Some Lambda Calculus and Type Theory Formalized

James McKinna
University of Durham, Department of Computer Science, DH1 3LE, U.K.
*(*j.h.mckinna@durham.ac.uk*)*

Robert Pollack
University of Glasgow, Department of Computing Science, G12 8QQ, Scotland
*(*pollack@dcs.gla.ac.uk*)*

Abstract. We survey a substantial body of knowledge about lambda calculus and Pure Type Systems, formally developed in a constructive type theory using the LEGO proof system. On lambda calculus, we work up to an abstract, simplified, proof of standardization for beta reduction, that does not mention redex positions or residuals. Then we outline the meta theory of Pure Type Systems, leading to the strengthening lemma. One novelty is our use of named variables for the formalization. Along the way we point out what we feel has been learned about general issues of formalizing mathematics, emphasizing the search for formal definitions that are convenient for formal proof and convincingly represent the intended informal concepts.

Keywords: formal mathematics, lambda calculus, Pure Type Systems, type theory, LEGO proof checker.

1. Introduction

"This paper is about our hobby." That is the first sentence of [30], 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 [30, 49, 36, 37]. 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, simplified, proof of standardization for beta reduction, not previously published, that does not mention redex positions or residuals. On 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 [27] was used to check the work in an implementation of the Extended Calculus of Constructions (ECC) with inductive types [26]. LEGO is a refinement style proof

checker, publicly available by ftp and WWW, with a User's Manual [27] and a large collection of examples. Section 1.3 contains information on accessing the formal development described in this paper. Other interesting examples formalized in LEGO include program specification and data refinement [25], strong normalization of System F [1], synthetic domain theory [40, 41], and operational semantics for imperative programs [43, 23].

1.1. Why?

PTS have a beautiful meta-theory, developed informally in [2, 6, 17, 48, 16]. 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 level of detail needed to write a computer program for typechecking PTS and prove to 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 [37]. 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.[49] reports work on typechecking for PTS, done after 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 3.5. 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, since the 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 [44, 20, 32, 33]; the only formalization of a standardization theorem we know of is for lazy combinator expressions [8]. Formalizations of type theory include [11, 4]; both of these address limited aspects of very special type theories (essentially the Calculus of Constructions), although [4] 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 [4], 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") [44, 1, 20, 32, 4] or higher order abstract syntax [33] 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 say that de Bruijn representation

"really is" what we informally mean by lambda terms because there is no need to quotient terms by alpha-conversion; intensional equality on de Bruijn terms corresponds with informal identity of terms. Nonetheless, de Bruijn representation is a coding of the informal notion of binding, and doesn't address the relationship between free and bound variable names: a free variable actually occurs, but its identity is lost when it becomes bound. For example, a presentation using de Bruijn indexes has some very different theorems than the informal presentation with names, because permuting a context of free variables causes judgements under that context to change as well; see the thinning lemma in section 5.3.

In our formalization, syntactic terms using named variables are themselves concrete: the names of bound variables actually occur in object level notations (terms, judgements) containing them, just as the names of free variables do. This is done using a formulation suggested by Coquand [9], based on syntactically distinguishing free from bound variables[1]. Other work on formalization of binding and substitution using names includes [10, 13, 18, 42, 45]. It would be interesting to compare our approach with one of these approaches, but we know of no other non-trivial example formalized using named variables. We guess these approaches would need some technique similar to our derived generalized induction principles (section 2.4.1 and 5.2.1).

Martin-Löf's calculus of explicit substitutions [47] is a deep analysis of type theory which treats named variables concretely. However, there has been no attempt to actually formalize a significant meta theory of this notation. We view it as a failure of concreteness of Martin-Löf's system that some alpha-variants of derivable judgements are not derivable. Our system does not have this weakness (section 5.5.3).

1.3. This paper and the formal development

LEGO, its documentation, basic libraries, and examples are publicly available from the LEGO WWW homepage [24]. The development surveyed in this paper has been checked by LEGO version 1.3.1, the current release at time of writing.

LEGO uses a module system (described in [21]) based on Cardelli's *mock modules* [7]. 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

[1] This distinction is already present in Gentzen [15] (pp. 71–2, 116–7, 141, 216–7) and Prawitz [39]

`strengthening.l` (which contains the proof of strengthening for PTS) will be loaded, preceeded by every module it depends on[2].

There are over 60 proof source files with extension `.l`[3] containing over 2000 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 suppress 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 [38] for discussion of believing a large formal development.

In [30] 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 document and the files drifted apart over time. In [36] 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 sanitized this presentation so that very little purely formal detail shows through. For example, we mostly suppress 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.

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

A few basic notations The development uses LEGO's built-in library of impredicative definitions for the usual logical connectives and their properties; we use standard notation for these connectives. Quantifiers are typed in ECC, but here 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, 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, and `assoc` and `member` for the usual list operations. All functions 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. (`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* is a triple $(\mathtt{PP}, \mathtt{VV}, \mathtt{SS})$ where

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

PP, VV and SS have decidable equality. That PP and 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 PP, VV and 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 PP has decidable equality.

2.1. Terms

The terms of a pure language, `Trm`, ranged over by M, N, A, ..., E, a, b, are given by the grammar

M	::=	$v \mid p \mid s$	*atoms: variable, parameter, sort*
	$\mid$	$[v{:}M]M \mid \{v{:}M\}M$	*binders: lambda, pi*
	$\mid$	$M\,M$	*application*

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`.

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 implementation of LEGO does have such a uniform representation for binders lambda, pi and sigma. In formalization (programs or proofs), representation counts.

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 in 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.

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

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

2.3. Substitution

For the machinery on terms, we use 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 shadowing of a parameter name by a binder.

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

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

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

$[\_/p]\_$ 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.

$$\textsc{Vcl-atom}\quad \texttt{Vclosed}(\alpha) \qquad \alpha \in \mathsf{PP} \cup \mathsf{SS}$$

$$\textsc{Vcl-bind}\quad \frac{\texttt{Vclosed}(A) \qquad \texttt{Vclosed}([p/v]B)}{\texttt{Vclosed}(\langle v{:}A\rangle B)}$$

$$\textsc{Vcl-app}\quad \frac{\texttt{Vclosed}(A) \qquad \texttt{Vclosed}(B)}{\texttt{Vclosed}(A\,B)}$$

Figure 1. Inductive definition of the relation `Vclosed`.

Some important lemmas can now be proved:

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

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 (figure 1). This definition 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. This relation is a simple case of ideas that recur many times in what follows, so 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 choice possible: 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 formalization correctly captures her

informal understanding. But a formalizer 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. `Vclosed` is equivalent to having no free variables in a conventional sense (see file `voccur.l`). This observation is of informal interest, to see that the definition of `Vclosed` is reasonable, but we do not use it formally because `Vclosed` allows us to avoid all talk of free variables.

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* [2], or *inversion principles* [3]. Note that inversion principles are determined by the *shape* of a definition, not by its extension. LEGO now has has very useful tactics to automate the use of inversion [29], but what we describe in this paper was done before these 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`:

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

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 right premise of the rule for binders, which requires $\texttt{aVclosed}([p/v]B)$ for *every* p:

$$\text{aVCL-BIND}\quad \frac{\texttt{aVclosed}(A) \qquad \forall p\,.\,\texttt{aVclosed}([p/v]B)}{\texttt{aVclosed}(\langle v{:}A\rangle B)}$$

We will show that `Vclosed` and `aVclosed` derive the same judgements.

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. This type of definition is called *generalized induction*. Notice also that for any term, A, there is at most one derivation of $\texttt{aVclosed}(A)$, while there may be many derivations of $\texttt{Vclosed}(A)$, differing in the parameters used in the right premises of instances of VCL-BIND.

What will be gained? By showing `aVclosed` and `Vclosed` to be *extensionally* equivalent (i.e. deriving the same judgements), we can view the induction principle of `aVclosed` as a "derived induction principle" for the extension of `Vclosed`, eliminating `Vclosed` derivations but providing a stronger induction hypothesis than `Vclosed` induction: namely that the induction property holds for *all* parameters, not just the one that actually appears in the eliminated `Vclosed` derivation.[5] 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.

