard Reus. *Program Verification in Synthetic Domain Theory*. PhD thesis, Ludwig-Maximilians-Universität München, 1995.

[Reu96] Bernhard Reus. Synthetic domain theory in type theory: Another logic of computable functions. In *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs'96*, volume 1125 of *LNCS*, pages 363–381. Springer-Verlag, 1996.

[Sat83] Masahiko Sato. Theory of symbolic expressions, I. *Theoretical Computer Science*, 22:19–55, 1983.

[Sch97] Thomas Schreiber. Auxiliary variables and recursive procedures. In *TAPSOFT'97*, LNCS. Springer-Verlag, 1997. To appear.

[Sha85] N. Shankar. A mechanical proof of the Church-Rosser theorem. Technical Report 45, Institute for Computing Science, University of Texas at Austin, March 1985.

[Sto88] A. Stoughton. Substitution revisited. *Theoretical Computer Science*, 17:317–325, 1988.

[Tak95] Masako Takahashi. Parallel reductions in λ-calculus (revised version). *Information and Computation*, 118(1):120–127, April 1995.

[Tas93] A. Tasistro. Formulation of Martin-Löf's theory of types with explicit substitutions. Master's thesis, Chalmers Tekniska Högskola and Göteborgs Universitet, May 1993.

[vBJ93] L.S. van Benthem Jutting. Typing in Pure Type Systems. *Information and Computation*, 105(1):30–41, July 1993.

[vBJMP94] L.S. van Benthem Jutting, James McKinna, and Robert Pollack. Checking algorithms for Pure Type Systems. In Henk Barendregt and Tobias Nipkow, editors, *Types for Proofs and Programs: International Workshop TYPES'93, Nijmegen, May 1993, Selected Papers*, volume 806 of *LNCS*, pages 19–61. Springer-Verlag, 1994.