Why don't we just use `aVclosed` in place of `Vclosed`? For one thing, while `aVclosed` is stronger for elimination, `Vclosed` is stronger for introduction: the right premise of VCL-BIND is easier to prove than that of aVCL-BIND. This trick of proving a judgement defined with ordinary induction by eliminating a judgement defined with generalized induction is used extensively in our work; see section 3.1.2 for an example. Also, `Vclosed` may be more natural and believable for some readers. Readers must decide for themselves if our formal definitions have the informal meaning we claim for them, so different but provably equivalent definitions make the meaning of a formal development

[5] Compare with the induction principle of Gordon and Melham [18] (section 3.2).

clearer. Finally, the generalized inductive definition of `aVclosed` cannot be formalized in weak systems like Primitive Recursive Arithmetic, or Feferman's FS0 [12]. Our development could be carried out without generalized induction, but this would be very inconvenient, and possibly "infeasible".

Equivalence of `Vclosed` and `aVclosed`

$$\texttt{aVclosed}(A) \;\Leftrightarrow\; \texttt{Vclosed}(A). \qquad \begin{array}{l}(\texttt{aVclosed\_Vclosed})\\(\texttt{Vclosed\_aVclosed})\end{array}$$

Both directions follow easily by structural inductions once we have the following claim:

$$\texttt{aVclosed}([p/v]B) \;\Rightarrow\; \forall q\,.\,\texttt{aVclosed}([q/v]B). \quad (\texttt{aVclosed\_alpha})$$

Proof. The claim is proved by induction on $\texttt{lngth}(B)$. 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

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

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

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

By `aVclosed` inversion applied to assumption vclp we also know

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

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 `lngth`, the first of these holds by ih and h1. For the second, let r be an arbitrary parameter, and consider cases. If $v = n$ then we are done by h2; i.e. $[q/v]B$ doesn't actually appear in the goal, and $[p/v]B$ doesn't actually appear in h2. Finally the interesting case: if $v \neq n$ we use a straightforward lemma

$$\begin{array}{l}\forall v, w\,.\,v \neq w \Rightarrow \\ \quad \forall r, q, A\,.\,[r/v][q/w]A = [q/w][r/v]A\end{array} \qquad (\texttt{alpha\_commutes\_alpha})$$

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 h2 after again rewriting the order of substituting p and r. □

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.

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

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.

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

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:

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

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

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

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

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

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:

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

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

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 parameter, just as in the definition of `Vclosed`. What is new here are the *eigenvariable* side conditions $q \notin M$, $q \notin N$ of rule ξ[6] which ensure that the parameter q correctly indicates the position of the variables bound in the conclusion.

Here is an instance of ξ where incorrect capture might occur (contracting the underlined redex):

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

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

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

because

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

Thus incorrect capture is avoided. This shows the need for the new variable y appearing in rule ξ.

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 emphasized by Takahashi in her beautiful account [46].

Following the ideas discussed above, one step parallel reduction $\xrightarrow{\mathsf{p}}$ (`par_red1`) is defined in figure 2. Only `Vclosed` terms participate in

[6] To treat such conditions Kleene [22] (§78, on *pure variable* proofs) explicitly considers operations on derivations (*i.e.* dependent elimination) whereas our methods require only rule induction (*i.e.* non-dependent elimination).

$$\textsc{pr1-atom}\quad \alpha \xrightarrow{\mathsf{p}} \alpha \qquad\qquad \alpha \in \mathtt{PP} \cup \mathtt{SS}$$

$$\textsc{pr1-beta}\quad \frac{A \xrightarrow{\mathsf{p}} A' \qquad [p/u]B \xrightarrow{\mathsf{p}} [p/v]B'}{([u{:}U]B)\, A \xrightarrow{\mathsf{p}} [A'/v]B'} \qquad \begin{array}{r} p \notin B, p \notin B' \\ \mathtt{Vclosed}(U) \end{array}$$

$$\textsc{pr1-bind}\quad \frac{A \xrightarrow{\mathsf{p}} A' \qquad [p/u]B \xrightarrow{\mathsf{p}} [p/v]B'}{\langle u{:}A\rangle B \xrightarrow{\mathsf{p}} \langle v{:}A'\rangle B'} \qquad p \notin B, p \notin B'$$

$$\textsc{pr1-app}\quad \frac{A \xrightarrow{\mathsf{p}} A' \qquad B \xrightarrow{\mathsf{p}} B'}{A\, B \xrightarrow{\mathsf{p}} A'\, B'}$$

Figure 2. 1-step of parallel reduction (`par_red1`).

$\xrightarrow{\mathsf{p}}$:

$$A \xrightarrow{\mathsf{p}} B \;\Rightarrow\; \mathtt{Vclosed}(A) \wedge \mathtt{Vclosed}(B). \qquad (\mathtt{par\_red1\_Vclosed})$$

This property is established by PR1-ATOM, and passed from premises to conclusion by all rules except PR1-BETA, where U doesn't appear in the premises. This explains the one `Vclosed` side condition in figure 2. Similarly, $\xrightarrow{\mathsf{p}}$-reflexivity on `Vclosed` terms

$$\mathtt{Vclosed}(A) \;\Rightarrow\; A \xrightarrow{\mathsf{p}} A \qquad (\mathtt{par\_red1\_refl})$$

is inherited from rule PR1-ATOM.

Remark. Rule PR1-BETA could be replaced by

$$\frac{A \xrightarrow{\mathsf{p}} A' \qquad [p/u]B \xrightarrow{\mathsf{p}} B'}{([u{:}U]B)\, A \xrightarrow{\mathsf{p}} [A'/p]B'} \qquad \begin{array}{r} p \notin B \\ \mathtt{Vclosed}(U) \end{array}$$

without changing the extension of $\xrightarrow{\mathsf{p}}$. See proof file `pred1.l`.

A stronger induction principle for $\xrightarrow{\mathsf{p}}$. As with `Vclosed` (section 2.4.1), we introduce an alternative relation, $\xrightarrow{\mathsf{ap}}$ (`par_ared1`), with a strong congruence rule for binders

$$\textsc{apr1-bind}\quad \frac{A \xrightarrow{\mathsf{ap}} A' \qquad \forall p\,.\, [p/u]B \xrightarrow{\mathsf{ap}} [p/v]B'}{\langle u{:}A\rangle B \xrightarrow{\mathsf{ap}} \langle v{:}A'\rangle B'}$$

and prove that $\xrightarrow{\mathsf{p}}$ and $\xrightarrow{\mathsf{ap}}$ are extensionally equivalent, giving a strong derived induction principle for $\xrightarrow{\mathsf{p}}$. Because of the eigenvariable conditions in PR1-BIND, renaming is used to show the equivalence. We omit the details; a similar argument is used in section 5.2.1.

The strong induction principle is used to show the essential substitution lemma:

$$(M \xrightarrow{\mathsf{p}} M' \wedge N \xrightarrow{\mathsf{p}} N') \;\Rightarrow\; [N/p]M \xrightarrow{\mathsf{p}} [N'/p]M'. \quad (\texttt{par\_red1\_psub})$$

Many-step parallel reduction $\xrightarrow{\mathsf{p}}\!\!\!\!\rightarrow$ (`par_redn`), is the transitive closure of $\xrightarrow{\mathsf{p}}$. It inherits properties `par_redn_Vclosed` and `par_redn_refl` from the corresponding properties of $\xrightarrow{\mathsf{p}}$.

3.1.1. *Alpha-conversion*
We define α-conversion, $\overset{\alpha}{\sim}$, to be the least congruence, *i.e.* $\overset{\alpha}{\sim}$ is exactly $\xrightarrow{\mathsf{p}}$ without the rule PR1-BETA, so $\overset{\alpha}{\sim} \subseteq \xrightarrow{\mathsf{p}}$. 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.[7] 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:

$$\begin{array}{l}\texttt{Vclosed}(\langle v{:}A\rangle B) \Rightarrow \\ \qquad \forall x\, \exists C\,.\, \langle v{:}A\rangle B \overset{\alpha}{\sim} \langle x{:}A\rangle C.\end{array} \qquad (\texttt{true\_alpha\_conv\_pi})$$

α-conversion is decidable (`decide_alpha_conv`):

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

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

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, standardization, subject reduction) achieve this goal.

Informal name-carrying syntax is an abbreviation for a *quotient* modulo α-conversion, so that when we formalize a relation R, such as parallel reduction, 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 *closed* (resp. *strongly closed*) wrt α if $\alpha\circ R \subseteq R\circ\alpha$ (resp. $\alpha\circ R \subseteq R$). Similarly R is *full* (resp. *strongly full*) wrt α if $R\circ\alpha \subseteq \alpha\circ R$ (resp. $R\circ\alpha \subseteq R$).

$\xrightarrow{\mathsf{p}}$ is strongly closed wrt α-conversion; the proof is similar to that for transitivity of $\overset{\alpha}{\sim}$. However $\xrightarrow{\mathsf{p}}$ is not full wrt $\overset{\alpha}{\sim}$-classes. For example

$$([x{:}q]x\,x)\,([w{:}q]w) \xrightarrow{\mathsf{p}} ([y{:}q]y)\,([y{:}q]y) \qquad \text{for every } y,$$

but no α-variant of the LHS $\xrightarrow{\mathsf{p}}$-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.

[7] Compare with Gallier's [14] meticulous but long-winded treatment.

3.1.2. *The Church-Rosser theorem*
We prove the first Church-Rosser theorem (`par_redn_DP`)

$$(M \xrightarrow{\mathsf{p}}\!\!\!\!\rightarrow M' \wedge M \xrightarrow{\mathsf{p}}\!\!\!\!\rightarrow M'') \;\Rightarrow\; \exists N \,.\, M' \xrightarrow{\mathsf{p}}\!\!\!\!\rightarrow N \wedge M'' \xrightarrow{\mathsf{p}}\!\!\!\!\rightarrow N$$

by the usual strip lemma argument using the diamond property of $\xrightarrow{\mathsf{p}}$ (`par_red1_DP`). We prove the latter by the argument of Tait and Martin-Löf, as modernized by Takahashi [46].

Define the relation of *complete development* $\xrightarrow{\mathsf{cd}}$ (`comp_dev`) inductively by the same rules as $\xrightarrow{\mathsf{p}}$, except for the application rule:

$$\text{CD-APP} \qquad \frac{A \xrightarrow{\mathsf{cd}} A' \qquad B \xrightarrow{\mathsf{cd}} B'}{A\,B \xrightarrow{\mathsf{cd}} A'\,B'} \qquad A \text{ is not a lambda}$$

As usual, we define a strong form $\xrightarrow{\mathsf{acd}}$ (`comp_adev`), and use it to prove a strong derived induction principle for $\xrightarrow{\mathsf{cd}}$.

By the side condition in CD-APP, only the β rule applies to a redex; hence $\xrightarrow{\mathsf{cd}}$ is a sub-relation of $\xrightarrow{\mathsf{p}}$ that deterministically contracts every redex. In contrast, Takahashi [46] defines $\xrightarrow{\mathsf{cd}}$ as a *function* by structural recursion on terms. Thus we must prove that complete developments always exist:

$$\texttt{Vclosed}(M) \;\Rightarrow\; \exists N \,.\, M \xrightarrow{\mathsf{cd}} N. \qquad (\texttt{comp\_dev\_exists})$$

This proof, using the strong derived induction on $\texttt{Vclosed}(M)$, exemplifies the technique of "strong elimination and weak introduction" mentioned in section 2.4.1. In the case for binders we must prove $\exists N \,.\, \langle x{:}A\rangle B \xrightarrow{\mathsf{cd}} N$ given, among other things, the induction hypothesis, $\forall p\, \exists N \,.\, [p/x]B \xrightarrow{\mathsf{cd}} N$. It is unclear how to use this induction hypothesis to introduce the strong form, $\xrightarrow{\mathsf{acd}}$, because lacking a choice principle, we cannot eliminate the existential quantifier under the universal. However, to introduce $\xrightarrow{\mathsf{cd}}$ we just choose a fresh enough parameter, p, to instantiate the universal quantifier of the induction hypothesis.

Another main lemma is that any $\xrightarrow{\mathsf{p}}$-step can be completed,

$$(M \xrightarrow{\mathsf{p}} N \wedge M \xrightarrow{\mathsf{cd}} M') \;\Rightarrow\; N \xrightarrow{\mathsf{p}} M', \qquad (\texttt{comp\_dev\_preCR})$$

which is proved by induction on $M \xrightarrow{\mathsf{cd}} M'$ with inversion of $M \xrightarrow{\mathsf{p}} N$. The interesting redex/redex case uses `par_red1_psub`. Our approach improves on Takahashi's informal proof (by structural induction on M) in that, in inductions on $\xrightarrow{\mathsf{cd}}$, we do not need subsidiary induction (case-analysis) to resolve the redex/non-redex distinction in the application case; this is already delineated by the definition of $\xrightarrow{\mathsf{cd}}$.

The diamond property of $\xrightarrow{\mathsf{p}}$ follows easily from these two lemmas: for every term M there is a complete development $M \xrightarrow{\mathsf{cd}} M'$, and every $\xrightarrow{\mathsf{p}}$ step from M can be completed to M'.

These proofs do not mention α-conversion because both $\xrightarrow{\mathsf{p}}$ and $\xrightarrow{\mathsf{cd}}$ are strongly closed wrt α-conversion. Indeed, we may show the following two properties:

$$(M \xrightarrow{\mathsf{cd}} M' \wedge M \xrightarrow{\mathsf{cd}} M'') \;\Rightarrow\; M' \overset{\alpha}{\sim} M'', \qquad (\texttt{comp\_dev\_unique})$$
$$M \overset{\alpha}{\sim} M' \;\Rightarrow\; \exists M'' \,.\, M \xrightarrow{\mathsf{cd}} M'' \wedge M' \xrightarrow{\mathsf{cd}} M''. \qquad (\texttt{comp\_dev\_ex\_uni})$$

3.2. Conversion

We define conversion, $\simeq$ (`conv`), as the symmetric and transitive closure of $\xrightarrow{\mathsf{p}}\!\!\!\!\rightarrow$. It inherits properties `conv_Vclosed` and `conv_refl` from those of $\xrightarrow{\mathsf{p}}\!\!\!\!\rightarrow$ mentioned above. The second Church-Rosser theorem is straightfoward to prove:

$$M \simeq M' \;\Rightarrow\; \exists N \,.\, M \xrightarrow{\mathsf{p}}\!\!\!\!\rightarrow N \wedge M' \xrightarrow{\mathsf{p}}\!\!\!\!\rightarrow N. \qquad (\texttt{convCR})$$

3.3. Beta normal form

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

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

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

$$A \downarrow N \triangleq \quad \texttt{beta\_norm}(N) \wedge A \xrightarrow{\mathsf{p}}\!\!\!\!\rightarrow N. \qquad (\texttt{normal\_form})$$

$\xrightarrow{\mathsf{p}}\!\!\!\!\rightarrow$ 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`)

$$A \xrightarrow{\mathsf{p}}\!\!\!\!\rightarrow B \;\Rightarrow\; \texttt{beta\_norm}(A) \;\Rightarrow\; \texttt{beta\_norm}(B).$$

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

$$A \xrightarrow{\mathsf{p}}\!\!\!\!\rightarrow B \;\Rightarrow\; \texttt{beta\_norm}(A) \;\Rightarrow\; A \overset{\alpha}{\sim} B.$$

Hence, by Church-Rosser, normal forms of a term are unique up to alpha-conversion (`nf_unique`). Since $\overset{\alpha}{\sim} \subseteq \xrightarrow{\mathsf{p}}$, the converse also holds:

$$A \downarrow M \;\Rightarrow\; (A \downarrow N \;\Leftrightarrow\; M \overset{\alpha}{\sim} N). \qquad (\texttt{nf\_alpha\_class})$$

$$\textsc{wh1-app}\quad \frac{A \xrightarrow{\mathsf{wh}} A'}{A\,B \xrightarrow{\mathsf{wh}} A'\,B}$$

$$\textsc{wh1-beta}\quad ([v{:}V]B)\,A \xrightarrow{\mathsf{wh}} [A/v]B$$

Figure 3. One step of weak-head reduction (`wh_red1`).

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 3.1.1).

3.4. Weak-head reduction

One step of weak-head reduction $\xrightarrow{\mathsf{wh}}$ (`wh_red1`) is defined in figure 3. By inversion, we see that there are no weak-head reducts of a lambda, so in the rule WH1-APP A cannot be a lambda. This definition will only be used in a context in which all terms are `Vclosed`.

Many-step weak-head reduction $\xrightarrow{\mathsf{wh}}\!\!\!\!\rightarrow$ (`wh_redn`), is the reflexive transitive closure of $\xrightarrow{\mathsf{wh}}$. It is closed under renamings, substitution and application on the right:

$$M \xrightarrow{\mathsf{wh}}\!\!\!\!\rightarrow M' \;\Rightarrow\; [N/p]M \xrightarrow{\mathsf{wh}}\!\!\!\!\rightarrow [N/p]M' \quad (\texttt{psub\_resp\_wh\_redn})$$
$$M \xrightarrow{\mathsf{wh}}\!\!\!\!\rightarrow M' \;\Rightarrow\; M\,N \xrightarrow{\mathsf{wh}}\!\!\!\!\rightarrow M'\,N \quad (\texttt{wh\_redn\_app})$$

Weak-head normal form, `whnf`(_), is defined by the same rules as beta normal form, except for the application rule, which is replaced by:

$$\textsc{whnf-app}\quad \frac{\texttt{whnf}(A)}{\texttt{whnf}(A\,B)} \qquad A \text{ is not a lambda}$$

3.5. The standardization theorem

Our work on type checking, in particular syntax-directed typing systems, requires us to consider *deterministic* reduction relations, of which weak-head reduction is the simplest. A typical property required of such a relation is the following counterpart to the quasi-normalization theorem (`wh_standardisation_lemma`):

$$(A \xrightarrow{\mathsf{p}}\!\!\!\!\rightarrow B \wedge \texttt{whnf}(B)) \;\Rightarrow\; \exists C\,.\; A \xrightarrow{\mathsf{wh}}\!\!\!\!\rightarrow C \wedge \texttt{whnf}(C) \wedge C \xrightarrow{\mathsf{p}}\!\!\!\!\rightarrow B$$

$$\text{IP1-ATOM}\quad \alpha \xrightarrow{\mathsf{ip}} \alpha \qquad\qquad \alpha \in \mathtt{PP} \cup \mathtt{SS}$$

$$\text{IP1-BIND}\quad \frac{A \xrightarrow{\mathsf{p}} A' \qquad [p/u]B \xrightarrow{\mathsf{p}} [p/v]B'}{\langle u{:}A\rangle B \xrightarrow{\mathsf{ip}} \langle v{:}A'\rangle B'} \qquad p \notin B, p \notin B'$$

$$\text{IP1-APP}\quad \frac{A \xrightarrow{\mathsf{ip}} A' \qquad B \xrightarrow{\mathsf{p}} B'}{A\,B \xrightarrow{\mathsf{ip}} A'\,B'}$$

Figure 4. One step of internal parallel reduction (`ipar_red1`).

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 [31]. We adapted her methods to the case of weak-head reduction, and the corresponding modified notion of internal reduction. Our arguments, slightly simpler than Takahashi's, also work for head reduction and internal reduction in their classical senses.

Although all the properties which we needed to analyse type checking, such as the lemma above, are already corollaries to the semi-standardization lemma, we rounded off this line of development by proving a standardization theorem for pure languages. The main novelty is avoiding any mention of residuals (so the reader may be unconvinced that we have formalized *the* standardization theorem).

3.5.1. *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 the definition of internal parallel reduction, $\xrightarrow{\mathsf{ip}}$ (`ipar_red1`), shown in figure 4. $\xrightarrow{\mathsf{ip}}$ allows arbitrary parallel reduction in each compound term, except in the rator position of applications, where we restrict to internal reduction.

It is immediate that $\xrightarrow{\mathsf{ip}} \subseteq \xrightarrow{\mathsf{p}}$. $\xrightarrow{\mathsf{ip}}$-inversion is powerful because $\xrightarrow{\mathsf{ip}}$ preserves the outermost constructor of terms. This, together with the next lemma, is the key to the proof of the quasi-normalization result with which we opened this discussion.

Lemma 1. Semi-standardization for $\xrightarrow{\mathsf{p}}$:

$$M \xrightarrow{\mathsf{p}} M' \Rightarrow \exists M_w \,.\, M \xrightarrow{\mathsf{wh}} M_w \xrightarrow{\mathsf{ip}} M' \qquad (\mathtt{par\_red1\_wh\_redn\_ipar\_red1})$$

Proof. By induction on $M \xrightarrow{\mathsf{p}} M'$; the only tricky case is that of the parallel β step. By inductive hypothesis, we obtain (introducing Skolem constants A_w, B_w)

$$\begin{array}{ll} \mathsf{ihA} & : \; A \xrightarrow{\mathsf{wh}}\!\!\!\!\rightarrow A_w \xrightarrow{\mathsf{ip}} A' \\ \mathsf{ihB} & : \; [p/u]B \xrightarrow{\mathsf{wh}}\!\!\!\!\rightarrow [p/w]B_w \xrightarrow{\mathsf{ip}} [p/v]B' \end{array}$$

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

$$([u{:}U]B)\, A \xrightarrow{\mathsf{wh}}\!\!\!\!\rightarrow M_w \xrightarrow{\mathsf{ip}} [A'/v]B'.$$

Since

$$([u{:}U]B)\, A \xrightarrow{\mathsf{wh}} [A/u]B = [A/p][p/u]B \xrightarrow{\mathsf{wh}}\!\!\!\!\rightarrow [A/p][p/w]B_w$$

by `psub_resp_wh_redn` and `vsub_is_psub_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 [46]:

$$\frac{M \xrightarrow{\mathsf{ip}} M' \qquad N \xrightarrow{\mathsf{wh}}\!\!\!\!\rightarrow N_w \xrightarrow{\mathsf{ip}} N' \qquad N \xrightarrow{\mathsf{p}} N'}{\exists P\,.\, [N/p]M \xrightarrow{\mathsf{wh}}\!\!\!\!\rightarrow P \xrightarrow{\mathsf{ip}} [N'/p]M'} \qquad (\texttt{wh\_ipar\_red1\_psub})$$

This is proved by induction on $M \xrightarrow{\mathsf{ip}} M'$. We show the case of an application $M = A\,B$, where $A \xrightarrow{\mathsf{ip}} A'$ and $B \xrightarrow{\mathsf{p}} B'$. By induction hypothesis, there exists some P_A such that $[N/p]A \xrightarrow{\mathsf{wh}}\!\!\!\!\rightarrow P_A \xrightarrow{\mathsf{ip}} [N'/p]A'$. The proof of the claim, and hence of the whole lemma, is completed by taking $P \triangleq P_A\,([N/p]B)$. □

To get the full semi-standardization result for $\xrightarrow{\mathsf{p}}\!\!\!\!\rightarrow$, we must also show the commutation property:

$$\begin{array}{l} M \xrightarrow{\mathsf{ip}} M_w \xrightarrow{\mathsf{wh}} M' \Rightarrow \\ \qquad \exists M'_w\,.\, M \xrightarrow{\mathsf{wh}}\!\!\!\!\rightarrow M'_w \xrightarrow{\mathsf{ip}} M' \end{array} \qquad (\texttt{ipar\_wh\_red1\_commutes})$$

Proof. Induction on $M \xrightarrow{\mathsf{ip}} M_w$, with inversion of $M_w \xrightarrow{\mathsf{wh}} M'$. All cases are simple, except that of application,

$$M = A\,B, \quad \text{where} \quad M_w = A'\,B', \quad A \xrightarrow{\mathsf{ip}} A' \quad \text{and} \quad B \xrightarrow{\mathsf{p}} B'.$$

We show the case where M_w is a weak-head redex,

$$M_w = ([v{:}V]A'')\,B', \quad \text{with} \quad A' = [v{:}V]A'' \quad \text{and} \quad M' = [B'/v]A''.$$

Inverting $A \xrightarrow{\mathsf{ip}} A' = [v{:}V]A''$, there are u, U and A''' such that

$$A = [u{:}U]A''' \quad \text{and} \quad \forall p\,.\, [p/u]A''' \xrightarrow{\mathsf{p}} [p/v]A''.$$

$$\text{STD-ATOM}\quad \alpha \xrightarrow{\mathsf{s}} \alpha \qquad \alpha \in \mathsf{PP} \cup \mathsf{SS}$$

$$\text{STD-BIND}\quad \frac{A \xrightarrow{\mathsf{s}} A' \qquad [p/u]B \xrightarrow{\mathsf{s}} [p/v]B'}{\langle u{:}A\rangle B \xrightarrow{\mathsf{s}} \langle v{:}A'\rangle B'} \qquad p \notin B, p \notin B'$$

$$\text{STD-APP}\quad \frac{A \xrightarrow{\mathsf{s}} A' \qquad B \xrightarrow{\mathsf{s}} B'}{A\,B \xrightarrow{\mathsf{s}} A'\,B'}$$

$$\text{STD-WH}\quad \frac{A \xrightarrow{\mathsf{wh}} \!\!\!\rightarrow B \qquad B \xrightarrow{\mathsf{s}} C}{A \xrightarrow{\mathsf{s}} C}$$

Figure 5. Standard reduction (`standard`), adapted from Plotkin.

We have $[B/u]A''' \xrightarrow{\mathsf{p}} [B'/v]A''$ by `par_red1_psub`, and by lemma 1 there is some M'_w such that

$$M = ([u{:}U]A''')\,B \xrightarrow{\mathsf{wh}} [B/u]A''' \xrightarrow{\mathsf{wh}} \!\!\!\rightarrow M'_w \xrightarrow{\mathsf{ip}} [B'/v]A'' = M'$$

as required. □

Throughout we used induction on the definitions of 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, which gives slightly weaker arguments, and consequently requires stronger inductive invariants. Such refinements would be inconceivable without machine support.

3.5.2. *Standard reduction*
The notion of *standard reduction* $\xrightarrow{\mathsf{s}}$ (`standard`) is often defined in intensional geometric terms (e.g. [31]). To formalize this notion directly, we would have to enrich the datatype of terms to speak of redex positions, as is done by Huet [20]. Instead we adapt Plotkin's notion of *standard sequence* [34]. Standard reduction (figure 5) is defined as the least congruence closed under prefixing by weak-head reductions. $\xrightarrow{\mathsf{s}}$ is of arbitrary length.[8] We leave implicit the sequence of redexes contracted (this may be computed by recursion) and its "left-to-right" character, avoiding mention of residuals or redex positions. This definition is both strongly closed and strongly full wrt α-conversion.

The standardization lemma. Our final aim is:

$$A \xrightarrow{\mathsf{p}} \!\!\!\rightarrow B \;\Rightarrow\; A \xrightarrow{\mathsf{s}} B. \qquad (\texttt{standardisation\_lem})$$

[8] $\xrightarrow{\mathsf{s}}$ could be forced to go to normal form by adding the side conditions "A' is not a lambda" to the STD-APP rule, and "$\texttt{whnf}(B)$" to the STD-WH rule.

Induction on $A \xrightarrow{\mathsf{p}}\!\!\!\!\rightarrow B$ suffices if we can show that our notion of standard reduction absorbs single steps of parallel reduction:

$$D \xrightarrow{\mathsf{p}} B \xrightarrow{\mathsf{s}} C \;\Rightarrow\; D \xrightarrow{\mathsf{s}} C.$$

By lemma 1, and STD-WH, it suffices to prove that standard reduction absorbs single steps of internal parallel reduction:

$$D \xrightarrow{\mathsf{ip}} B \xrightarrow{\mathsf{s}} C \;\Rightarrow\; D \xrightarrow{\mathsf{s}} C. \qquad (\texttt{standard\_absorbs\_ipar\_red1})$$

Proof. By induction on $B \xrightarrow{\mathsf{s}} C$; we avoid reconsidering the tricky application case of the commutation lemma above, and exploit $\xrightarrow{\mathsf{ip}}$-inversion. The price we pay is the need for strong derived induction on $B \xrightarrow{\mathsf{s}} C$, as the ancillary hypothesis $D \xrightarrow{\mathsf{ip}} B$ is then an *expansion* step. In each non-atomic case we make a subsidiary appeal to semi-standardization in order to apply the induction hypotheses. This gives a mechanical and "abstract nonsense" flavour to the argument, emphasizing that the real complexity lies in the the proof of lemma 1.

We focus on the case of a binder, $D \xrightarrow{\mathsf{ip}} \langle u{:}A\rangle B \xrightarrow{\mathsf{s}} \langle v{:}A'\rangle B'$, where we have:

$$\begin{array}{ll} \mathsf{ihA} & : \forall D \,.\, D \xrightarrow{\mathsf{ip}} A \;\Rightarrow\; D \xrightarrow{\mathsf{s}} A' \\ \mathsf{ihB} & : \forall p, D \,.\, D \xrightarrow{\mathsf{ip}} [p/u]B \;\Rightarrow\; D \xrightarrow{\mathsf{s}} [p/v]B' \\ \mathsf{H} & : D \xrightarrow{\mathsf{ip}} \langle u{:}A\rangle B \end{array}$$

By inversion of H, there are w, A_D and B_D such that

$$D = \langle w{:}A_D\rangle B_D, \quad \text{where} \quad A_D \xrightarrow{\mathsf{p}} A \quad \text{and} \quad \forall p \,.\, [p/w]B_D \xrightarrow{\mathsf{p}} [p/u]B.$$

Now by semi-standardization (choosing a fresh parameter p), there are A_w and B_w such that

$$A_D \xrightarrow{\mathsf{wh}}\!\!\!\!\rightarrow A_w \xrightarrow{\mathsf{ip}} A \qquad \text{and} \qquad [p/w]B_D \xrightarrow{\mathsf{wh}}\!\!\!\!\rightarrow B_w \xrightarrow{\mathsf{ip}} [p/u]B.$$

By ihA, we have $A_w \xrightarrow{\mathsf{s}} A'$, and using STD-WH to fold back the weak-head steps $A_D \xrightarrow{\mathsf{wh}}\!\!\!\!\rightarrow A_w$, we obtain $A_D \xrightarrow{\mathsf{s}} A'$. Similarly, by ihB we have $[p/w]B_D \xrightarrow{\mathsf{s}} [p/v]B'$. Now $D = \langle w{:}A_D\rangle B_D \xrightarrow{\mathsf{s}} \langle v{:}A'\rangle B'$ as required. □

Having proved that standard reduction absorbs internal reduction, and hence parallel reduction; we conclude that every parallel reduction sequence may be standardized.

4. Pure Type Systems

PTS is a class of type theories given by a set of derivation rules (figure 6) parameterized by a pure language (PP, VV, SS), and by two relations

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

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

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

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

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

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

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

Figure 6. The typing rules of PTS (gts).

- *axioms*, $\mathtt{ax} \subseteq \mathtt{SS} \times \mathtt{SS}$, written informally as $\mathtt{ax}(s_1{:}s_2)$
- *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 ([36, 37]), 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 $\langle \Gamma, M \rangle$) the *subject* and A the *predicate* of a judgement. Contexts, ranged over by Γ, Δ, bind parameters to types:

$$\Gamma \quad ::= \quad \bullet \ \mid\ \Gamma[p{:}A]. \qquad\qquad (\mathtt{nilCxt}, \mathtt{CONS})$$

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 figure 6 on another relation, $\leq$ (called *abstract conversion*), occurring in the

side condition of rule tConv. In conventional presentations of PTS [2], 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 [36]. Even for PTS, there is a notorious open problem, the *Expansion Postponement* problem [49, 36], which asks if the conversion relation in figure 6 can be replaced by reduction without changing the typability of any terms. (Our abstraction does not solve this problem, but a better one might.) We know of one other work on PTS using an abstract conversion relation: [5].

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

$$\begin{array}{ll}
(\texttt{cnv_refl}) & \texttt{Vclosed}(A) \;\Rightarrow\; A \leq A \\
(\texttt{cnv_trans}) & A \leq D \;\Rightarrow\; D \leq B \;\Rightarrow\; A \leq B \\
(\texttt{psub_resp_cnv}) & \texttt{Vclosed}(N) \;\Rightarrow\; A \leq B \;\Rightarrow\; [N/p]A \leq [N/p]B
\end{array}$$

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:

$$\begin{array}{ll}
(\texttt{cnv_conv}) & \simeq \;\subseteq\; \leq \\
(\texttt{cnvCR_pi}) & \{u{:}A\}B \leq \{v{:}a\}b \;\Rightarrow\; a \leq A \wedge \forall q \,.\, [q/u]B \leq [q/v]b
\end{array}$$

Notice the contravariance in `cnvCR_pi`, and, because $\simeq$ is symmetric while $\leq$ is not, in `cnv_conv`. 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:

$$s_1 \simeq s_2 \;\Rightarrow\; s_1 = s_2, \qquad A \simeq s \;\Rightarrow\; A \xrightarrow{\mathsf{p}}\!\!\!\!\rightarrow s, \qquad A \xrightarrow{\mathsf{p}}\!\!\!\!\rightarrow s \;\Rightarrow\; s \in A.$$

Some differences between abstract-conversion-PTS and β-conversion-PTS are detailed in [36]. This kind of analysis, of which properties are actually used in some body of work, is greatly aided by formalization.

4.2. Is it a formalization of PTS?

Leaving aside abstract conversion, the rules of figure 6 differ from the standard informal presentation [2] in several ways. First, the handling of parameters and variables in the Pi and Lda rules, is similar to that in the rules of figure 2. Other differences are restriction of weakening to atomic subjects, and the Lda rule.

Binding and substitution The treatment of operating under binders in figure 6 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

$$\Gamma \vdash M : A \;\Rightarrow\; \texttt{Vclosed}(M) \wedge \texttt{Vclosed}(A) \qquad (\texttt{gts\_Vclosed\_lem})$$

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 (`weakening`)

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

whereas our rule Weak restricts M to atomic terms. Our presentation derives the same judgements as the standard one (`weakening` is seen to be admissible in section 5.3), but allows 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 The rule for typing a lambda in informal presentations [2, 16] 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.[9] 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

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

This was our first attempt, and surprisingly, this system derives the same judgements as the system of figure 6 (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 [36] 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 [2] 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 have to be established, usually by simple structural induction.

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

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

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

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

[9] However, in the left premise of rule λ, free occurrences 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.

5.2. A better induction principle for PTS

As for previous relations, we define an alternative typing relation, $\vdash_{\mathtt{a}}$ (`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. $\vdash_{\mathtt{a}}$ differs from $\vdash$ only in the right premise of the Pi rule and the left premise of the Lda rule.

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

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

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 $\vdash_{\mathtt{a}}$ 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 parameters can occur in B.

5.2.1. $\vdash_{\mathtt{a}}$ *is equivalent to* $\vdash$

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

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

Proof. Direction $\Rightarrow$ is straightfoward by structural induction on $\Gamma \vdash_{\mathtt{a}} M : A$. For direction $\Leftarrow$, first prove a lemma that bijective renamings respect $\vdash_{\mathtt{a}}$

$$(\mathtt{inj}(\rho) \wedge \mathtt{sur}(\rho)) \Rightarrow \Gamma \vdash_{\mathtt{a}} M : A \;\Rightarrow\; \rho\Gamma \vdash_{\mathtt{a}} \rho M : \rho A \qquad (\mathtt{bij\_ren\_resp\_apts})$$

by $\vdash_{\mathtt{a}}$-structural induction.[10]

Now we proceed to prove $\Gamma \vdash M : A \;\Rightarrow\; \Gamma \vdash_{\mathtt{a}} M : A$ by structural induction on a derivation of $\Gamma \vdash M : A$. All cases are trivial except for the rules Pi

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

and LDA. Consider the case for PI: we must prove $\Gamma \vdash_{\mathsf{a}} \{n{:}A\}B : s_3$ under the assumptions

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

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

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

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 bijective.) Thus we are finished if we can show

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

It is clear that the first equation holds from nopG, noccG, norA and nopA. For the second equation, 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). Using `vsub_renTrm_commutes` (section 2.5) we have

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

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

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

This is the definition used informally in [2, 17, 48]; 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`):

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

A naive attempt to prove the thinning lemma by structural induction on $\Gamma \vdash M : A$ encounters serious difficulties with parameter side conditions (see [30, 36] 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

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

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 type correctness theorem:

$$\Gamma \vdash M : A \;\Rightarrow\; \exists s \,.\, A = s \vee \Gamma \vdash A : s \qquad (\mathtt{type\_correctness})$$

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

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 use all five properties of abstract conversion (section 4.1).

5.5.1. *Non-overlapping reduction*

Our goal is to prove

$$\Gamma \vdash M : A \;\Rightarrow\; M \xrightarrow{\mathtt{p}\!\!\twoheadrightarrow} M' \;\Rightarrow\; \Gamma \vdash M' : A \qquad (\mathtt{gtsSR})$$

usually called the subject reduction theorem. A naive strategy is to show that one step of reduction preserves typing, by induction on $\Gamma \vdash M : A$. One difficulty is that in rules Pi and Lda, the type label, A, of the subject of the conclusion appears in the context part, $\Gamma[p{:}A]$, of a premise; thus in the Pi and Lda cases of an induction argument, a contraction in the subject of the conclusion may result in a contraction in the context of a premise. To address this problem, [17, 2] strengthen the induction hypothesis to simultaneously treat reduction in the context and the subject, leading to the goal

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

where $\rightarrow$ is one step of ordinary β-reduction. This is an improvement over previous proofs, but still produces a large number of case distinctions, based on which part 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 introduced by rule App. This suggested to us that the proof would be smoother using a reduction relation that is congruent simultaneously in all branches of the subject, in the same sense that $\xrightarrow[\mathsf{p}]{}\!\!\!\!\rightarrow$ is congruent simultaneously in all branches of a term. However, to simplify the critical case of rule App when the introduced application is actually a redex that is contracted, we want to avoid overlapping redexes, as allowed in the β-rule of $\xrightarrow[\mathsf{p}]{}$. Thus we define one step *non-overlapping* reduction, $\xrightarrow[\mathsf{no}]{}$ (`no_red1`), by the same rules as $\xrightarrow[\mathsf{p}]{}$ (figure 2) except for the β-rule, which is modified to prevent overlapping redexes:[11]

$$\textsc{nor1-beta} \qquad ([u{:}U]B)\,A \xrightarrow[\mathsf{no}]{} [A/u]B \qquad\qquad \texttt{Vclosed}(([u{:}U]B)\,A)$$

Clearly $\xrightarrow[\mathsf{no}]{} \subseteq \xrightarrow[\mathsf{p}]{}$, so using `cnv_conv` (section 4.1), we have

$$A \xrightarrow[\mathsf{no}]{} B \;\Rightarrow\; A \leq B \wedge B \leq A.$$

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

$$M \xrightarrow[\mathsf{no}]{}\!\!\!\!\twoheadrightarrow N \;\Leftrightarrow\; M \xrightarrow[\mathsf{p}]{}\!\!\!\!\twoheadrightarrow N. \qquad (\texttt{no\_par\_redn},\ \texttt{par\_no\_redn})$$

5.5.2. *The main lemma*

$$\Gamma \vdash M : A \Rightarrow \langle \Gamma, M\rangle \xrightarrow[\mathsf{no}]{} \langle \Gamma', M\rangle' \;\Rightarrow\; \Gamma' \vdash M' : A. \qquad (\texttt{subject\_reduction\_lem})$$

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

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

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

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

[11] An anonymous referee pointed out that essentially the same proof of the main lemma goes through using $\xrightarrow[\mathsf{p}]{}$ (extended compositionally to subjects) in place of $\xrightarrow[\mathsf{no}]{}$. This is true for β-conversion PTS, but in our abstract setting a stronger assumption on $\leq$ than `psub_resp_cnv` is required, precisely because a redex may be contained in the critical redex (see proof file `par_subject_reduction.l`). $\xrightarrow[\mathsf{no}]{}$ really is "better" than $\xrightarrow[\mathsf{p}]{}$ for structuring the argument we give.

By type correctness of gtsDM, for some s

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

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

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

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)

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

Now there are two subcases.

$\boldsymbol{R = M'\,L'}$ where $M \xrightarrow{\mathsf{no}} M'$ and $L \xrightarrow{\mathsf{no}} L'$.
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, $\Delta \vdash [v{:}A']b : \{x{:}A\}B$. Inverting this we have, for some w, B' and s'

$$\begin{array}{ll}\mathsf{gtsDb}: & \forall p \notin \Delta \,.\, \Delta[p{:}A'] \vdash [p/v]b : [p/w]B' \\ \mathsf{c'} \quad : & \{w{:}A'\}B' \leq \{x{:}A\}B\end{array}$$

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

$$\begin{array}{ll}\mathsf{cnvA}: & A \leq A' \\ \mathsf{cnvB}: & \forall q \,.\, [q/w]B' \leq [q/x]B\end{array}$$

By `(tCnv cnvA gtsDL)`

$$\mathsf{gtsDL'}: \ \Delta \vdash L : A'$$

By `psub_resp_cnv` and cnvB we have $[L/p]B' \leq [L/p]B$, so by tCnv and gtsDBsub, it suffices to show $\Delta \vdash [L/v]b : [L/x]B'$, which follows by `cut_rule` 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*

$$\Gamma \vdash M : A \ \Rightarrow \ A \xrightarrow{\mathsf{p}} A' \ \Rightarrow \ \Gamma \vdash M : A'. \qquad (\texttt{gtsPR})$$

Finally, extending $\xrightarrow{\mathsf{p}}$ compositionally to contexts,[12] $\vdash$ is closed under reduction (`gtsAllRed`)

$$\Gamma \vdash M : A \ \Rightarrow \ (\Gamma \xrightarrow{\mathsf{p}} \Gamma' \wedge M \xrightarrow{\mathsf{p}} M' \wedge A \xrightarrow{\mathsf{p}} A') \ \Rightarrow \ \Gamma' \vdash M' : A'.$$

[12] This relation is equivalent to transitively closing the compositional extension of of $\xrightarrow{\mathsf{no}}$ to contexts.

There is a trivial but useful lemma:

$$(\Gamma \vdash M : A \wedge A \simeq s) \;\Rightarrow\; \Gamma \vdash M : s. \qquad (\texttt{predicate\_conv})$$

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 at a finite level fails to have such a property because of the sort at the top of the hierarchy.

Closure under alpha-conversion Since $\overset{\alpha}{\sim} \subseteq \overset{\mathsf{p}}{\longrightarrow}$ it follows from `gtsAllRed` that $\vdash$ judgements are preserved by $\overset{\alpha}{\sim}$ (`gts_alpha_closed`):

$$\Gamma \vdash M : A \;\Rightarrow\; (\Gamma \overset{\alpha}{\sim} \Gamma' \wedge M \overset{\alpha}{\sim} M' \wedge A \overset{\alpha}{\sim} A') \;\Rightarrow\; \Gamma' \vdash M' : A'.$$

Hence an implementation may typecheck a judgement as stated by a user, rather than searching for an alpha-equivalent judgement which is derivable. For example, standardizing the variables of a judgement apart, which is necessary in Martin-Löf's calculus of explicit substitutions [35], is not necessary in our formalization of PTS.

5.6. Another presentation of PTS

In several rules of $\vdash$ the context Γ occurs more than 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 figure 7, and show it is equivalent to $\vdash$. The idea is originally due to Martin-Löf [28], and is used in [19].

This system has two judgements, a type judgement 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. Hence the condition $\Gamma \vdash_{\mathsf{lv}} A : s$ of rule lvCons is a side condition, not a premise.

We have proved that $\vdash_{\mathsf{lv}}$ characterizes $\vdash$:

$$\Gamma \vdash M : A \;\Leftrightarrow\; (\Gamma \vdash_{\mathsf{lv}} M : A \wedge \Gamma \vdash_{\mathsf{lv}}) \qquad (\texttt{iff\_gts\_lvcxt\_lvtyp})$$

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 [36].

$$\textsc{lvSrt}\quad \Gamma \vdash_{\mathsf{lv}} s_1 : s_2 \qquad \mathtt{ax}(s_1{:}s_2)$$

$$\textsc{lvPar}\quad \Gamma \vdash_{\mathsf{lv}} p : A \qquad [p{:}A] \in \Gamma$$

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

$$\textsc{lvLda}\quad \frac{\Gamma[p{:}A] \vdash_{\mathsf{lv}} [p/x]M : [p/y]B \qquad \Gamma \vdash_{\mathsf{lv}} \{y{:}A\}B : s}{\Gamma \vdash_{\mathsf{lv}} [x{:}A]M : \{y{:}A\}B} \qquad \begin{array}{r}p \notin M\\ p \notin B,\ p \notin \Gamma\end{array}$$

$$\textsc{lvApp}\quad \frac{\Gamma \vdash_{\mathsf{lv}} M : \{x{:}A\}B \qquad \Gamma \vdash_{\mathsf{lv}} N : A}{\Gamma \vdash_{\mathsf{lv}} M\,N : [N/x]B}$$

$$\textsc{lvConv}\quad \frac{\Gamma \vdash_{\mathsf{lv}} M : A \qquad \Gamma \vdash_{\mathsf{lv}} B : s}{\Gamma \vdash_{\mathsf{lv}} M : B} \qquad A \leq B$$

$$\textsc{lvNil}\quad \bullet \vdash_{\mathsf{lv}}$$

$$\textsc{lvCons}\quad \frac{\Gamma \vdash_{\mathsf{lv}}}{\Gamma[p{:}A] \vdash_{\mathsf{lv}}} \qquad p \notin \Gamma,\ \Gamma \vdash_{\mathsf{lv}} A : s$$

Figure 7. 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 [36] 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 an admissible rule similar to `substitution_lem` of section 5.4.) This theory, leading to the strengthening theorem and typechecking algorithms for classes of PTS, is detailed in [49].

6.1. Strengthening

Strengthening for all PTS, a hard result, was proved by Jutting [48]:

$$(q \notin d \wedge q \notin D \wedge q \notin \Delta) \Rightarrow \Gamma[q{:}C]\Delta \vdash d : D \ \Rightarrow\ \Gamma\Delta \vdash d : D. \qquad (\mathtt{gts\_strengthening})$$

The development we formalize, in which strengthening is a corollary to work on typechecking, is described in detail in [49].

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` makes sense for abstract-conversion PTS, and is useful in that setting: 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 [36]. For β-conversion PTS, this determinism propagates through derivations to give a property that types are unique up to conversion:

$$\texttt{conv\_unique\_types} \triangleq \forall \Gamma, M, A, B \,.\, (\Gamma \vdash M : A \wedge \Gamma \vdash M : B) \Rightarrow A \simeq B$$

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

Because $\leq$ is only a partial order, the correct generalization to abstract-conversion PTS is a *principal types* lemma, saying that any type of term M is "above" some principal type of M, but we cannot hope that every two types are comparable [36].

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

$$\texttt{conv\_unique\_types} \Rightarrow (M \xrightarrow{\mathrm{p}} N \wedge \Gamma \vdash N : A \wedge \Gamma \vdash M : B) \Rightarrow \Gamma \vdash M : A.$$

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 [49].

References

1. Altenkirch, T.: 1993, '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*, Vol. 664 of *LNCS*.
2. Barendregt, H.: 1992, 'Lambda Calculi with Types'. In: Abramsky, Gabbai, and Maibaum (eds.): *Handbook of Logic in Computer Science*, Vol. II. Oxford University Press.

3. Barras et al.: 1998, 'The Coq Proof Assistant Reference Manual'. INRIA-Rocquencourt. `http://pauillac.inria.fr/coq/`.
4. Barras, B.: 1996, 'Coq en Coq'. Rapport de Recherche 3026, INRIA.
5. Barthe, G. and P.-A. Melliès: 1997, 'On the Subject Reduction Property for Algebraic Type Systems'. In: *CSL'96: Proceedings of the 10th Annual Conference of the European Association for Computer Science Logic, Utrecht*, Vol. 1258 of *LNCS*. pp. 34–57.
6. Berardi, S.: 1990, 'Type Dependence and Constructive Mathematics'. Ph.D. thesis, Dipartimento di Informatica, Torino, Italy.
7. Cardelli, L.: 1991, 'F-sub, the System'. Technical report, DEC Systems Research Centre.
8. Coquand, C.: 1996a, 'Combinator Shared Reduction and Infinite Objects in Type Theory'. Manuscript obtained from `http://www.cs.chalmers.se`.
9. Coquand, T.: 1991, 'An Algorithm for Testing Conversion in Type Theory'. In: G. Huet and G. Plotkin (eds.): *Logical Frameworks*.
10. Coquand, T.: 1996b, 'An Algorithm for Type-Checking Dependent Types'. *Science of Computer Programming* **26**(1–3), 167–177.
11. Dowek, G. and R. Boyer: 1993, 'Towards Checking Proof Checkers'. In: H. Geuvers (ed.): *Informal Proceedings of the Nijmegen Workshop on Types for Proofs and Programs*.
12. Feferman, S.: 1988, 'Finitary Inductively Presented Logics'. In: *Logic Colloquium '88, Padova*. North-Holland.
13. Gabbay, M. and A. Pitts: 1998, 'A New Approach to Abstract Syntax Involving Binders'. Draft.
14. Gallier, J.: 1990, 'On Girard's *'Candidats de reductibilité'*'. In: P. Odifreddi (ed.): *Logic and Computer Science*, Vol. 31 of *APIC Studies in Data Processing*. Academic Press, pp. 123–203.
15. Gentzen, G.: 1969, *The Collected Papers of Gerhard Gentzen*, Studies in Logic and the Foundations of Mathematics. North-Holland. editor M. Szabo.
16. Geuvers, H.: 1993, 'Logics and Type Systems'. Ph.D. thesis, Department of Mathematics and Computer Science, University of Nijmegen.
17. Geuvers, H. and M.-J. Nederhof: 1991, 'A Modular Proof of Strong Normalization for the Calculus of Constructions'. *Journal of Functional Programming* **1**(2), 155–189.
18. Gordon, A. and T. Melham: 1996, 'Five Axioms of Alpha Conversion'. In: Von Wright, Grundy, and Harrison (eds.): *Ninth Conference on Theorem Proving in Higher Order Logics TPHOL'96, Turku*, Vol. 1125 of *LNCS*. pp. 173–190.
19. Huet, G.: 1989, 'The Constructive Engine'. In: R. Narasimhan (ed.): *A Perspective in Theoretical Computer Science*. World Scientific Publishing. Commemorative Volume for Gift Siromoney.
20. Huet, G.: 1994, 'Residual Theory in λ-Calculus: A Formal Development'. *Journal of Functional Programming* **4**(3), 371–394.
21. Jones, C. and R. Pollack: 1993, 'Incremental Changes in LEGO: 1993'. See [24].
22. Kleene, S. C.: 1952, *Introduction to Metamathematics*. Princeton: van Nostrand.
23. Kleymann, T.: 1998, 'Hoare Logic and VDM: Machine-Checked Soundness and Completeness Proofs'. Ph.D. thesis, Edinburgh Univ. LFCS Technical Report ECS-LFCS-98-392.
24. LEGO: 1998, 'The LEGO Proof Assistant WWW page'. `http://www.dcs.ed.ac.uk/home/lego/`.

25. Luo, Z.: 1991, 'Program Specification and Data Refinement in Type Theory'. In: *TAPSOFT '91 (Volume 1)*. pp. 143–168.
26. Luo, Z.: 1994, *Computation and Reasoning: A Type Theory for Computer Science*, International Series of Monographs on Computer Science. Oxford University Press.
27. Luo, Z. and R. Pollack: 1992, 'LEGO Proof Development System: User's Manual'. Technical Report ECS-LFCS-92-211, Computer Science Dept., Univ. of Edinburgh. Updated version. See [24].
28. Martin-Löf, P.: 1971, 'A Theory of Types'. Technical Report 71-3, University of Stockholm.
29. McBride, C.: 1998, 'Inverting Inductively Defined Relations in LEGO'. In: E. Gimnez and C. Paulin-Mohring (eds.): *TYPES'96: Workshop on Types for Proofs and Programs, Aussois; Selected Papers*. To appear.
30. McKinna, J. and R. Pollack: 1993, 'Pure Type Systems Formalized'. In: M.Bezem and J.F.Groote (eds.): *Proceedings of the International Conference on Typed Lambda Calculi and Applications, TLCA'93, Utrecht*. pp. 289–305.
31. Mitschke, G.: 1979, 'The Standardisation Theorem for λ-calculus'. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* **25**, 29–31.
32. Nipkow, T.: 1996, 'More Church-Rosser Proofs (in Isabelle/HOL)'. In: *Automated Deduction – CADE-13*, Vol. 1104 of *LNCS*. pp. 733–747.
33. Pfenning, F.: 1992, 'A Proof of the Church-Rosser Theorem and its Representation in a Logical Framework'. Technical Report CMU-CS-92-186, Carnegie Mellon University.
34. Plotkin, G.: 1975, 'Call-by-name, call-by-value, and the λ-calculus'. *Theoretical Computer Science* **1**.
35. Pollack, R.: 1994a, 'Closure Under Alpha-Conversion'. In: H. Barendregt and T. Nipkow (eds.): *TYPES'93: Workshop on Types for Proofs and Programs, Nijmegen, May 1993, Selected Papers*, Vol. 806 of *LNCS*. pp. 313–332.
36. Pollack, R.: 1994b, 'The Theory of LEGO: A Proof Checker for the Extended Calculus of Constructions'. Ph.D. thesis, University of Edinburgh.
37. Pollack, R.: 1995, 'A Verified Typechecker'. In: M.Dezani-Ciancaglini and G.Plotkin (eds.): *Proceedings of the Second International Conference on Typed Lambda Calculi and Applications, TLCA'95, Edinburgh*.
38. Pollack, R.: 1998, 'How to Believe a Machine-Checked Proof'. In: G. Sambin and J. Smith (eds.): *Twenty Five Years of Constructive Type Theory*.
39. Prawitz, D.: 1965, *Natural Deduction; A Proof-Theoretical Study*, Stockholm Studies in Philosophy 3. Almqvist and Wiksell.
40. Reus, B.: 1995, 'Program Verification in Synthetic Domain Theory'. Ph.D. thesis, Ludwig-Maximilians-Universität München.
41. Reus, B.: 1996, 'Synthetic Domain Theory in Type Theory: Another Logic of Computable Functions'. In: *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs'96*, Vol. 1125 of *LNCS*. pp. 363–381.
42. Sato, M.: 1983, 'Theory of Symbolic Expressions, I'. *Theoretical Computer Science* **22**, 19–55.
43. Schreiber, T.: 1997, 'Auxiliary Variables and Recursive Procedures'. In: *TAPSOFT'97*, Vol. 1214 of *LNCS*.
44. Shankar, N.: 1988, 'A Mechanical Proof of the Church-Rosser Theorem'. *Journal of the ACM* **35**(3), 475–522.
45. Stoughton, A.: 1988, 'Substitution Revisited'. *Theoretical Computer Science* **17**, 317–325.

46. Takahashi, M.: 1995, 'Parallel Reductions in λ-Calculus (Revised version)'. *Information and Computation* **118**(1), 120–127.
47. Tasistro, A.: 1993, 'Formulation of Martin-Löf's Theory of Types with Explicit Substitutions'. Master's thesis, Chalmers University of Technology.
48. van Benthem Jutting, L.: 1993, 'Typing in Pure Type Systems'. *Information and Computation* **105**(1), 30–41.
49. van Benthem Jutting, L., J. McKinna, and R. Pollack: 1994, 'Checking Algorithms for Pure Type Systems'. In: H. Barendregt and T. Nipkow (eds.): *TYPES'93: Workshop on Types for Proofs and Programs, Nijmegen, May 1993, Selected Papers*, Vol. 806 of *LNCS*. pp. 19–61.